GOLDSTEIN/GOLDSTEIN

GOLDSTEINS IBM PC BUCH

Betriebssystem,
Programmierung,
Anwendungen

IBM

GOLDSTEIN/GOLDSTEIN
Goldsteins IBM PC-Buch

Larry Joel Goldstein
Martin Goldstein

Goldsteins IBM PC-Buch

übersetzt und bearbeitet von
Rüdiger Gritsch

Carl Hanser Verlag München Wien

Titel der Originalausgabe:
IBM Personal Computer
An Introduction to Programming and Applications

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Goldstein, Larry Joel:
[IBM-PC-Buch]
Goldsteins IBM-PC-Buch / Larry Joel Goldstein ; Martin Goldstein. Übers. u. bearb. von Rüdiger Gritsch.
- München ; Wien : Hanser, 1985.
Einheitssacht.: IBM personal computer ⟨dt.⟩
ISBN 3-446-13956-7
NE: Goldstein, Martin:; Gritsch, Rüdiger [Bearb.]

Satz und Druck: Appl, Wemding
Buchbinderische Verarbeitung: Wagner, Nördlingen
Umschlaggestaltung: Christa Quilici und Günther Gerstmayer
Printed in Germany

Vorwort zur 1. amerikanischen Ausgabe

Mit diesem Buch beabsichtigen wir, dem Computeranfänger beizubringen, wie er mit dem IBM Personalcomputer umzugehen hat. Die Entwicklung der Personalcomputer wird von uns als eine der erregendsten technischen Errungenschaften unserer Zeit angesehen. In der Tat verspricht der preiswerte Personalcomputer, die durch die Computer ausgelöste technische Revolution Millionen Menschen nahezubringen, ihnen zu zeigen, wie sich unsere Art zu denken, zu lernen, zu arbeiten und zu spielen ändern wird. Dieses Buch ist dazu bestimmt, als Einführung in diese Umwälzung unserer Lebensauffassungen zu dienen. Demgemäß verfolgt es zwei Absichten. Einmal soll es den Leser bzw. die Leserin in den Gebrauch des IBM Personalcomputers einführen und zweitens soll es einige der Methoden erläutern, wie er verwendet werden kann.

Wir wollen den Leser bzw. die Leserin durch das Buch hindurch führen und begleiten, beginnend von dem Augenblick an, in dem der Computer zum ersten Mal eingeschaltet wird. Wir wollen darüberhinaus aber auch die Anfangsgründe der Programmierung diskutieren und uns dabei der problemorientierten Programmiersprache BASIC bedienen. Da dieses Buch auch als Lehrbuch gedacht ist, sind in jedes Kapitel Aufgabengruppen aufgenommen worden. Die Lösungen ausgewählter Aufgaben sind bewußt an das Ende des Buches gestellt worden. Das Buch soll aber auch zum Selbststudium benutzt werden können. Deshalb sind in fast alle Abschnitte Fragen und persönliche Übungen eingestreut worden, die mit der Überschrift „Testübungen" versehen sind. Diese Testübungen sollen das Verständnis über den soeben durchgenommenen Stoff überprüfen; sie bilden gewissermaßen einen Leitfaden durch das Selbststudium. Die Antworten auf die Testübungen sind am Schluß des Abschnittes, zu dem sie gehören, aufgeführt.

Wir haben in diesem Buch mehr Themen abgedeckt, als die meisten anderen Bücher über die elementaren Grundlagen der Programmierung enthalten. Diese Handlungsweise stammt von unserer Überzeugung, daß der Anfänger zusätzlich zur Erlernung der BASIC-Sprache einen Überblick über möglichst viele Anwendungen aus dem täglichen Leben erwerben sollte. Deshalb haben wir viele Erörterungen solchen Anwendungen gewidmet, einschließlich eines kurzen Blickes auf die Textverarbeitung. Alle Diskussionen über die von uns gestreiften Anwendungen sind so abgefaßt, daß sie des Lesers Aufmerksamkeit wecken und schärfen; sie können als Einleitung für weitergehende Studien herangezogen werden.

Allzu begeisterte Benutzer von Personalcomputern statten sehr rasch ihre persönlichen Computer mit zusätzlichen Geräten unterschiedlichster Art aus. Aus diesem Grunde haben wir hier zusätzlich eine Einführung in das Disketten-BASIC sowie kurzgefaßte Besprechungen über Drucker und Verbindungsschnittstellen (Interfaces) aufgenommen. Zum Schluß des Buches bringen wir schließlich Hinweise auf Themen, die wir für anschließende Studien empfehlen.

Jedes Buch verdankt seine Existenz der Arbeit, der Mühe und den Eingebungen vieler Menschen. In unserem Fall wurden wir von unseren Frauen *Sandy* und *Rose* ständig unterstützt. Unsere Kinder bzw. Enkel *Melissa* und *Jonathan* gesellten sich begeistert hinzu, als es darum ging, mit unserem IBM Personalcomputer eine Vielzahl von Aufgaben zu lösen. Ihr Enthusiasmus und ihre gesunden kindlichen Standpunkte haben uns einen Schimmer der Zukunft der durch die Computer ausgelösten technischen Revolution vermittelt. Unser aufrichtiger Dank gilt den Rezensenten *Mel Hallerman, Parker Foley, Philipp Lemmons* und *Jessie Katz* für ihre gewissenhafte, sorgfältige Prüfung des Manuskripts und für ihre zahlreichen wertvollen Anregungen. Danksagungen gehen ebenfalls an *Michael Rogers,* dem verantwortlichen Redak-

teur, für seine professionelle Art und Weise, mit der er die Arbeit an diesem Buch und seine Produktion managte. Schließlich möchten wir auch *Harry Gaines,* dem Präsidenten der Brady Company, und *David Culverwell,* dem Chefredakteur der Brady Company, für ihre fortwährende Unterstützung danken. Ihre freundlichen Bemühungen förderten unseren Anreiz und unsere Arbeitslust beim Schreiben dieses Buches und kann deshalb als Modell für ideale Beziehungen zwischen Herausgebern und Autoren angesehen werden.

Dr. Larry Joel Goldstein
Silver Spring, Maryland

Martin Goldstein
West Palm Beach, Florida

Vorwort zur 2. amerikanischen Ausgabe

Die Welt der IBM Personalcomputer hat sich seit dem Erscheinen der ersten Ausgabe dieses Buches erheblich gewandelt. Das Betriebssystem DOS unterlag mehreren Verbesserungen. Neben dem ursprünglichen PC kam ein erweitertes Modell, PC/XT genannt, auf den Markt. Schließlich wurde auch die Programmiersprache BASIC für den PC erweitert. Diese Ausgabe berücksichtigt diese tiefgründigen Veränderungen.

Wir haben die Gelegenheit benutzt, um Inhalt und Themenkreis dieses Buches wie folgt zu ergänzen bzw. zu erweitern:

- Wir haben der Diskussion über das Betriebssystem DOS mehr Raum gewidmet.
- Wir haben weitere Programmiertips aufgenommen und sind hinweismäßig mehr als bisher auf die strukturierte Programmierung eingegangen.
- Wir haben die Besprechung der graphischen Datenverarbeitung ausgedehnt und einen Abschnitt über die Tonerzeugung hinzugefügt.
- Wir haben das Buch durch die Behandlung des Gebrauches der Funktionstasten, des Anhängens von Programmen, des Aufsuchens von Fehlern und von INKEY erweitert.
- Wir haben viele Programme überarbeitet und eine Reihe neuer aufgenommen.

Das Ziel dieser Ausgabe bleibt das gleiche wie das der ersten Ausgabe. Uns ging und geht es darum, eine klare, wohlbegründete, die Anwendungen betonende Einführung über Personalcomputer bereitzustellen.

Unser aufrichtiger Dank gilt allen Leserinnen und Lesern, die sich die Zeit genommen haben, um sich mit uns in Verbindung zu setzen und uns teilhaben ließen an ihren Ideen, ihrer Begeisterung, aber auch an ihren Enttäuschungen. Viele ihrer Gedanken und Anregungen fanden Eingang in diese Ausgabe.

Wieder einmal wollen wir unsere tiefste Anerkennung allen unseren Freunden bei der *Robert J. Bradey Co.* aussprechen und ihnen für ihre Bemühungen danken; dieser Dank bleibt bestehen, auch über das Erscheinen dieses Buches hinaus. Unsere ganz besonderen Danksagungen gelten *Mike Rogers* und *Tony Melendez* für ihre Verantwortung über den Herstellungsprozeß dieses Buches, *Jessie Katz* für die Unterstützung bei den Absprachen mit den Rezensenten und für ihre Hilfe bei der Abwicklung der Korrespondenz mit unseren Leserinnen und Lesern. Unser Dank gilt auch *Don Sellers* für die ideenreiche künstlerische Gestaltung dieses Buches, *George Dodson* für seine großartigen Photographien und *Joan Caldwell* für ihre Bemühungen in der Werbung und im Verkauf. Zuletzt, aber umso herzlicher, wollen wir unseren Dank den uns nahestehenden Freunden und Kollegen abstatten, die uns beraten haben, sowie *David Culverwell,* Chefredakteur, und *Harry Gaines,* Präsident, für ihre uneigennützige Hilfe und ihre nimmermüde Unterstützung.

Dr. Larry Joel Goldstein
Martin Goldstein

Vorwort zur deutschen Ausgabe

Nach nahezu 25jähriger Tätigkeit auf allen Gebieten der Datenverarbeitung von der Programmierung und Organisation bis hin zum Entwurf und der Realisierung komplexer Systeme kann man wohl mit Recht von sich behaupten, daß man die hauptsächlichen Entwicklungen und Anwendungen der Computer verfolgt hat.

Umfang und Komplexität der in der Praxis ständig auftauchenden Probleme und die Herausforderung, sie sinnvoll und nutzbringend zu lösen, verlangten stets den persönlichen Einsatz, das persönliche Engagement und die ständige Arbeitsbereitschaft. Es verblieb deshalb kaum Zeit, sich mit den kleinen Computern auseinanderzusetzen, die in den letzten Jahren verstärkt auf den Markt drängten. Allzu leichtfertig wurden sie oft geringschätzig als Heimcomputer betrachtet und zu bloßen Spielcomputern abqualifiziert. Den ihnen anfangs innewohnenden Fähigkeiten war man vor rund 30 Jahren bereits begegnet. Sie waren nur für einzelne Benutzer ausgelegt, sie boten nur dürftige Verarbeitungsmöglichkeiten, die Eingabe und die Ausgabe war auf bescheidene technische Konzeptionen beschränkt und sie ermöglichten auch nicht die Speicherung großer Datenmengen auf peripheren Speichern. Im Laufe der Jahre hatte man ja miterlebt, wie diese Barrieren durch konsequente technische und technologische Entwicklungen und Errungenschaften überwunden wurden. Es war aber kein Weg zurück, wie viele der in der Datenverarbeitung Beschäftigten zunächst meinten. Sie hatten infolge der rasant aufeinanderfolgenden Neuerungen einfach nicht beachtet, daß eine Fülle von Anwendungen auf ihre Erschließung durch Computer geradezu harrt, zumal wenn deren Einsatz neben einer Arbeitserleichterung auch eine präzisere, ermüdungsfreiere und mit weniger Fehlern behaftete Durchführung der Aufgaben verspricht. Aber für die Lösung isolierter, fachspezifischer, geringe Datenmengen beinhaltender oder rein persönlicher Aufgaben die Kapazitäten größerer Datenverarbeitungsanlagen bereitzustellen, und sei es auch nur im Teilnehmerbetrieb, konnte und kann nicht immer gerechtfertigt werden; deren Betriebs- bzw. Arbeitsweise entspricht vielfach nicht den Vorstellungen der potentiellen Benutzer.

Gewerbetreibende, Handwerker, Kaufleute, Vertreter, Berater usw. legen nach wie vor entscheidenden Wert darauf, ihre Geschäfte individuell und unabhängig führen zu können. Andererseits wollen sie aber auch nicht auf das gesamte gesammelte Wissen ihres Berufszweiges verzichten. Ihren Bedürfnissen, aber auch den Bedürfnissen zahlreicher Sachbearbeiter in den größeren Unternehmen kommt ein persönlicher Computer, ein Computer am Arbeitsplatz eher entgegen, ein Computer also, der alle notwendigen Fähigkeiten besitzt und der darüberhinaus im Bedarfsfall mit großen Datenverarbeitungsanlagen verbunden werden kann, auf deren Speichern umfassende Informationen abrufbereit zur Verfügung gehalten werden. – Die Mikrocomputer haben mittlerweile einen Entwicklungsstand erreicht, der die Inangriffnahme derartiger Anwendungen ermöglicht.

Die immensen Fähigkeiten, die bereits die heutigen Mikrocomputer auszeichnen, sollten jedoch nicht zu dem Fehlschluß führen, daß sie die größeren Datenverarbeitungsanlagen ablösen oder ersetzen können. Vielmehr wird sich gerade in der Zukunft zeigen, daß die Datenverarbeitung auf einer sachgerechten Symbiose von Klein- und Großcomputern fußt. Den Mikrocomputern wird dabei die Rolle zufallen, in alle unsere primären Lebensbereiche, angefangen bei Haushalt und Schule, einzudringen. Waren es am Anfang dieses Jahrhunderts die Autos, die zu einer ungeahnten Freiheit in der körperlichen Mobilität der Menschen führten, so werden es zukünftig die Computer sein, die zu einer ebenso ungeahnten Freiheit in der geistigen Mobilität des Menschen führen werden. Allerdings müssen wir auch eines festhalten: Gerade, weil wir heute zu erkennen beginnen, daß die gewonnene körperliche Mobilität uns

Menschen nicht nur Vorteile und Nutzen brachte, sondern auch Nachteile, sollten wir von vornherein aufpassen, daß uns der Einsatz der Computer niemals zum Schaden gereicht, uns niemals der gedanklichen Freiheit beraubt.

Computer richtig einzusetzen und sinnvoll zu benutzen, wird eine Aufgabe sein, vor die sich früher oder später die meisten berufstätigen Menschen gestellt sehen werden. Das Zugehörigkeitsgefühl zu einer elitären Gruppe, von dem sich noch heute viele Fachkräfte der Datenverarbeitung leiten lassen, wird weichen müssen. Grundlegendes Computerwissen möglichst rasch möglichst vielen Menschen zu vermitteln, wird zu einer der Hauptaufgaben werden müssen, mit der wir uns in der nächsten Zeit zu befassen haben werden. Eine umfassende Einweisung und Unterrichtung erfordert die Aufteilung des Wissensstoffes in einzelne Bausteine, die unabhängig voneinander nach Bedarf absolviert werden können. Aus diesem Gedanken heraus schuf das EDV-Bildungszentrum München im Rahmen der Control Data Institute eine zweckentsprechende Ausbildungslinie, die als Ziel den Erwerb eines Computerpasses in mehreren Stufen vorsieht, ein gewisses Analogon zum Erwerb des in mehrere Klassen eingeteilten Führerscheines also. Basisbausteine dieser wie auch anderer ähnlich gearteter Lehrgänge werden sicher Einführungen in die Hardware der Personalcomputer, in ein Betriebssystem und schließlich in die Programmierung sein.

Computerwissen zu lehren und zu vermitteln, kann auf unterschiedlichen Wegen erfolgen. Ein Weg, und sicher ein sehr guter, wird nach wie vor der klassische Weg sein. Ein fundiert aufgebautes, leicht verständliches Lehr- und Lernbuch wird nicht nur jede Art von Unterricht unterstützen können, sondern auch als Grundlage eingehender Selbststudien dienen können. Von diesen Gedanken haben sich auch L. und M. Goldstein leiten lassen. Der durchschlagende Erfolg ihres Buches im englischen Sprachraum hat ihre Bemühungen gerechtfertigt, zumal es die o. a. Basisbausteine voll abdeckt.

Die deutsche Ausgabe wurde gegenüber dem Original bearbeitet, um einerseits den deutschen Verhältnissen besser gerecht zu werden und um andererseits die neuesten Erkenntnisse berücksichtigen zu können. In diesem Zusammenhang sollte ausdrücklich erwähnt werden, daß ein großer Teil der mit der Erschaffung der deutschen Ausgabe verbundenen Arbeiten auf einem IBM Personalcomputer mit entsprechenden Softwareprodukten ausgeführt wurde, so z. B. die Erstellung des Sachregisters, zu der das Datenbankverwaltungssystem dBASE II herangezogen wurde.

An der Schaffung der deutschen Ausgabe dieses in den USA so außerordentlich erfolgreichen Werkes waren viele beteiligt. Besonderen Dank schulde ich meinem Würzburger Freundeskreis und den ungenannt bleiben wollenden Würzburger Studenten, die sämtliche anfallenden datentechnischen Arbeiten (Sachregister usw.) auf den ihnen in Würzburg zur Verfügung stehenden IBM Personalcomputern ausgeführt haben. Zu besonderem Dank bin ich auch M. Adamski verpflichtet, die die meisten Schreibarbeiten sehr gewissenhaft und sorgfältig durchführte. Anerkennend muß auch bemerkt werden, daß die beachtenswerten, in Würzburg durchgeführten Druckarbeiten der ins Buch aufgenommenen Listen und Programmauflistungen wesentlich die Gestaltung des Buches beeinflußten. Dem Carl Hanser Verlag gebührt das Verdienst, das Buch in einer Form herausgegeben zu haben, die uneingeschränkt zu loben ist.

Sollte sich im Verlauf der Zeit herausstellen, daß von den Lernenden und Studierenden die im Buch aufgeführten und behandelten Beispiele auf einer Diskette gespeichert zur Verfügung gestellt werden sollten, so behält sich der Carl Hanser Verlag ausdrücklich das Recht vor, eine solche Diskette zusätzlich zum Buch auf den Markt zu bringen.

Gilching und Würzburg im Mai 1985 Rüdiger Gritsch

Inhalt

Vorbemerkungen

1. **Hinweis für LeserInnen und Leser**

Einige der in diesem Buch behandelten Programme erfordern einen Bildschirm mit einer Anzeigenbreite von 80 Stellen pro Zeile. In dieser Hinsicht treten keine Schwierigkeiten auf, wenn die Schnittstelle für die einfarbige Anzeige und ein Schwarz-Weiß-Bildschirm vorhanden sind. Wenn jedoch der Personalcomputer entweder mit dem Farbmonitor der IBM oder mit dem eigenen Farbfernsehgerät verbunden ist, ist es erforderlich, nach dem Einschalten des Computers den Befehl (Command)

```
WIDTH 80
```

einzugeben, um zu gewährleisten, daß die Anzeigen genau so erscheinen, wie es im Buch beschrieben ist.

2. **Beschränkung der Haftpflicht und Ausschließung von Gewährleistungsansprüchen**

Autoren und Herausgeber dieses Buches bemühten sich nach besten Kräften, dieses Buch und die in ihm beschriebenen Programme gewissenhaft vorzubereiten und abzufassen. Hierzu sind die Bemühungen bei der Entwicklung, den Untersuchungen und dem Testen der Theorien und der Programme zu zählen; eingeschlossen ist dabei ebenfalls die Bestimmung ihrer Wirksamkeit. Autoren und Herausgeber geben keine Garantien jedweder Art, weder ausdrücklich noch stillschweigend inbegriffen, auf diese Programme und auf die Dokumentationen dieser Programme, soweit sie in dieses Buch aufgenommen wurden. Autoren und Herausgeber können nicht zur Verantwortung herangezogen werden, wenn in Verbindung mit diesen Programmen oder aus diesen Programmen heraus entstehend, zufällige oder fortlaufend bestehen bleibende Fehler auftreten sollten. Das gilt sowohl für die Auslieferung, das Leistungsvermögen als auch für die Benutzung dieser Programme.

3. **Hinweise auf Warenzeichen und Handelsmarken**

Hinsichtlich der Warenzeichen des in diesem Buch besprochenen Materials gilt folgendes und ist daher entsprechend zu beachten:

a) *IBM PC* und *IBM PC/XT* sind registrierte Warenzeichen der *IBM (International Business Machines Corporation), USA.*
b) *VisiCalc* ist ein registriertes Warenzeichen von *VisiCorp, Inc., USA.*
c) *CP/M-86* ist ein registriertes Warenzeichen von *Digital Research Corporation, USA.*
d) *UNIX* ist ein registriertes Warenzeichen von *Bell Telephone Laboraties, Inc., USA.*
e) *Lotus* ist ein registriertes Warenzeichen von *Lotus Development Corp., USA.*
f) *WordStar* ist ein registriertes Warenzeichen von *MicroPro International Corp., USA.*

1 Ein erster Ausblick auf die Computer

1.1 Einführung

Seit knapp 30 Jahren erst sprechen wir von einem Zeitalter der Computer, aber es wirkt sich bereits tiefschürfend auf unsere Lebensverhältnisse aus. Alltäglich werden wir mit ihnen bei Behörden, in Fabriken, ja sogar in Supermärkten konfrontiert. In den letzten drei oder vier Jahren drangen sie sogar in unsere Häuser, in unsere Wohnzimmer ein: Millionen von Computerspielen wurden gekauft, aber auch Hunderttausende von kleinen Computern für persönliche Zwecke. Diese kleinen Computer werden wegen dieser Verwendung heute als Personal- oder auch als Arbeitsplatzcomputer bezeichnet. Wir haben uns inzwischen so an die Computer gewöhnt, daß wir uns nicht einen Tag mehr vorstellen können, an dem wir nicht wenigstens mit einem von ihnen in Berührung gekommen sind.

Trotz des explosionsartig gewachsenen Einsatzes der Computer in unserer Gesellschaft wissen die meisten Menschen nur sehr wenig über sie Bescheid. Sie sehen einen Computer als ein „Elektronengehirn" an. Sie wissen dabei aber nicht, wie ein Computer arbeitet, wie er eingesetzt werden kann und wie sehr er uns die verschiedenartigsten täglichen Aufgaben erleichtern kann. Diese Tatsache zeugt jedoch nicht unbedingt von einem Mangel an Interesse. Die meisten Menschen haben längst erkannt, daß die Computer in immer weiterem Umfang unser Leben bestimmen werden und sind deshalb daran interessiert, ihre Arbeitsweise kennenzulernen, um herauszufinden, wie sie eingesetzt werden können. Für jeden, der sich mit diesen Gedankengängen beschäftigt, ist dieses Buch bestimmt.

Dieses Buch stellt eine Einführung in die Wirkungsweise und den Gebrauch von Personalcomputern dar, bestimmt für den Anfänger, mag er nun Student, Lehrer, Buchhalter, Geschäftsmann, Hausfrau oder irgendeine sonstige wißbegierige Person sein. Wir nehmen an, daß der Leser wenig oder gar keine Vorkenntnisse über Computer besitzt und die Grundlagen kennenlernen will. Wir wollen den Leser von dem Augenblick an geleiten, in dem er zum ersten Mal seinen IBM[1] PC[2] anschaltet. (Bis zu diesem Augenblick gibt es wirklich nichts zu überlegen!). Danach werden wir den Leser in die Grundlagen der BASIC-Sprache[3] einführen und ihm damit die Möglichkeit geben, den Computer anzusprechen. Durch Übungsaufgaben werden wir dafür Sorge tragen, daß der Leser fortwährend sein Verständnis des Textes überprüfen kann. Wir werden dem Leser durch sie auch die vielen Möglichkeiten aufzeigen, wie er seinen eigenen Computer benutzen kann. Die Übungsaufgaben werden den Leser somit sicher auch zum Schreiben von Programmen anregen, denn viele derselben sind so abgefaßt worden, daß sie einen Einblick in den Einsatz der Computer in Handel und Industrie gewähren. Wir werden ferner eine Fülle von Anregungen geben, wie man den Computer im eigenen Haushalt einsetzen kann. Bis zu einem vertretbaren Grade werden wir sogar Anreize vermitteln, ein paar Computerspiele selbst zu „basteln".

Was versteht man nun unter „persönlichem Rechnen"? Um diese Frage zu beantworten, wollen wir ein klein wenig die Entwicklungsgeschichte der Computer zurückverfolgen. In der ersten Zeit des automatisierten Rechnens, d. h. in den 40-er und 50-er Jahren unseres Jahrhunderts, bestand ein Computer aus einer riesigen Menge mechanischer und elektronischer Bauteile, die in mehreren Räumen untergebracht werden mußten. Damals war es oft notwen-

---

[1] IBM ist die Abkürzung von „**I**nternationale **B**üro**M**aschinen"
[2] PC ist die Abkürzung von „**P**ersonal **C**omputer"
[3] BASIC ist die Abkürzung von „**B**eginners **A**ll Purpose **S**ymbolic **I**nstruction **C**ode"

dig, die Böden der Räume zu verstärken, in denen die Computer aufgestellt werden sollten. Die Installation spezieller Klimaanlagen war erforderlich, um eine ordnungsgemäße Funktion der Rechenanlagen sicherzustellen. Zudem kosteten die Frühzeitcomputer mehrere Millionen DM. – Im Verlauf der Jahre sanken die Computerkosten in geradezu dramatischer Weise und, dank der Mikro-Miniaturisierung, schrumpfte ihr Umfang und damit ihre Größe sogar noch schneller, als ihr Preis fiel.

Ende der 70-er Jahre kamen die ersten Personalcomputer auf den Markt. Diese waren vergleichsweise verhältnismäßig preiswert und außerdem so entworfen, daß ein Durchschnittsmensch soviel über die Computer und ihren Einsatz erlernen konnte, um sie zur Lösung von Alltagsproblemen heranziehen zu können. Die Personalcomputer erwiesen sich als unglaublich populär und haben die Einbildungskraft der Menschen in allen Lebensbereichen in Bewegung gesetzt. Wenn Millionen von Menschen sich anschicken zu lernen, wie man den Computer in sein tägliches Leben einbezieht, kann man wohl ohne Übertreibung sagen, daß eine Computerrevolution im Gang ist.

Personalcomputer sollte man nicht als Spielzeug ansehen. Sie sind „echte" Computer, die die meisten Eigenschaften ihrer größeren Brüder, den sogenannten Haupt- oder Großcomputern, die nach wie vor Millionen DM kosten, besitzen. Personalcomputer können so ausgerüstet werden, daß ihre Kapazität ausreicht, um die steuernden Aufgaben des Rechnungswesens und der Lagerbestandsführung vieler kleinerer Geschäfte ausüben zu können. Ebenso können sie Berechnungen für Ingenieure und Wissenschaftler ausführen, und sie können sogar zur Verfolgung der häuslichen Finanzen herangezogen werden, um diese auf dem laufenden zu halten. Außerdem können sie z.B. die persönlichen Schreibarbeiten nachhaltig unterstützen. Es würde völlig unmöglich sein, eine umfassende Zusammenstellung aller Anwendungen zu geben, für die sich die Personalcomputer eignen. Die folgende Aufzählung kann deshalb nur eine Vorstellung darüber verschaffen, welchen breiten Spielraum die Anwendungen bereits heute umfassen.

● *Anwendungen für den Geschäftsmann*
- Rechnungsstellung
- Buchführung
- Schreibarbeiten
- Kassenführung und -verwaltung
- Vorbereitung und Erstellung von graphischen Darstellungen und Diagrammen
- Textverarbeitung
- Datenanalyse

● *Anwendungen für Zuhause*
- Haushaltsführung
- Verwaltung des Haushaltsgeldes
- Investmentanalyse
- Korrespondenz
- Haussicherheit

● *Anwendungen für den Studenten und Schüler*
- Bildung und Weiterbildung mit Hilfe des Computers
- Erstellung von Beleg-, Semester- und Jahresarbeiten
- Vorbereitung und Erstellung von graphischen Darstellungen und Diagrammen
- Arbeitspläne für Projekte
- Speicherung, Organisierung und Gliederung von Aufzeichnungen und Notizen

- *Anwendungen für und im Beruf*
  - Reklame
  - Marketing
  - Analyse von Daten
  - Erzeugung von Berichten
  - Korrespondenz
- *Anwendungen in der Freizeit, zur Entspannung und zur Unterhaltung*
  - Computerspiele
  - Computergraphik
  - Computerkunst

Wie man aus der Aufzählung ersehen kann, ist diese zwar reichlich umfassend, aber aus verständlichen Gründen unvollständig. Sollten spezielle, den einzelnen Leser interessierende Gebiete vermißt werden, so mache man sich darüber weiter keine Gedanken. Es ist auch genügend Freiraum für diejenigen gelassen, die gerade wißbegierig hinsichtlich der Computer sind und den Umgang mit ihnen als Hobby erlernen wollen.

Dieses Buch wird sich mit dem „IBM Personalcomputer"[1] beschäftigen. Diese Maschine ist ein unvorstellbar hochentwickeltes Gerät, das in sich viele der Eigenschaften vereinigt, die die Großcomputer aufweisen. Bevor wir mit der Diskussion der besonderen Eigenschaften des IBM Personalcomputers beginnen, wollen wir zunächst über die Eigenschaften sprechen, die allen Computern gemeinsam sind.

1.2 Was ist ein Computer?

Das Herzstück eines jeden Computers bildet die Zentraleinheit, kurz ZE genannt (englisch: Central Processing Unit, abgekürzt CPU), die die vom Benutzer gegebenen Befehle ausführt. Sie verrichtet also die arithmetischen Operationen, macht logische Entscheidungen usw. Somit ist die ZE ihrem Wesen nach das „Gehirn" des Computers. Der Speicher des Computers ermöglicht es ihm, sich an Zahlen, Wörter, Paragraphen genau so zu „erinnern" wie an eine Folge von Befehlen, die der Benutzer vom Computer ausführen lassen will. Die Eingabeeinheit gestattet es dem Benutzer, Informationen an den Computer zu senden; die Ausgabeeinheit hingegen erlaubt dem Computer die Überstellung von Informationen an den Benutzer. Die Beziehungen dieser vier Basiskomponenten eines Computers zueinander sind in der Abb. 1.1 dargestellt.

Bei einem IBM Personalcomputer besteht die ZE aus einem winzigen Scheibchen mit einer elektronischen Schaltung; ein solches Scheibchen wird als Chip (Mikrobaustein) bezeichnet. Dieser Chip führt den Namen „8088 Mikroprozessor". Für einen Computerneuling ist es nicht erforderlich zu wissen, wie die Elektronik der Computer funktioniert. Zu diesem Zeitpunkt sollte der Leser die ZE einfach als ein magisches Gerät, als einen schwarzen Kasten ansehen, dessen Inneres die Fähigkeiten eines Computers besitzt. Jeder andere Gedankengang sollte tunlichst vermieden werden, am besten gar nicht erst aufkommen.

Das Haupteingabegerät des IBM Personalcomputers ist die Computertastatur. Mit den speziellen Einrichtungen dieser Tastatur wollen wir uns im Abschnitt 1.5 beschäftigen. Zum jetzi-

[1] „IBM Personal Computer" ist ein registriertes Warenzeichen der „International Business Machines Corporation", USA

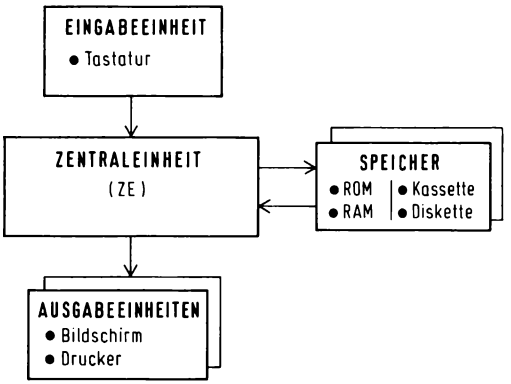


Abb. 1.1 Die Hauptkomponenten eines Computers

gen Zeitpunkt genügt es, wenn wir die Tastatur als simple Schreibmaschine betrachten. Durch Anschlagen von Symbolen auf der Tastatur werden die diesen zugeordneten Zeichen in den Computer eingegeben.

Der IBM Personalcomputer kann mit verschiedenen Ausgabegeräten ausgerüstet werden. Fast immer wird ein TV-Bildschirm benutzt, der mitunter auch als Videomonitor (kurz: Monitor) oder Videoanzeige bezeichnet wird. Natürlich kann auch ein Drucker angeschlossen werden, der Ausgaben „zu Papier" bringt. In der Fachsprache der Computerwelt ist dann von Hartkopien (engl.: Hard Copy, abgekürzt HC) die Rede.

Es gibt fünf verschiedene Speichertypen bei den IBM Personalcomputern: ROM, RAM, Kassette, Diskette und Festplatte oder Hartplatte. Jede dieser Speichertypen hat ihre Vorteile und ihre Nachteile. Wir sollten deshalb immer versuchen, den Speicher des Computers so vielseitig wie möglich zu machen, indem wir die guten Eigenschaften aller Typen miteinander kombinieren.

Die Kürzel ROM steht für „read only memory", d.h. für einen Speicher, der nur das Lesen zuläßt (Lesespeicher). Der Inhalt dieses Speichertyps kann vom Computer, genauer gesagt, von der ZE gelesen werden, aber es kann auf ihn nichts aufgezeichnet werden. Der ROM ist für die Computersprache reserviert, die die ZE verwendet. Diese Sprache wollen wir später diskutieren. Augenblicklich sollten wir uns nur merken, daß der ROM die Informationen enthält, die für den Computer notwendig sind, um die Befehle des Benutzers zu verstehen. Diese Informationen sind im Herstellungswerk im voraus aufgezeichnet worden und bleiben im ROM auf Dauer gespeichert.

Die Kürzel RAM steht für „random access memory", d.h. für einen Speicher, der einen wahlfreien (direkten) Zugriff zuläßt. Auf diesen Speicher kann auch geschrieben werden. Wenn der Benutzer auf der Tastatur Zeichen eingibt, werden sie im RAM gespeichert. Entsprechend dazu werden die Ergebnisse von Berechnungen im RAM festgehalten, d.h. sie warten dort auf ihre Ausgabe. Eine extrem wichtige Eigenschaft des RAM verdient gesondert erwähnt zu werden; der Leser möge sich an sie ständig erinnern:

Wenn der Computer ausgeschaltet wird, wird der RAM gelöscht

Wegen dieser Eigenschaft kann der RAM nicht zur permanenten Speicherung von Daten benutzt werden. Nichtsdestotrotz wird er infolge seiner großen Schnelligkeit als Hauptspeicher des Computers verwendet. Es dauert nur ungefähr eine einzige Mikrosekunde (Millionstel Sekunde), um ein Datenelement auf dem RAM zu speichern oder aus dem RAM herauszuholen. Die Kapazität, d.h. das Aufnahmevermögen des Hauptspeichers wird in Bytes oder Speicherstellen angegeben. Ein Byte nimmt normalerweise ein einziges Zeichen auf, beispielsweise A,! oder 4. Öfters wird man Aussagen der Art „der IBM PC von Herrn X ist mit einem RAM von 128k ausgerüstet" vernehmen. Das Kürzel k steht hier für die Zahl 1024. Somit bedeutet die Angabe 128k das Produkt 128 · 1024, d.h. der erwähnte Computer besitzt einen RAM von 131072 Bytes oder Speicherstellen.

Um permanente Kopien von Programmen oder Daten zu erzeugen, können wir entweder einen Kassettenrecorder oder ein Diskettenlaufwerk oder ein Festplattenlaufwerk (Hartplattenlaufwerk) verwenden. Der Kassettenrecorder ist weiter nichts als ein Bandaufzeichnungsgerät, das die Aufzeichnung von Informationen in einer Form ermöglicht, die der Computer versteht. Das Magnetband ist von der gleichen Art, wie es für gewöhnliche Musik- und Sprachaufzeichnungen benutzt wird.

Ein Diskettenlaufwerk zeichnet die Informationen auf flexiblen Disketten (Scheiben), die den Schallplatten ähneln, auf. Die Disketten werden wegen ihrer Biegsamkeit oftmals als „Schlaffplatten" oder „floppy disks" bezeichnet; jede kann mehrere Hunderttausend Zeichen speichern. (Nebenbei sei vermerkt, daß auf ein Blatt Schreibmaschinenpapier vom Format DIN A4 bis zu 70 Zeilen mit je 80 Zeichen, d.h. rd. 5600 Zeichen untergebracht werden können). Ein Diskettenlaufwerk ermöglicht im Durchschnitt den Zugriff zu Informationen in einer weitaus geringeren Zeit als ein Kassettenrecorder. Andererseits kosten Diskettenlaufwerke erheblich mehr als Kassettenrecorder.

Bei einer Hart- oder Festplatte, auch als „Winchesterplatte" bezeichnet, werden, wie es schon der Name besagt, die Informationen auf einer unbiegsamen Platte gespeichert. Diese befindet sich entweder in der Platteneinheit, d.h. im Plattengerät, selbst oder wird in einer festen Plastikhülle aufbewahrt. Winchesterplatten sind die teuersten Speichermedien des Computers. Sie erlauben jedoch zu den permanent gespeicherten Daten den schnellsten Zugriff überhaupt und können mindestens 5 Millionen Zeichen, maximal bis zu 100 Millionen Zeichen aufnehmen.

In diesem Buch wollen wir uns auf die „Straßenmitte" konzentrieren, d.h. auf ein Computersystem mit mittlerer Ausrüstung einstellen, auf einen Computer also mit einem Diskettenlaufwerk oder mit zwei Diskettenlaufwerken.

1.3 Die Bestandteile des IBM Personalcomputers

Bevor wir den Computer anschalten, wollen wir uns mit den verschiedenen Bausteinen des Systems vertraut machen. In der Abbildung 1.2 sind die drei Grundkomponenten dargestellt:

- Monitor
- Tastatur
- Systemeinheit

Wir wollen annehmen, daß der Leser der Anleitung der IBM gefolgt ist und die verschiedenen Kabelverbindungen zwischen diesen Einheiten richtig bewerkstelligt hat. Dabei ist zu beachten, daß die Computerkabel so ausgelegt sind, daß sie nur in einer Richtung eingesteckt wer-

Abb. 1.2 Der IBM Personalcomputer

den können; die Chancen, beim Stecken der Verbindungen Fehler zu machen, werden dadurch auf ein Minimum reduziert.

Wir sollten besonders erwähnen, daß wir den Monitor auf die Systemeinheit plaziert haben; dadurch erreichen wir ein bequemeres Sehen. Falls der Benutzer es wünscht, kann er selbstverständlich den Monitor auch neben die Systemeinheit stellen. Dem Benutzer obliegt also die Wahl der Anordnung der Komponenten; er sollte sie in jedem Fall jedoch so treffen, daß er seinen Computer praktisch, leicht und bequem bedienen kann. Man bedenke, daß die Benutzung eines Systems über mehrere Stunden hinweg die Augen sehr anstrengt und damit eine Ermüdung herbeiführt, sofern nicht für einen ausreichenden Bedienungskomfort gesorgt ist.

Die Tastatur ist mit der Systemeinheit durch eine gerollte elastische Leitungsschnur verbunden. Dadurch kann die Tastatur dorthin gestellt werden, wo sie für den Benutzer am bequemsten zu bedienen ist. Wir sollten zudem auf die Stützen hinweisen; durch sie kann die Tastatur in einem Winkel zur Tischebene schräggestellt werden. Wir haben es also auch hier mit einer Einrichtung zu tun, die die „menschlichen Faktoren" berücksichtigt, d.h. die dazu beitragen soll, Ermüdungserscheinungen bei der Benutzung hinauszuzögern. Die verschiedenen anderen Einrichtungen der Tastatur werden wir im nächsten Abschnitt besprechen.

Von den Grundbausteinen ist die Systemeinheit am geheimnisvollsten. Deshalb wollen wir nunmehr ihren Aufbau untersuchen. Auf ihrer Vorderseite befinden sich die Diskettenlaufwer-

ke. Jedes Laufwerk weist einen Schlitz auf, in den eine Diskette eingesteckt werden kann. Um die Verriegelung eines Laufwerkes zu öffnen, sollte der Benutzer einen Finger in die Vertiefung unterhalb des Schlitzes legen und behutsam die Verriegelung nach vorn ziehen (siehe Abb. 1.3). Die Verriegelung öffnet sich dadurch nach oben. In dieser Position der Verriegelung kann nun die Diskette in den Schlitz eingeführt werden, mehr darüber später. Durch einen Fingerdruck auf die Verriegelung nach unten und nach hinten ist diese anschließend wieder in ihre ursprüngliche Lage zurückzuführen und damit zu schließen.

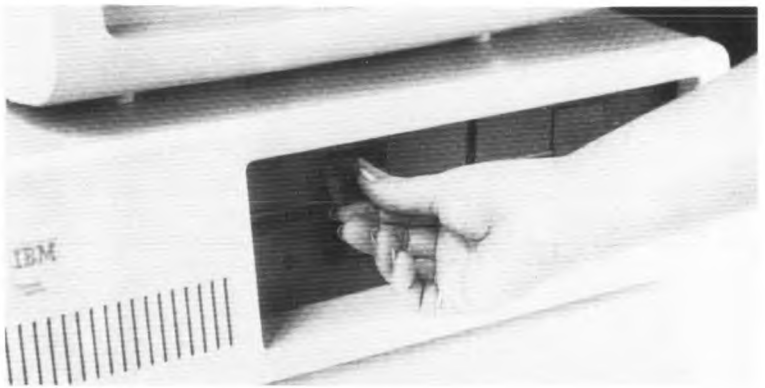


Abb. 1.3 Öffnen der Verriegelung eines Diskettenlaufwerkes

In Abb. 1.4 zeigen wir die Rückseite der Systemeinheit. Man beachte hierbei besonders, daß auf ihr viele Anschlüsse vorhanden sind, in die Kabel verschiedener Ausmaße und Formen hineingesteckt werden können. Die Systemeinheit, die dem Leser zur Verfügung steht, kann unterschiedlich aussehen, abhängig von den jeweils installierten Geräten. Die in der Abbil-

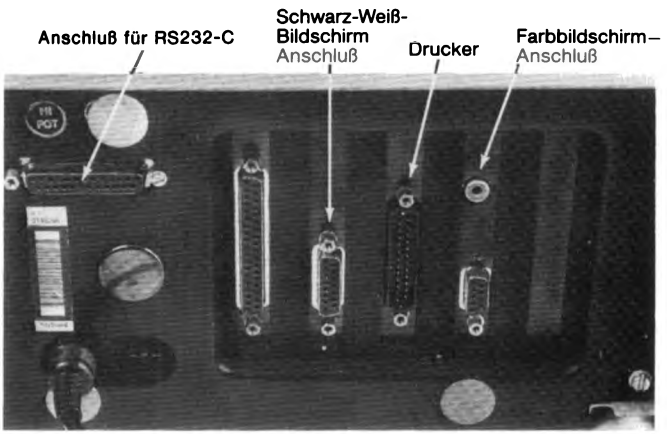


Abb. 1.4 Ansicht der Rückseite der Systemeinheit

dung dargestellte Systemeinheit weist Anschlüsse für einen Drucker, für einen Schwarz-Weiß-Bildschirm und einen Farbbildschirm auf und ist zudem mit einer Kommunikationsschnittstelle, d.h. einen Anschluß RS232-C (mehr darüber später), ausgerüstet.

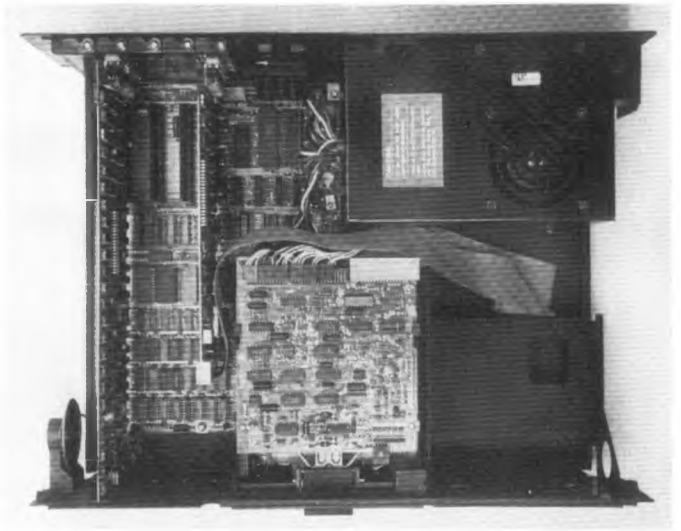


Abb. 1.5 Das Innere des IBM Personalcomputers

In der Abb. 1.5 wird das Innere des Computers gezeigt. (Ist der Leser ein Neuling, so möge er bitte nicht das Gehäuse des Computers abnehmen, er möge sich vielmehr an die Abbildung halten). Der große Bauteil in der Ecke enthält die Spannungsquelle und ein Kühlgebläse. Man beachte auch den Lautsprecher in der linken unteren Ecke. Die Systemplatine, die die elektronische Hauptschaltung des Computers enthält, ist horizontal an der Unterseite des Computers untergebracht. Der 8088-Chip befindet sich auf dieser Platine. Die vertikalen Platinen sind optionale Bestandteile, die in Fassungen auf der Systemplatine eingesteckt werden können, die Erweiterungsschlitze genannt werden. Man kann aus einem ansehnlichen Vorrat von Schaltungen auswählen und dadurch die Fähigkeiten des eigenen Computers erweitern. Die Platinen in der Abb. 1.5 gehören von links nach rechts zu folgenden Komponenten:

- Schnittstelle zum Schwarz-Weiß-Bildschirm
- Speichererweiterung
- Diskettensteuerung
- Schnittstelle zum Farbbildschirm und zur graphischen Darstellungsmöglichkeit

Um dem Leser einen Eindruck über die Komplexität dieser Platinen zu vermitteln, zeigen wir in der Abb. 1.6 als abgeschlossenes Ganzes die Schnittstelle für den Farbbildschirm und die graphische Darstellungsmöglichkeit. In der Abb. 1.7 ist in gleicher Weise ein ROM-Chip dargestellt; zum Vergleich der Größenverhältnisse liegt daneben eine Büroklammer.

Man darf sich beim Anblick der elektronischen Schaltung nicht bange machen lassen. Um den eigenen Computer zu gebrauchen, bedarf es keiner Kenntnisse über die Arbeitsweise der Schaltkreise.

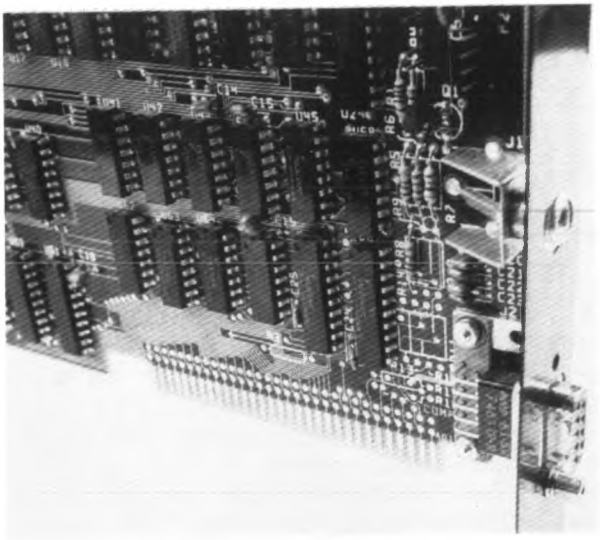


Abb. 1.6 Schnittstelle für Farbe und Graphik

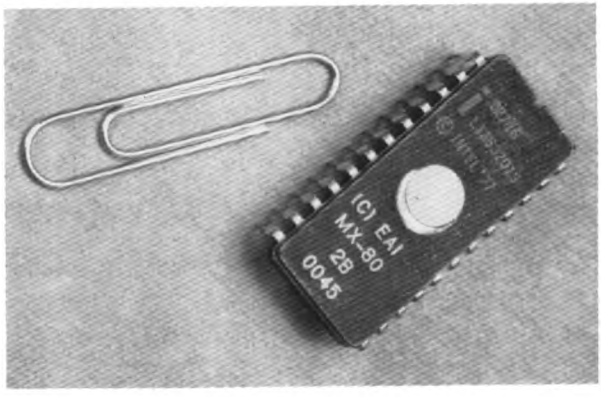


Abb. 1.7 ROM-Chip

1.4 Der Gebrauch von Diskettenlaufwerken

Bei einem Computersystem stellen die Diskettenlaufwerke wichtige Bestandteile dar, denn sie ermöglichen mittels der in sie eingelegten Disketten die Speicherung und das Wiederaufsuchen sowohl von Daten als auch von Programmen. Bevor wir mit dem Stoff fortfahren, wollen wir uns deshalb zunächst mit diesen für den Betrieb der Personalcomputer so wichtigen Bauteilen vertraut machen.

1.4.1 Der Aufbau der Disketten

Zur Speicherung der Informationen dienen bei den Diskettenlaufwerken flache Scheiben (Disketten) mit einem Durchmesser von 5,25 Zoll (13,34 cm). Auf jeder Diskette können annähernd 163000 Zeichen untergebracht werden, was nahezu dem Inhalt von 60 Schreibmaschinenseiten im Format DIN A4 entspricht (mit einer Zeile Zwischenraum zwischen je zwei Textzeilen).

Die Abb. 1.8 spiegelt die wesentlichen Teile einer Diskette wider. Die Diskette selbst ist eine kreisförmige flache Scheibe aus Mylar-Plastikmaterial, die frei in einer steifen Hülle rotiert. Die Hülle dient zum Schutz der Diskette. Das Innere der Hülle enthält ein Gleitmittel, das dazu beiträgt, daß die Diskette ungehindert in der Hülle rotieren kann. Man sollte niemals versuchen, die schützende Hülle gewaltsam zu öffnen. Durch auf den Hüllen angebrachte Etiketten läßt sich der Inhalt der Disketten jederzeit ausreichend identifizieren.

Das Diskettenlaufwerk liest von der bzw. schreibt auf die Diskette mittels eines Lese/Schreib-Spaltes, auch Lese/Schreib-Fenster genannt. Man sollte unter keinen Umständen die Oberflächen der Disketten berühren. Disketten sind sehr empfindlich. Ein Staubkorn oder sogar nur ein wenig Fett von einem Fingerabdruck können bei Disketten bereits Schaden anrichten und sich auf die aufgezeichneten Informationen nachteilig bis hin zu ihrer völligen Unbrauchbarkeit auswirken.

Durch die Schreibschutzkerbe kann verhindert werden, daß die auf der betreffenden Diskette aufgezeichneten Informationen geändert werden. Wenn die Kerbe nicht durch eines der mit der Diskette mitgelieferten metallischen Schildchen zugedeckt ist, kann der Computer von der Diskette lesen, aber es ist ihm nicht möglich, Informationen auf die Diskette zu schreiben oder bereits aufgezeichnete Informationen zu ändern. Um auf eine Diskette schreiben zu können, muß die Schreibschutzkerbe abgedeckt sein.

Man beachte, daß die von der IBM mitgelieferte Diskette mit dem Betriebssystem DOS keine Schreibschutzkerbe aufweist; somit kann auf sie niemals geschrieben werden. Auf dieser Diskette befinden sich die Kopien der Originale einiger sehr wichtiger Programme; bei diesen haben jegliche Änderungen zu unterbleiben.

1.4.2 Verhaltensmaßregeln bei der Handhabung von Disketten

Wie schon gesagt, sind Disketten extrem empfindlich. Deshalb werden hier jetzt einige Tips für ihre sorgfältige Behandlung gegeben:

1. Bei Nichtgebrauch sollten die Disketten stets in ihre Schutztasche (aus Papier) gesteckt werden.
2. Wie Schallplatten sollten auch die Disketten in vertikaler Stellung aufbewahrt werden.
3. Die Oberflächen von Disketten sollten niemals mit bloßen Händen berührt noch sollten sie mit einem Taschentuch, einem Handtuch oder irgendeinem anderen Lappen abgewischt werden.

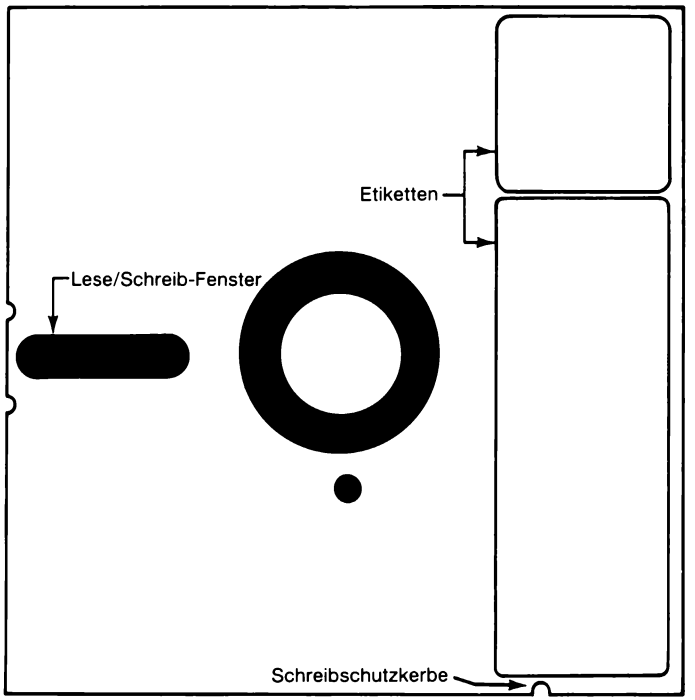


Abb. 1.8 Diskette

4. Disketten sollten niemals großer Wärme ausgesetzt werden; sie sollten sich also unter keinen Umständen in der Nähe von Heizkörpern befinden noch starker Sonneneinstrahlung unterliegen noch sonst irgendwie von Wärmequellen beeinflußt werden können.
5. Eine Diskette sollte niemals gekrümmt oder gebogen werden.
6. Zur Beschriftung von Diskettenetiketten sollten nur Filzschreiber benutzt werden; kein Schreibwerkzeug mit scharfer Spitze sollte dazu herangezogen werden. Die Etiketten sollten stets vor dem Aufkleben beschriftet werden.
7. Disketten sollten von magnetischen Feldern ferngehalten werden; es ist hierbei an solche gedacht, die von elektrischen Motoren, von Rundfunk- und Fernsehgeräten, von Kassettenrecordern und von anderen elektrischen Geräten erzeugt werden.
8. Von einem rotierenden Laufwerk sollte niemals eine Diskette abgenommen werden. Man erkennt ein sich bewegendes Laufwerk daran, daß der Motor summt und daß die Betriebsanzeigelampe rot aufleuchtet. Geschieht das trotzdem, so kann ein irreparabler Schaden angerichtet werden.

Die soeben zusammengestellten Vorsichtsmaßregeln können durchaus Anfänger überfordern. Wenn sich jedoch erst zweckmäßige persönliche Verfahren bei der Handhabung von Disketten eingespielt haben, wird man ohne weiteres erkennen, daß Disketten ein zuverlässiges, betriebssicheres Speichermedium von hoher Lebensdauer darstellen.

1.4.3 Verwendung von Disketten

Um eine Diskette in ein Laufwerk plazieren zu können, ist es zunächst notwendig, die Verriegelung des Laufwerkes zu öffnen. Die Diskette ist so zu drehen, daß die Etikettenseite oben liegt und die Schreibschutzkerbe auf der Benutzerseite liegt. Anschließend schiebe man die Diskette behutsam in das Laufwerk, bis ein leichtes Klicken zu hören ist. Nach dem Schließen der Laufwerkverriegelung kann nunmehr die Diskette vom Computer angesprochen werden.

Um die Diskette aus dem Laufwerk zu entfernen, muß man zunächst darauf achten, daß die Anzeigelampe links unterhalb des Schlitzes verloschen ist. Nach Öffnen der Verriegelung kann die Diskette vorsichtig aus dem Laufwerk herausgezogen werden.

1.4.4 Starten des Computers

Um den Informationsfluß zu und von den Diskettenlaufwerken steuern und kontrollieren zu können, benötigen wir ein Programm, das Betriebssystem genannt wird. Ein solches Programm wirkt wie ein Manager, der alle Aktivitäten innerhalb des Computers lenkt und der den Informationsfluß zwischen der Tastatur, dem Bildschirm, dem RAM, dem ROM, den Diskettenlaufwerken und jedweden anderen peripheren Geräten, die an den Computer zusätzlich angeschlossen sind, koordiniert. Die IBM stellt ein Betriebssystem bereit, das „IBM DOS", was die Abkürzung von „IBM Disk Operating System (Plattenbetriebssystem)" darstellt, genannt wird, auch als MS-DOS[1], PC-DOS[2] oder kurz als DOS bekannt. Eine Originalfassung dieses Betriebssystemes ist auf einer DOS-Diskette gespeichert, die man zur gleichen Zeit käuflich erwerben kann, wenn man Disketten ersteht.

Zu Beginn aller Arbeiten mit dem Computer muß man zuerst die deutsche Fassung des DOS in den Computer einlesen. Wenn wir zukünftig einfach vom DOS sprechen, meinen wir stets die deutsche Fassung des DOS. Die deutsche Fassung des DOS muß man zunächst einmal aus der käuflich erworbenen Originalfassung des DOS generieren. Wie man dabei vorzugehen hat, ist im Abschnitt 1.5 ausführlich beschrieben. Wir wollen für die weiteren Ausführungen voraussetzen, daß der Generierungslauf bereits erfolgt ist und wir folglich über eine Diskette mit der deutschen Fassung des DOS verfügen. Nach dem Einlesen managt das DOS alle weiteren, von uns dem Computer übertragenen Arbeiten. Zum Einlesen des DOS ist das folgende Verfahren einzuhalten:

1. Die Diskette, auf der das DOS aufgezeichnet ist, ist in das linke Diskettenlaufwerk einzustecken (bzw. in das einzige Diskettenlaufwerk, über das der entsprechende Computer verfügt), wobei die Etikettenseite nach oben zeigen muß. Die Diskette ist nun solange nach hinten in das Laufwerk einzuschieben, bis ein leichtes Klicken zu vernehmen ist. Anschließend ist die Verriegelung des Laufwerkes zu schließen.
2. Der Monitor ist einzuschalten. (Bei Vorhandensein des einfarbigen IBM-Bildschirmes braucht man nichts zu unternehmen, da dieser automatisch eingeschaltet wird).
3. Wenn ein Drucker zum Computer gehört, ist dieser nunmehr ebenfalls einzuschalten.
4. Hinten auf der rechten Seite der Systemeinheit (dem Gerät, in dem die Diskettenlaufwerke untergebracht sind) befindet sich der rote Kippschalter zum Einschalten des Computers (siehe Abb. 1.9); er ist in die obere Stellung, die mit I beschriftet ist, umzulegen.

---

[1] MS ist die Abkürzung von „**M**icro**s**oft Corp."
[2] PC ist die Abkürzung von „**P**ersonal **C**omputer"

Abb. 1.9 Ein/Aus-Schalter des Computers

Nach 30 bis 45 Sekunden sollte der Computer mit der Nachricht und mit der Systemanfrage

```
Datum ist:  (TT-MM-JJ): 01-01-1980
Neues Datum eingeben:
```

reagieren.

5. Nunmehr ist das gegenwärtige Datum im angeforderten Format, beispielsweise 13-5-82, einzugeben (Beispiel: 13. Mai 1982). Danach ist die Eingabetaste zu drücken, die große Taste rechts in der Mitte der Tastatur mit dem Symbol ←┘
   Beispiel: Eingabe von 12-10-84

6. Im Anschluß daran fordert der Computer zu einer weiteren Eingabe auf; er zeigt nämlich auf dem Monitor den folgenden Text an (Nachricht und Systemanfrage):

```
Zeit ist:  00:03:14
Neue Zeit eingeben:
```

Nehmen wir einmal an, es ist gerade 16 Uhr 39 Minuten und 7 Sekunden; in diesem Fall geben wir ein:

```
16:39:07
```

Die Uhrzeit ist also im Format

```
hh:mm:ss
```
(Stunden:Minuten:Sekunden)

einzutasten. Auf die Eingabe der Sekunden (und auch der Minuten) kann durch Weglassung von :ss bzw. :mm:ss verzichtet werden.

7. Nach Drücken der Eingabetaste kommt der Computer mit einer Meldung auf den Benutzer zu (dieses Mal ist keine Antwort fällig):

```
The IBM Personal Computer DOS
Version 2.00 (C) Copyright IBM Corp. 1981,1982,1983
A>
```

Beide Nachrichtenzeilen werden eingeleitet durch das hier nicht aufgeführte Wort REM; REM steht für „remark" (Bemerkung). Damit wird angedeutet, daß es sich um einen Kommentar handelt.

Die Zeichenfolge A> wird als DOS-*Systemanfrage* oder DOS-*Systemmeldung* bezeichnet und sagt dem Benutzer, daß das Betriebssystem DOS geladen und damit in der Lage ist, die Befehle (Commands) des Benutzers zu akzeptieren. Immer, wenn die genannte Zeichenfolge erscheint, dürfen wir einen Befehl eingeben. In manchen Fällen findet man auch den Begriff „Rückmeldung" statt „Systemmeldung".

8. Zu diesem Zeitpunkt wollen wir den Befehl zum Laden der Programmiersprache BASIC eingeben. Zu dieser Sprache gehört ein Programm auf der DOS-Diskette; dieser Sprache wollen wir im weiteren Verlauf dieses Buches unsere Aufmerksamkeit widmen. Zum Laden von BASIC ist der Computer durch Eingabe des Befehles

```
BASIC
```

aufzufordern.
Nach Drücken der Eingabetaste wird der Computer mit einer Antwort aufwarten, die der nachstehenden gleichkommt oder ähnelt:

```
The IBM Personal Computer Basic
Version D2.00 Copyright IBM Corp.1981, 1982, 1983
61330 Bytes Free
Ok
```

Die Zeichenfolge Ok wird als BASIC-*Systemanfrage* (BASIC-*Rückmeldung*) bezeichnet. Sie zeigt dem Benutzer an, daß die Computersprache (BASIC genannt) bereit zum Empfang von Befehlen und/oder Anweisungen ist. In der dritten Zeile wird angezeigt, wieviel Stellen (Bytes) des Hauptspeichers noch frei für eigene Zwecke des Benutzers sind. Die mitgeteilte Zahl hängt natürlich von der Kapazität (dem Aufnahmevermögen) des Hauptspeichers ab, der in unserem Fall so groß war, daß 61330 Bytes frei sind. Der schmale blinkende Strich wird *Positionsanzeiger* oder *Cursor* genannt. Er markiert die Stelle auf dem Bildschirm, auf der das nächste eingegebene Zeichen erscheinen wird.

9. Wenn man von der Programmiersprache BASIC zum Betriebssystem DOS zurückzukehren beabsichtigt, ist einfach der Befehl

```
I SYSTEM
```

einzugeben. Nach Drücken der Eingabetaste erscheint dann wieder die DOS-Systemanfrage A> auf dem Bildschirm.

Anmerkung: Alle Eingaben können in Großbuchstaben oder in Kleinbuchstaben oder gemischt in Groß- und Kleinbuchstaben erfolgen.
Im Text dieses Buches wollen wir jedoch alle Eingaben, die den Vorschriften des Betriebssystemes DOS bzw. der Computersprache BASIC entsprechen, durch Benutzung von Großbuchstaben hervorheben.
Bei der Eingabe von Zahlen können die führenden Nullen stets weggelassen werden.

Testübungen zur Überprüfung des Verständnisses 1.4.1[1)]

a) Der Computer ist einzuschalten und das Betriebssystem DOS zu laden.

b) Die Programmiersprache BASIC ist zu laden.

c) Es ist die Rückkehr zum Betriebssystem DOS zu veranlassen.

1.4.5 Weitere Ausführungen zu den Diskettenlaufwerken

Der IBM Personalcomputer kann mit bis zu vier Diskettenlaufwerken ausgerüstet werden. In der Systemeinheit können ein oder zwei Laufwerke eingebaut sein. Die weiteren Laufwerke sind in einem gesonderten Gehäuse untergebracht, das über eine Kabelverbindung an die Systemeinheit angeschlossen wird; sie werden deshalb als externe Laufwerke bezeichnet.

Den einzelnen Diskettenlaufwerken sind die Namen

A:
B:
C:
D:

gegeben (man beachte bei den Namen die zu ihnen gehörenden Doppelpunkte). Das linke Laufwerk in der Systemeinheit heißt A:, das rechte B:. Die externen Laufwerke besitzen die Namen C: und D:. Die Laufwerknamen müssen dann benutzt werden, wenn man sich in einem Befehl auf ein Laufwerk beziehen muß oder will.

In jedem Augenblick ist ein Laufwerk als „*Standardlaufwerk*" *(laufendes Laufwerk)* festgelegt. Bleibt bei der Eingabe eines Befehles in diesem der Laufwerkname unerwähnt, unterstellt das

---

[1)] Die Testübungen zur Überprüfung des Verständnisses werden zukünftig nur kurz als „Testübungen" bezeichnet. Die Antworten zu den Übungen folgen jeweils am Schluß des jeweiligen Abschnittes.

Betriebssystem von sich aus, daß sich der Benutzer auf das Standardlaufwerk beziehen will. Beim erstmaligen Einschalten des Computers ist als Standardlaufwerk das mit A: bezeichnete Laufwerk vorgegeben.

Die Systemanfrage des DOS verweist immer auf das Standardlaufwerk. Wenn die Systemanfrage

```
A>
```

erscheint, dann gilt das mit A: benannte Laufwerk als Standardlaufwerk. Erscheint auf dem Bildschirm dagegen B>, dann wird das mit B: bezeichnete Laufwerk als Standardlaufwerk angesehen.

Wenn das Standardlaufwerk geändert werden soll, ist das folgende Vorgehen zu absolvieren:

1. Die DOS-Systemanfrage

   A> oder B>

   ist abzuwarten.

2. Der Name des Laufwerkes, das ab diesem Augenblick als Standardlaufwerk betrachtet werden soll, ist einzugeben und die Eingabetaste zu drücken.

Testübungen 1.4.2

a) Der Computer ist einzuschalten und das Standardlaufwerk von A: auf B: zu ändern.

b) Das Standardlaufwerk ist anschließend von B: auf A: zu setzen, also wieder auf den Ausgangszustand zu bringen.

1.4.6 Formatieren und Kopieren von Disketten

Der Benutzer wird den Gegebenheiten der Praxis am besten gerecht, wenn von allen Disketten Kopien vorliegen. Dadurch kann man sich effektiv vor dem Verlust von Programmen und Daten schützen. Zu einem solchen Verlust kann es rein zufällig kommen, sei es durch einen Spannungsausfall oder -abfall, sei es durch Verschütten von Kaffee auf eine Diskette oder durch ein anderes unglückseliges Ereignis. Es ist sogar sehr sinnvoll, sich eine Kopie der Diskette mit dem Betriebssystem DOS zuzulegen, und zwar sofort bei ihrer erstmaligen Benutzung. Späterhin sollte man dann nur noch die Kopie verwenden. Die Originaldiskette mit dem DOS sollte man an einem sicheren Platz aufbewahren, so daß von ihr jederzeit erneut eine Kopie gezogen werden kann, falls die erste Kopie schadhaft geworden ist. Nachfolgend wollen wir das Verfahren kennenlernen, das uns eine solche Ausweichkopie verschafft. Das Verfahren selbst wird für *Ein-Laufwerk-Systeme* beschrieben; bezüglich der *Zwei-* bzw. *Mehr-Laufwerk-Systeme* verweisen wir auf den Kommentar im Anschluß an die Verfahrensbeschreibung. Das Verfahren gliedert sich in zwei Hauptschritte.

I. Formatieren einer Diskette

Um eine Diskette verwendungsbereit zu präparieren, müssen auf sie zunächst die „elektronischen" Grenzen der Spuren geschrieben werden. Dieser Prozeß wird Formatieren genannt; er läuft wie folgt ab:

1. Die weiter oben beschriebene Startprozedur (siehe Unterabschnitt 1.4.4) ist mit Ausnahme der Anforderung von BASIC durchzuführen. Nach Erscheinen der Systemanfrage A> ist der Befehl

```
FORMAT A:
```

einzugeben. Danach ist die Eingabetaste zu betätigen.

2. Das Betriebssystem DOS antwortet mit der Nachricht:

```
Neue Diskette einlegen in Laufwerk A:
und anschl. eine Taste betätigen
```

3. Die Diskette mit dem DOS ist aus dem Laufwerk herauszunehmen und durch eine leere Diskette von 5,25 Zoll Durchmesser zu ersetzen. Danach ist irgendeine Taste zu drücken.

4. Nunmehr geht der Computer zum Formatieren der leeren Diskette über. Die während und nach Beendigung des Formatierens erscheinenden Nachrichten könnten wie folgt aussehen:

```
Formatieren läuft ...                          (während der Formatierung)
Formatieren beendet                            (nach Beendigung der Formatierung)
Weitere Diskette formatieren (J/N)?
```

Durch Beantwortung der Anfrage mit N, was für „Nein" steht, zeigen wir an, daß wir keine weitere Diskette formatieren wollen. Die leere Diskette ist nunmehr formatiert, d.h. es kann jetzt auf sie geschrieben werden.

5. Zum Schluß ist die formatierte Diskette aus dem Laufwerk herauszunehmen und an ihrer Stelle die Diskette mit dem DOS wieder ins Laufwerk einzustecken.

Kommentar zu Zwei-Laufwerk-Systemen

Das soeben beschriebene Verfahren für das Formatieren einer Diskette gilt für Ein-Laufwerk-Systeme. Bei Zwei-Laufwerk-Systemen läßt sich die Vorgehensweise vereinfachen, indem man die Leerdiskette in das Laufwerk B: einschiebt und gemäß der obigen Punkte verfährt. Beim anschließenden Kopieren entfällt dann die Notwendigkeit, die Disketten im Laufwerk ständig auszuwechseln.

II. Kopieren des Inhalts einer Diskette auf eine andere

Der nächste Hauptschritt bewirkt das eigentliche Kopieren, d.h. der Übernahme des Inhalts der DOS-Diskette auf die soeben formatierte Leerdiskette. Dieser Hauptschritt wird im folgenden beschrieben.

1. Die Leerdiskette ist aus dem Laufwerk zu entnehmen und die DOS-Diskette einzuschieben. Der Befehl

```
DISKCOPY
```

ist einzugeben und die Eingabetaste zu drücken. Der Computer antwortet mit der Nachricht:

```
Quellendiskette einlegen in Laufwerk A:
Wenn bereit, eine Taste betätigen
```

2. Die Ausgangs- oder Quellendiskette, nämlich die DOS-Diskette, befindet sich bereits im Laufwerk A:. Somit kann sofort irgendeine Taste angeschlagen werden. Der Computer kopiert nun einen Teil des DOS in den RAM. Nachdem er soviel wie möglich kopiert hat, d.h. soviel er aufgrund des zur Verfügung stehenden RAM-Platzes kann, schickt er folgende Nachricht auf den Bildschirm:

```
Zieldiskette einlegen in Laufwerk A:
Wenn bereit, eine Taste betätigen
```

Die DOS-Diskette ist nun zu entnehmen und die formatierte Leerdiskette einzuschieben. Danach ist irgendeine Taste auf der Tastatur zu drücken. Der Computer wird daraufhin die im RAM zwischengespeicherten Daten auf die Diskette übertragen. Wenn weitere Daten von der DOS-Diskette kopiert werden müssen, müssen die Schritte 1 und 2 mehrere Male wiederholt werden. Nach dem Kopieren aller Daten zeigt der Computer die folgende Nachricht an:

```
Kopieren beendet
Eine weitere Kopie erstellen (J/N)?
```

Durch die Antwort N (für nein) sagen wir, daß wir keine weitere Diskette kopieren wollen. Der Computer wird daraufhin mit der normalen DOS-Systemanfrage

```
A>
```

reagieren.

3. Die formatierte Diskette enthält nunmehr eine exakte Kopie des Inhalts der Originaldiskette, hier der DOS-Diskette. Auf die DOS-Systemanfrage hin kann nunmehr wie üblich ein neuer DOS-Befehl oder die Anforderung von BASIC eingegeben werden.

Kommentar zu Zwei-Laufwerk-Systemen

Um den Inhalt der Diskette A: auf die Diskette B: zu kopieren, ist so vorzugehen, wie es beschrieben wurde; jedoch lautet der einzugebende Befehl

```
DISKCOPY A: B:
```

Man beachte hierbei, daß in diesem Befehl der Name des Quellenlaufwerkes, d.h. des Laufwerkes, in das die Diskette mit dem zu kopierenden Inhalt, eingeschoben ist, zuerst genannt werden muß. Das Ziellaufwerk hingegen, das Laufwerk also, in das die Diskette eingeschoben ist, auf die die Kopie des Inhalts der Quellendiskette aufzuzeichnen ist, ist erst an zweiter Stelle zu erwähnen.

Testübung 1.4.3

Es ist eine Kopie der DOS-Systemdiskette zu erzeugen.

Anmerkung:

Wenn die Version 1.10 oder eine spätere Version des Betriebssystemes DOS eingesetzt wird, kann das Formatieren weggelassen werden, sofern von einer Diskette auf eine andere Diskette kopiert werden soll.

Das Formatieren ist hier in den Befehl DISKCOPY mit eingeschlossen. Wenn jedoch eine Diskette vorzubereiten ist, auf die später Programme oder Daten aufgezeichnet werden sollen, dann muß der Formatierungsschritt nach wie vor ausgeführt werden.

1.4.7 Ratschläge für die Verwaltung der Disketten

Das soeben beschriebene Sicherungsverfahren für die Diskette mit dem Betriebssystem DOS kann in gleicher Weise dazu dienen, den Inhalt einer beliebigen anderen Diskette durch Kopieren zu sichern.

Da Disketten äußerst empfindlich sind, ist es von enormer Bedeutung, daß man sich von allen Disketten, auf denen relevante Programme oder Daten gespeichert sind, Duplikate anfertigt und aufbewahrt. Vorteilhaft ist es, wenn man unmittelbar nach einer Sitzung am Computer sich Kopien derjenigen Disketten anfertigt, deren Inhalt während der Sitzung verändert worden ist. Bei einer solchen Vorgehensweise mag man nur allzu ungern an eine Art „großen Bruder" denken, aber man kann sich dadurch unermeßlichen Kummer ersparen, falls durch irgendeine ungewollte Panne eine Diskette, die „kritische" Programme oder Daten enthält, gelöscht oder beschädigt wird.

Antworten zu den Testübungen

1.4.1 a) Man befolge die Punkte 1. bis 7. des im Unterabschnitt 1.4.4 beschriebenen Verfahrens.
b) Man befolge den Punkt 8. des im Unterabschnitt 1.4.4 beschriebenen Verfahrens.
c) Man befolge den Punkt 9. des im Unterabschnitt 1.4.4 beschriebenen Verfahrens.

1.4.2 a) Nach Einschalten des Computers ist die DOS-Systemanfrage abzuwarten. Danach ist

```
B:
```

einzugeben und die Eingabetaste zu betätigen.

b) Es ist

```
A:
```

einzugeben und die Eingabetaste zu betätigen.

1.4.3 Es ist nach dem im Unterabschnitt 1.4.6 beschriebenen Verfahren vorzugehen.

1.5 Generierung der deutschen Fassung des Betriebssystems DOS

Aus der käuflich erworbenen Diskette und der Originalfassung des DOS ist zunächst eine deutsche Fassung des DOS zu generieren.
Zur Anpassung der Originalfassung des DOS an deutsche Verhältnisse ist wie folgt zu verfahren.

I. Vorbereitung der Generierung

Die Diskette mit der Originalfassung, zukünftig als Quellendiskette (Source disk) bezeichnet, ist in der geschilderten Weise ins linke Laufwerk einzuschieben. Nach einer gewissen Zeit erscheint auf dem Bildschirm eine Anleitung, die aufzeigt, daß die Generierung der deutschen Fassung des DOS in fünf Schritten vorzunehmen ist. Um diese fünf Schritte durchführen zu können, muß eine Leerdiskette bereitgelegt werden, auf die die deutsche Fassung des DOS aufgezeichnet werden soll. Wenn wir zukünftig vom DOS reden, so meinen wir stets diese deutsche Fassung (bzw. die Diskette mit der deutschen Fassung). Die bereitgelegte Leerdiskette wird in den durchzuführenden Schritten stets als Zieldiskette (Target disk) bezeichnet.

II. Durchführung der Generierung

Die einzelnen Schritte werden fortlaufend auf dem Bildschirm angezeigt und beschrieben. Im Text der Beschreibung kommt ständig das Wort „REM" vor. Dieses Wort stellt eine Abkürzung des englischen Wortes „Remark" dar, das hier soviel wie „Bemerkung" heißt. Es ist stets unbeachtet zu lassen.

1. Schritt 1: Kopieren
   Im ersten Schritt wird der Inhalt der Quellendiskette auf die Zieldiskette kopiert. Während des Kopierens erscheinen auf dem Bildschirm Nachrichten, die man gewissenhaft verfolgen sollte, da sie den Fortgang der Operationen beschreiben, die zum Kopieren erforderlich sind.
   Da im Schritt 1 auch mehrere Kopien gleichzeitig angefertigt werden können, endet dieser Schritt mit der Frage

   ```
   Eine weitere Kopie erstellen (J/N)?
   ```

   J steht für „Ja", N für „Nein". Wir antworten mit N, da wir uns mit einer Kopie begnügen wollen. Jede Eingabe kann entweder mit Großbuchstaben oder mit Kleinbuchstaben erfolgen.

2. Schritt 2: Vergleichen
   Im zweiten Schritt wird der Inhalt der Zieldiskette mit dem Inhalt der Quellendiskette verglichen, um sicherzustellen, daß die Zieldiskette den gleichen Inhalt wie die Quellendiskette aufweist. Der erste Schritt wird also überprüft.
   Beim zweiten Schritt spielt es keine Rolle, in welchen Laufwerken Quellen- bzw. Zieldiskette sich befinden. Wir belassen sie zweckmäßig in den Laufwerken, in denen sie sich bereits befinden. Auch während der Durchführung des zweiten Schrittes erscheinen auf dem Bildschirm Nachrichten, die den Fortgang des Vergleichens signalisieren. Der zweite Schritt endet mit der Frage (Schlußnachricht):

   ```
   Weitere Disketten vergleichen (J/N)?
   ```

Da wir im ersten Schritt nur eine Kopie erstellt haben und somit keine weitere Diskette zu vergleichen haben, geben wir n oder N ein.

3. Schritt 3: Maßnahmen im Fehlerfall
   Wenn der Schritt 2 erfolgreich verlaufen ist, angezeigt während des zweiten Schrittes durch die Fortgangsnachricht

   ```
   Diskettenvergleich ok
   ```

   auf dem Bildschirm vor der Schlußnachricht, ist im dritten Schritt nichts zu tun, d.h. es kann sofort der 4. Schritt durchgeführt werden. Dazu ist nur eine beliebige Taste anzuschlagen.
   Sind hingegen Fehler aufgetreten, so muß nach der auf dem Bildschirm angezeigten Anleitung vorgegangen werden, die praktisch beinhaltet, daß eine neue Zieldiskette zu benutzen ist (Wiederholung der bisherigen Schritte), da die bisherige Zieldiskette unbrauchbar ist.

4. Schritt 4: Diskettenwechsel
   Die Originalfassung des DOS wird nunmehr nicht mehr benötigt. Die Quellendiskette wird also aus dem linken Laufwerk nach Öffnen der Verriegelung herausgezogen, in ihre Aufbewahrungstasche gesteckt und zur sicheren Aufbewahrung weggelegt. Die Zieldiskette hingegen wird aus dem rechten Laufwerk entnommen und ins linke Laufwerk eingeführt. Wenn diese Manipulationen erledigt sind, ist irgendeine Taste auf der Tastatur anzuschlagen.

5. Schritt 5: Anpassung an deutsche Verhältnisse
   Auf der im linken Laufwerk befindlichen Zieldiskette befindet sich nach Absolvierung der ersten vier Schritte nichts anderes als eine gewissenhaft überprüfte Kopie der Originalfassung des Betriebssystemes DOS.
   Erst im 5. Schritt wird durch vorgefertigte Prozeduren aus der Originalfassung die deutsche Fassung erzeugt.
   Gemäß der auf dem Bildschirm erschienenen Nachrichten betätigen wir zunächst eine beliebige Taste auf der Tastatur. Dadurch wird der Bildschirm gelöscht und anschließend auf der ersten Bildschirmzeile die Nachricht

   ```
   1=USA 2=Francais 3=Deutsch 4=Italiana
   5=Español 6=English 0=exit
   ```

   angezeigt, hier der Übersicht wegen in zwei Zeilen niedergeschrieben. Durch Eingabe der Kennung 3 zeigen wir an, daß wir die Generierung einer deutschen Fassung wünschen. Nach der Eingabe erscheinen auf den Zeilen 2 und 3 des Bildschirmes folgende Nachrichten:

   ```
   Prüfen: Ist die deutsche DOS-Diskette im Laufwerk A?
   Um fortzufahren, eine Taste betätigen
   ```

   Da das linke Laufwerk als Laufwerk A gilt, und sich die Zieldiskette, hier als „deutsche DOS-Diskette" bezeichnet, darin befindet, können wir sofort irgendeine Taste anschlagen.
   Die Anpassung dauert einige, für den Benutzer lang erscheinende, Sekunden und endet mit dem Starten der generierten deutschen Fassung des Betriebssystemes.

Anmerkungen:

1. Das linke Laufwerk in der Systemeinheit wird im Betriebssystem DOS als Laufwerk A: bezeichnet; die Laufwerkbezeichnung des rechten Laufwerkes ist B:.

2. Eines der beiden Laufwerke (A bzw. B) gilt als Standardlaufwerk. Welches Laufwerk momentan als Standardlaufwerk gilt, wird auf dem Bildschirm ständig durch die DOS-Systemmeldung (DOS-Systemanfrage)

```
        A>
bzw.    B>
```

angezeigt.

1.6 Die Tastatur

Am schnellsten lernt man die Bedienung des Computers kennen und beherrschen, wenn man dieses Buch, am Computer sitzend, liest und die einzelnen Aussagen sofort beim Studieren ausprobiert. Der Leser sollte also vor dem Computer Platz nehmen und ihn zunächst einmal einschalten. Danach sollte er, wie im vorhergehenden Abschnitt beschrieben, das Betriebssystem DOS und danach BASIC laden. Falls ein Computer nicht verfügbar sein sollte, möge sich der Leser auf die Abb. 1.10 stützen.

Wir wollen uns zunächst einmal die Tastatur anschauen. Ohne Zweifel ähnelt sie der Tastatur einer Schreibmaschine; einige wesentliche Unterschiede sind allerdings bereits auf den ersten Blick zu erkennen. Bei vielen Schreibmaschinen ist die gleiche Taste für die Ziffer 1 und für den Kleinbuchstaben l vorgesehen. Beim Computer darf jedoch keine Zweideutigkeit vorhanden sein. Infolgedessen gibt es für diese zwei verschiedenen Zeichen auch zwei verschiedene Tasten. Ebenso kann es auch leicht zu Verwechslungen zwischen dem Großbuchstaben O und der Ziffer 0 kommen. Um hier Verwechslungen zu vermeiden, wird beim Schreiben gewöhnlich die Ziffer 0 mit einem durch sie hindurchführenden Schrägstrich geschrieben, also zu Ø. Um Irrtümern von vornherein vorzubeugen, sollte man sich unbedingt diese Schreibweise zu eigen machen, sofern es wir mit Zeichensätzen zutun haben, die keinen Unterschied zwischen dem Großbuchstaben O und der Ziffer 0 kennen.

Man beachte, daß die Tastatur eine ganze Reihe von Sondertasten, die nicht zur standardisierten Schreibmaschinentastatur gehören, aufweist. Davor brauchen wir nicht gleich zu erschrekken. Die Funktion und die Bedeutung dieser Tasten werden wir zu gegebener Zeit besprechen und erläutern.

Um ein Gefühl für die Tastatur zu bekommen, wollen wir zunächst einige beliebige Tasten anschlagen. Beim Anschlagen erscheinen die den Tasten entsprechenden Zeichen auf dem Bildschirm. Dabei wandert der Positionsanzeiger (Cursor) auf dem Bildschirm über die

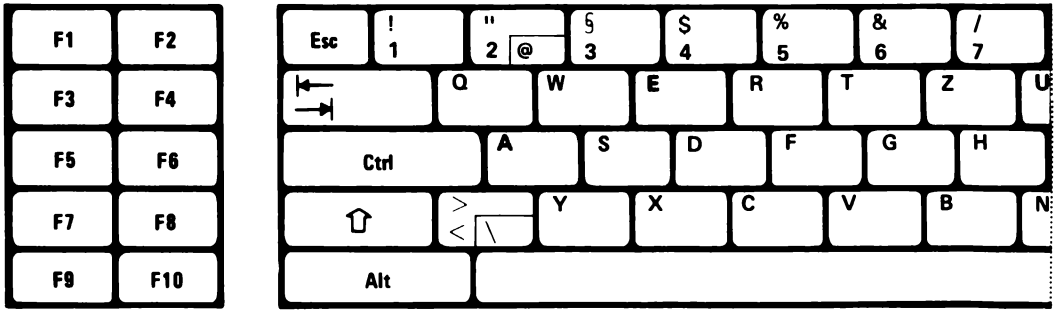


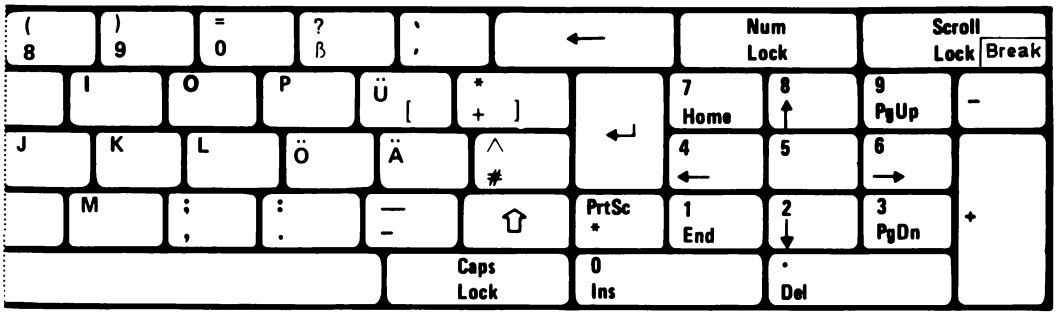
Abb. 1.10 Die Tastatur des IBM Personalcomputers

Schreibzeile hinweg. Er sitzt stets auf der Stelle, auf der das nächste eingegebene Zeichen angezeigt wird.

Beim Eintippen der Zeichen wird man sicher die Ähnlichkeiten zwischen den Tastaturen von Computer und Schreibmaschine bemerken. Jedoch sollte man auch die Unterschiede erkennen. Bei Schreibmaschinen hat man am Ende einer Zeile für den Wagen- bzw. Schreibkopfrücklauf zu sorgen, entweder manuell oder durch das Drücken der Rücklauftaste. Natürlich weist der Bildschirm keinen Wagen und keinen Schreibkopf auf, der auf den Anfang der nächsten Zeile zurückgeführt werden muß. Man muß vielmehr dem Computer sagen, daß man auf die nächste Zeile übergehen will. Das wird einfach durch das Anschlagen der *Eingabetaste (Taste ENTER)* bewerkstelligt. Die Eingabetaste ist mit dem Symbol ↵ bezeichnet und befindet sich rechts auf der Tastatur. Dadurch wird der Positionsanzeiger (Cursor) auf den Anfang der nächsten Zeile (äußerste linke Position auf dem Schirm) eingestellt. Die Eingabetaste besitzt darüberhinaus noch eine andere Funktion. Sie signalisiert nämlich dem Computer, daß er die gerade eingetippte Zeile übernehmen soll: bis zum Anschlagen der Eingabetaste existiert die eingetippte Zeile für den Computer nicht.

Wir bleiben beim Eintippen, bis wir die letzte Bildschirmzeile gefüllt haben. Wenn wir jetzt die Eingabetaste anschlagen, wird zusätzlich der gesamte Bildschirminhalt um eine Zeile nach oben gerückt, d.h. die erste bisher auf dem Bildschirm angezeigte Zeile verschwindet. Dieser Vorgang wird B i l d b e w e g u n g genannt. Die vom Bildschirm verschwundene Zeile geht natürlich nicht verloren. Ihr Inhalt verbleibt im RAM und kann mittels des dafür geeigneten Befehles zurück auf den Bildschirm geholt werden.

Wie der Benutzer bereits festgestellt haben wird, antwortet der Computer auf einige oder auf alle der eingetippten Zeilen mit einer Fehlernachricht (error message). Über diese sollte man sich zu diesem Zeitpunkt noch keine Gedanken machen. Dem Computer ist „beigebracht" worden, daß er nur auf bestimmte eingegebene Befehle reagieren soll. Wenn ihm also eine Kette von Zeichen begegnet, die er nicht als Befehl auslegen kann, muß er mit einer Fehlernachricht antworten. Es ist außerordentlich wichtig, daß sich der Benutzer im klaren darüber ist, daß diese Fehlernachrichten in keiner Weise die Arbeitsweise des Computers beeinträchtigen. – Tatsächlich kann man kaum etwas tun, was dem Computer Schaden zufügen würde, es sei denn durch physischen Mißbrauch seiner Komponenten. Man sollte sich also keinesfalls durch die gelegentlichen, vom Computer aus dem „Handgelenk" ausgeteilten „Schläge" entmutigen lassen. Was auch immer geschieht, man sollte sich dadurch nicht vom Experi-



mentieren abhalten lassen. Das Schlimmste, was jemals passieren kann, ist, daß man den Computer abschalten und anschließend wieder von vorn mit der Startprozedur beginnen muß.

Bis zu diesem Zeitpunkt haben wir ein rechtes Wirrwar auf dem Bildschirm erzeugt. Um den Bildschirm zu löschen, haben wir gleichzeitig die beiden Tasten zu drücken, die die Aufschriften *Ctrl*[1] (Control, d.h. Steuerung) bzw. *Home* aufweisen. Dadurch werden alle Zeichen auf dem Bildschirm gewissermaßen „ausradiert", nur der Positionsanzeiger (Cursor) bleibt zurück. Dieser wird jedoch in die linke obere Ecke des Schirms gestellt, die deshalb auch als „Heimposition" (Home) gilt.

Die Tastatur des IBM Personalcomputers zeichnet sich durch mehrere andere, sehr interessante Einrichtungen aus. So erscheint u.a. jede der Ziffern 0 bis 9 zwei Mal auf der Tastatur (siehe Abb. 1.11). Einmal tauchen die Zifferntasten an ihrem gewohnten Platz in der oberen Tastenreihe auf und ein zweites Mal auf der rechten Tastaturseite, wo sie wie bei einem Tischrechner angeordnet sind und somit eine leichtere und bequemere Eingabe der Ziffern gestatten. Es ist völlig gleichgültig, welche Tastengruppe für die Eingabe der Ziffern benutzt wird. Tatsächlich kann man zwischen den beiden Tastengruppen willkürlich wechseln, z.B. zuerst eine 1 durch Drücken der entsprechenden Taste der oberen Tastenreihe eingeben, dann eine 5 durch Anschlagen der Taste auf der rechten Seite usw. Die zusammengefaßten Zifferntasten der rechten Tastengruppe werden als „numerischer Block" bezeichnet.

Die Tasten des numerischen Blockes erfüllen zwei Aufgaben. Sie dienen nämlich auch zum Edieren, d.h. zum Aufbereiten und Ändern von bereits eingegebenen Texten. Wir werden erst im Kapitel 4 behandeln, was bei IBM Personalcomputern beim Edieren zu beachten ist. Wir wollen uns zunächst nur merken, daß durch die mit *Num Lock* (numeric lock, d.h. numerische Eingabesperre) beschriftete Taste gesteuert wird, welche Funktionen den Tasten des numerischen Blockes zugeordnet sind. Wenn die Taste *Num Lock* eingerastet ist, funktioniert der numerische Block wie die Tastatur eines Tisch- oder eines Taschenrechners. Ist diese Taste jedoch ausgekuppelt, so ist der numerische Block tauglich fürs Edieren. Beim Einschalten des Computers wird der numerische Block auf die Funktionen gesetzt, die man zum Edieren benötigt. Somit ist es notwendig, daß vorm erstmaligen Gebrauch des numerischen Blockes zur Eingabe von Ziffern die Taste *Num Lock* eingerastet wird.

Es gibt weiterhin auf der Tastatur zwei *Umschalttasten*; das sind die beiden Tasten mit dem Symbol ⇧ . (Diese Tasten dürfen nicht mit der Taste des numerischen Blockes verwechselt werden, die mit dem Symbol ↑ markiert ist). Die beiden Umschalttasten wirken genau so wie die Umschalttasten von Schreibmaschinen, d.h. sie ermöglichen die Eingabe von Großbuchstaben bzw. der oberen Zeichen derjenigen Tasten, die mit zwei Symbolen versehen sind. So befindet sich z.B. in der oberen Tastenreihe eine Taste, die mit den Symbolen ! (oben) und 1 (unten) markiert ist. Um das Ausrufungszeichen eingeben zu können, muß man also diese Taste anschlagen, während man zugleich eine der beiden Umschalttasten niederdrückt.

Bei den meisten Anwendungen, zu denen der Computer herangezogen wird, ist es sinnvoller, beim Eintippen nur Großbuchstaben zu benutzen, denn diese sind nun einmal größer und daher deutlicher und leichter auf dem Bildschirm zu erkennen. Man kann die Eingabe von Klein-

---

[1] Die Taste *Ctrl* wirkt praktisch wie eine Umschalttaste. Durch ihr Niederdrücken können andere Tasten Alternativfunktionen ausüben. Beim Gebrauch solcher Tasten ist es ratsam, erst die Taste *Ctrl* niederzudrücken, bevor man die betreffende Taste selbst anschlägt.

Abb. 1.11 Numerischer Block

buchstaben gänzlich unterdrücken, indem man die Taste *Caps Lock* niederdrückt. Unter dieser Vorbedingung werden die angeschlagenen Buchstabentasten automatisch Großbuchstaben „abliefern". Hierbei gilt es jedoch zu beachten, daß die Nichtbuchstabentasten, wie z.B. die Taste mit den beiden Symbolen ! und 1, nach wie vor ihre zwei Bedeutungen behalten. Um das Zeichen einzutippen, das zum oberen Symbol solcher Tasten gehört, muß nach wie vor eine der Umschalttasten mit betätigt werden. Um die reine Großbuchstaben-Arbeitsweise wieder zu verlassen, muß die Taste *Caps Lock* (capitals lock, d.h. Eingabesperre für Großbuchstaben) erneut gedrückt werden.

Testübungen 1.6.1

a) Es ist der eigene Name so einzutippen, daß er auf dem Bildschirm erscheint.

b) Der Bildschirm ist zu löschen.

c) Die Übung a) ist zu wiederholen, jedoch sind dabei nur Großbuchstaben einzugeben.

Anmerkung: Bei diesen Übungen kümmere man sich nicht um die Antworten des Computers als Reaktion auf die Eingaben.

Beim Eingeben wird man mitunter Tippfehler machen, es sei denn, man ist eine Schreibkraft der absoluten Spitzenklasse (die meisten von uns sind gerade das Gegenteil). So laßt uns also über die Korrektur solcher Fehler sprechen. Dazu tippe man zunächst einige willkürliche Zeichen ein, schlage jedoch die Eingabetaste anschließend *nicht* an. Man drücke vielmehr auf die *Rücktaste* (backspace) oben auf der Tastatur. Diese Taste trägt das Symbol ←; diese Taste darf nicht verwechselt werden mit der Taste des numerischen Blockes, die unten mit dem

Symbol ← beschriftet ist. Das Drücken der Rücktaste hat zur Folge, daß der Positionsanzeiger um eine Stelle zurückgesetzt wird. Dabei wird gleichzeitig das Zeichen gelöscht, das er bei dieser Rückwärtsbewegung passiert. Wir haben es hier demnach mit einem weiteren Unterschied zwischen den Tastaturen von Schreibmaschinen und Computern zu tun. Man merke sich jedoch unbedingt, daß das Rücksetzen zur Korrektur von Zeilen nur dann brauchbar ist, wenn die Zeile noch nicht zum Computer gesendet ist, d.h. die Eingabetaste noch nicht betätigt wurde.

Wenn man sich hoffnungslos verfahren hat und deshalb von vorn beginnen will, betätigt man die Tasten *Ctrl, Alt* und *Del* gleichzeitig. Diese Tastenkombination führt den Computer auf den Betriebszustand zurück, den er unmittelbar vor dem Einschalten aufwies. Sowohl der RAM als auch der Bildschirm werden dadurch gelöscht.

Es gibt andere Möglichkeiten, um Tippfehler zu beseitigen. Bei diesem Stand unserer Kenntnisse wollen wir es aber bei der soeben diskutierten Methode bewenden lassen.

1.6.1 Zeilenbreite

Der IBM Personalcomputer arbeitet mit Zeilen, die entweder eine Schreibbreite von 40 oder von 80 Stellen aufweisen. Um von einer Zeilenbreite auf die andere umzuschalten, müssen wir den Befehl WIDTH einsetzen. Um zu 40-stelligen Zeilen überzugehen, haben wir somit den Befehl

```
I   WIDTH 40
```

einzugeben. Zur Rückkehr auf 80-stellige Zeilen ist deshalb folgerichtig der Befehl

```
I   WIDTH 80
```

einzutippen. (Nebenbei bemerkt, wir sollten nicht vergessen, nach der Befehlseingabe die Eingabetaste zu betätigen!). Bei den anschließenden Besprechungen wollen wir grundsätzlich von einer Zeilenbreite von 80 Stellen ausgehen. Bei einer Zeilenbreite von 40 Stellen werden die Anzeigen, die auf dem Bildschirm erscheinen, etwas anders aussehen als die dargestellten.

1.6.2 Anzeigen von Funktionstasten

Nach kurzer Benutzungszeit des Computers werden wir schon gemerkt haben, daß die letzte Bildschirmzeile mit Daten gefüllt ist, die sich während des Eintippens nicht verändern. Diese Daten zeigen die Zuweisungen zu bestimmten, vom Benutzer programmierbarer Tasten an; die sogenannten *Funktionstasten* F1 bis F10 sind auf der linken Seite der Tastatur angeordnet.

Die Funktionstasten können so programmiert werden, daß bei ihrem Niederdrücken gewisse, oft benutzte Folgen von Tastenanschlägen oder Wörtern erzeugt werden. Sicherheitshalber sollte man als Anfänger vorerst auf diesen Vorteil verzichten. Deshalb auch ist es höchstwahrscheinlich am besten, wenn man ganz und gar die Anzeige des Status der vom Benutzer programmierbaren Tasten, d.h. der Funktionstasten, unterdrückt. Das Nichterscheinen dieser Zeile erreicht man durch die Eingabe des Befehles

```
I   KEY OFF
```

Nachfolgend ist wie üblich die Eingabetaste zu drücken. Wünscht man die Wiederanzeige dieser Zeile, so hat man einfach den Befehl

```
I   KEY ON
```

einzugeben.

Durch das Ausschalten dieser Anzeigezeile kann man auch die letzte Bildschirmzeile für Programmzwecke zur Verfügung stellen. Bei den nachfolgenden Ausführungen nehmen wir immer an, daß die Anzeigezeile unterdrückt ist.

Die Tastatur des IBM Personalcomputers weist außer den hier vorgestellten noch eine Anzahl weiterer Tasten auf. Aber . . ., wir sollten vorerst einmal endlich damit beginnen, mit dem Computer zu arbeiten. Mit der Besprechung der weiteren Tasten können wir uns Zeit lassen.

Aufgabengruppe 1

Auf dem Bildschirm sollen die folgenden Zeilen angezeigt werden:

1. `10 PRINT "Hallo."`
2. `10 ARITH1=1.5378`
3. `10 PRINT  3+7`
4. `20 LET  A = 3 - 5`
5. `20 5% of 68`
6. `10 IF  38 > -5`
7. `10 X=5:PRINT X`
8. `20 IF X>0 THEN 50`
9. `10 LET X=10`
   `20 LET Y=50.35`
10. `200 Y = X*2 - 5`
    `300 PRINT Y, "Y"`

Nach Durchführung jeder Aufgabe soll jeweils der Bildschirm gelöscht werden.

Antworten zu den Testübungen

1.6.1 a) Es ist der Name einzutippen und danach die Eingabetaste zu drücken.

b) Es sind gleichzeitig die Tasten Ctrl und Home zu betätigen.

c) Es ist die Taste Caps Lock anzuschlagen; danach ist a) zu wiederholen.

1.7 Der Personalcomputer IBM PC/XT

Unter der Bezeichnung PC/XT wird ein Personalcomputer mit umfangreicherer Ausrüstung und damit erweiterten Fähigkeiten angeboten. Als Hauptunterschied ist wohl mit anzusehen, daß er statt eines zweiten Diskettenlaufwerkes auf der rechten Seite der Systemeinheit ein Laufwerk für eine Hartplatte besitzt. Eine Hartplatte gestattet bekanntlich die Speicherung von etwa genausoviel Zeichen, wie auf 25 Disketten Platz finden.

Der IBM PC/XT weist im übrigen standardmäßig eine Reihe von Einrichtungen auf, die sonst nur zusätzlich erhältlich sind. Darüberhinaus ist seine Systemplatine mit insgesamt acht Erweiterungsschlitzen ausgerüstet gegenüber nur fünf bei den normalen Personalcomputern.

Die nachfolgende Abb. 1.12 zeigt die Fotografie eines Personalcomputers des Modells IBM PC/XT.

Abb. 1.12 Personalcomputer IBM PC/XT

1.8 Erweiterungsmöglichkeiten der Personalcomputer

Es gibt viele andere Platinen, die zur Erweiterung der Möglichkeiten von IBM Personalcomputern zur Verfügung stehen. Einige von ihnen wollen wir hier erwähnen:

a) Adapter für Spiele
Diese Einrichtung ermöglicht die Benutzung von Spielpulten und von einem Lichtstift

b) Speicherplatinen
Durch diese Platinen kann der Hauptspeicher (RAM) kapazitätsmäßig aufgestockt werden.

c) Kalendarium (Zeit und Datum)
Durch diese Einrichtung kann der Personalcomputer eigenständig die Zeit und das Datum verfolgen, sogar bei ausgeschaltetem Computer. Dadurch erspart man sich die Eingabe von Datum und Zeit.

d) Platinen für mehrere Funktionen (Multifunktionsplatinen)
Die Platinen gestatten verschiedenartige Kombinationen von

- Druckeradaptern
- Spieleadaptern
- Adaptern für serielle Kommunikation
- Speichererweiterungen
- Adapter für einfarbige Bildschirme oder für Bildschirme mit Farbe/Graphik

Diese Platinen sind weitverbreitet, weil sie auf eine sinnvolle Weise die Personalcomputer mit vielen Funktionen ausstatten und dabei kostbare Erweiterungsschlitze für andere Zwecke aufsparen.

2 Grundlagen der Programmierung in BASIC

2.1 Überblick über die BASIC-Versionen

Im ersten Kapitel lernten wir, mit der Tastatur und dem Bildschirm des IBM Personalcomputers umzugehen. Nunmehr wollen wir uns damit beschäftigen, wie man sich mit dem Computer in Verbindung setzt und wie man ihm Anweisungen erteilt.

Menschen gebrauchen Sprachen, um miteinander in Verbindung zu treten. Den Computern dienen Sprachen dazu, um mit anderen angeschlossenen Geräten (z. B. Druckern), mit Benutzern und sogar mit anderen Computern zu kommunizieren. Heuzutage gibt es mittlerweile Hunderte von Computersprachen. Der IBM Personalcomputer versteht durchaus einige von ihnen.

Unter den dem IBM Personalcomputer verständlichen Sprachen befindet sich BASIC, eine vielseitige und dabei leicht zu begreifende Sprache. Sie wurde von *John Kemeny* und *Thomas Kurtz* am *Dartmouth College* entwickelt und zwar speziell für Anfänger in den Computerwissenschaften. In den nächsten Kapiteln wollen wir uns auf das Erlernen der Grundlagen von BASIC konzentrieren. Dabei werden wir auch eine ganze Menge darüber erfahren, wie man Computer zur Lösung von Problemen einsetzt.

Der IBM Personalcomputer kann mit drei verschiedenen Versionen der BASIC-Sprache betrieben werden. Die schwächste Version von BASIC wird „Kassetten-BASIC" genannt. Diese Version kann für alle IBM Personalcomputer, unabhängig von ihrer Ausrüstung, geliefert werden; sie ist im ROM gespeichert. Wenn ein Computer mit ein oder zwei Diskettenlaufwerken ausgestattet ist, kann man schon von einer sprachlich stärkeren Version von BASIC Gebrauch machen, die „Disketten-BASIC" genannt wird. Diese Version schließt alle Anweisungen von Kassetten-BASIC ein und enthält darüber hinaus zusätzliche Statements, durch die die Benutzung der Diskettenlaufwerke ermöglicht wird. Die dritte Sprachebene von BASIC heißt „Erweitertes BASIC" und weist zunächst einmal alle Sprachelemente des Disketten-BASIC auf. Darüberhinaus sieht das erweiterte BASIC auch Anweisungen vor, die fortgeschrittene graphische Funktionen ansprechen, die Musik spielen, und die verschiedene zusätzliche Einrichtungen steuern und kontrollieren können. Zu diesen Einrichtungen zählen z. B. Spielpulte für elektronische Spiele und Lichtstifte. Es sollte jedoch ausdrücklich betont werden, daß viele Statements des erweiterten BASIC verlangen, daß der Computer mit der Schnittstelle für Farbe und Graphik ausgerüstet ist.

Die im Kapitel 1 besprochenen Verfahren ermöglichen das Laden des Disketten-BASIC. Mit dieser Version von BASIC wollen wir uns daher in den nächsten Kapiteln auseinandersetzen, d. h. ihre Statements kennen- und gebrauchen lernen. Später werden wir uns dann mit einigen Sprachelementen beschäftigen, die uns zusätzlich im erweiterten BASIC zur Verfügung stehen.

2.2 Ausführung von BASIC-Programmen

Eine Folge von Computerinstruktionen wird P r o g r a m m genannt. Wir werden lernen, Programme zu verfassen, die

- arithmetische Operationen durchführen,
- Zeichnungen anfertigen
  und sogar
- Spiele wie „Tic Tac Toe" betreiben.

Bevor wir aber dazu übergehen, unsere eigenen Programme zu schreiben, wollen wir einen Blick auf ein Programm werfen, das von der IBM stammt und welches uns die Leistungsfähigkeit ihrer Personalcomputer demonstrieren soll.

Zu diesem Zweck ist die Diskette mit dem Betriebssystem DOS ins Laufwerk A: einzustekken, der Computer einzuschalten und bis zur Ok-Rückmeldung von BASIC vorwärtszuschreiten, d.h. es ist so vorwärts zu gehen, wie es im Kap. 1 beschrieben worden ist. Danach ist die zweite, von der IBM zusammen mit der DOS-Diskette ausgelieferte und mit „Beispiel" beschriftete Diskette statt der DOS-Diskette ins Laufwerk A: einzuschieben. Die IBM hat auf dieser Diskette viele interessante Programme aufgezeichnet. Um eine Übersicht über diese Programme zu erhalten, muß der Befehl

```
FILES "*.BAS"
```

eingetippt und danach die Eingabetaste gedrückt werden. Auf dem Bildschirm werden dadurch die Bezeichnungen der Dateien angezeigt, in denen BASIC-Programme stehen. Die Bezeichnungen solcher Dateien bestehen grundsätzlich aus einem beliebigen Dateinamen, dem der einheitliche Dateinamenzusatz

```
.BAS
```

folgt. Die Anführungszeichen schließen im Befehl FILES die Dateibezeichnung ein, auf die sich der Befehl beziehen soll; dabei steht das Sternzeichen symbolisch für „alle" Dateinamen, die den Zusatz .BAS besitzen. Der Programmname von BASIC-Programmen ist stets gleich dem Dateinamen.

Stehen zwei Diskettenlaufwerke zur Verfügung, so schieben wir die Diskette mit der Aufschrift „Beispiel" ins Laufwerk B: und belassen die Diskette mit dem Betriebssystem DOS im Laufwerk A:. Der einzugebende Befehl muß jetzt zusätzlich auf das Laufwerk B: bezug nehmen, da dieses nicht das Standardlaufwerk ist. Er lautet deshalb wie folgt:

```
FILES "B:*.BAS"
```

Eines der eindrucksvollsten Programme heißt MUSIC; man beachte, daß es in der Auflistung unter der Bezeichnung

```
MUSIC.BAS
```

erscheint. Die Buchstabenfolge BAS besagt, daß das Programm in BASIC geschrieben ist (vorläufig noch brauchen wir das nicht zu beachten). Um das Programm MUSIC von der Diskette in den RAM zu laden, müssen wir den Befehl

```
LOAD "MUSIC"          (Diskette „Beispiel" im Laufwerk A:)
bzw.
LOAD "B:MUSIC"        (Diskette „Beispiel" im Laufwerk B:)
```

eingeben und anschließend die Eingabetaste betätigen. Den Programmnamen müssen wir bei der Eingabe in Anführungszeichen einschließen. Nach Betätigung der Eingabetaste leuchtet das Anzeigelämpchen des Diskettenlaufwerks auf, man hört das Laufwerk arbeiten, und das Programm MUSIC wird in den RAM geladen. Anschließend verlöscht das Anzeigelämpchen wieder, und das Laufwerk stoppt.

Nunmehr wollen wir erreichen, daß der Computer die Instruktionen des Programms befolgt; in der Fachsprache redet man von der *Ausführung* des Programms. Zu diesem Zweck geben wir den Befehl

```
RUN
```

ein und drücken die Eingabetaste. Zu Beginn der Programmausführung zeichnet der Computer auf dem Bildschirm die Tastatur eines Klavieres und zeigt außerdem die Titel verschiedener Melodien an. Um eine Melodie zu spielen, ist die angezeigte Taste zu betätigen. Warum sollen wir nicht einige Minuten die vom Computer generierte Musik genießen? Man beachte auch, wie der Computer die Klaviatur „belebt", indem er eine sich bewegende Note anzeigt, die auf die gerade „angeschlagene" Taste der Klaviatur verweist.

Früher oder später wird man einmal ein gerade ablaufendes Computerprogramm unterbrechen wollen. Dies kann durch das gleichzeitige Drücken der beiden Tasten, die mit

Ctrl und *Break*

beschriftet sind, erreicht werden. Aus gutem Grund wird eine Unterbrechung nur durch diese zweihändige Operation bewirkt. Die beiden Tasten sind zudem so angeordnet, daß Programme nicht zufällig unterbrochen werden können. Um einmal die Programmunterbrechung zu illustrieren, wollen wir das Programm MUSIC ablaufen lassen und eine Melodie spielen. Mitten im Lied wollen wir gleichzeitig die beiden Tasten Ctrl und Break anschlagen. Dadurch wird der Programmablauf angehalten und auf dem Bildschirm eine Nachricht angezeigt, die wie folgt aussieht:

`Break in xxxx`
`Ok`
`-`
	Unterbrechung in xxxx

Die Information „Unterbrechung in xxxx" signalisiert die Programmzeile, auf der der Programmablauf gestoppt wurde. Die BASIC-Rückmeldung „Ok" weist darauf hin, daß BASIC nunmehr die Eingabe eines Befehles erwartet. Die Unterbrechung eines Programmablaufes löscht das Programm keineswegs aus dem RAM. Um das Programm erneut auszuführen, genügt die Eingabe des Befehles RUN und anschließendes Betätigen der Eingabetaste.

Genug Musik für jetzt! Wir wollen die Programmausführung beenden. Entsprechend der auf dem Bildschirm angezeigten Instruktionen, können wir die Einstellung der Programmausführung dadurch veranlassen, indem wir die Taste

Esc

anschlagen; diese Taste ist links oben auf der Tastatur plaziert. Nach dem Anschlagen wird die BASIC-Systemanfrage „Ok" angezeigt. Das bedeutet, daß BASIC die Eingabe eines Befehls erwartet. – Einen Programmablauf einzustellen, heißt den Ausgang oder Exit eines Programmes anzusteuern.

Der Benutzer wird sicher manchmal neugierig sein und sich die Menge der Instruktionen ansehen wollen, die zusammen das Programm MUSIC bilden. Nichts ist leichter zu bewerkstelligen. Es ist einfach der Befehl

```
LIST
```

einzutippen und danach wie üblich die Eingabetaste zu drücken. Hierdurch werden die Anweisungen des Programmes auf dem Bildschirm angezeigt. Da es weit mehr sind, als der Bildschirm Zeilen besitzt, rauschen sie natürlich viel zu schnell vorüber, um sie lesen zu können. Später werden wir lernen, wie wir die Bildschirmanzeige dort anhalten können, wo wir es wollen, oder wie wir uns eine gedruckte Kopie (Druckerausgabe) verschaffen können.

Testübung 2.2.1

Von den auf der Diskette „Beispiel" in Dateien gespeicherten Programmen ist ein anderes auszuwählen, in den Speicher (RAM) zu laden und auszuführen. Einige Programme mögen den Farbe/Graphik-Adapter benötigen; das schadet nichts. Wenn diese Schalttafel beim Computer nicht installiert ist, meldet sich der Computer und teilt dem Benutzer mit, daß er das betreffende Programm nicht ausführen kann und zeigt erneut die BASIC-Rückmeldung an.

Antwort zur Testübung 2.2.1

Es ist bei der BASIC-Rückmeldung zu starten. Zunächst ist

LOAD *"programmname"* bzw. LOAD "B:*programmname*"

einzugeben und die Eingabetaste zu drücken. Unter *programmname* ist der Name desjenigen Programms gemeint, das ausgeführt werden soll. Zum Schluß ist der Befehl

```
RUN
```

einzugeben und erneut die Eingabetaste zu drücken.

2.3 Abfassen von Programmen in BASIC

Man mag durch die Vielzahl der Anweisungen, aus denen das Programm MUSIC besteht, verunsichert sein. Wir werden in diesem Buch derart komplizierte Programme überhaupt nicht schreiben. Wir wollen vielmehr Schritt für Schritt unsere Kenntnisse aufbauen und zuerst lernen, wie man einfache Programme in BASIC abfaßt.

Wir wollen annehmen, daß das im letzten Kapitel beschriebene Startverfahren erledigt wurde, und daß der Computer bereit ist, weitere Instruktionen entgegenzunehmen. Er zeigt das bekanntlich durch die BASIC-Rückmeldung.

```
Ok
```

an. Von diesem Punkt an wird eine typische Computersitzung etwa wie folgt aussehen:

1. Eintippen des Programmes
2. Lokalisieren und Korrigieren der Programmfehler

3. Ausführen des Programmes
4. Beschaffung der durch das Programm angeforderten Ausgabe.
5. Durchführung einer der drei folgenden Alternativen:
   a) Wiederholung der Programmausführung
   b) Wiederholung der Schritte 1. bis 4. für ein neues Programm
   c) Beendigung der Sitzung (Ausschalten des Computers und . . . zum Essen gehen)

Um voll verstehen zu können, was diese fünf Schritte alles beinhalten, wollen wir ein einzelnes einfaches Beispiel betrachten. Nehmen wir dazu einmal an, daß wir durch den Computer die Zahlen 9 und 7 addieren lassen wollen. Zuerst müssen wir die folgenden beiden Anweisungen eintippen:

```
10 PRINT  7 + 9
20 END
```

Diese Sequenz von zwei Anweisungen bildet ein Programm zur Berechnung der Summe der beiden Zahlen 7 und 9. Man beachte, daß der Computer während des Eintippens des Programmes die Anweisungen aufzeichnet, aber nicht ausführt. Zur Zeit der Programmeingabe sorgt der Computer jedoch dafür, daß der Benutzer Gelegenheiten zum Ändern, Löschen und Korrigieren der Zeilen mit den Anweisungen hat. Wie das im einzelnen zu geschehen hat, wollen wir später behandeln. Irgendwann wird man jedoch mit dem Programm zufrieden sein. Dann muß man dem Computer mitteilen, daß das Programm ablaufen soll, d.h. daß er die Anweisungen des Programmes ausführen soll. Man veranlaßt den Programmablauf durch Eingabe des Befehles[1)]

```
RUN
```

Nunmehr wird der Computer das Programm ablaufen lassen und die gewünschte Antwort anzeigen:

```
16
```

Wenn man das Programm ein zweites Mal ausführen lassen will, ist erneut der Befehl RUN einzugeben.

Nach der Ausführung wird ein Programm nicht im RAM gelöscht. Deshalb kann man, wenn man weitere Anweisungen zum Programm hinzufügen oder das Programm ändern will, einfach mit dem Eintippen fortfahren, gerade so, als ob der Befehl RUN nicht dazwischen geschoben wurde. Wenn wir also beispielsweise das Programm um die Berechnung der Differenz 7–9 ergänzen wollen, so haben wir einfach die zusätzliche Zeile

```
15 PRINT  7 - 9
```

einzugeben. Damit wir uns das gegenwärtig gespeicherte Programm ansehen können, haben wir den Befehl

---

[1)] Man vergesse unter keinen Umständen, daß nach der Befehlseingabe die Eingabetaste gedrückt werden muß. Man rufe sich auch ins Gedächtnis zurück, daß der Computer eingegebene Zeilen nur dann anerkennt, wenn sie zu ihm durch Anschlagen der Eingabetaste gesendet worden sind.

```
LIST
```

einzugeben und die Eingabetaste zu betätigen. Das Programm besteht jetzt aus den folgenden drei Zeilen, die infolge des soeben eingegebenen Befehles auf dem Bildschirm angezeigt werden:

```
10 PRINT  7 + 9
15 PRINT  7 - 9
20 END
```

Man beachte, daß der Computer die nachträglich eingegebene Zeile 15 in die richtige Reihenfolge gebracht hat. Wenn wir anschließend wieder den Befehl RUN eingeben, erhalten wir zwei Antwortzeilen, nämlich

```
16
-2
```

Falls wir nun zu einem anderen Programm übergehen wollen, haben wir den Befehl

```
NEW
```

einzutippen.

Durch diesen Befehl wird das vorhergehende Programm in RAM gelöscht und damit ist der Computer präpariert, ein neues Programm zu akzeptieren. Man sollte sich immer der wichtigen Tatsache erinnern:

Der RAM kann stets nur ein einziges Programm enthalten.

Testübungen 2.3.1

a) Es ist ein BASIC-Programm zu schreiben und einzugeben, das die Summe aus 12.1, 98 und 5.32 errechnet.

b) Es ist zu veranlassen, daß das Programm von Übung a) ausgeführt wird.

c) Das Programm von Übung a) ist im RAM zu löschen

d) Es ist ein Programm zur Berechnung der Differenz 48.75–1.674 zu schreiben.

e) Das in Übung d) geschriebene Programm ist einzutippen und anschließend ablaufen zu lassen.

2.3.1 Befehlsmodus und Ausführungsmodus

BASIC auf IBM Personalcomputern arbeitet in zwei verschiedenen Zuständen. Im *Befehlsmodus* akzeptiert der Computer eingegebene Programmzeilen und solche Befehle, wie RUN und NEW, die zur Manipulation von Programmen dienen. Der Computer erkennt und identifiziert eine Programmzeile durch ihre Zeilennummer. Programmzeilen werden nicht sofort ausge-

führt. Sie werden vielmehr in RAM gespeichert, bis man dem Computer sagt, was mit ihnen geschehen soll. Befehle hingegen werden unmittelbar nach ihrer Eingabe, d.h. so bald als möglich, ausgeführt.

Im *Ausführungsmodus* führt der Computer ein Programm aus. In diesem Modus wird der Bildschirm vom Programm gesteuert und kontrolliert.

Nach dem Einschalten wird der Computer automatisch in den Befehlsmodus versetzt, angezeigt durch die auf dem Bildschirm erscheinende Rückmeldung Ok. Der Befehl RUN bringt den Computer in den Ausführungsmodus. Nach Beendigung des Programmablaufes wird erneut die Rückmeldung Ok angezeigt. Damit wird angedeutet, daß der Computer den Ausführungsmodus verlassen hat und in den Befehlsmodus zurückgekehrt ist.

2.3.2 Gegenüberstellung von Groß- und Kleinschreibung, gesonderte Leerstellen (Zwischenräume)

Der Computer ist ein strenger „Arbeitgeber". Es besitzt einen eng begrenzten Wortschatz (BASIC) und dieses Vokabular muß darüberhinaus auch noch den sehr spezifischen Regeln genügen, die für die Anordnung der Wörter und für die Zeichensetzung usw. gelten. BASIC gestattet jedoch eine gewisse Freiheit bei der Bildung von Ausdrücken. So ist es z.B. möglich, daß BASIC-Anweisungen in Großbuchstaben oder in Kleinbuchstaben oder in einer Mixtur von beiden eingegeben werden. Außerdem bleiben zusätzliche Leerstellen, die die Zwischenräume zwischen den Wörtern vergrößern, unbeachtet. Somit werden z.B. die in der Abb.2.1 dargestellten BASIC-Anweisungen als gleichwertig interpretiert.

```
10  PRINT A
10  print  a
10  PRINT A
10  print A
10    print    A
```

usw.

Abb.2.1 Gleichwertige BASIC-Anweisungen

An gewissen Stellen allerdings erwartet BASIC Zwischenräume. So muß z.B. bei den oben aufgeführten Anweisungen mindestens eine Leerstelle die Wörter PRINT und A voneinander trennen. Andernfalls würde nämlich BASIC die Anweisung PRINTA lesen, die wiederum in seinem Vokabular nicht existiert.

2.3.3 Tröstende Worte für den Programmierer

Viele Menschen nehmen von Computern an, daß sie „Elektronengehirne" seien, denen irgendwie die Kraft menschlicher Gedanken innewohne. Solche Vorstellungen gehen völlig an der Wirklichkeit vorbei. Die Elektronik der Computer und die Regeln der BASIC-Sprache ermöglichen nur die Erkennung eines engbegrenzten Wortschatzes und die Ausübung verschiedener Aktionen, die auf den Daten fußen, die ihnen zugewiesen wurden. Es ist außerordentlich wichtig, daß man sich im klaren darüber ist, daß die Computer nicht das besitzen, was

man gewöhnlich als „gesunden Menschenverstand" zu bezeichnen pflegt. Der Computer versucht nur zu interpretieren, welche Daten auch immer eingegeben werden. Wenn die eingegebenen Zeichen einen erkennbaren Befehl darstellen, wird dieser ausgeführt. Es spielt dabei keine Rolle, ob der Befehl in einem bestimmten Zusammenhang einen Sinn ergibt oder nicht. Der Computer kann keineswegs derartige Urteile fällen. Er kann nur das verrichten, was ihm zu tun aufgetragen wird. Wegen dieser Inflexibilität bei der Auslegung der Befehle muß man dem Computer exakt mitteilen, was man tun will. Man mache sich also keine Sorgen, wenn der Computer in Verwirrung gerät. - Wenn man einer Maschine einen Befehl in einer nicht korrekten Form mitteilt, wird man sie keineswegs beschädigen! Um jedoch zu erreichen, daß die Maschine unseren Geboten gehorcht, ist es notwendig, daß man lernt, die Computersprache präzis zu „sprechen".

Antwort zu den Testübungen 2.3.1

a)
```
10  PRINT  12.1 + 98 + 5.32
20  END
```

b) Eingabe des Befehles RUN; Drücken der Eingabetaste

c) Eingabe des Befehles NEW; Drücken der Eingabetaste

d)
```
10  PRINT  48.75 - 1.674
20  END
```

e) – Eingabe der beiden Zeilen des Programmes von d)
– Eingabe des Befehles RUN
– Drücken der Eingabetaste

2.4 Einige elementare Programme in BASIC

Will man den Gebrauch einer Programmiersprache erlernen, wollte man sich zuerst um das *Alphabet* der Sprache kümmern. Danach sollte man sich mit dem *Wortschatz,* dem *Vokabular,* der Sprache befassen. Schließlich muß man die Art und Weise studieren, in der die Wörter in Sätzen zusammengestellt werden müssen. Im Kapitel 1 lernten wir die Zeichen kennen, die auf der Tastatur des IBM Personalcomputers vorhanden sind. Diese Zeichen bilden zugleich das Alphabet von BASIC.

Darauf aufbauend wollen wir uns nun mit dem Grundvokabular von BASIC befassen. Die einfachsten „Wörter" stellen die sogenannten Konstanten dar.

2.4.1 Konstanten in der Programmiersprache BASIC

BASIC gestattet die Handhabung sowohl von Zahlen als auch von Texten. Natürlich weichen die Regeln, die für die Handhabung numerischer Daten gelten, von den Regeln ab, die die Verarbeitung von Texten betreffen. In BASIC werden diese beiden Datentypen wie folgt unterschieden:

- eine numerische Konstante stellt eine Zahl dar,
- eine Kettenkonstante ist eine Folge von „Tastaturzeichen"; diese Folge schließt Buchstaben, Ziffern und die anderen auf der Tastatur vorhandenen Symbole ein.

Einige Beispiele für numerische Konstanten seien nunmehr aufgeführt:

5	23456
-2	456.78345676543987
3.14159	27134566543

Anschließend sollen einige Kettenkonstanten folgen:

```
"JOHN"
"Ausstehende Rechnungen"
"Betrag 2714.56 DM"
"9. Juli 1924"
```

Man beachte, daß die Kettenkonstanten immer in Anführungszeichen eingeschlossen sind. Um Unklarheiten zu vermeiden, können Anführungszeichen nicht als Teil einer Kettenkonstanten auftreten; in der Praxis könnte ein Auslassungszeichen ' verwendet werden, um ein Anführungszeichen " in einer Kettenkonstanten zu ersetzen. Obgleich Zahlen in einer Kettenkonstanten erscheinen dürfen, kann man solche enthaltenen Zahlen nicht in arithmetische Operationen einsetzen. Nur die nicht von Anführungszeichen umgebenen Zahlen dürfen zu Rechenoperationen herangezogen werden.

Bei gewissen Anwendungen wird man die numerischen Konstanten im *Potenzformat* angeben wollen. Als besonders nützlich wird das empfunden, wenn sehr große oder sehr kleine Zahlen vorliegen. Betrachten wir z.B. die Zahl 15300000000. Es ist äußerst unbequem, wenn man die zahlreichen Nullen eingeben muß. Solche Zahlen können in einem handlichen Kurzformat geschrieben werden, in diesem Fall zu 1.53E10. Die Zeichenfolge 1.53 zeigt die ersten drei geltenden Ziffern der Zahl an, während E10 bedeutet, daß der Dezimalpunkt 10 Stellen nach rechts verrückt werden muß. Um bei einem weiteren Beispiel zu bleiben: Die Zahl −237.000 kann in der Potenzform in gleicher Weise zu −2.37E5 geschrieben werden. Die Potenzform kann ebenso auch für sehr kleine Zahlen eingesetzt werden. Beispielsweise kann die Zahl 0.00000000054 zu 5.4E−10 notiert werden; −10 zeigt hier an, daß der Dezimalpunkt in 5.4 um 10 Stellen nach links verschoben werden muß.

Testübungen 2.4.1

a) Die nachfolgenden Zahlen sind in der Potenzform zu schreiben:
0.00048
−1374.5

b) Die nachfolgenden, in Potenzform geschriebenen Zahlen sind in der gewöhnlichen Form darzustellen:
−9.7E3
9.7E−3
−9.7E−3

Wir werden später noch einiges mehr über die Konstanten zu sagen haben. Beispielsweise werden wir uns mit der Anzahl der Ziffern zu befassen haben, die die Genauigkeit der Zahlen

bestimmen, wir werden über das Runden sprechen müssen usw. Genug für jetzt, das erworbene Wissen reicht aus, um mit dem Abfassen von Programmen zu beginnen. Anstelle der Verfeinerung unserer bisherigen Kenntnisse wollen wir lieber dazu übergehen, den Computer zur Durchführung von Arbeiten zu bringen.

2.4.2 Programme in BASIC

Wir wollen uns das Programm, dem wir im Abschnitt 2.1. begegnet sind, erneut anschauen. Es lautete bekanntlich so, wie die Abb. 2.2 es zeigt.

```
10  PRINT  7 + 9
20  END
```

Programmende
Zeilennummern

Abb. 2.2 Programm zum Addieren zweier Zahlen

Dieses Programm macht uns bereits mit zwei wichtigen Besonderheiten vertraut, die allgemein für alle BASIC-Programme gelten:

1. Die Anweisungen eines Programmes müssen numeriert werden. Jede Programmzeile beginnt mit einer Zeilennummer. Der Computer führt die Anweisungen in der Reihenfolge steigender Zeilennummern aus.
2. Das Statement END kennzeichnet das Ende jeden Programmes. Stößt die Programmausführung auf dieses Statement, stoppt der Computer den Programmablauf und zeigt auf dem Bildschirm die Rückmeldung Ok an.

Man beachte, daß die Zeilennummern nicht aufeinanderfolgend vorliegen müssen. Zum Beispiel wird auch ohne weiteres ein Programm beanstandungslos entgegengenommen, dessen Zeilennummern der Reihe nach 10, 23, 47 und 100 lauten. Man merke sich ebenfalls, daß es nicht notwendig ist, die Statements in ihrer numerischen Reihenfolge einzutippen. Man kann z. B. zuerst die Zeile mit der Zeilennummer 20 eingeben und danach erst die Zeile mit der Zeilennummer 10. Der Computer sondert die Zeilen aus und ordnet sie neu an, und zwar entsprechend ihrer aufsteigenden Zeilennummern. Diese typische Eigenschaft ist besonders dann sehr hilfreich, wenn man zufällig beim Eingeben eines Programmes eine Zeile ausgelassen hat. (Das soll mitunter schon vorkommen, selbst beim tüchtigsten Programmierer).

Hinsichtlich der Zeilennumerierung ist noch ein anderer wichtiger Tatbestand zu erwähnen. Wenn zwei Zeilen mit gleicher Nummer eingegeben werden, löscht der Computer die zuerst eingegebene Version und behält die zweite bei. Dieses Merkmal ist bei der Fehlerkorrektur sehr nützlich: Weist eine Zeile Fehler auf, so braucht man sie nur neu einzutippen.

Der IBM Personalcomputer kann natürlich alle gängigen Rechenoperationen ausführen, die auch mit einem Tisch- oder Taschenrechner erledigt werden können. Da die meisten Menschen mit der Bedienung und der Anwendung solcher Rechner vertraut sind, sollten wir rasch damit beginnen, Programme zum Lösen arithmetischer Probleme zu schreiben.

Die meisten arithmetischen Operationen werden in der üblichen Weise niedergeschrieben. Zum Beispiel werden Additionen und Subtraktionen in der herkömmlichen Art abgefaßt:

```
5 + 4
9 - 8
```

Die Multiplikation jedoch ist mit dem Symbol * zu formulieren; das Sternzeichen teilt sich eine Taste mit der Ziffer 8. Somit ist das Produkt aus den beiden Faktoren 5 und 3 wie folgt einzugeben:

```
5 * 3
```

Die Division ist mit dem Symbol / niederzuschreiben. Somit nimmt z. B. der Quotient mit dem Dividenden 8.2 und dem Divisor 15 die Form

```
8.2 / 15
```

an.

Beispiel 1:

Es ist ein BASIC-Programm zur Berechnung der Summe aus 54.75, 78.33 und 548 zu schreiben.

Die Summe ergibt sich zu

54.75 + 78.33 + 548

Das Ausgeben von Daten auf dem Bildschirm besorgt die Anweisung PRINT. Das gesuchte BASIC-Programm ist demnach wie folgt zu formulieren:

```
10 PRINT  54.75 + 78.33 + 548
20 END
```

BASIC führt die arithmetischen Operationen in einer bestimmten festgelegten Reihenfolge aus. Ein arithmetischer Ausdruck wird zunächst hinsichtlich der in ihm enthaltenen arithmetischen Operationen untersucht. Die Multiplikationen und Divisionen werden zuerst ausgeführt, und zwar in der Reihenfolge ihres Auftretens von links nach rechts. Anschließend erfolgt eine Rückkehr zur linken Seite des Ausdrucks. Nunmehr kommen die Additionen und Subtraktionen an die Reihe; sie werden ebenfalls von links nach rechts ausgeführt. Erscheinen in einem Ausdruck Klammern, so werden diese vorrangig unter Zugrundelegung der soeben besprochenen Regeln ausgewertet. Treten Klammern innerhalb von Klammern auf, so werden die Operationen in den innersten Klammernpaaren zuerst ausgeführt.

Beispiel 2:

Wie lauten die Ergebnisse, die BASIC aus den nachfolgenden Ausdrücken ermittelt?

a) `(5+7)/2`
b) `5+7/2`
c) `5+7*3/2`
d) `(5+7*3)/2`

Zu den Lösungen ist folgendes zu sagen:

a) Der Computer wendet die Regeln für die Reihenfolge der Rechenoperationen an. Zuerst wird die

Klammer ausgewertet. Es ergibt sich dabei der Wert 12. Danach wird 12 durch 2 dividiert. Als Endresultat wird also 6 gefunden.

b) Der Computer prüft den Ausdruck von links nach rechts durch und führt alle Multiplikationen und Divisionen in der Reihenfolge durch, in der sie ihm begegnen. Somit wird zuerst 7 durch 2 dividiert, was zum Ergebnis 3.5 führt. Danach wird der Ausdruck erneut von links nach rechts durchgegangen, um die Additionen und Subtraktionen in dieser Reihenfolge auszuführen. Somit wird nun 5 + 3.5 errechnet, was zum Endresultat 8.5 führt.

c) Zuerst werden die Multiplikation und Divisionen von links nach rechts ausgeführt. Dadurch geht der Ausgangsausdruck in den Ausdruck 5 + 10.5 über. Nach der anschließenden Ausführung der verbleibenden Addition ergibt sich als Endresultat 15.5.

d) Der in Klammern stehende Teilausdruck wird vom Computer zuerst ausgewertet; in diesem Fall wird zunächst 7*3 errechnet, was 21 ergibt. Die anschließende Addition von 5 führt zu 26, dem Wert des Klammerausdruckes. Zum Schluß kommt die verbleibende Division an die Reihe: 26/2. Als Endresultat wird somit 13 ermittelt.

Testübung 2.4.2

Es sind die beiden Ausdrücke

$5+3/2+2$

und $(5+3)/(2+2)$

zu berechnen.

Beispiel 3:

Es ist ein BASIC-Programm zu schreiben, das den Wert des Bruches

$$\frac{22\cdot 18+34\cdot 11-12.5\cdot 8}{27.8}$$

errechnet!

Als Lösung ergibt sich das folgende Programm:

```
10 PRINT  (22*18 + 34*11 - 12.5*8) / 27.8
20 END
```

Bei dieser Lösung ist zu beachten, daß wir in Zeile 10 Klammern benutzt haben. Sie besagen dem Computer, daß der gesamte, in Klammern stehende Ausdruck durch 27.8. zu dividieren ist. Würden wir die Klammern weglassen, dann würden zunächst von links nach rechts die drei Produkte errechnet und anschließend das letztere durch 27.8 dividiert. Danach wird wiederum von links nach rechts addiert bzw. subtrahiert.

Testübung 2.4.3

Es sind BASIC-Programme zu schreiben

a) Der Ausdruck
$((4\cdot 3 + 5\cdot 8 + 7\cdot 9) / (7\cdot 9 + 4\cdot 3 + 8\cdot 7)) \cdot 48.7$
ist zu errechnen!

b) 27.8% von (112 + 38 + 42)

c) Der Mittelwert der Zahlen 88, 78, 84, 49 und 73 ist zu ermitteln!

2.4.3 Ausgabe von Wörtern

Bisher haben wir die Anweisung PRINT nur dazu benutzt, die Ergebnisse numerischer Probleme anzuzeigen. Diese Anweisung ist jedoch weitaus vielseitiger; sie gestattet nämlich auch die Anzeige von Kettenkonstanten. Betrachten wir ein erstes Beispiel:

```
10 PRINT "Krankengeschichte"
```

Während der Programmausführung kommt es, durch diese Anweisung bedingt, zur Anzeige von

```
Krankengeschichte
```

auf dem Bildschirm.

Um mehrere Kettenkonstanten auf der gleichen Zeile anzeigen zu können, müssen sie, durch Kommas getrennt, in der gleichen PRINT-Anweisung aufgeführt werden. Sehen wir uns dazu ein Beispiel an:

```
10 PRINT  "ALTER", "GESCHL.", "GEBURTSORT", "ADRESSE"
```

Durch dieses Statement werden damit bei der Programmausführung vier Wörter auf einer Zeile ausgegeben, nämlich

```
ALTER          GESCHL.        GEBURTSORT     ADRESSE
```

Sowohl numerische Konstanten als auch Kettenkonstanten können in ein einziges PRINT-Statement eingeschlossen werden, beispielsweise

```
10 PRINT  "ALTER", 65.43, "FAM.-ANG."
```

Hier kommt das Problem auf, wie der Computer die Raumaufteilung auf einer Zeile vornimmt. Dazu ist folgendes zu sagen: Jede Zeile ist in sogenannte *Ausgabezonen* oder *Druckzonen* eingeteilt. Jede Druckzone besteht aus 14 Stellen. Durch das Hineinstellen eines Kommas in

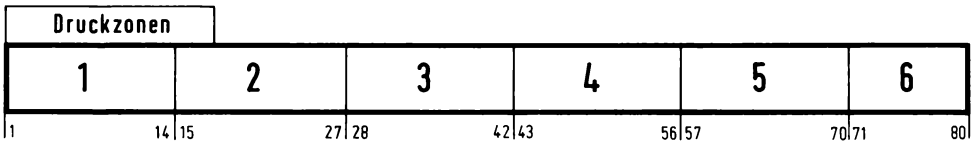


Abb. 2.3 Druckzonen

ein PRINT-Statement sagt man dem Computer, daß er die Ausgabe der nächsten Zeichenfolge am Anfang der nächsten Druckzone beginnen soll. Daher beginnen z. B. die vier Wörter ALTER, GESCHL., GEBURTSORT und ADRESSE des zuvor erwähnten Beispiels auf den Stellen 1,15,29 bzw. 43 (siehe Abb. 2.3).

Testübung 2.4.4

Es ist ein Programm zu schreiben, das zu der folgenden Ausgabe führt:

```
NACHNAME     1. VORNAME    2. VORNAME    RANG
WIESNER      ALBERT        FRANZ         56
```

Beispiel 4:

Angenommen, ein Verkäufer eines Möbelmarktes verkauft 50 Stühle und 5 Tische. Jeder Stuhl kostet 59,70 DM und jeder Tisch 247,90 DM. Für beide Artikel wird ein Nachlass von 30% gewährt. Es ist eine Versandrechnung vorzubereiten; dafür ist ein Programm zu erstellen.

Zur Lösung ist folgendes zu bemerken: Auf der Rechnung soll zunächst eine Zeile mit vier Spaltenüberschriften, nämlich

```
ARTIKEL, MENGE, EINZELPREIS, GESAMTPREIS
```

erscheinen. Anschließend sollen nach einer Leerzeile zwei Zeilen ausgegeben werden entsprechend der beiden verkauften Artikel (Artikelzeilen). Nach einer weiteren Leerzeile soll die Rechnung mit Rechnungssummenzeile abgeschlossen werden; in dieser ist der errechnete Rechnungsbetrag auszuweisen. Diese Vorgaben führen zu dem Programm, das in der Abb. 2.4 aufgelistet ist.

```
10 PRINT "ARTIKEL","MENGE","EINZELPREIS","GESAMTPREIS"
20 PRINT
30 PRINT "STUHL",50,59.70,50*(59.70-0.3*59.70)
40 PRINT "TISCH",5,247.90,5*(247.90-0.3*247.90)
50 PRINT
60 PRINT "RECHNUNGSSUMME",,,50*(59.70-0.3*59.70)+5*(247.90-0.3*247.90)
70 END
```

Abb. 2.4 Programm zur Rechnungsausgabe

```
10 PRINT "ARTIKEL","MENGE","EINZELPREIS","GESAMTPREIS"
20 PRINT
30 PRINT "STUHL",50,59.70,50*(59.70-0.3*59.70)
40 PRINT "TISCH",5,247.90,5*(247.90-0.3*247.90)
50 PRINT
60 PRINT "RECHNUNGSSUMME",,,50*(59.70-0.3*59.70)+5*(247.90-0.3*247.90)
70 END

RUN

ARTIKEL        MENGE          EINZELPREIS    GESAMTPREIS

STUHL           50             59.70          2089.50
TISCH           5              247.90         867.65

RECHNUNGSSUMME                                2957.15
```

Abb. 2.5 Ausgegebene Rechnung

Bei diesem Programm sei besonders auf die Zeilen 20 und 50 hingewiesen. In ihnen wird festgelegt, daß jeweils eine Leerzeile auszugeben ist. Man beachte ferner die Aufeinanderfolge von drei Kommas in der Zeile 60. Die zusätzlichen zwei Kommas bewirken, daß die nachfolgende Ausgabe erst am Anfang der vierten Zone beginnt, wodurch die Rechnungssumme direkt in die Spalte plaziert wird, die mit der Überschrift GESAMTPREIS versehen ist. Wenn wir nun RUN eingeben, gefolgt vom Anschlagen der Eingabetaste, dann sieht unser Bildschirm so aus, wie es die Abb. 2.5 zeigt.

Man mag nun gegen diese Form der Rechnungen einwenden, daß sie irgendwie schlampig aussieht, weil die Beträge in den Spalten nicht ausgerichtet (Komma unter Komma) aufgeführt sind. Geduld, die Ausrichtung der Spalten werden wir auch noch kennenlernen und zwar dann, wenn wir uns noch ein bißchen mehr über die Programmierung erarbeitet haben.

Testübung 2.4.5

Es soll ein Programm geschrieben werden, das zu der folgenden Ausgabe führt:

```
                    BUDGET-APRIL
LEBENSMITTEL        387.50
AUTO                123.71
ENERGIE             100.00
VERSCHIEDENES       146.00
UNTERHALTUNG        100.00
                    ------
GESAMT              (errechnete Summe)
```

2.4.4 Potenzierung

Nehmen wir einmal an, daß wir unter A eine beliebige Zahl verstehen wollen. Ferner sei N eine natürliche Zahl, d.h. eine Zahl aus der Menge 1,2,3, ··· Dann verstehen wir unter der N-ten Potenz von A ein Produkt, das dadurch zustandekommt, daß man A insgesamt N-mal mit sich selbst multipliziert. Das Ergebnis wollen wir B nennen. In der mathematischen Schreibweise sieht das ganze wie folgt aus:

$$B = A^N$$

(*gelesen:* B ist gleich A hoch N)

Dieser Rechenprozeß wird „Potenzierung" genannt. Einige Beispiele mögen diese kurzgefaßte Erklärung untermauern:

$2^3 = 2 \cdot 2 \cdot 2$	$5^7 = 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5 \cdot 5$
$2^3 = 8$	$5^7 = 78125$

Man kann natürlich Potenzen durch fortlaufende Multiplikationen errechnen lassen. Wenn jedoch N eine große Zahl darstellt, dann kann es ermüdend sein, das Produkt einzugeben. BASIC sieht für die Potenzierung eine Abkürzung vor. Das Symbol ∧ dient zur Kennzeichnung der Potenzierung; es ist auf der gleichen Taste untergebracht, der auch das Nummernzeichen # zugewiesen ist (mittlere Reihe der Tastatur). Unter Benutzung dieses Symbols kann die Potenz

$$2^3$$

in BASIC wie folgt geschrieben und eingegeben werden:

```
2∧3
```

Hinsichtlich der Reihenfolge der Rechenoperationen in einem arithmetischen Ausdruck hat die Potenzierung Vorrang vor Multiplikation und Division. Man betrachte dazu das folgende Beispiel:

Beispiel 5

Es ist der Wert zu bestimmen, der sich aus dem in BASIC niedergeschriebenen Ausdruck ergibt.

```
20*3-5*2∧3
```

Die Lösung findet man, indem man zunächst potenziert. Somit ergibt sich nach dem 1. Schritt

```
20*3-5*8
```

Die nächsten Schritte, in der bekannten und besprochenen Reihenfolge ausgeführt, führen schließlich zum Ergebnis 20.

Testübung 2.4.6

Es sind zunächst manuell und anschließend durch ein Programm mit dem IBM Personalcomputer die folgenden Ausdrücke zu berechnen!

a) $2^4 \cdot 3^3$

b) $2^2 \cdot 3^3 - 12^2 / 3^2 \cdot 2$

Aufgabengruppe 2

Es sind Programme in BASIC zu schreiben, mit deren Hilfe die nachfolgenden Berechnungen ausgeführt werden können:

1. 57 + 23 + 48
2. 57.83 · (48.27 - 12.54)
3. 127·86/38
4. $365/0.005 + 1.02^5$
5. Es ist eine Tabelle anzulegen, die die 1., 2., 3., und 4. Potenz der Zahlen 2, 3, 4, 5 und 6 enthält. Alle 1. Potenzen sind in einer Spalte für sich aufzuführen, alle 2. Potenzen in einer anderen Spalte usw.
6. Frau Monika Schmidt sucht ihren Arzt wegen eines gebrochenen Beines auf. Ihr Arzt schickt ihr nach Abschluß der Behandlung eine Rechnung zu, in der folgende Beträge ausgewiesen sind:
   45,- DM für das Entfernen der Schienen,
   35,- DM für die Heilbehandlung,
   5,- DM für Medikamente.
   Ihre Krankenversicherung wird 80% der Kosten direkt an den Arzt überweisen. Man benutze den Computer zum Schreiben der Rechnung für Frau Schmidt.

7. Bei einer Bürgermeisterwahl stellen sich drei Kandidaten zur Wahl, und zwar die Herren Tischler, Hoffmann und Weber. Die nach der Wahl erfolgte Auszählung der Stimmen in den vier Ortsteilen der betreffenden Gemeinde ergab folgendes Resultat:

Name	Ortsteil 1	Ortsteil 2	Ortsteil 3	Ortsteil 4
Tischler	698	732	129	487
Hoffmann	148	928	246	201
Weber	379	1087	148	641

Das Computerprogramm soll einmal die Gesamtzahl der Stimmen errechnen, die die einzelnen Kandidaten erzielt haben, sowie die Gesamtzahl der abgegebenen Stimmen.

Die von den nachfolgenden Programmen produzierte Ausgabe ist zu beschreiben!

8.
```
10 PRINT  8*2 - 3*(2^4-10)
20 END
```

9.
```
10 PRINT  "SILBER", "GOLD", "KUPFER", "PLATIN"
20 PRINT  327,448,1052,2
30 END
```

10.
```
10 PRINT  ,"LEBENSMITTEL", "FLEISCH", "DROGERIE"
20 PRINT   "MON", "1,245", "2,348", "2,531"
30 PRINT   "DIE", ":248", "3,459", "2,148"
40 END
```

Die nachfolgenden Zahlen sind in die Potenzform umzuwandeln!

Die Trennung der ganzzahligen Anteile der Zahlen von den gebrochenen Anteilen erfolgte durch Dezimalpunkte

11. 23,000,000
12. 175.25
13. -200,000,000
14. 0.000,14
15. -0.000000000275
16. 53,420,000,000,000,000

Die nachfolgenden Zahlen sind von der Potenzform in die gewöhnliche Schreibweise zu überführen.

17. 1.59E5
18. -20.3456E6
19. -7.456E-12
20. 2.39456E-18

Antworten zu den Testübungen

2.4.1	a)	4.8E-4	b)	-9700
		-1.3745E3		0.0097
				-0.0097

2.4.2 8.5 und 2

2.4.3 a)
```
10 PRINT  ((4*3+5*8+7*9)/(7*9+4*3+8*7))/48.7
20 END
```
b)
```
10 PRINT  .278*(112+38+42)
20 END
```
c)
```
10 PRINT  (88+79+84+49+63)/5
20 END
```

2.4.4
```
10 PRINT  ,"NAME"
20 PRINT
30 PRINT  "NACHNAME", "1. VORNAME", "2. VORNAME", "RANG"
40 PRINT
50 PRINT  "WIESNER", "ALBERT", "FRANZ",56
```

2.4.5
```
10 PRINT  ,"   BUDGET-APRIL"
20 PRINT  "LEBENSMITTEL",  387.50
30 PRINT  "AUTO",          475.00
40 PRINT  "ENERGIE",       123.71
50 PRINT  "VERSCHIEDENES", 146.00
60 PRINT  "UNTERHALTUNG",  100.00
70 PRINT  ,"------"
80 PRINT  "GESAMT", 387.50+475.00+123.71+146.00+100.00
90 END
```

2.4.6 a) 432
b) 76

2.5 Namensgebung für Zahlen und Wörter

Wir haben inzwischen sicherlich festgestellt, daß wir bei den Beispielen und Aufgaben des vorhergehenden Abschnittes erheblich viel Zeit damit verschwendet haben, daß wir bestimmte Zahlen immer und immer wieder eingegeben haben. Nicht nur das ständige Wiederholen der Eingabe kostet uns Zeit, sondern es ist auch eine ständige Fehlerquelle. Wenn wir Variablen benutzen, ist solch immer wiederkehrendes Eingeben unnötig.

Unter einer *Variablen* versteht man einen Buchstaben, der benutzt wird, um eine Zahl darzustellen. Jeder Buchstabe des Alphabets kann zu diesem Zweck herangezogen werden. (Darüberhinaus gibt es für die Variablen weitere mögliche Namen; siehe dazu die Erläuterungen weiter unten). Mögliche Variablen sind also A, B, C, X, Y oder Z. In einem gegebenen Augenblick besitzt jede Variable einen eigenen Wert. Zum Beispiel kann die Variable A den Wert 5 aufweisen, während die Variable B den Wert −2.137845 besitzt. Eine Methode, den Wert einer Variablen zu ändern, steht uns durch den Gebrauch der Anweisung LET zur Verfügung. Das Statement

```
10 LET  A=7
```

setzt die Variable A auf den Wert 7. Irgendein vorangegangener, A zugewiesener Wert wird damit gleichzeitig gelöscht.

Nachdem einer Variablen einmal ein Wert zugewiesen ist, kann die Variable das ganze Programm hindurch angesprochen werden. Der Computer fügt den zugewiesenen Wert überall

dort ein, wo die Variable auftritt. Um bei einem Beispiel zu bleiben: Wenn A den Wert 7 besitzt, dann wird der Ausdruck

```
A + 5
```

zu 7 + 5 oder 12 errechnet. Der Ausdruck

```
3*A - 10
```

wird zu $3 \cdot 7 - 10 = 21 - 10$ oder 11 ausgewertet. Und schließlich ergibt der Ausdruck

```
2*A∧2
```

den Wert 2*7∧2 = 2*49 oder 98.

Testübung 2.5.1

Es sei angenommen, daß die Variable A den Wert 4 und die Variable B den Wert 3 besitzt. Welcher Wert ergibt sich dann bei der Auswertung des Ausdrucks

```
A∧2/2*B∧2?
```

Man merke sich folgenden wichtigen Tatbestand:

> Wenn für eine arithmetische Variable kein Wert festgelegt ist, dann wird ihr durch BASIC der Wert 0 zugewiesen.

An dieser Stelle wollen wir dem Leser weiterhin drei nützliche Hinweise für die Programmierung geben.

1. Das Wort LET ist ein *optionales* Wort, d.h. seine Benutzung ist freigestellt. Somit kann z.B. das Statement

   ```
   10 LET A=5
   ```

   kürzer auch zu

   ```
   10 A=5
   ```

   geschrieben werden.

2. Auf einer Zeile können mehrere Statements untergebracht werden, doch sind sie dann voneinander durch einen Doppelpunkt zu trennen. Im besonderen könnte eine einzige Zeile benutzt werden, um mehreren Variablen Werte zuzuweisen. Hierfür sei ein Beispiel aufgeführt

```
100  LET C=18:  LET D=23:  LET E=2.718
```

Diese Anweisungsfolge bewirkt die Zuweisung des Wertes 18 zur Variablen C, des Wertes 23 zu D und des Wertes 2.718 zu E. Unter Benutzung der unter 1. besprochenen Abkürzungsmöglichkeiten können wir die Zeile 100 auch wie folgt schreiben:

```
100  C=18: D=23: E=2.718
```

3. Statements können über eine Zeile hinweg fortgesetzt werden. Uns erscheint das besonders nützlich, wenn wir vielen Variablen Werte zuweisen wollen (siehe Pkt. 2.). Wir bewerkstelligen die Ausdehnung einer Zeile durch die folgende Maßnahme: Nach Erreichung des physischen Zeilenendes (40 oder 80 Stellen) hören wir mit dem Eintippen nicht auf, fahren vielmehr damit fort. Wir drücken die Eingabetaste erst dann, wenn wir mit dem Eintippen der Zeichenfolge, die wir der betreffenden Zeilennummer zuordnen wollen, fertig sind. Eine dermaßen erweiterte Zeile *(logische Zeile)* kann bis zu 255 Zeichen umfassen. Nach Erreichung der 256. Stelle einer logischen Zeile, d.h. nach Eingabe von 255 Zeichen, beendet BASIC automatisch die Zeile gerade so, als ob wir die Eingabetaste betätigt haben.

Variablen können genau so gut auch in PRINT-Anweisungen benutzt werden. Z.B. verursacht das Statement

```
10 PRINT A
```

die Ausgabe des der Variablen A zugewiesenen Wertes, natürlich in der ersten Ausgabezone. Das Statement

```
20 PRINT A, B, C
```

wiederum führt zur Ausgabe der gegenwärtigen Werte der Variablen A, B bzw. D in den Ausgabezonen 1, 2 bzw. 3.

Testübung 2.5.2

Wir nehmen an, daß der Variablen A der Wert 5 zugewiesen ist. Welche Ausgabe wird durch die Anweisung

```
10 PRINT A,A^2,2*A^2
```

hervorgerufen?

Beispiel 1:

Es sind die drei Zahlen 5.71, 3.23 und 4.05 gegeben. Es sind ihre Summe, ihr Produkt sowie die Summe ihrer Quadrate (d.h. die Summe der 2. Potenzen, oft in der Statistik benutzt) zu berechnen.

Um zu einer einfachen überschaubaren Lösung zu gelangen, wollen wir die drei Variablen A, B und C einführen und ihnen die drei gegebenen Zahlen als Werte zuweisen. Anschließend können die gewünschten Resultate ermittelt werden. Das unter diesem Gedankengang erstellte Programm ist in der Abb. 2.6 aufgelistet.

```
10 LET A=5.71:B=3.23:C=4.05
20 PRINT "ALS SUMME ERGIBT SICH:",A+B+C
30 PRINT "ALS PRODUKT ERGIBT SICH:",A*B*C
40 PRINT
50 PRINT "ALS SUMME DER QUADRATE ERGIBT SICH:",A^2+B^2+C^2
60 END
```

Abb. 2.6 Programm zur Berechnung von Summen, Produkten und von Quadratsummen

Testübung 2.5.3

Es liegen die sechs Zahlen 101, 102, 103, 104, 105 und 106 vor. Es ist ein Programm zu schreiben, das der Reihe nach das Produkt der ersten zwei, der ersten drei, der ersten vier, der ersten fünf und schließlich aller sechs Zahlen errechnet.

Die folgende geistige Vorstellung ist sicher sehr nützlich zum Verständnis darüber, wie BASIC die Variablen handhabt. Wenn BASIC zum ersten Mal auf eine Variable, angenommen A, stößt, richtet es ein Kästchen ein (in Wirklichkeit einen Speicherplatz) und etikettiert es mit A (siehe Abb. 2.7). In dieses Kästchen wird der laufende Wert von A gespeichert. Wenn nun eine Änderung des Wertes von A erfolgt, wirft der Computer den augenblicklichen Inhalt aus dem Kästchen heraus und stellt den neuen Wert hinein.

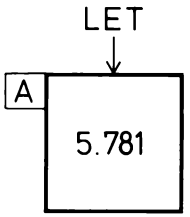


Abb. 2.7 Variable A

Dieser Gedankenflug vermittelt uns die Erkenntnis, daß der Wert einer Variablen während des Programmablaufes nicht gleich bleiben muß. An jeder Programmstelle kann man den Wert einer Variablen ändern, z. B. mit Hilfe des Statements (der Anweisung) LET. Wenn ein Programm den Wert eines Ausdrucks zu bestimmen hat, in den eine Variable verwickelt ist, so geht in die Auswertung immer der augenblickliche Wert der Variablen ein; Werte, die eine Variable an früher erreichten Programmstellen angenommen hatte, bleiben prinzipiell unbeachtet.

Testübung 2.5.4

Nehmen wir einmal an, daß uns ein Darlehen von 5000,- DM mit 1,5% monatlichen Zinsen, bezogen auf den Kontenstand am Ende eines jeden Monats, gewährt wurde.

Es ist ein Programm zu schreiben, das die Zinsen berechnet, die am Ende des 1. Monats anfallen. Wir nehmen weiterhin an, daß wir am Ende des 1. Monats eine Ratenzahlung (Abzahlung) von 150,- DM tätigen, terminlich nach Hinzufügung der angefallenen Zinsen. Das Programm soll so entworfen werden, daß es außer den Zinsen auch den Kontostand nach der Zahlung der Rate berechnet.

Die Variablen sollen wie folgt benannt werden:

K	Kontostand
Z	Zinsen
A	Ratenzahlung (Abzahlung)

Der neue Kontostand (Saldo) ergibt sich dann zu K+Z−A.

Beispiel 2:

Welche Ausgabe ergibt sich aus dem folgenden Programm, dargestellt in Abb. 2.8?

```
10 LET A=10:B=20
20 LET A=5
30 PRINT A+B+C,A*B*C
40 END
```

Abb. 2.8 Programm für Beispiel 2

Zur Beantwortung der Fragestellung ist zunächst festzustellen, daß der Variablen C von BASIC der Wert 0 zugewiesen wird, da im Programm selbst kein Statement vorhanden ist, das C einen Wert zuweist. Der Wert von A beträgt anfangs 10 (Zeile 10); er wird jedoch in Zeile 20 auf 5 geändert. Somit besitzen in Zeile 30 die Variablen A, B und C der Reihe nach die Werte 5, 20 und 0. Die Ausgabe führt also zu:

25 (in der 1. Ausgabezone)
0 (in der 2. Ausgabezone)

Für den Computer bedeutet das Statement

```
LET A = ···
```

die Ersetzung des augenblicklichen Wertes von A durch den Wert, der sich aus der Auswertung des Ausdruckes rechts vom Gleichheitszeichen ergibt. Durch die Niederschrift der Anweisung

```
LET A = A + 1
```

sagen wir also dem Computer, daß er auf den Wert, der augenblicklich A zugewiesen ist, eine 1 addieren soll; die entstehende Summe wird dann wieder A zugewiesen, d.h. ersetzt als neuer Wert den alten Wert von A. Wenn also beispielsweise der augenblickliche Wert von A gleich 4 ist, dann wird nach Ausführung dieser Anweisung A den Wert 5 besitzen.

Testübung 2.5.5

Welche Ausgabe wird in der Zeile 50 von den nachfolgend aufgelisteten Programm erzeugt?

```
10 LET A=5.3
20 LET A=A+1
30 LET A=2*A
40 LET A=A+B
50 PRINT A
60 END
```

2.5.1 Gültige Namen und Variablen

Wie wir zuvor schon erwähnt haben, kann man jeden Buchstaben des Alphabets als Variablennamen verwenden. Der IBM Personalcomputer ist jedoch ziemlich flexibel, was die Variablennamen betrifft. Jede Folge von bis zu 40 Zeichen, die mit einem Buchstaben beginnt, wird als gültiger Variablenname angesehen. Bezüglich von Ausnahmen weisen wir auf die weiteren Erklärungen dieses Unterabschnittes hin. Deshalb können Variablennamen gewählt werden, die bezüglich der Anwendungen sehr aussagekräftig sind, wie z.B. STEUERN, ZINSEN, KONTENSTAND, SALDO, LOHNLISTE, GEHALT, BERUF usw. Tatsächlich darf aber nicht jede Zeichenfolge als Variablenname gebraucht werden. Es muß auf diejenigen Zeichenfolgen verzichtet werden, die von BASIC reserviert sind, d.h. die in BASIC eine bestimmte Bedeutung besitzen. Hierzu zählen u.a. die Wörter

```
IF, ON, OR, TO, THEN, GOTO
```

Wenn wir erst vertrauter mit BASIC geworden sind, wird es uns zur zweiten Natur werden, diese und die anderen reservierten Wörter bei der Vergabe von Variablennamen zu vermeiden. Betont werden sollte auch, daß die Variablen A, AA und A1 für den Computer stets verschiedene Variablen darstellen.

Hinweis: Ein Variablenname muß stets mit einem Buchstaben beginnen

Ein Variablenname darf niemals mit einer Ziffer anfangen. 1A ist deshalb kein gültiger Variablenname.

2.5.2 Kettenvariablen

Alle bisher besprochenen Variablen repräsentierten numerische Werte. BASIC gestattet jedoch auch den Gebrauch von Variablen, die Kettenkonstanten als Werte annehmen können. Derartige Variablen werden *Kettenvariablen* genannt. Man erkennt sie daran, daß sie durch einen Variablennamen, dem ein *Währungszeichen* (hier: Dollarzeichen) angehängt ist, bezeichnet werden. Somit sind u.a. A$, B1$ und ZZ$ gültige Namen von Kettenvariablen. Zur Zuweisung eines Wertes zu einer Kettenvariablen verwenden wir das schon bekannte LET-Statement; der zuzuweisende Wert muß rechts vom Gleichheitszeichen aufgeführt werden und in Anführungszeichen eingeschlossen sein. Um z.B. A auf den Kettenwert "Kontoauszug" zu setzen, können wir die Anweisung

```
LET  A = "Kontoauszug"
```

codieren.

Der Wert einer Kettenvariablen kann genau so ausgegeben werden wie der Wert einer numerischen Variablen. Wird z.B. das Statement

```
PRINT A$
```

niedergeschrieben, so wird aus dem Bildschirm die Zeichenkette

```
Kontoauszug
```

angezeigt, falls A$ noch den soeben durch das LET-Statement zugewiesenen Wert besitzt.

Beispiel 3:

Wie sieht die Ausgabe des in der Abb. 2.9 aufgelisteten Programmes aus?

```
10 LET A$="MONATLICHE EINNAHMEN":B$="MONATLICHE AUSGABEN"
20 LET A=20373.10:B=17584.31
30 PRINT A$,B$
40 PRINT A,,B
50 END
```

Abb. 2.9 Programm für Beispiel 3

Als Lösung ergibt sich:

```
MONATLICHE EINNAHMEN        MONATLICHE AUSGABEN
20373.10                    17584.31
```

Wir sollten die Lösung noch unter verschiedenen Aspekten kommentieren. Wir haben im gleichen Programm sowohl die Variablen A und A$, wie auch B und B$ benutzt. Die Variablen A und A$ werden vom Computer als verschiedene Variablen angesehen; dasselbe gilt für die Variablen B und B$. Es sollte auch auf die Anwesenheit des zweiten Kommas in Zeile 40 hingewiesen werden. Das geschieht notwendigerweise wegen der Tatsache, daß der Wert von A$, die Kettenkonstante "MONATLICHE EINNAHMEN", 20 Stellen belegt und damit um 6 Stellen in die zweite Ausgabezone hineinragt. Um also zwischen den beiden Spaltenüberschriften einen Zwischenraum zu erhalten, wird der Wert von B$ automatisch an den Anfang der 3. Ausgabezone gestellt. Um die Werte von A und B nun korrekt auf die passenden Spaltenüberschriften auszurichten, müssen wir bei der Ausgabe dieser numerischen Werte die 2. Ausgabezone leer lassen. Das erreichen wir, wie wir früher festgestellt haben, durch das zweite Komma im PRINT-Statement der Zeile 40. Wenn wir nun die Ausgabe der Überschriften mit der Ausgabe der Zahlen vergleichen, werden wir feststellen, daß die entsprechenden Werte nicht exakt untereinander stehen: Die Zahlen sind um eine Stelle nach rechts verschoben. Das geschieht deshalb, weil BASIC Platz für ein Vorzeichen (+ oder −) vor der ersten Ziffer einer Zahl vorsieht. Sind die Zahlen jedoch positiv, wird das Vorzeichen bei der Ausgabe unterdrückt. Die für das Vorzeichen reservierte Stelle bleibt leer.

2.5.3 Kommentare bzw. Bemerkungen in Programmen

Es ist außerordentlich sinnvoll, Programme mit Hilfe von *Kommentaren* bzw. *Bemerkungen* zu erläutern. Einesteils hat dies zur Folge, daß Programme leichter lesbar sind, andererseits tragen sie dazu bei, Fehler zu finden und gewünschte Änderungen einzuarbeiten.

Um Kommentare bzw. Bemerkungen in Programme einzufügen, können wir uns des Statements REM bedienen. Betrachten wir dazu als Beispiel die Zeile:

```
520 REM   X BEDEUTET DIE KOSTENBASIS
```

Da diese Zeile mit dem Wort REM beginnt, bleibt sie bei der Programmausführung unbeachtet. Anstelle von REM kann auch einfach ein Apostroph (Auslassungszeichen) gebraucht werden; siehe dazu das nachfolgende Beispiel:

```
540   ' Y BEDEUTET DIE GESAMTKOSTEN
```

Um einen Kommentar bzw. eine Bemerkung auf der gleichen Zeile wie eine Programmanwei-

sung unterzubringen, verwende man den Doppelpunkt und lasse auf ihn ein Auslassungszeichen (oder das Wort REM) folgen; das nachstehende Beispiel illustriert dieses Vorgehen:

```
10 LET A=PI*R∧2:  ' A FLÄCHENINHALT, R RADIUS
```

Alle Zeichen, die nach dem Auslassungszeichen stehen, werden grundsätzlich als zur Bemerkung gehörig betrachtet. Somit ist es unmöglich, nach einer Bemerkung noch eine Anweisung auf der betreffenden Zeile unterzubringen. Beispielsweise wird in der nachfolgenden Zeile 20 die Zeichenfolge C = B + 8 nicht als Statement angesehen, sondern als Teil der Bemerkung.

```
20 LET B=A∧2:  ' B bedeutet Flächeninhalt:  C=B+8
```

Die Bedeutung von Kommentaren bzw. Bemerkungen kann nicht genug betont werden. Beim Schreiben von BASIC-Programmen ist es sehr leicht, die Programme so zu gestalten, daß niemand, der Verfasser inbegriffen, sie später zu lesen vermag. Man sollte deshalb unbedingt danach streben, Programme so zu schreiben, daß sie wie ein normaler Text gelesen werden können. Als bedeutsamster Schritt in dieser Richtung ist wohl die Aufnahme möglichst vieler Kommentare in ein Programm anzusehen. In den folgenden Ausführungen werden wir deshalb großzügig mit dem Gebrauch von Kommentaren umgehen, nicht nur, um unsere Programme leichter lesbar zu gestalten, sondern auch, um ein Beispiel guten Programmierstils zu geben.

Testübung 2.5.6

Welches Resultat ergibt die nachfolgende Programmzeile?

```
10 LET  A=7:B$="KOSTEN":C$="GESAMT":PRINT C$,B$,"=",A
```

2.5.4 Benutzung eines Druckers

Bei der Niederschrift von Programmen und der Analyse ihrer Ergebnisse ist es oft einfacher, sich auf gedruckte Ausgaben zu verlassen als auf Ausgaben, die auf dem Bildschirm erscheinen. In der Computerterminologie spricht man von *Hartkopien,* wenn man gedruckte oder geschriebene Ausgaben meint. Hartkopien können von einer großen Mannigfaltigkeit von Druckern geliefert werden, angefangen bei den Punktmatrixdruckern, die nur wenige Hundert DM kosten, bis hin zu den Typenraddruckern mit Kosten von mehreren Tausend DM. Gerade, wenn man anfängt, Drucker ernsthaft einzusetzen, findet man es schwierig, ohne Hartkopien zu arbeiten. Tatsächlich erweist sich die Programmierung als wesentlich einfacher, wenn man in den verschiedenen Stadien der Programmentwicklung Hartkopien zu Rate ziehen kann. Ein Grund dafür dürfte darin liegen, daß man, hat man gedruckte Listen zur Hand, beim Anschauen eines Programmes keinen Begrenzungen unterliegt. Bei Bildschirmen kann man sich immer nur „Schnappschüsse" von 25 Zeilen ansehen. Ebenso will man natürlich Drucker auch zur Erzeugung von Programmausgaben einsetzen, die von umfangreichen Zahlentabellen bis zu Adreßverzeichnissen und Dokumententexten reichen können.

Über den Drucker eines Personalcomputers können Hartkopien mittels des BASIC-Statements LPRINT produziert werden. So wird z. B. durch die Anweisung

```
10 LPRINT  A,A$
```

bewirkt, daß die augenblicklichen Werte der beiden Variablen A und A$ über den Drucker ausgegeben werden und zwar in den Druckzonen 1 und 2. Wie bei den Bildschirmen wird von BASIC eine Druckzeile in Druckzonen unterteilt; jede Druckzone weist 14 Stellen auf. Das Statement

```
20 LPRINT  "Kunde", "Kreditlimit", "Letzter Kauf"
```

resultiert also in der Ausgabe von drei Spaltenüberschriften für die ersten drei Druckzonen, nämlich

```
Kunde          Kreditlimit   Letzter Kauf
```

Wichtig ist es, daß man sich unbedingt eines immer vor Augen hält: Sollen die Ergebnisse eines Programmablaufes sowohl auf dem Bildschirm erscheinen als auch gedruckt werden, so ist es notwendig, die beiden Anweisungen PRINT und LPRINT zu benutzen. Wünscht man also z. B. die Ausgabe der Werte, die den Variablen A und A$ zugewiesen sind, sowohl auf dem Bildschirm als auch auf einer Hartkopie, so muß man in das entsprechende Programm die beiden Zeilen

```
10 PRINT   A,A$
20 LPRINT  A,A$
```

aufnehmen.

Aufgabengruppe 3

Bei den Aufgaben 1. bis 6. ist die Ausgabe des vorliegenden Programmes zu bestimmen.

1.
```
10 LET A=5: B=5
20 PRINT A+B
30 END
```

2.
```
10 LET AA=5
20 PRINT AA*B
30 END
```

3.
```
10 LET A1=5
20 PRINT  A1∧2 + 5*A1
30 END
```

4.
```
10 LET A=2: B=7: C=9
20 PRINT A+B, A-C, A*C
30 END
```

5.
```
10 LET A$="HANS ABEL"
20 LET B$="ALTER":  C=38
30 PRINT A$,B$,C
40 END
```

6.
```
10 LET  X=11: Y=19
20 PRINT 2*X
30 PRINT 3*Y
40 END
```

Welche Fehler weisen die nachfolgenden BASIC-Statements auf?

7. `10 LET A = "JUGEND"`

8. `10 LET AA = -12`

9. `10 LET A$ = 57`

10. `LET ZZ$ = Adresse`

11. `250 LET AAA = -9`

12. `10000 LET 1A = -2.34567`

13. Es seien die Zahlen 2.3758, 4.58321 und 58.11 gegeben. Schreibe ein Programm, das ihre Summe, ihr Produkt und die Summe ihrer Quadrate ermittelt.

14. Eine Firma ist u.a. in die drei folgenden Abteilungen gegliedert: Schreibbüro, Datenverarbeitung und Pressedienst. Diese drei Abteilungen konnten im vorangegangenen Quartal der Reihe nach die folgenden Einkünfte in DM erzielen: 346712, 459321 und 367872. Gleichzeitig fielen in dem betreffenden Quartal die folgenden Ausgaben in DM an: 176894, 584837 und 402195. Es ist ein Programm zu schreiben, das diese Daten auf dem Bildschirm anzeigt und zwar mit entsprechenden erklärenden Spaltenüberschriften und Zeilenhinweisen. Das Programm sollte darüberhinaus noch den Reinverdienst (bzw. den Verlust) jeder Abteilung sowie der drei Abteilungen insgesamt errechnen und ebenfalls anzeigen.

Antworten zu den Testübungen

2.5.1 72

2.5.2 Ausgabe in der 1. Druckzone: 5
Ausgabe in der 2. Druckzone: 25
Ausgabe in der 3. Druckzone: 50

2.5.3
```
10 LET A=101:B=102:C=103:D=104:E=105:F=106
20 PRINT A*B
30 PRINT A*B*C
40 PRINT A*B*C*C
50 PRINT A*B*C*D*E
60 PRINT A*B*C*D*E*F
70 END
```

2.5.4
```
10 LET K=5000:F=.015:A=150.00
20 LET Z=F*K
30 PRINT "ZINSEN",Z
40 K=K+Z
50 PRINT "KONTOSTAND MIT ZINSEN",K
60 K=K-A
70 PRINT "KONTOSTAND NACH ZAHLUNG",K
80 END
```

2.5.5 12.6

2.5.6 Es wird eine auf vier Druckzonen verteilte Ausgabe erzeugt.
1. Druckzone: GESAMT
2. Druckzone: KOSTEN
3. Druckzone: =
4. Druckzone: 7

2.6 Einige Befehle

Bis hierhin konzentrierten wir unsere Aufmerksamkeit auf das Erlernen von *Statements (Anweisungen),* die in ein Programm eingefügt werden können. Nunmehr wollen wir zu den Befehlen übergehen. Durch *Befehle* können wir Programme selbst handhaben bzw. die Arbeitsweise des Computers steuern und kontrollieren. Der Befehl NEW, den wir zuvor diskutieren, gehört zu dieser Kategorie.

Bevor wir mit der Besprechung beginnen, sollten wir uns die bisher bekannten Erkenntnisse über Befehle ins Gedächtnis zurückrufen. Wir können sie in den folgenden vier Punkten zusammenfassen:

1. Befehle werden ohne Zeilennummer eingegeben.
2. Nach Eingabe eines Befehles muß die Eingabetaste gedrückt werden.
3. Ein Befehl darf immer dann eingegeben werden, wenn sich der Computer im Befehlsmodus befindet. (Man erinnere sich, daß der Computer die Rückmeldung Ok anzeigt, wenn er zum ersten Mal in den Befehlsmodus eintritt; er verbleibt solange im Befehlsmodus, bis ein RUN-Befehl eingegeben wird).
4. Der Computer führt die Befehle sofort nach ihrem Empfang aus.

2.6.1 Das Auflisten eines Programmes

Um eine Liste aller Statements des augenblicklich im RAM gespeicherten Programmes zu erhalten, kann man den Befehl

```
LIST
```

eingeben.
Nehmen wir z.B. einmal an, daß der RAM das folgende Programm enthält:

```
10 PRINT 5+7, 5-7
20 PRINT 5*7, 5/7
30 END
```

(Dieses Programm kann im Augenblick der Eingabe des Befehles auf dem Bildschirm angezeigt sein oder auch nicht). Jedenfalls wird nach Eingabe von LIST und Drücken der Eingabetaste dieser Befehl sofort ausgeführt und mündet in der Anzeige der drei Programmzeilen, gefolgt von der üblichen Nachricht Ok.

Bei der Entwicklung eines Programmes wird man es zweifellos als notwendig erachten, daß man die Zeilen nicht in aufsteigender lückenloser Reihenfolge eingeben kann und daß man bereits eingegebene Zeilen korrigieren kann. Wenn sich ein solcher Zwang ergibt, wird der Bildschirm gewöhnlich nicht die laufende Version des Programmes anzeigen. Die Eingabe des Befehles LIST unterstützt den Benutzer bei seinem Bemühen, bezüglich des Programmzustandes immer auf dem laufenden zu bleiben, vor allem hinsichtlich der unternommenen Änderungen. Insbesondere ist LIST nützlich beim Überprüfen eines Programmes und bei Untersuchungen zur Klärung der Frage, warum ein Programm nicht läuft.

Hier ist der Hinweis angebracht, daß der Bildschirm des IBM Personalcomputers bis zu 25 Zeilen Text anzeigen kann. Das bedeutet, daß man sich nie mehr als 25 Programmzeilen gleichzeitig anzeigen lassen kann. Um nur diejenigen Zeilen aufzulisten, die durch die Zeilennummern 1–25 gekennzeichnet sind, verwenden wir den folgenden Befehl:

```
LIST 1-25
```

In ähnlicher Weise können wir jede Folge von Zeilen mit aufeinanderfolgenden Zeilennummern auflisten lassen.

Es gibt mehrere andere Variationen des Befehles LIST. Um die Programmzeilen vom Anfang des Programmes bis zur Zeile 75 aufzulisten, können wir den Befehl

```
LIST -75
```

verwenden. In gleicher Weise bewirkt der Befehl

```
LIST 100-
```

die Auflistung der Programmzeilen ab der Zeile mit der Zeilennummer 100 bis zum Programmende. Um sich die Zeile 300 anzusehen, genügt die Eingabe des Befehles

```
LIST 300
```

Testübung 2.6.1

Es sind Befehle zu formulieren, die die Auflistung der Zeilen eines Programmes bewirken:

a) Zeile 200

b) Zeilen 300 bis 330

c) Zeile 300 bis zur letzten Zeile des Programmes

Die einzelnen Befehle sind an einem beliebigen Beispielprogramm zu testen.

2.6.2 Drucken von Auflistungen

Man wird schon nach kurzer Programmierpraxis herausfinden, daß es schwierig ist, ein umfangreiches Programm zu schreiben und sich dabei nur auf den Bildschirm stützen. Solche Programme erfordern geradezu die Einsichtnahme in Auflistungen. Unter Benutzung des Druckers können diese ohne weiteres erstellt werden; Voraussetzung hierfür ist natürlich der Anschluß eines Druckers an die Systemeinheit. Um das augenblicklich im RAM gespeicherte Programm aufzulisten, ist der Befehl

```
LLIST
```

einzugeben und die Eingabetaste zu drücken. Alle Abarten des LIST-Befehles, die wir kennengelernt haben, gelten sinngemäß auch für den Befehl LLIST. Man kann also Zeilen herausdrucken, die in einem gewissen Bereich liegen, die Zeilen vom Programmanfang bis zu einer gegebenen Zeilennummer usw.

2.6.3 Löschen von Programmzeilen

Bei der Eingabe eines Programmes oder bei der Durchsicht eines existierenden Programmes ist es oftmals nötig, Zeilen zu löschen, die bereits zum Bestandteil des Programmes geworden sind. Ein einfacher Weg, das zu bewerkstelligen, besteht darin, die Zeilennummer einzutippen und sofort danach die Eingabetaste zu drücken. So bewirkt z. B. die Eingabe von

```
275
```

die Löschung der Zeile mit der Zeilennummer 275, wenn man die Eingabe mit der Betätigung der Eingabetaste abschließt. Für den gleichen Zweck kann jedoch auch der Befehl DELETE benutzt werden:

```
DELETE 275
```

Der DELETE-Befehl besitzt eine Menge von Variationen, die ihn zweifellos ziemlich flexibel machen. Zum Löschen der Zeilen 200 bis einschließlich 500 können wir diesen Befehl in der Form

```
DELETE 200-500
```

benutzen. Zum Löschen aller Zeilen vom Programmanfang bis zur Zeile 350 einschließlich nimmt der Löschbefehl die folgende Gestalt an:

```
DELETE -350
```

Der DELETE-Befehl muß stets auf eine letzte Zeilennummer Bezug nehmen. Dadurch soll unglückseligen Pannen vorgebeugt werden, durch die irrtümlich ungewollt große Programmteile gelöscht werden könnten. Wenn alle Zeilen, beginnend ab 100, bis zum Ende eines Programmes getilgt werden sollen, dann muß somit die Löschung von 100 bis zur letzten Zeilennummer spezifiziert werden. Erinnert man sich nun nicht mehr an diese, so muß man eben zunächst durch Eingabe des Befehles LIST die Schlußnummer ermitteln. Danach kann man den geeigneten DELETE-Befehl ausführen lassen. Ist allerdings das Programm sehr lang, kann man das Verlangen haben, die Auflistung zu vermeiden. Eine Löschung von Zeilen bis zum Programmende kann dann wie folgt vorgenommen werden:

Es sind zwei Eingaben zu tätigen. Nach Eintippen von

```
65535 END
```

folgt die Eingabe von

```
DELETE 100-65535
```

Durch Hinzufügen einer „künstlichen" Hilfszeile zum Ende des Programmes (65535 ist die größtmögliche Zeilennummer, die vergeben werden kann) wird also hier das Löschen möglich gemacht.

Testübung 2.6.2

Welche Fehler liegen bei den nachfolgenden Befehlen vor?

a) `DELETE 450-`

b) `LIST 450-`

c) `DELETE 300-200`

2.6.4 Sicherstellung von Programmen

Nachdem man ein Programm in den RAM eingetippt hat, besteht möglicherweise der Wunsch, eine Kopie desselben auf einer Diskette sicherzustellen. Später kann man dann nämlich die Diskettenkopie in den RAM wieder einlesen. Das Neueintippen des Programmes entfällt somit. Nach dem Wiedereinlesen kann man das Programm erneut ausführen lassen oder es modifizieren oder es erweitern. Um der Konkretheit willen wollen wir einmal annehmen, daß sich das folgende Programm im RAM befindet:

```
10 PRINT 5+7
20 END
```

Um ein Programm sicherstellen zu können, muß man ihm zunächst einen Namen geben, es benennen. Ein *Programmname* ist eine Zeichenkette aus Buchstaben und Ziffern, die bis zu 40 Zeichen umfassen darf. Zusätzlich kann eine Erweiterung an den Namen angehängt werden; die Erweiterung muß aus einem Punkt und nachfolgenden drei Zeichen bestehen. Einige Beispiele für gültige Programmnamen folgen nun:

```
ABRECHNUNG1
SPIELE.IDA
GESCHICHTE.003
```

Wenn keine Erweiterung angegeben wird, hängt BASIC automatisch an den vergebenen Programmnamen die Erweiterung .BAS an.

Nehmen wir einmal an, wir hätten uns für den Programmnamen AUFBEWAHRUNG entschieden. Wir können unser Programm auf einer Diskette sicherstellen, die wir in irgendein Laufwerk eingeführt haben. Wenn wir z.B. das Programm auf die Diskette im Laufwerk B: speichern wollen, müßten wir den Befehl

```
SAVE "B:AUFBEWAHRUNG"
```

eingeben. Nach Beendigung der Aufzeichnung der Programmkopie auf die designierte Diskette, meldet sich der Computer wieder mit der üblichen Rückmeldung Ok und ist damit bereit für andere Arbeiten. Die Sicherstellung eines Programmes beeinflußt das Programm im RAM nicht; dieses bleibt also in seiner bestehenden Form erhalten.

Die ersten acht Zeichen des Programmnamens werden als Dateiname derjenigen Datei angesehen, in die das Programm abgelegt wird.

2.6.5 Zurückrufen von Programmen

Um ein Programm von einer Diskette zurück in den RAM zu lesen, bedienen wir uns des Befehles LOAD. Um z.B. das Programm AUFBEWAHRUNG von der Diskette im Laufwerk B: zurück in den RAM zu bringen, verwenden wir den Befehl

```
LOAD "B:AUFBEWAHRUNG"
```

Man sollte die in den Unterabschnitten 2.6.4 und 2.6.5 beschriebenen Befehle einmal mit dem kleinen, in 2.6.4 gegebenen Beispielprogramm ausprobieren. Nach Sicherstellung des Programmes mittels des Befehles SAVE ist es im RAM durch Eintippen des Befehles NEW zu löschen. Danach ist das Programm mittels des Befehles LOAD wieder in den RAM zu laden. Die Überprüfung, daß das Programm nun wieder im RAM gespeichert ist, kann durch Eingabe des Befehles LIST erfolgen.

2.6.6 Löschen von auf Disketten gespeicherten Programmen

Mit Hilfe des Befehles KILL kann ein Programm gelöscht werden, das sich in einer Datei auf einer Diskette befindet. Beispielsweise ist das in den vorhergehenden Unterabschnitten erwähnte Programm AUFBEWAHRUNG tatsächlich unter der Bezeichnung

```
AUFBEWAH.BAS
```

in einer Datei auf der Diskette gespeichert. Um dieses Programm zu löschen, ist die Eingabe des Befehles

```
KILL "AUFBEWAH.BAS"
```

erforderlich. Der Dateinamenszusatz, hier .BAS, muß im KILL-Befehl stets aufgeführt, kann also nicht weggelassen werden. Dadurch wird eine gewisse Sicherheit gegenüber dem unbeabsichtigten Löschen von Dateien mit BASIC-Programmen erreicht.

2.6.7 Manipulierung von Zeilennummern

Es sind mehrere Befehle vorgesehen, die den Benutzer von der Last, sich intensiv mit Zeilennummern beschäftigen zu müssen, weitgehend befreit.

Der Befehl AUTO kann zur automatischen Generierung von Zeilennummern eingesetzt werden. Um diese Möglichkeit auszunutzen ist der Befehl

```
AUTO
```

einzugeben und die Eingabetaste zu drücken. Als Folge davon werden nun die Zeilennummern 10, 20, 30, ··· erzeugt. Das wirkt sich so aus, daß eine Zeilennummer angezeigt und der Positionsanzeiger (Cursor) auf die zweite Stelle hinter der Zeilennummer geführt wird. Als Reaktion hierauf kann nun die entsprechende Programmzeile eingegeben werden; man erspart sich somit das Eintippen der Zeilennummer. Wie gewöhnlich ist die Eingabe durch Drücken der Eingabetaste abzuschließen. Nunmehr ist der Computer an der Reihe: er zeigt auf dem Bildschirm die nächste Zeilennummer an. Danach kommt wieder der Benutzer zum Zuge. Dieser Dialog zwischen Benutzer und Computer kann so lange fortgesetzt werden, wie man es wünscht.

Um die automatische Zeilennumerierung außer Kraft zu setzen, sind gleichzeitig die beiden Tasten

Ctrl und *Break*

niederzudrücken. Als Resultat dieser Aktivität erscheint die bekannte BASIC-Rückmeldung Ok.

Es wird dem Leser sicherlich schon aufgefallen sein, daß wir stets Zeilennummern benutzt haben, die Vielfache von 10 darstellen. Es liegen gute Gründe dafür vor, daß „anscheinend mit den Zeilennummern gewüstet wird". Öfters wird es sich nämlich als notwendig erweisen, Anweisungen zwischen bereits bestehende Anweisungen einzuschieben. Unser Nummernschema läßt Platz für die Einfügung von bis zu 9 Ergänzungen. So können wir beispielsweise zwischen die Zeilen mit den Zeilennummern 40 und 50 bis zu neun Zeilen mit den Zeilennummern 41, 42, · · ·, 49 einschieben.

Es gibt mehrere sinnvolle Variationen des AUTO-Befehles. Man kann die automatische Generierung der Zeilennummern an jeder beliebigen Stelle beginnen lassen. Zum Beispiel ist zur Erzeugung der Zeilennummern

55,65,75

einfach der Befehl

```
AUTO 55
```

zu benutzen. Ebenso kann man auch die Schrittweite zwischen den zu generierenden Zeilennummern beeinflussen. Um z. B. die Folge der Zeilennummern

38,43,48,53,58

erzeugen zu lassen (Anfangsnummer 38, Schrittweite 5), ist der Befehl

```
AUTO 38,5
```

erforderlich.

Es ist ferner für eine automatische Neunumerierung der Zeilen Sorge getragen. Diese Möglichkeit erweist sich sicher dann als vorteilhaft, wenn es notwendig ist, eine Anweisungsmenge aus zwei einzelnen Anweisungsmengen zusammenzustellen, deren Zeilennummern sich überlappen. Der Befehl

```
RENUM
```

bewirkt, daß alle Zeilen eines Programmes neu numeriert werden. Die Numerierung beginnt dabei mit der Zeilennummer 10. Wie beim Befehl AUTO können auch beim Befehl RENUM mehrere sinnvolle Abarten eingesetzt werden. Um die Zeilen eines Programmes so neu zu numerieren, daß sie mit der 1000 beginnen, ist der Befehl

```
RENUM 1000
```

einzugeben.

Die Neunumerierung kann auf einen Programmteil beschränkt werden. Damit die Zeilen mit den Zeilennummern von 200 an aufwärts mit neuen Zeilennummern, beginnend mit 1000, versehen werden, muß der Befehl

```
RENUM 1000,200
```

verwendet werden. Alle Zeilennummern unterhalb 200 bleiben durch diesen Befehl unberührt. Beenden wir die Besprechung mit einem letzten Beispiel. Um die Zeilen von 200 an aufwärts neu zu numerieren, beginnend bei der Zeilennummer 1000 und fortfahrend mit der Schrittweite 100, ist der Befehl

```
RENUM 1000,200,100
```

einzugeben.

Aufgabengruppe 4

Die Aufgaben 1 bis 7 beziehen sich auf das folgende Programm

```
10 LET  A = 19.1: B= 17.5
20 PRINT A+B,A*B
30 END
```

1. Das obige Programm ist in den RAM einzugeben und auszuführen. Zur Erzeugung der Zeilennummern ist die AUTO-Einrichtung zu verwenden.
2. Der Bildschirm ist zu löschen, der RAM jedoch nicht. Das Programm ist danach aufzulisten.
3. Das Programm ist sicherzustellen; danach ist der RAM zu löschen.
4. Das Programm ist zurückzurufen und aufzulisten; danach soll erneut seine Ausführung veranlaßt werden.
5. Die folgende Zeile ist in das Programm einzufügen:

   ```
   25 PRINT  A∧2 + B∧2
   ```

   Dabei soll das gesamte Programm nicht erneut eingegeben werden; es ist vielmehr nur die neue Zeile hinzuzufügen. Das erweiterte Programm ist aufzulisten. Es ist ferner seine Ausführung zu veranlassen.
6. Das neue Programm ist ohne Zerstörung des alten Programmes sicherzustellen.
7. Das neue Programm ist zurückzurufen. Danach ist die Zeile 20 zu löschen. Anschließend ist die Ausführung des geänderten Programmes zu veranlassen.
8. Die Zeilen des bei 7. geänderten Programmes sollen mit 100, 200, 300 neu numeriert werden.
9. Die Zeilennummern des Programmes von Aufgabe 8. sollen geändert werden, und zwar in 10, 2000, 2005.

Antworten zu den Testübungen

2.6.1 a) `LIST 200`
b) `LIST 300-330`
c) `LIST 300-`

2.6.2 a) Die Zeilennummer der letzten zu löschenden Zeile muß genannt werden. Soll bis Zeile 450 einschließlich gelöscht werden, so muß der Befehl

```
DELETE -450
```

lauten.

b) Der Befehl ist fehlerfrei.

c) Die niedrigere Zeilennummer muß zuerst genannt werden. Demzufolge muß der Befehl

```
DELETE 200-300
```

lauten.

2.7 Einige Programmiertips

Das Programmieren in BASIC ist nicht schwer. Es erfordert jedoch stets eine gewisse Sorgfalt und peinlich genaue Aufmerksamkeit im Detail. Jeder muß danach trachten, seinen eigenen, seinen individuellen Programmierstil zu entwickeln.

Nachstehend wollen wir einige Hinweise für den Anfangsprogrammierer geben, die ihm helfen sollen, sich über die Widrigkeiten hinwegzusetzen, die beim Schreiben der ersten Programme auftauchen.

Programmiertips

1. Man überdenke sorgfältig das zu schreibende Programm und breche die vom Programm zu verrichtende Arbeit in einzelne Schritte auf. Jeder einzelne Schritt sollte klar und unmißverständlich in der deutschen Sprache beschrieben werden. (*Merke:* Wenn man sich selbst nicht verständlich machen kann, was der Computer tun soll, kann man es auch kaum dem Computer sagen!).
2. Für jeden herausgearbeiteten Schritt ist eine Anweisungsmenge zu schreiben. Die Anweisungen sind visuell sorgfältig zu überprüfen; mit „Adleraugen" suche man dabei nach falschen Schreibweisen, nach fehlenden Klammern und nach anderen Flüchtigkeitsfehlern.
3. Das Programm ist mit Kommentaren und Bemerkungen reichlich zu „pfeffern".
4. Das Programm ist so zu gestalten, daß es wie eine Geschichte gelesen werden kann. (Wie man das bewerkstelligt, darüber mehr im nächsten Kapitel).
5. Das geschriebene Programm sollte per Hand durchgegangen werden (Schreibtischtest!). Dabei sollte man so tun, als wäre man selbst der Computer. Man überstürze nichts! Das Programm ist Schritt für Schritt durchzugehen und dabei zu überprüfen, ob es so arbeitet, wie man es beabsichtigt hatte.
6. Sind allen Variablen die Werte zugewiesen worden, die sie besitzen sollen? Man erinnere sich, daß BASIC automatisch den arithmetischen Variablen den Wert 0 zuweist, wenn man für sie keinen Wert spezifiziert hat. Der Wert 0 braucht keinesfalls der Wert zu sein, der für eine Variable beabsichtigt war.

In den folgenden Kapiteln wollen wir dem Leser nicht nur beibringen, wie er in BASIC zu programmieren hat. Wir wollen ihn auch dazu bringen, gute Programmierungsunterlagen zu entwickeln, und zu einem nützlichen, sinnvollen Programmierstil zu ermutigen. Dieses Vorhaben führt uns dazu, die obige Zusammenstellung von Programmiertips laufend zu vervollständigen.

2.8 Benutzung des Basiseditors (Zeileneditors)

Nehmen wir einmal an, daß wir eine fehlerhafte Programmzeile entdeckt haben. Wie können wir sie korrigieren?

Diese Frage werden wir uns mit Sicherheit öfters stellen müssen. Bis zu diesem Zeitpunkt kennen wir nur den Weg, die ganze Zeile neu einzutippen. Es gibt aber eine viel bessere Methode. Der IBM Personalcomputer ist mit einem sehr wirkungsvollen *Editor* ausgerüstet. Er gestattet uns das *Hinzufügen, Löschen* und *Ändern* von Textstellen in existierenden Programmzeilen. In diesem Abschnitt wollen wir uns intensiv mit dem Editor beschäftigen.

Der *Edierungs-* oder *Aufbereitungsprozeß* (der Prozeß der Änderung oder Korrektur von bereits eingegebenen Zeichen also) besteht aus drei Schritten:

1. Anzeige der zu ändernden Stelle
2. Eingabe der Änderung
3. Übertragen der Änderung an den Computer mittels Niederdrückens der Eingabetaste

Diese Schritte erfordern den Gebrauch einer Anzahl spezieller Edierungstasten. Die meisten dieser Tasten sind im numerischen Block der Tastatur zu finden. Wie wir bereits früher gesagt haben, kann der numerische Block zur Eingabe von Ziffern verwendet werden, gerade so, wie es bei Tisch- und Taschenrechnern geschieht. Um den numerischen Block in dieser Weise benutzen zu können, muß die Taste *Num Lock* eingerastet sein. Ist diese Taste jedoch ausgekuppelt, übernehmen die Tasten des numerischen Blockes andere Funktionen. Diese sind auf den Tasten durch Symbole, wie ↑, ←, →, ↓, DEL usw., vermerkt. Diese Alternativfunktionen dienen dem Edieren. Es sei in diesem Zusammenhang noch einmal auf einen bereits früher gebrachten Hinweis verwiesen: die Taste *Num Lock* befindet sich beim Einschalten des Computers in der ausgekuppelten Position, d.h. der numerische Block ist für das Edieren präpariert.

Am besten versteht man das Edieren, wenn man sich durch mehrere Beispiele hindurcharbeitet. Soweit es überhaupt möglich ist, sollte man diesen Beispielen folgen, indem man sie Schritt für Schritt auf der Tastatur nachspielt. Nehmen wir zunächst einmal an, daß wir die folgenden Programmzeilen eingetippt haben:

```
10 PRIMT  X,Y,Z
20 IF  A = 5  THN 50  ELSE 30
_
```

Die dritte Zeile zeigt die Stellung des Positionsanzeigers (Cursors) an. Wir erkennen sofort zwei Schreibfehler, nämlich PRIMT und THN. (Hätte der Computer einen gesunden Menschenverstand, dann würde er jetzt wissen, was der Benutzer gemeint hat!). Zusätzlich nehmen wir an, daß wir die Variablenfolge X, Y, Z in der ersten Zeile in die Folge A, X, Y, Z ändern müssen. Schließlich wollen wir noch die Klausel

```
ELSE 30
```

auf der zweiten Zeile entfernen. Wir wollen in einem Edierungsprozeß alle von uns beabsichtigten Korrekturen vornehmen. Zum ersten Schritt müssen wir den Cursor auf die Stelle positionieren, die das erste zu berichtigende Zeichen enthält. Wir können das erreichen, indem wir den Cursor bewegen, wozu uns verschiedene Tasten auf dem numerischen Block zur Verfügung stehen. Hierbei handelt es sich um folgende Tasten:

Symbol auf der Tastatur	*Bewegung des Cursors um · · ·*
↑	eine Zeile nach oben
↓	eine Zeile nach unten
←	eine Stelle nach links
→	eine Stelle nach rechts

Es sind noch weitere Tasten vorhanden, mit denen der Cursor bewegt werden kann, aber wir wollen vorerst darauf verzichten, sie kennenzulernen. Um den Fehler im Wort PRIMT zu korrigieren, müssen wir zunächst den Positionsanzeiger auf die Stelle positionieren, auf der der Buchstabe M steht. Dazu schlagen wir zuerst zweimal die Taste ↑ an. Der Cursor rückt damit um zwei Zeilen nach oben. Die Bildschirmanzeige sieht nun wie folgt aus:

```
10 PRIMT  X,Y,Z
20 IF  A = 5  THN 50  ELSE 30
```

Danach drücken wir sechs Mal auf die Taste →, wodurch der Cursor sechs Stellen nach rechts rückt; hierbei ist natürlich die Leerstelle zwischen 0 und P mitzuzählen. Die Bildschirmanzeige nimmt jetzt die folgende Form an:

```
10 PRIMT  X,Y,Z
20 IF  A = 5  THN 50  ELSE 30
```

Wir haben damit den Schritt 1 vollzogen: Der Positionsanzeiger befindet sich auf der Stelle, die das fehlerhafte Zeichen enthält. Jetzt kann der Schritt 2 erledigt werden. Wir tippen die vorgesehene Änderung ein; in diesem Fall schlagen wir also die Taste mit dem Buchstaben N an. Der falsche Buchstabe M wird somit durch N ersetzt. Die Anzeige auf dem Bildschirm hat sich damit wie folgt geändert:

```
10 PRINT  X,Y,Z
20 IF  A = 5  THN 50  ELSE 30
```

Der erste Fehler ist nunmehr berichtigt. Wir haben jedoch noch nicht die korrigierte Zeile zum Computer geschickt, weil wir anschließend nicht die Eingabetaste gedrückt haben. Wir könnten das zwar in diesem Augenblick tun, doch hätte das wenig Sinn, denn es gibt auf dieser Zeile noch eine weitere Berichtigung vorzunehmen. Wir wollen uns deshalb zuerst diesem Fehler widmen: Es sind die beiden Zeichen A, vor dem Buchstaben X einzufügen. Wir beginnen damit, daß wir den Cursor um drei Stellen nach rechts bewegen, so daß die Bildschirmanzeige die folgende Gestalt annimmt:

```
10 PRINT  X,Y,Z
20 IF  A = 5  THN 50  ELSE 30
```

Um Zeichen an der Stelle einzufügen, auf der der Cursor steht, drücken wir die Taste *Ins* (Insert-Einfügen) und tippen anschließend die fehlenden Zeichen, nämlich A, ein . Die Taste *Ins* versetzt die Tastatur in den *Einfügungszustand.* In diesem Zustand werden die eingetippten Zeichen dort eingefügt, wo sich der Cursor befindet, der restliche Text der Zeile wird um

die entsprechende Stellenzahl nach rechts verschoben. Nach dem Einfügen ergibt sich die folgende Bildschirmanzeige:

```
10 PRINT  A,X,Y,Z
20 IF  A = 5  THN 50  ELSE 30
```

Nach Beendigung des Einfügens müssen wir natürlich den Einfügungszustand wieder aufheben. Das kann auf mehreren Wegen geschehen. Eine Methode besteht darin, die Taste *Ins* erneut zu drücken. Das würde uns erlauben, mit weiteren Korrekturen auf der gleichen Zeile fortzufahren. Eine andere Methode, die wir in diesem Fall vorziehen, besteht darin, daß wir die Eingabetaste betätigen. Dadurch wird ebenfalls der Einfügungszustand aufgehoben und gleichzeitig die korrigierte Zeile zum Computer geschickt. Bei dieser Methode ist es völlig gleichgültig, auf welcher Position der Zeile sich der Cursor in dem Moment befindet, in dem die Eingabetaste gedrückt wird. Die Anzeige auf dem Bildschirm sieht nunmehr wie folgt aus:

```
10 PRINT  A,X,Y,Z
20 IF  A = 5  THN 50  ELSE 30
```

Der Positionsanzeiger ist, wie wir erkennen, durch das Drücken der Eingabetaste auf die 1. Stelle der nächsten Zeile, der Zeile mit der Zeilennummer 20 also, gerückt. Wir korrigieren die falsche Schreibweise des Wortes THEN, indem wir nacheinander die folgenden Schritte durchführen:

- Einstellen des Cursors auf die Stelle, die das N enthält (Bewegung um 16 Stellen nach rechts)
- Drücken der Taste *Ins*
- Drücken der Taste mit dem Buchstaben E
- Drücken der Taste *Ins*

Die Bildschirmanzeige hat sich dadurch wie folgt geändert:

```
10 PRINT  A,X,Y,Z
20 IF  A + 5  THEN 50  ELSE 30
```

Die letzte durchzuführende Berichtigung ist die Entfernung der Klausel ELSE 30. Wir erreichen das durch Benutzung der Taste *Del*. Zuerst müssen wir allerdings den Cursor auf die Stelle führen, die das erste E des Wortes ELSE enthält. Danach können wir sieben Mal die Taste *Del* betätigen. Jedes Anschlagen dieser Taste hat zur Folge, daß das Zeichen auf der augenblicklichen Cursorposition verschwindet und der verbleibende Text auf der Zeile um eine Stelle nach links verschoben wird. Nach dem erstmaligen Drücken der Taste *Del* wird deshalb z. B. die Bildschirmanzeige das folgende Aussehen aufweisen:

```
10 PRINT  A,X,Y,Z
20 IF  A = 5  THEN 50  LSE 30
```

Nach sechsmaligem Wiederholen, also nach insgesamt sieben Anschlägen der Taste *Del* wird die Bildschirmanzeige wie folgt aussehen:

```
10 PRINT  A,X,Y,Z
20 IF  A = 5  THEN 50  _
```

Die Berichtigungen sind dann abgeschlossen: wir schicken deshalb die Zeile zum Computer, d.h. wir betätigen die Eingabetaste.

Das soeben besprochene Beispiel illustriert einige der mannifgaltigen Edierungseinrichtungen, über die der IBM Personalcomputer verfügt. Wir können die Edierungstasten in der gleichen, soeben beschriebenen Weise benutzen, um irgendeine Zeile auf dem Bildschirm zu ändern. Wenn wir dagegen eine Zeile modifizieren müssen, die augenblicklich nicht auf dem Bildschirm angezeigt wird, müssen wir zuvor für die Anzeige der entsprechenden Zeile sorgen. Hierzu verhilft uns der Befehl LIST. Danach kann das Edieren so stattfinden, wie wir es besprochen haben.

Es gibt eine Reihe anderer Tasten, mit deren Hilfe das Edieren schneller erledigt werden kann. Um z.B. die Bewegung des Positionsanzeigers zu beschleunigen, können wir die folgenden Tasten anschlagen:

- *Home*
  Durch das Anschlagen dieser Taste wird der Cursor in die linke obere Ecke des Bildschirms gestellt, d.h. in die sogenannte Heimposition.

- *Ctrl* und *Home*
  Die gleichzeitige Betätigung dieser zwei Tasten löscht den Bildschirm und setzt den Cursor auf seine Heimposition.

- *Ctrl* und → (im numerischen Block)
  Die gleichzeitige Betätigung dieser zwei Tasten führt den Cursor auf den Platz unmittelbar rechts vom Beginn des nächsten Wortes. (Man denke sich ein Wort als irgendeine Folge von Zeichen, die frei von Leerzeichen, d.h. Zwischenraumzeichen, ist. Wir haben uns hier zwar nicht korrekt ausgedrückt, aber für die praktischen Bedürfnisse reicht diese Erklärung aus.).

- *Ctrl* und ← (im numerischen Block)
  Die gleichzeitige Betätigung dieser zwei Tasten führt den Cursor auf den Platz unmittelbar links vom Beginn des nächsten Wortes.

- *End*
  Das Drücken dieser Taste führt den Cursor auf das Ende des Textes auf der Zeile, in der der Cursor vor der Betätigung der Taste stand.

Zusätzlich zu den soeben abgehandelten *Edierungstasten* können noch die folgenden Tastenkombinationen nützliche Funktionen beim Edieren ausüben.

- *Ctrl* und *End*
  Das gleichzeitige Drücken dieser zwei Tasten löscht auf dem Bildschirm alle Zeichen von der augenblicklichen Stellung des Cursors ab bis zum Zeilenende.

- *Ctrl* und *Break*
  Das gleichzeitige Drücken dieser zwei Tasten streicht alle beim Edieren vorgenommenen Änderungen in der gegenwärtig bearbeiteten Zeile.

Der Wichtigkeit wegen sollten wir zum Schluß noch einmal darauf hinweisen, daß die beim Edieren vorgenommenen Änderungen nur in die Programmkopie eingehen, die im RAM gespeichert ist. Wenn Änderungen auf die Programmkopien durchschlagen sollen, die auf Kassetten oder Disketten gespeichert sind, dann ist es notwendig, die im RAM vorliegende edierte Kopie mittels des Befehles SAVE sicherzustellen.

Aufgabengruppe 5

Durch welche Tastenanschläge werden bei den Aufgaben 1. bis 10. die genannten Edierungsfunktionen veranlaßt?

1. Bewegung des Positionsanzeigers um vier Stellen nach rechts
2. Löschung des vierten Zeichens rechts vom Positionsanzeiger
3. Einfügung der Zeichenfolge 538 auf die augenblickliche Position des Cursors
4. Löschung des Teils der Zeile rechts von der augenblicklichen Cursorposition
5. Bewegung des Positionsanzeigers um acht Zeilen nach oben
6. Bewegung des Positionsanzeigers um drei Stellen nach links
7. Auflistung der gegenwärtigen Fassung einer Zeile
8. Änderung der 0 auf der gegenwärtigen Position des Cursors in die 1
9. Löschung des acht Stellen links von der gegenwärtigen Position des Cursors befindlichen Buchstaben a
10. Streichung aller Änderungen in der gerade bearbeiteten Zeile.

Bei den Aufgaben 11. bis 15. ist der Zeileneditor zu benutzen, um die geforderten Änderungen in der Programmzeile durchzuführen, deren Inhalt augenblicklich wie folgt lautet

```
300 FOR M=11 TO 99, SETP .5:  X = M∧2 - 5
```

Die Änderungen sind in der Reihenfolge der Aufgaben zu erledigen:

11. Löschung des Kommas
12. Korrektur der falschen Schreibweise des Wortes STEP (hier zu SETP geschrieben)
13. Änderung des arithmetischen Ausdrucks

    ```
    M∧2 - 5
    ```

    in den Ausdruck

    ```
    M∧3 - 2
    ```

14. Änderung der numerischen Konstante .5 in die numerische Konstante −1.5
16. Hinzufügung der Zeichenkette

    ```
    :  Y = M
    ```

    am Ende der Zeile

3 Steuerung des Programmablaufes

In diesem Kapitel wollen wir mit der Einführung in die Programmiersprache BASIC (Diskettenversion) für den IBM Personalcomputer fortfahren. Die Anweisungen zur Steuerung des Programmablaufes sollen dabei in den Mittelpunkt unserer Diskussionen gestellt werden.

3.1 Ausführung von Operationswiederholungen (Schleifen)

Wir wollen einmal annehmen, daß 50 einander ähnliche Multiplikationen durchgeführt werden sollen. Grundsätzlich ist es natürlich möglich, die 50 geforderten Multiplikationen auf einmal nacheinander einzutippen und sie anschließend dem Computer zur Lösung übergeben. Wenn man auf diese Weise an ein solches Problem herangeht, beschreitet man einen ziemlich plumpen Weg. Bei 50 vorgegebenen Multiplikationen wäre er vielleicht gerade noch gangbar. Was soll aber geschehen, wenn stattdessen 500 oder gar 5000 Multiplikationen anfallen? Eine solche Anzahl von Multiplikationsaufgaben auf einmal einzutippen, erweist sich sicher nicht als praktikabel. Gelingt es uns jedoch, dem Computer die gesamte zu lösende Aufgabe in einer geschlossenen Form zu beschreiben, dann benötigen wir nur wenige BASIC-Anweisungen, um den Computer zu veranlassen, dieses Problem in Angriff zu nehmen.

Um diese Behauptung zu erhärten, wollen wir ein konkretes Problem betrachten. Angenommen, es sollen der Reihe nach die Potenzen

$$1^2,\quad 2^2,\quad 3^2,\qquad\qquad ,\quad 10^2$$

berechnet werden. Es soll also eine Tabelle der Quadrate der ganzen Zahlen von 1 bis 10 erstellt werden. Der eigentliche Rechengang kann dem Computer, das wissen wir bereits, durch den Ausdruck

```
N∧2
```

beschrieben werden. In diesem Ausdruck stellt N eine Variable dar, der der Reihe nach die Werte 1, 2, 3, · · ·, 10 zugewiesen werden müssen. Die Folge von BASIC-Statements in der Abb.3.1 vollführt die von uns ins Auge gefaßten Berechnungen.

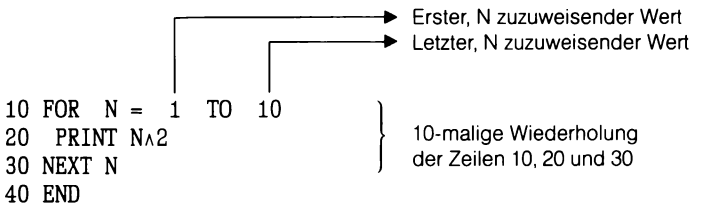


Abb.3.1 Aufbau einer Schleife

Die Statementfolge in den Zeilen 10, 20 und 30 wird *Schleife* genannt. Wenn der Computer auf das Statement FOR stößt, wird die Variable N auf den Wert 1 gesetzt und mit der Ausfüh-

rung der Anweisungen fortgefahren. Das Statement auf Zeile 20 bewirkt die Ausgabe von N∧2. Da N gleich 1 gesetzt wurde, erhalten wir also

N∧2 → 1∧2 = 1

Somit gibt der Computer die Zahl 1 aus. Als nächstes kommt das Statement auf Zeile 30 an die Reihe; dieses Statement verlangt das „nächste" N, besser gesagt: den nächsten Wert von N. Damit wird also der Computer instruiert,

- zum FOR-Statement in Zeile 10 zurückzukehren,
- den Wert, der N zugewiesen ist, um 1 zu erhöhen (hiermit wird also N der Wert 2 zugewiesen),
- die Anweisungen auf den Zeilen 20 und 30 zu wiederholen.

Somit ergibt sich jetzt bei der erneuten Auswertung des Ausdrucks

N∧2 → 2∧2 = 4

Durch die Anweisung in Zeile 20 wird folglich der Computer veranlaßt, die Zahl 4 auszugeben. Zeile 30 besagt wiederum, nach Zeile 10 zurückzugehen, den Wert von N auf 3 zu erhöhen usw. Die Zeilen 10, 20 und 30 werden insgesamt 10-mal wiederholt! Nach Ausführung der Statements auf den Zeilen 10 bis 30 mit N = 10 wird die Schleife verlassen, und die Zeile 40 kommt zum Zuge.

Wir wollen nunmehr das soeben besprochene Programm und anschließend den Befehl RUN eingeben. Es kommt zur folgenden Ausgabe auf dem Bildschirm:

```
1
4
9
16
25
36
49
64
81
100
Ok
```

Die Variable N wird als *Laufvariable* der Schleife bezeichnet. Sie kann innerhalb der Schleife genau so benutzt werden wie jede andere Variable. Sie kann also in arithmetische Berechnungen eingehen, aber auch in Ausgabestatements aufgeführt werden.

Testübungen 3.1.1

a) Es ist eine Schleife zu ersinnen, die es ermöglicht, daß die Variable N die Werte von 3 bis einschließlich 77 annimmt.

b) Es ist ein Programm zu schreiben, das N∧2 für N = 3 bis N = 77 errechnet!

3.1.1 Übersichtliche Gestaltung von Schleifen

Man beachte, daß wir den Textteil der Zeile 20 unseres Beispielprogrammes (siehe Abb. 3.1) eingerückt haben. Dadurch haben wir die Lesbarkeit der Schleife verbessert: Man kann deut-

lich Anfang und Ende der Schleife auf den ersten Blick erkennen. Es ist eine begrüßenswerte Programmierpraxis, wenn man stets die Statements innerhalb der Schleifen auf diese Weise einrückt. Die Übersichtlichkeit der Programme wird dadurch immens gesteigert. Zum Einrükken kann die Tabulatortaste, d.h. die Taste mit den beiden Symbolen |← und →| benutzt werden. In BASIC sind nämlich standardmäßig Tabulatoren nach jeweils 8 Stellen gesetzt. Diese Taste erfüllt ihre Funktion genau so wie die entsprechenden Einrichtungen bei den Schreibmaschinen. Immer, wenn die Tabulatortaste angeschlagen wird, bewegt sich der Cursor bis zur nächsten Tabulatorstellung.

Wir wollen nunmehr unser erstes Beispielprogramm für Schleifen etwas modifizieren. Jede Ausgabezeile soll nicht nur den Wert von N∧2 , sondern auch den Wert von N enthalten. Um außerdem die entstehende Tabelle übersichtlicher zu gestalten, wollen wir ferner noch zwei Spaltenüberschriften hinzufügen. Das neue Programm sieht unter diesen Umständen so aus, wie es die Abb.3.2 zeigt.

```
10 PRINT "N","N∧2"
20 FOR  N=1 TO 10
30  PRINT N,N∧2
40 NEXT N
50 END
```

Abb.3.2 Programm mit Schleife

Als Programmausgabe erhalten wir jetzt:

N	N∧2
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100
Ok	

Testübung 3.1.2

Welches Resultat zeitigt die Änderung der Zeilennummer 10 in die Zeilennummer 25 bei unserem letzten Beispielprogramm?

Wir wollen nun einige Beispiele für Schleifen besprechen, einige von vielen Anwendungen wohlgemerkt.

Beispiel 1:

Es ist ein BASIC-Programm zu schreiben, das die Summe der natürlichen Zahlen 1, 2, 3, ···, 100 berechnet!

Um die Lösung übersichtlich zu gestalten, wählen wir S als Namen für die Variable, die die Summe aufnimmt. Zu Beginn des Programmes setzen wir S auf 0 (Zuweisung eines Anfangswertes). In der Schleife lassen wir sukzessiv auf den jeweiligen Wert von S die Zahlen 1, 2, 3, ···, 100 addieren. Bei Zugrundelegung dieser Gedankengänge ergibt sich das Programm, das in Abb. 3.3 dargestellt ist.

```
10 LET  S = 0
20 FOR  N=1 TO 100        }
30  LET  S = S + N        }  Diese Anweisungen
40 NEXT N                 }  werden 100mal angesprochen
50 PRINT  S
60 END
```

Abb. 3.3 Programm zur Summenbildung

Beim ersten Eintreten in die Schleife besitzt S den Wert 0 und N den Wert 1. Durch das Statement auf Zeile 30 wird S durch S + N ersetzt, d.h. durch 0 + 1. Das Statement auf Zeile 40 führt die Programmablaufsteuerung zurück auf Zeile 20, wodurch der Wert von N nunmehr 2 beträgt. In Zeile 30 wird wiederum S durch S + N, jetzt also 0 + 1 durch 0 + 1 + 2, wodurch S der Wert 3 zugewiesen wird. Das Statement auf Zeile 40 bewirkt erneut das Zurückgehen auf Zeile 20; N erhält dadurch den Wert 3. In Zeile 30 erfolgt wiederum das Ersetzen von S, das bekanntlich infolge der vorhergehenden Schleifendurchläufe den Wert 0 + 1 + 2 = 3 aufweist, durch S + N. S besitzt somit jetzt den Wert 0 + 1 + 2 + 3 = 4. Dieser Prozeß wiederholt sich ständig. Der letzte, d.h. der 100. Schleifendurchlauf ersetzt den bisherigen Wert von S durch 0 + 1 + 2 + 3 + ··· + 100, wodurch die beabsichtigte Summe gebildet ist. Lassen wir nun durch Eingabe des Befehles RUN das Programm ablaufen, so kommt es auf dem Bildschirm zur Ausgabe

```
5050
Ok
```

Testübung 3.1.3

Es ist ein BASIC-Programm zur Berechnung der Summe

$$101 + 102 + 103 + \cdots + 110$$

zu schreiben.

Testübung 3.1.4

Es ist ein BASIC-Programm zu schreiben, mit dem die Potenzen von 2 errechnet und angezeigt werden können, d.h. es sollen auf dem Bildschirm die Werte von

$$2,\ 2\wedge 2,\ 2\wedge 3,\ \cdots,\ 2\wedge 20$$

erscheinen.

Beispiel 2

Es ist ein Programm zur Berechnung der Summe

$$1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \cdots + 49 \cdot 50$$

zu schreiben.

Zur Lösung ist folgendes zu bemerken: Die Summe soll in der Variablen S gebildet werden, so wie es wir bereits beim vorhergehenden Beispiel vorgesehen hatten. Die zu addierenden Summanden sind die Produkte

N*(N+1) für N=1,2,3,···,49

Unser Programm nimmt unter den genannten Voraussetzungen die Gestalt an, die in Abb. 3.4 dargestellt ist.

```
10 LET  S = 0
20 FOR N=1 TO 49
30   LET  S = S + N*(N+1)
40 NEXT N
50 PRINT S
60 END
```

Abb. 3.4 Programm zur Berechnung der Summe von Produkten

3.1.2 Fehlerursachen bei Schleifen

Zwei häufig bei der Codierung von Schleifen gemachte Fehler sollen jetzt besprochen werden.

1. Jedes FOR-Statement muß in ein korrespondierendes NEXT-Statement einmünden. Ist das nicht der Fall, stoppt BASIC das Programm und zeigt die folgende Fehlermeldung an:

FOR without NEXT in xxxx (FOR ohne NEXT in xxxx	Unter xxxx ist die Nummer der Zeile zu verstehen, die das FOR-Statement enthält

   In gleicher Weise muß jedem NEXT-Statement ein korrespondierendes FOR-Statement vorausgehen, anderenfalls kommt es zum Anhalten des Programmes und der damit verbundenen Fehlernachricht:

NEXT without FOR in xxxx (NEXT ohne FOR in xxxx)	Unter xxxx ist die Nummer der Zeile zu verstehen, die das NEXT-Statement enthält

2. Man stelle sicher, daß die Laufvariable nicht bereits mit anderer Bedeutung verwendet wurde. Nehmen wir z. B. einmal an, daß die Laufvariable N bereits vor Schleifenbeginn benutzt wurde. Das hätte zur Folge, daß die Schleifensteuerung bei Eintritt in die Schleife den alten Wert von N zerstört. Es besteht keine Möglichkeit, ihn nach Schleifenende wieder zu bekommen.

3.1.3 Ineinandergeschachtelte Schleifen (Schleifennester)

Bei vielen Anwendungen erweist es sich als notwendig, innerhalb einer Schleife eine weitere Schleife ausführen zu lassen. Nehmen wir z.B. einmal an, daß wir die folgende Zahlenanordnung berechnen lassen wollen:

```
1∧2, 2∧2, 3∧2, ··· , 10∧2
101∧2, 102∧2, 103∧2, ··· ,110∧2
⋮
2001∧2, 2002∧2, 2003∧2, ··· , 2010∧2
```

Solche Zahlenanordnungen bezeichnet man als *Matrix*. Die hier vorgelegte Matrix besteht aus 21 *Zeilen* mit je 10 *Spalten*. Die Berechnung jeder Zeile erfordert die Codierung einer Schleife. Die erste Zeile zu berechnen, heißt die nachstehende Statementfolge niederzuschreiben:

```
100 FOR  I=1 TO 10
110   PRINT I∧2
120 NEXT I
```

In gleicher Weise erfordert die Berechnung der in der zweiten Zeile auftretenden Elemente die Codierung:

```
100 FOR  I=1 TO 10
110   PRINT (100+I)∧2
120 NEXT I
```

Unsere Überlegungen, in gleicher Weise fortgesetzt, führen uns schließlich zu dem Code, mit dem die Elemente der 21. und damit der letzten Zeile berechnet werden können.

```
100 FOR  I=1 TO 10
110   PRINT (2000+I)∧2
120 NEXT I
```

Wenn wir unsere bisher angestellte Überlegungen kritisch betrachten, so können wir schlußfolgern, daß, im Grunde genommen, die zu ermittelnden Werte durch eine 21-malige Wiederholung dieser Anweisungsfolge bestimmt werden können.

Die jeweils auf I zu addierenden Zahlen beginnen bei 0 (ist als 0*100 anzusehen) für die erste Zeile, gehen über zu 100 (ist als 1*100 anzusehen) für die zweite Zeile und enden schließlich bei 2000 (ist als 20*100 anzusehen) für die letzte Zeile. Uns wird durch diese Betrachtungen der Gedanke nahegelegt, die zu addierenden Zahlen zu J*100 darzustellen, wobei J eine Laufvariable ist, die die Werte von 0 bis 20 annimmt. Unter dieser Voraussetzung können wir jetzt die Ausgabe der Zahlenmatrix durch das in Abb.3.5 dargestellte Programm abdecken.

```
10  FOR  J=0 TO  20
100   FOR I=1  TO  10
110     PRINT (100*J+I)∧2
120   NEXT I
200 NEXT J
```

Abb.3.5 Programm mit Schleifennest

Die Anweisungen, die gegenüber der ersten Anweisung um zwei Stellen eingerückt sind, werden 21-mal ausgeführt, entsprechend der Werte von J=0 bis J=20. Beim ersten Schleifendurchlauf (J=0) werden durch die Statements auf den Zeilen 100 bis 120 die Werte der ersten Matrixzeile ausgegeben, beim zweiten Schleifendurchlauf (J=1) die Werte der zweiten Zeile usw. Wir glauben, daß jeder an diesem Beispiel die Überzeugung gewonnen hat, wie die Einrückungen bei Schleifen erheblich zur Lesbarkeit eines Programmes beitragen können und damit geradezu musterhaft einen guten Programmierstil prägen helfen.

Wenn eine Schleife in einer Schleife enthalten ist, spricht man von einem *Schleifennest.* BASIC gestattet, die Nestbildung in der Tiefe so weit zu treiben, wie man sie aufgrund der Problemstellung treiben muß, d.h. man kann eine Schleife in einer Schleife in einer Schleife usw. bilden.

Ineinandergeschachtelte Schleifen dürfen sich nicht überlappen. Demzufolge ist die Statementfolge in Abb.3.6 nicht erlaubt:

```
10 FOR  J=1 TO  100
20   FOR K=1  TO  50
:
80 NEXT J
90   NEXT K
```

Abb.3.6 Nicht gestattete Statementfolge

Das Statement

```
NEXT K
```

muß auf jeden Fall dem Statement

```
NEXT J
```

vorausgehen, d.h. die innere, durch K gesteuerte Schleife muß komplett innerhalb der äußeren, durch J gesteuerten Schleife liegen.

Testübung 3.1.5

Es ist ein BASIC-Programm zu schreiben, das die folgende Tabelle von Zahlen ausgibt:

```
1    11   21   31
2    12   22   32
:    :    :    :
9    19   29   39
```

3.1.4 Anwendungen mit Schleifen

Beispiel 3

Um ein Auto zu kaufen, leiht sich ein Angestellter 7000,– DM. Der Kredit hat eine Laufzeit von 36 Monaten bei einem Zinssatz von 1% pro Monat auf die jeweilige Restschuld. Die monatliche Ratenzahlung ist auf 232,50 DM festgelegt. Es ist ein Programm zu schreiben, das am Ende jeden Monats die folgenden Werte errechnet:

- Zinsen
- Tilgungsrate
- Kontostand

Zur Lösung führen wir die folgenden Variablen ein:

K	Kontostand in DM
Z	(monatliche) Zinsen in DM
T	(monatliche) Tilgungsrate in DM
R	(monatliche) Ratenzahlung in DM

Zu Programmbeginn haben wir K den Wert 7000 als Anfangswert zuzuweisen. Am Ende eines jeden Monats sind die angefallenen Zinsen Z zu berechnen, nämlich 0.01*K; diese betragen z.B. am Ende des ersten Monats 0.01*7000 = 70.00 DM. Nach Vereinbarung sind monatlich R = 232.50 DM zu zahlen. Zur Tilgung bleiben also T = R – Z; nach dem ersten Monat sind das T = 232.50 – 70.00, d.h. T = 162.50 DM. Daraus wiederum resultiert der neue Kontostand K = 7000 – 162.5, d.h. K = 6837.50 DM. Ein Programm, das die der Aufgabenstellung gerecht werdenden Berechnungen durchführt, ist in der Abb. 3.7 aufgelistet.

```
10 PRINT "MONAT","ZINSEN","TILGUNG","KONTOSTAND"
20 LET  K = 7000          : 'K ---> KONTOSTAND ZU BEGINN
25 LET  R = 232.50        : 'R ---> MONATLICHE ZAHLUNG
30 FOR  M = 1  TO  36     : 'M ---> ANZAHL DER ABGELAUFENEN MONATE
40   LET  Z = .01*K       : '  BERECHNUNG DER MONATLICHEN ZINSEN
50   LET  T = R - Z       : '  BERECHNUNG DER MONATLICHEN TILGUNGSRATE
60   LET  K = K - T       : '  BERECHNUNG DES NEUEN KONTOSTANDES
70   PRINT M,Z,T,K        : '  AUSGABE DER MONATLICHEN DATEN
80 NEXT M                 : 'UEBERGANG ZUM NAECHSTEN MONAT
90 END
```

Abb. 3.7 Programm zur Berechnung eines Tilgungsplanes

Wir sollten nunmehr versuchen, das Programm ablaufen zu lassen. Dabei werden wir feststellen, daß es läuft, . . . aber wir können mit den Resultaten kaum etwas anfangen, da auf dem Bildschirm nicht Platz genug für die gesamte Ausgabe ist. Die meisten Ausgabezeilen rauschen vorüber, bevor sie gelesen werden können. Ein Weg, um diese Situation zu meistern, besteht darin, gleichzeitig die beiden Tasten *Ctrl* und *Num Lock* zu drücken, während auf dem Bildschirm die Ausgabezeilen erscheinen. Dadurch pausiert die Ausführung des Programmes, folglich wird der Bildschirminhalt zum Zeitpunkt des Drückens der Tasten gewissermaßen eingefroren. Soll der Computer die Programmausführung wieder aufnehmen und mit der Ausgabe fortfahren, ist irgendeine Taste anzuschlagen. Die Ausgabezeilen beginnen dann wieder, sich über den Bildschirm zu bewegen. Die Benutzung dieser Technik erfordert ein gewisses Fingerspitzengefühl. Darüberhinaus kann nicht im voraus bestimmt, geschweige denn garantiert werden, wo die Bildbewegung gestoppt wird.

Testübung 3.1.6

Man lasse das Programm des Beispieles 3 ablaufen und praktiziere versuchsweise das „Einfrieren" des Bildschirminhaltes. Mehrere Abläufe können erforderlich sein, bevor man in der Lage ist, diese Technik einigermaßen zu beherrschen.

Eine andere Methode, die Ausgabe des Programmes vom Beispiel 3 in den Griff zu bekommen, besteht darin, die PRINT-Statements durch LPRINT-Statements zu ersetzen. Die Ausgabe wird dann freilich zum Drucker geleitet. Die Ausgabeliste muß man hier eben später lesen.

Zum Schluß wollen wir uns noch mit einer anderen, eleganteren Möglichkeit befassen. Sie fußt darauf, daß die Menge der jeweiligen Ausgabezeilen dem Bildschirmumfang angepaßt wird. Bei unserem Beispiel 3 begrenzen wir die Anzeige auf die Daten von jeweils 12 aufeinanderfolgenden Monaten; wir müssen also für drei aufeinanderfolgende Bilder sorgen. Zu diesem Zweck führen wir eine zweite, die vorhandene Schleife umschließende Schleife ein, durch die die Daten von jeweils 12 Monaten gehandhabt werden. Die Laufvariable der neuen äußeren Schleife soll mit J (J steht für Jahre) benannt werden; sie variiert von 0 bis 2. Die andere Laufvariable, d. h. die Laufvariable der alten, nunmehr inneren Schleife soll nach wie vor M (M steht für Monate) heißen, doch nunmehr variiert M nur noch von 1 bis 12. Die Anzahl der abgelaufenen Monate bestimmt sich jetzt nach der Formel

```
12*J+M,
```

d. h. sie ist gleich der Summe aus dem 12-fachen von J und der Zahl der abgelaufenen Monate innerhalb des betreffenden Jahres. Das unter diesen Überlegungen entwickelte Programm ist in der Abb. 3.8 aufgelistet.

```
 10 LET  K = 7000          : 'K ---> KONTOSTAND ZU BEGINN
 20 LET  R = 232.50        : 'R ---> MONATLICHE ZAHLUNG
 30 FOR  J = 0  TO  2      : 'J ---> ANZAHL DER ABGELAUFENEN JAHRE
 40   PRINT "MONAT","ZINSEN","TILGUNG","KONTOSTAND"
 50   FOR  M = 1  TO  12   : 'M ---> ANZAHL DER ABGELAUFENEN MONATE
                                     INNERHALB EINES JAHRES
 60     LET  Z = .01*K     : '  BERECHNUNG DER MONATLICHEN ZINSEN
 70     LET  T = R - Z     : '  BERECHNUNG DER MONATLICHEN TILGUNGSRATE
 80     LET  K = K - T     : '  BERECHNUNG DES NEUEN KONTOSTANDES
 90     PRINT 12*J+M,Z,T,K: '  AUSGABE DER MONATLICHEN DATEN
100   NEXT M               : 'UEBERGANG ZUM NAECHSTEN MONAT
110   STOP                 : 'STOPPEN DER AUSFUEHRUNG
120   CLS                  : 'LOESCHEN DES BILDSCHIRMES
130 NEXT J                 : 'UEBERGANG ZUM NAECHSTEN JAHR
140 END
```

Abb. 3.8 Neufassung des Programmes zur Berechnung eines Tilgungsplanes

Das in der Abb. 3.8 dargestellte Programm gebraucht mehrere neue Statements. So liegt z. B. in der Zeile 110 das Statement STOP vor; es bewirkt, daß der Computer die Ausführung des Programmes unterbricht und damit stoppt. Der Computer merkt sich natürlich, wo er gestoppt hat. Er bewahrt auch die Werte aller im Programm verwendeten Variablen auf. Das STOP-

```
Ok

RUN

MONAT          ZINSEN          TILGUNG         KONTOSTAND
 1              70              162.5           6837.5
 2              68.375          164.125         6673.375
 3              66.73375        165.7662        6507.609
 4              65.07608        167.4239        6340.184
 5              63.40184        169.0981        6171.086
 6              61.71086        170.7891        6000.297
 7              60.00297        172.497         5827.8
 8              58.278          174.222         5653.578
 9              56.53578        175.9642        5477.614
 10             54.77614        177.7238        5299.89
 11             52.9989         179.5011        5120.389
 12             51.20389        181.2961        4939.093

Break in 100
Ok
CONT
```

Bild 1

```
MONAT          ZINSEN          TILGUNG         KONTOSTAND
 13             49.39093        183.109         4755.984
 14             47.55984        184.9401        4571.043
 15             45.71043        186.7895        4384.254
 16             43.84254        188.6574        4195.596
 17             41.95596        190.544         4005.052
 18             40.05052        192.4494        3812.603
 19             38.12603        194.3739        3618.229
 20             36.18229        196.3177        3421.911
 21             34.21911        198.2808        3223.630
 22             32.2363         200.2637        3023.367
 23             30.23367        202.2663        2821.1
 24             28.211          204.289         2616.811

Break in 100
Ok
CONT
```

Bild 2

```
MONAT          ZINSEN          TILGUNG         KONTOSTAND
 25             26.16811        206.33189       2410.479
 26             24.10479        208.39521       2202.084
 27             22.02084        210.47916       1991.605
 28             19.91605        212.58395       1779.021
 29             17.79021        214.70979       1564.311
 30             15.64311        216.85689       1347.454
 31             13.47454        219.02546       1128.429
 32             11.28429        221.21571       907.2137
 33             9.07213         223.42787       683.7859
 34             6.83785         225.66215       458.1237
 35             4.58123         227.91877       230.2049
 36             2.30204         230.19796       7.019999E-03
Ok
```

Bild 3

Abb. 3.9 Ausgabe des Programmes zur Berechnung eines Tilgungsplanes (siehe Abb. 3.8)

Statement läßt den Bildschirminhalt ebenfalls ungeändert. Folglich kann man die auf dem Schirm angezeigten Ergebniszeilen so lange betrachten, wie man es wünscht. Wenn die Programmausführung vom Computer wieder aufgenommen werden soll, tippt man einfach den Befehl CONT (von Continue→Fortfahren) ein. Der Computer fährt dann von der Stelle aus fort, an der er angehalten wurde. Die erste Anweisung, auf die er jetzt stößt, ist die Anweisung CLS auf Zeile 120. Diese Anweisung hat das Löschen des Bildschirmes zur Folge, somit verschwindet der bisherige Bildschirminhalt. Danach erfolgt der Übergang zum nächsten Wert von J und damit zu den 12 Monaten des darauffolgenden Jahres. In der Abb. 3.9 ist die Ausgabe des soeben besprochenen Programmes aufgelistet. Der Übersichtlichkeit wegen sind in der Auflistung die vom Benutzer eingegebenen Befehle unterstrichen.

Die Abb. 3.9 bedarf noch einer Kommentierung. Die Ausgabedaten erscheinen mit bis zu sieben Ziffern, obgleich den Rechengängen DM und Pfg. zugrunde liegen. Dem Rundungsproblem bei Zahlen werden wir erst später die gebührende Aufmerksamkeit schenken. Es sei ferner noch auf den Kontostand am Ende des 36. Monats hingewiesen; er erscheint nämlich in der technisch-wissenschaftlichen Schreibweise. Die Zeichenfolge E-03 besagt, daß der Dezimalpunkt drei Stellen nach links verschoben werden muß. Die aufgelistete Zahl ist also bei Verwendung der normalen Zahlenschreibweise gleich 0,007019999 DM oder gleich rd. 0,7 Pfg, d. h. weniger als 1 Pfg! Die Darstellung dieser Zahl in der normalen Schreibweise erfordert, wie wir gesehen haben, mehr als sieben Ziffernstellen. Deshalt ist der Computer zur technisch-wissenschaftlichen Darstellungsform übergegangen, denn die Wahl der Darstellungsform obliegt dem Computer.

3.1.5 Der Gebrauch von Schleifen zur Erzeugung von Verzögerungen

Durch Codierung einer geeigneten Schleife können wir eine *Verzögerung* innerhalb des Computers hervorrufen. Man betrachte hierzu die nachstehende Anweisungsfolge:

```
10 FOR  N=1 TO 3000
20 NEXT N
```

Diese Schleife führt quasi *nichts* aus! Jedoch wiederholt der Computer die Anweisungen auf den Zeilen 10 und 20 dreitausendmal! Das scheint eine beträchtliche Arbeitslast zu sein, jedoch nicht für einen Computer. Um ein Gefühl für die Geschwindigkeit zu bekommen, mit der ein Computer arbeitet, sollte man einmal die Zeit für die Ausführung dieser Anweisungsfolge bestimmen. – Eine solche Schleife kann zur Erzielung einer Verzögerung eingesetzt werden. Wenn man z. B. wünscht, daß einige Daten auf dem Bildschirm stehen bleiben, ohne daß der Computer den Programmablauf stoppen muß, dann baue man einfach in das betreffende Pro-

```
10 PRINT "GRAPHISCHES PROGRAMM ZUM ANZEIGEN VON UMSAETZEN"
20 PRINT "DES LAUFENDEN JAHRES BIS ZUM HEUTIGEN DATUM"
30 FOR N=1 TO 5000
40 NEXT N:                    'VERZOEGERUNGSSCHLEIFE
50 CLS
60 PRINT "ES SIND DIE FOLGENDEN PARAMETER BEREITZUSTELLEN:"
70 PRINT "ARTIKEL, VERKAUFSBEZIRK, VERKAEUFER"
80 END
```

Abb. 3.10 Programm mit Verzögerungsschleife

gramm eine geeignete *Verzögerungsschleife* ein. Zur Demonstration unserer Gedankengänge wollen wir jetzt ein Programm vorstellen, das Text für zwei Bildschirmbilder ausgibt. Die eingebaute Verzögerungsschleife soll dem Benutzer Zeit geben, den auf dem ersten Bild ausgegebenen Text zu lesen. Das Programm ist in der Abb. 3.10 aufgelistet.

Beispiel 4:

Durch eine Verzögerungsschleife soll für ein Sicherheitssystem eine blinkende Anzeige auf dem Bildschirm erzeugt werden.

Zur Lösung nehmen wir an, daß das Sicherheitssystem direkt mit dem Computer verbunden ist und daß das System einen Eindringling in der Zone 2 des Warenhauses aufgespürt hat. In diesem Fall soll die Nachricht

SICHERHEITSSYSTEM ENTDECKT EINDRINGLING IN ZONE 2

ausgegeben werden.

Um die Aufmerksamkeit des Bedieners auf sich zu lenken, soll diese Nachricht blinken, d. h. abwechselnd auf dem Bildschirm erscheinen und nicht erscheinen. Gehandhabt werden kann das dadurch, daß man abwechselnd die Nachricht auf dem Schirm ausgeben läßt und dann wieder den Schirm löscht (siehe Programm in Abb. 3.11).

```
10 PRINT "GRAPHISCHES PROGRAMM ZUM ANZEIGEN VON UMSAETZEN"
20 PRINT "DES LAUFENDEN JAHRES BIS ZUM HEUTIGEN DATUM"
30 FOR N=1 TO 5000
40 NEXT N:                    'VERZOEGERUNGSSCHLEIFE
50 CLS
60 PRINT "ES SIND DIE FOLGENDEN PARAMETER BEREITZUSTELLEN:"
70 PRINT "ARTIKEL, VERKAUFSBEZIRK, VERKAEUFER"
80 END
```

Abb. 3.11 Blinkende Anzeige für ein Sicherheitssystem

Die Schleife, die durch die Anweisungen auf den Zeilen 30 und 40 gebildet wird, dient dazu, die Nachricht auf dem Schirm für einen Augenblick festzuhalten (Verzögerungsschleife). Das Statement auf Zeile 50 bringt die Nachricht zum Verlöschen, das Statement auf Zeile 20 erzeugt sie erneut. Das Blinken der Nachricht (siehe Statement auf Zeile 10) findet insgesamt 2000-mal statt.

Testübung 3.1.7

Es ist ein Programm zu schreiben, durch das der eigene Name auf dem Bildschirm angezeigt wird, und zwar 500-mal blinkend. Das Anzeigen des Namens soll durch eine Verzögerungsschleife gesteuert werden, die jeweils 50-mal durchlaufen werden soll.

3.1.6 Ergänzungen zur Codierung von Schleifen

Unsere bisher vorgestellten Beispiele über Schleifen hatten eins gemeinsam: Die Laufvariablen der Schleifen wurden bei jedem Schleifendurchlauf um 1 erhöht, besaßen also stets die

Schrittweite 1. Es ist jedoch möglich, in einer Schleife die Laufvariable bei jedem Durchlauf um einen beliebigen Wert zu erhöhen. Man betrachte dazu die folgenden Anweisungen:

```
10   FOR  N=1 TO 500 STEP 2
⋮
1000 NEXT N
```

Die so definierte Schleife mit N als Laufvariable zeichnet sich dadurch aus, daß die Laufvariable bei jedem Durchlauf um 2 erhöht wird und damit der Reihe nach die folgenden Werte annimmt:

1, 3, 5, 7, ···, 499

Ähnlich wirkt sich der Gebrauch der Klausel

```
STEP 0.5
```

auf die Laufvariable der obigen Schleife aus; N nimmt freilich jetzt der Reihe nach die Werte

1, 1.5, 2, 2.5, ···, 499.5, 500

an. Die Schrittweite der Laufvariablen ist also gleich 0.5. Man kann die Schrittweite sogar negativ angeben. In einem solchen Fall fallen die aufeinanderfolgenden Werte der Laufvariablen. Man verfolge hierzu die Anweisungsfolge:

```
10  FOR N=100 TO 1 STEP -1
⋮
100 NEXT N
```

Hier durchläuft die Laufvariable N die Werte von 100 bis 1 mit einer Schrittweite von -1, beginnend bei 100 und endend mit 1; es erfolgt also ein Herabzählen. In der Aufgabengruppe 6 sind Aufgaben enthalten, die das Herabzählen von Laufvariablen beinhalten.

Testübung 3.1.8

Es sind Anweisungen zu formulieren, die bewirken, daß N der Reihe nach die nachstehend aufgeführten Werte annimmt.

a) 95, 96.7, 98.4, ··· , 112

b) 200, 199.5, 199, ··· , 100

Aufgabengruppe 6

Für die Berechnung der nachfolgenden Summen sind BASIC-Programme zu schreiben!

1. $1^2 + 2^2 + 3^2 + \cdots + 25^2$

2. $1 + \left(\frac{1}{2}\right)^1 + \left(\frac{1}{2}\right)^2 + \cdots + \left(\frac{1}{2}\right)^{10}$

3. $1^3 + 2^3 + 3^3 + \cdots + 10^3$

4. $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{100}$

5. Schreibe ein Programm zur Berechnung der Potenzen

$N^2,\ N^3,\ N^4$

für N = 1, 2, 3, · · · , 12
Die Ausgabe soll dabei wie folgt gestaltet werden:

```
N          N∧2        N∧3        N∧4
1          ...
2
3
⋮
12
```

Es sollen also die ersten vier Ausgabezonen benutzt werden. Jede Ausgabezone soll außerdem mit einer Überschrift versehen werden.

6. Angenommen, jemand hat für einen Autokauf ein Darlehen von 4000,– DM aufgenommen. Monatlich ist eine Rate von 125,33 DM zu zahlen. Die monatlichen Zinsen betragen 1% der jeweils noch bestehenden Schuld.
Es ist ein Programm zu schreiben, durch das eine Tabelle mit dem jeweils fälligen Zinsbetrag und dem neuesten Kontostand für die nächsten 12 Monate ausgegeben wird.

7. Angenommen, jemand legt am 1. Januar jeden Jahres bei einer Kundenkreditbank den Betrag von 1000,– DM zu 10% Zinsen an. Die Zinsen werden am 1. Januar jeden Jahres berechnet und gutgeschrieben, basierend auf dem Guthaben des vergangenen Jahres.
Es ist ein Programm zu schreiben, das den Kontostand für die nächsten 15 Jahre berechnet und ausgibt.

8. Ein Börsenmakler schätzt, daß die Firma

GRISCHO Computer KG

in jedem der nächsten drei Jahre ein 20%-iges Wachstum bei den Verkaufserlösen verzeichnen wird. Er vermutet weiterhin, daß der Reingewinn pro Jahr sogar um 30% ansteigen wird. Die Verkaufserlöse des letzten Jahres betrugen 35 Millionen DM, der Reingewinn 5,54 Mill. DM. Durch ein Programm soll ein Ausblick auf die vermuteten Verkaufserlöse und die Reingewinne der nächsten drei Jahre vermittelt werden. Basis hierfür sollen die Voraussagen des Börsenmaklers sein.

Antworten auf die Testübungen

3.1.1 a)

```
10  FOR N=3 TO 77
⋮
100 NEXT N
```

3.1.1 b)

```
10 FOR N=3 TO 77
20     PRINT N∧2
30 NEXT N
40 END
```

3.1.2 Die Überschrift

```
N              N∧2
```

würde vor jeder Tabellenzeile erneut ausgegeben werden.

3.1.3
```
10 LET  S = 0
20 FOR  N=101 TO 110
30   S = S + N
40 NEXT N
50 PRINT S
60 END
```

3.1.4
```
10 FOR  N=1 TO 20
20   PRINT  2∧N
30 NEXT N
40 END
```

3.1.5
```
10 FOR  J=1 TO 9
20   FOR  I=0 TO 3
30     PRINT 10*I+J
40   NEXT I
50 NEXT J
```

3.1.6 Übung macht den Meister!

3.1.7
```
10 FOR  N=1 TO 500
20   PRINT "NAME, VORNAME DES BENUTZERS"      ← individuell ausfüllen
30   FOR  K=1 TO 50
40   NEXT K
50   CLS
60 NEXT N
70 END
```

3.1.8
```
a)      10 FOR  N=95 TO 112  STEP 1.7
b)      20 FOR  N=200 TO 100  STEP -.5
```

3.2 Entscheidungsfindung durch den Computer

Eine der wertvollsten Eigenschaften, die die Computer zu nützlichen Werkzeugen für Problemlösungen macht, ist ihre Fähigkeit zu Entscheidungsfindungen. BASIC enthält Anweisungen, die den Anwender die Stellung von Fragen gestattet. Der Computer ermittelt die Anwort und ergreift aufgrund der Antwort eine Maßnahme. Nachfolgend sind einige Fragen zusammengestellt, auf die der Computer eine Antwort finden kann.

- Ist der Wert von A größer als Null?
- Ist die Potenz A∧2 größer als oder gleich 200?
- Beginnt die Zeichenkette, die NAME$ zugewiesen ist, mit dem Buchstaben Z?
- Ist wenigstens einer der Werte, die den Variablen A, B oder C zugewiesen sind, negativ?

Zwei BASIC-Statements gestatten die Stellung solcher und ähnlicher Fragen:

```
IF        THEN ···
IF        THEN ···  ELSE ···
```

Das erste dieser beiden Statements kann in den beiden folgenden Formen eingesetzt werden:

```
IF   <frage>  THEN <statement>
IF   <frage>  THEN <zeilennummer>
```

Wie funktioniert nun dieses Statement? Seine Wirkungsweise ist durch drei Faktoren bestimmt.

1. Der Frageteil des Statements erlaubt die Formulierung von Fragen, die den oben aufgeführten Beispielen entsprechen.
2. Ergibt sich als Antwort auf die gestellte Frage „Ja" (englisch: YES), wird vom Programm der Teil des Statements angesprochen, der in der Klausel THEN enthalten ist.
   a) Folgt dem Wort THEN ein Statement, so wird dieses ausgeführt.
   b) Folgt dem Wort THEN eine Zeilennummer, so fährt das Programm mit der Ausführung des Statements fort, das auf der Zeile mit der angegebenen Zeilennummer steht.
3. Ergibt sich als Antwort auf die gestellte Frage „Nein" (englisch: NO), so geht das Programm zum nächsten Statement über, d.h. zu dem Statement, das auf das IF-Statement unmittelbar folgt.

Zur Untermauerung unserer Aussagen wollen wir ein Beispiel betrachten:

```
500 IF  N=0  THEN PRINT "ENDE DER BERECHNUNG"
```

Der Frageteil dieses Statements lautet „N = 0"; der Teil des Statements, der auf das Wort THEN folgt, ist zu

```
PRINT "ENDE DER BERECHNUNG"
```

formuliert. Stößt die Programmablaufsteuerung auf die Zeile 500, d.h. auf dieses Statement, wird zuerst untersucht, ob der Wert der Variablen N gleich Null ist. Ist das der Fall, so wird der Text

```
ENDE DER BERECHNUNG
```

ausgegeben und anschließend mit dem nächsten Statement fortgefahren, das auf der Zeile steht, die unmittelbar auf die Zeile 500 folgt. Ist der Wert von N jedoch ungleich Null, so geht das Programm sofort zur nächsten Anweisung über, d.h. zur Anweisung auf der Zeile, die unmittelbar auf die Zeile 500 folgt. Das Statement nach dem Wort THEN wird also ignoriert.

Wir wollen uns noch ein zweites Beispiel ansehen.

```
600 IF  A∧2<1  THEN 300
```

Wenn beim Programmablauf dieses Statement erreicht wird, wird der Wert der Potenz A∧2 untersucht. Ist dieser Wert kleiner als 1, geht das Programm zur Zeile mit der Zeilennummer 300

über; man sagt, es *„verzweigt"* zu dieser Zeile. Anderenfalls kommt es nach der Auswertung der Fragestellung zu Ausführung der nächsten Anweisung.

Das Statement:

```
I        IF        THEN ···  ELSE ···
```

ähnelt dem soeben besprochenen Statement

```
I        IF        THEN ···
```

Es ist jedoch flexibler als dieses, wenn die Antwort auf die Frage zu einem „Nein" führt. Es besitzt die Form

```
IF  <frage>  THEN <aktion1>  ELSE <aktion2>
```

Unter aktion1 bzw. aktion2 ist ein Statement oder eine Zeilennummer zu verstehen. Dieses Statement arbeitet wie folgt: Der Computer antwortet zunächst auf die gestellte Frage. Ist die Antwort „Ja", so wird der Teil des Statements ausgeführt, der zur Klausel THEN gehört, also aktion1 (der Teil, der auf das Wort THEN folgt, aber vor ELSE steht). Bei Verneinung der gestellten Frage, wird der zur Klausel ELSE gehörende Teil des Statements ausgeführt, also aktion2.

Ein erstes Beispiel soll das soeben besprochene illustrieren.

```
500 IF  N=0  THEN PRINT "ENDE DER BERECHNUNG"  ELSE 250
```

Bei der Programmausführung wird zuerst bestimmt, ob der Wert von N gleich Null ist. Ist das der Fall, erfolgt die Ausgabe der Zeichenkette

```
ENDE DER BERECHNUNG
```

Ist dagegen der Wert von N ungleich Null, fährt das Programm mit dem Statement fort, das auf der Zeile mit der Zeilennummer 250 steht.

Eine andere Möglichkeit, dieses Statement sinnvoll einzusetzen, besteht darin, beide Klauseln (THEN und ELSE) jeweils mit Anweisungen zu versehen. Man betrachte hierzu folgendes Beispiel:

```
600 IF  A+B>=100  THEN PRINT A+B  ELSE PRINT A
```

Bei der Ausführung dieses Statements wird zunächst untersucht, ob die Summe A + B größer als oder gleich 100 ist. Im bejahenden Falle wird der Wert der Summe ausgegeben, anderenfalls der Wert der Variablen A. In beiden Fällen wird anschließend die Programmausführung mit der nächsten Anweisung fortgesetzt, d.h. mit der Anweisung, die auf der Zeile steht, die unmittelbar auf die Zeile mit der Zeilennummer 600 folgt.

Unsere Betrachtungen wollen wir mit einer Bemerkung beschließen. Nach dem Schlüsselwort IF können die Fragestellungen in Form irgendwelcher *Vergleichsausdrücke* oder *logischen Ausdrücke* formuliert werden. Der Computer überprüft dann diese Ausdrücke auf ihren Wahrheitsgehalt, d.h. ob sie „wahr" oder „falsch" sind. Dafür einige Beispiele:

`N=0`	*(N gleich 0)*
`N>5`	*(N größer als 5)*
`N<12.9`	*(N kleiner als 12.9)*
`N>=0`	*(N größer als oder gleich 0)*
`N<=-1`	*(N kleiner als oder gleich −1)*
`A+B<>C`	*(A + B nicht gleich C)*
`A∧2+B∧2<=C∧2`	*($A^2 + B^2$ kleiner als oder gleich C^2)*
`N=0 OR A>B`	*(Entweder N gleich 0 oder A größer als B oder beides)*
`N>M AND I=0`	*(Sowohl N größer als M als auch I gleich 0)*

Unter aktion 1 bzw. aktion 2 können auch mehrere Statements aufgeführt werden, sofern sie durch das übliche Trennungszeichen, d.h. durch den Doppelpunkt, voneinander getrennt werden.

Testübung 3.2.1

Es sind Anweisungen zu formulieren, die die folgenden Probleme lösen!

a) Wenn A kleiner als B ist, dann ist der Wert der Summe A + B auszugeben; anderenfalls ist das Programm zu beenden.

b) Wenn A2 + D mindestens gleich 5000 ist, dann ist zum Statement auf der Zeile mit der Zeilennummer 300 zu verzweigen; anderenfalls ist zum Statement auf der Zeile mit der Zeilennummer 500 zu verzweigen.

c) Wenn N größer als die Summe von I und K ist, dann ist N diese Summe als Wert zuzuweisen; anderenfalls ist N gleich K zu setzen.

Die beiden Statements

```
     IF      THEN ···
und  IF      THEN ···  ELSE ···
```

können zu Folge haben, daß bei der Programmausführung die normale Reihenfolge der Programmzeilen nicht mehr gilt, basierend auf den Wahrheitsgehalt einer Fragestellung, d.h. einer Bedingung. Bei vielen Anwendungen *müssen* oder *wollen* wir jedoch Anweisungen ausführen lassen, die außerhalb der normalen Reihenfolge liegen, unabhängig von irgendeiner Bedingung. In diesen unabdingbaren Fällen können wir die Anweisung GOTO benutzen (hier liegt kein orthographischer Fehler vor, tatsächlich ist zwischen GO und TO kein Zwischenraum zu lassen!). Diese Anweisung besitzt die Form

```
I        GOTO <zeilennummer>
```

Die Anweisung

```
1000 GOTO 300
```

führt somit den Computer zurück zur Zeile mit der Zeilennummer 300; auf dieser findet er die nächste auszuführende Anweisung *(Rückwärtsverzweigung).*

Das folgende Beispiel illustriert die Verwendung der *bedingten* (IF) und der *unbedingten* (GOTO) *Verzweigungsanweisungen* in einem Programm.

Beispiel 1:

Erklärte Geschäftspolitik einer Bauwarenhandlung ist es, daß der Rechnungsbetrag von auf Kredit gekauften Waren den Betrag von 1000,– DM nicht übersteigen darf, eingeschlossen sind dabei die 10%-igen Bearbeitungskosten und 14% Mehrwertsteuer.

Ein Kunde bestellt 150 Pfosten 100 × 5 zu 1.99 DM/Stck., 30 Sperrholzbretter zu 14,00 DM/Stck., 150 kg Nägel zu 2,28 DM/kg und 2 doppelt verglaste Isolierfenster zu 189,75 DM/Stck. Es ist ein Programm zu schreiben, das den Rechnungsbetrag ermittelt und darüber befindet, ob dieser über dem Kreditlimit liegt oder nicht.

Zur Lösung ist vorab folgendes zu bemerken: Mit M1, M2, M3 und M4 sind die Bestellmengen (Pfosten, Sperrholzbretter, Nägel und Isolierfenster) bezeichnet, mit E1, E2, E3 und E4 die zugehörigen Einzelpreise der betreffenden vier Artikel. Der Warenwert der bestellten Artikel ergibt sich dann zu

```
M1*E1 + M2*E2 + M3*E3 + M4*E4
```

Auf den so errechneten Warenwert werden nun zusätzlich 10% addiert zur Deckung der Bearbeitungskosten. Die so erhaltene Summe ergibt den Gesamtwert der Bestellung. Danach berechnen wir 14% des Gesamtwertes als Mehrwertsteuer und addieren diese auf den bisher ermittelten Betrag.

Der auf diese Weise ermittelte Rechnungsbetrag R wird untersucht, ob er das Kreditlimit von 1000,– DM übersteigt. Ist das der Fall, so sollen die Nachrichten

```
RECHNUNGSBETRAG ÜBERSTEIGT 1000.-DM
VERKAUF AUF KREDIT NICHT GESTATTET
```

ausgegeben werden, anderenfalls die Nachricht

```
VERKAUF AUF KREDIT GESTATTET
```

Das unter diesen Vorüberlegungen geschriebene Programm ist in der Abb. 3.12 dargestellt.

```
10  LET M1=150:  M2=30:  M3=150:  M4=2:       'Zuweisung der Bestellmengen
20  LET E1=1.99: E2=14:  E3=2.28: E4=189.75:  'Zuweisung der Einzelpreise
30  LET W = M1*E1 + M2*E2 + M3*E3 + M4*E4:    'W ---> Warenwert
40  PRINT "WAKENWERT DER BESTELLUNG",W
50  LET B = .1*W:                             'B ---> Bearbeitungskosten
60  PRINT "BEARBEITUNGSKOSTEN",B
70  LET S = .14*(W+B):                        'S ---> Mehrwertsteuer
80  PRINT "MEHRWERTSTEUER",S
90  R = W + B + S:                            'R ---> Rechnungsbetrag
100 PRINT "RECHNUNGSBETRAG",R
110 IF  R > 1000  THEN 200  ELSE 300:         'Rechnungsbetrag > 1000.00 DM ?
200 PRINT "RECHNUNGSBETRAG ÜBERSTEIGT 1000.- DM"
210 PRINT "VERKAUF AUF KREDIT NICHT GESTATTET"
220 GOTO 400
300 PRINT "VERKAUF AUF KREDIT GESTATTET"
400 END
```

Abb. 3.12 Programm zur Untersuchung des Rechnungsbetrages

Man beachte die Entscheidungsfindung durch das Statement auf der Zeile mit der Zeilennummer 110. Wenn der Rechnungsbetrag R den Betrag von 1000,– DM überschreitet, erfolgt eine Verzweigung zum Statement auf der Zeile mit der Zeilennummer 200. Dieses und das nachfolgende Statement auf der Zeile mit der Zeilennummer 210 verweigern einen Verkauf auf Kredit, d.h. sie geben zwei Textzeilen mit den vorgesehenen Nachrichten aus. Das anschließend ausgeführte Statement auf der Zeile mit der Zeilennummer 220 bewirkt eine unbedingte Verzweigung zur Zeile mit der Zeilennummer 400, d.h. zum Programmende. Wenn andererseits der Rechungsbetrag R kleiner als oder gleich 1000,– DM ist, wird der Computer zur Zeile mit der Zeilennummer 300 geführt. Das Statement auf dieser Zeile billigt durch die von ihm ausgegebene Textzeile ausdrücklich einen Verkauf auf Kredit.

Testübungen

3.2.2 Angenommen, daß bei Käufen auf eine bestimmte Kreditkarte auf die offenstehenden Beträge folgende Zinsen berechnet werden:

- Die ersten 500,– DM werden mit 1,5% verzinst.
- Die diesen Sockelbetrag übersteigenden Schulden werden mit 1% verzinst.

a) Es ist ein Programm zu schreiben, daß die Schuldzinsen ermittelt und diese sowie den neuen Kontostand ausgibt.
b) Die Richtigkeit des geschriebenen Programmes ist anhand zweier Schuldbeträge zu überprüfen, und zwar anhand der beiden Beträge 1300,– DM und 275,– DM

3.2.3 Man betrachte die nachstehende Anweisungsfolge:

```
100 IF  A>=5  THEN 200
110 IF  A>=4  THEN 300
120 IF  A>=3  THEN 400
130 IF  A>=2  THEN 500
```

Angenommen, die Variable A besitzt den Wert 3. Es ist die Folge der ausgeführten Anweisungen niederzuschreiben, ausgedrückt durch die entsprechenden Zeilennummern.

Beispiel 2:

Eine Familie will sich zur Erinnerung an ihren Urlaub eine runde Stickdecke aus Seide mitnehmen. Diese Decken werden zu einem einheitlichen Preis von 0,20 DM pro dm^2 angeboten. Das restliche Urlaubsgeld reicht gerade für die Bezahlung von 500 dm^2. Wie groß kann der Radius der Stickdecke maximal gewählt werden, wenn er nur ein Vielfaches von einem dm betragen kann?

Die Formel für den Flächeninhalt eines Kreises beträgt bekanntlich

$$A = \pi \cdot r^2$$

```
10  LET  PI = 3.14159
20  LET  R = 1          :  'R  Radius
30  LET  A = PI*R^2     :  'A  Flaecheninhalt
40  'Bei  A >= 500 ---> Programmende, anderenfalls Ausgabe von R
50  IF  A >= 500  THEN 100  ELSE PRINT R
60  LET  R = R + 1      :  'Uebergang zum naechsten Radius
70  GOTO 30             :  'Rueckverzweigung zwecks Wiederholung
100 END                 :  'Programmende
```

Abb. 3.13 Berechnung der Flächeninhalte von Kreisen

Zur Lösung wollen wir sukzessive die Flächeninhalte von Kreisen berechnen lassen, beginnend beim Radius 1 dm und jeweils um 1 dm ansteigend. Die Berechnungen werden solange fortgesetzt, bis sich ein Flächeninhalt ergibt, der größer als oder gleich 500 dm ist. Das zur Lösung des Problems geschriebene Programm ist in der Abb. 3.13 aufgelistet.

Die Zeile 50 enthält ein IF-Statement, das sowohl eine THEN-Klausel als auch eine ELSE-Klausel aufweist. Wenn der Flächeninhalt A, der in Zeile 30 errechnet wird, größer als oder gleich 500 ist, geht die Programmablaufsteuerung zur Zeile 100 über; durch das dort gefundene END-Statement wird das Programm beendet. Ist jedoch A kleiner als 500, kommt das Statement in der ELSE-Klausel zum Zuge, wodurch der augenblickliche Radius gedruckt wird. Anschließend wird der Radius um 1 erhöht (Zeile 60) und zur Zeile 30 zurückverzweigt (GOTO auf Zeile 70). Die Zeilen 30 bis 70 werden also solange wiederholt, bis der Flächeninhalt A das vorgegebene Limit 500 erreicht oder überschritten hat. Die fünf Zeilen 30 bis 70 bilden somit eine Schleife. Wir haben jedoch zur Steuerung des Schleifenablaufes nicht die Statements FOR und NEXT gebraucht, weil wir von vornherein nicht wissen konnten, wieviele Male wir die Schleife zu durchlaufen haben. Wir mußten den Computer aufgrund der Rechenergebnisse selbst entscheiden lassen, wann die Schleife verlassen werden muß, was durch das von uns codierte IF-Statement geschehen ist.

Im Abschnitt 1 dieses Kapitels diskutierten wir die Codierung einer Schleife. In diesem Abschnitt haben wir über Entscheidungsfindungen gesprochen. Die beiden Statements WHILE und WEND können zur Kombination dieser beiden Erfordernisse herangezogen werden. Dieses Anweisungspaar besitzt die folgende Form:

```
WHILE <ausdruck>
:
WEND
```

Die Statements zwischen WHILE und WEND werden solange wiederholt, bis der bei WHILE codierte Ausdruck „wahr" ist. Man beachte jedoch, daß die genannte Statementfolge mindestens einmal ausgeführt wird. Die Prüfung des Ausdrucks auf seinen Wahrheitsgehalt findet nämlich erst am Ende der durch WHILE und WEND eingeschlossenen Statementfolge statt. Diese Form der Codierung von Schleifen erweist sich immer dann als nützlich, wenn nicht im voraus die Anzahl der durch eine Schleife zu erfassenden Wiederholungen bestimmt werden kann.

```
10  LET  PI = 3.14159
20  LET  R = 1          :  'R  Radius
30  WHILE  A < 500      :  'Schleifenbeginn
40     A = PI*R^2       :  'A  Flaecheninhalt
50     PRINT R          :  'Ausgabe des Radius
60     LET  R = R + 1   :  'Uebergang zum naechsten Radius
70  WEND                :  'Wiederholung (Schleifenende)
100 END                 :  'Programmende
```

Abb. 3.14 Berechnung der Flächeninhalte von Kreisen (2. Fassung)

Zur Demonstration des soeben besprochenen wollen wir das Programm, das wir zur Lösung des Beispieles 2 geschrieben haben, unter Benutzung von WHILE und WEND erneut codieren (Abb. 3.14).

Beispiel 3:

Für den Vorsitz des Gesamtelternbeirats einer Stadt bewerben sich zwei Kandidaten. An den einzelnen Schulen der Stadt brachte die Wahl die folgenden Ergebnisse:

Kandidat	Schule 1	Schule 2	Schule 3	Schule 4
Herr Schmidt	487	229	1540	1211
Herr Wieser	1870	438	110	597

Es soll zunächst die Gesamtzahl der Stimmen ermittelt werden, die jeder Kandidat erreicht hat. Ferner soll der Prozentsatz der Stimmen errechnet werden, die auf die beiden Kandidaten entfallen. Daraus resultierend soll schließlich entschieden werden, wer als Sieger aus der Wahl hervorgegangen ist.

Unter A1, A2, A3 und A4 wollen wir die Anzahl der Stimmen speichern, die Herr Schmidt in den Schulen 1, 2, 3 und 4 erhalten hat, desgleichen unter B1 bis B4 die Stimmen von Herrn Wieser. Die Summe der erhaltenen Stimmen wollen wir den Variablen GA (Herr Schmidt) und GB (Herr Wieser) zuweisen, die errechneten Prozentsätze den Variablen PA bzw. PB. Die Gesamtzahl der abgegebenen Stimmen wird unter G gebildet. Das zur Lösung des Problems geschriebene Programm ist in der Abb.3.15 aufgelistet.

```
10  LET A1=487:   A2=229:   A3=1540:   A4=1211
20  LET B1=1870:  B2=438:   B3=110:    B4=597
30  GA = A1 + A2 + A3 + A4: 'Gesamte Stimmenzahl für H. Schmidt
40  GB = B1 + B2 + B3 + B4: 'Gesamte Stimmenzahl für H. Wieser
50  G = GA + GB           : 'Gesamtzahl der abgegebenen Stimmen
60  PA = 100*GA/G         : 'Prozentsatz der Stimmen für H. Schmidt
70  'GA/G ist der Stimmenanteil für H. Schmidt
80  'Zur Ermittlung des Prozentsatzes ---> Multiplikation mit 100
90  PB = 100*GB/G         : 'Prozentsatz der Stimmen für H. Wieser
100 A$ = "HERR SCHMIDT"
110 B$ = "HERR WIESER"
120 'Zeilen 130 bis 150 beinhalten Ausgabe des Wahlergebnisses
130 PRINT "KANDIDAT","STIMMENZAHL","PROZENTSATZ"
140 PRINT A$,GA,PA
150 PRINT B$,GB,PB
160 'Zeilen 170 bis 400 beinhalten Entscheidung über den Wahlgewinner
170 IF  GA > GB  THEN 300 : 'Gewinner ist H. Schmidt
180 IF  GA < GB  THEN 400 : 'Gewinner ist H. Wieser
190 PRINT A$,"UND",B$,"UNENTSCHIEDENER WAHLAUSGANG"
200 GOTO 900              : 'Übergang zum Programmende
300 PRINT A$,"IST DER GEWINNER DER WAHL"
310 GOTO 900              : 'Übergang zum Programmende
400 PRINT B$,"IST DER GEWINNER DER WAHL"
900 END                   : 'Programmende
```

Abb.3.15 Bestimmung eines Wahlergebnisses

Man betrachte einmal die Logik, mit der der Gewinner der Wahl ermittelt wird. Auf Zeile 170 werden die Gesamtstimmenzahlen miteinander verglichen, die die beiden Kandidaten erreicht haben. Wenn GA (Schmidt) größer als GB (Wieser) ist, dann ist Herr Schmidt Gewinner der Wahl. Es erfolgt der Übergang zum Statement auf Zeile 300 (Ausgabe des Wahlergebnisses); anschließend wird zum Programmende verzweigt. Wenn andererseits GA>GB zum Resultat „falsch" führt, dann hat entweder Herr Wieser die Wahl gewonnen oder die Wahl endete unentschieden. Das nächstfolgende Statement auf Zeile 180 muß also eine erneute Entscheidungsfindung beinhalten. Es wird deshalb jetzt der Vergleichsausdruck

GA<GB untersucht. Ist dieser „wahr", so ist Herr Wieser Gewinner der Wahl; es erfolgt eine Verzweigung zum Statement auf Zeile 400 (Drucken des Ergebnisses) und anschließend wird das Programmende erreicht. Wenn GA<GB „falsch" ist, dann bleibt als einzige Möglichkeit übrig, daß GA gleich GB ist. Ist das der Fall, wird das nächste Statement (Zeile 190) ausgeführt, was zur Ausgabe einer Nachricht führt. Die anschließende unbedingte Verzweigung (Zeile 200) bedeutet ebenfalls das Ansteuern des Programmendes.

3.2.1 Endlose Schleifen und Unterbrechungen

Wie wir soeben gesehen haben, erweist es sich als sehr vorteilhaft, daß man Schleifen, von denen man im voraus nicht weiß, wieviele Male sie durchlaufen werden, auf eine sehr einfache Art und Weise codieren kann. Die Bequemlichkeit birgt aber auch eine Gefahr in sich. Es ist nämlich durchaus möglich, eine Schleife zu kreieren, die *endlos* durchlaufen wird. Dazu wollen wir uns ein Beispiel ansehen (Abb. 3.16).

```
10  LET  N = 1
20  PRINT N
30  LET  N = N + 1
40  GOTO 20
50  END
```

Abb. 3.16 Endlose Schleife in einem Programm

Die Variable N beginnt mit dem Anfangswert 1. Dieser wird ausgegeben. Anschließend wird N um 1 erhöht, d. h. auf 2 gesetzt. Wieder erfolgt die Ausgabe. N wird nun erneut erhöht (ist jetzt gleich 3) und der Wert ausgegeben. Dieses Spielchen wird laufend fortgesetzt. Das Programm läuft somit ewig! Natürlich sollte man beim Schreiben von Programmen solche endlosen Schleifen tunlichst vermeiden. Doch selbst erfahrene Programmierer bringen gelegentlich solche endlosen Schleifen zustande. Ist so etwas einmal geschehen, braucht man dennoch nicht in Panik zu geraten. Es gibt eine Möglichkeit, den Computer zu stoppen. Man muß dazu nur gleichzeitig die beiden Tasten *Ctrl* und *Break* niederdrücken. Zukünftig wollen wir diese Tastenkombination als Unterbrechungstaste bezeichnen. Die Betätigung der Unterbrechungstaste unterbricht die Ausführung des laufenden Programmes und versetzt den Computer in den Befehlsmodus, d. h. man kann nunmehr über die Tastatur einen Befehl eingeben. Es sei aber ausdrücklich darauf hingewiesen, daß das Programm im RAM dabei nicht berührt wird, es wird also nicht zerstört.

Testübung 3.2.4

Man tippe das Programm von Abb. 3.16 ein. Nach der Programmeingabe lasse man es durch Eingabe des Befehles RUN ausführen und stoppe es durch Betätigung der Unterbrechungstaste. Nach dem Anhalten lasse man es erneut ausführen, d. h. man gebe wieder den Befehl RUN ein.

3.2.2 Statement INPUT

Es ist sicher sehr zweckentsprechend, wenn man den Computer veranlassen könnte, Informationen während der Programmausführung anzufordern. Mittel des Statements INPUT kann

man das erreichen. Um zu sehen, wie das geschehen kann, wollen wir zuerst eine simple Anweisung betrachten:

```
570 INPUT A
```

Wenn der Computer beim Programmablauf auf dieses Statement stößt, gibt er ein Fragezeichen, als ?, aus und wartet darauf, daß der Benutzer einen Wert für A eintippt und anschließend die Eingabetaste betätigt. Danach weist der Computer der Variablen A den eingegebenen numerischen Wert zu und fährt mit der Programmausführung fort.

Man kann das Statement INPUT auch zur Eingabe mehrerer Werte benutzen. Mit einem Male können damit mehrere Variablen mit Werten versorgt werden. Die in der Anweisung INPUT aufgeführten Variablen können sowohl numerische Variablen als auch Kettenvariablen sein. Nehmen wir einmal an, daß der Computer dem Statement

```
50 INPUT A,B,C$
```

begegnet. Daraufhin gibt der Computer

```
?
```

aus. Als Konsequenz dieser Ausgabe hat der Benutzer nun die gewünschten Werte für A, B und C$ einzutippen, in derselben Reihenfolge und voneinander durch Kommas getrennt. Angenommen, der Benutzer tippt z.B. ein:

```
10.5, 11.42, ACHSLAGER
```

Anschließend beendet er die Eingabe mit der Betätigung der Eingabetaste. Der Computer führt folgende Zuweisungen durch

```
A=10.5,  B=11.42,  C$="ACHSLAGER",
```

bevor er zum nächsten Statement des gerade ablaufenden Programmes übergeht.
Wenn man als Antwort auf das vom Computer ausgegebene Fragezeichen nur einen Wert, beispielsweise 10.5, eingibt, reagiert der Computer mit einer Rückfrage, nämlich mit

```
? Redo from start
?
```

noch einmal vom Anfang an

Damit zeigt er an, daß er weitere Daten erwartet hat. Wenn man versucht, eine Kettenkonstante anstelle eines angeforderten numerischen Wertes einzugeben, so reagiert der Computer ebenfalls mit

```
? Redo from start
?
```

Er wartet numehr auf die Wiederholung der Eingabeoperation. Um bei der Eingabe während des Programmablaufes nichts falsch zu machen, ist es äußerst sinnvoll, das Programm um die vorausgehende Ausgabe einer Anforderungsnachricht zu ergänzen. Diese Anforderungs-

nachricht sollte die vom Computer erwarteten Werte beschreiben. Um das zu erreichen, muß man die Anforderungsnachricht, umgeben von Anführungszeichen, hinter das Schlüsselwort INPUT stellen; erst nach einem folgenden Semikolon notiert man die Variablen, für die Werte eingegeben werden sollen. Man betrachte z.B. das Statement

```
175 INPUT "EINGABE KUNDENNAME, BETRAG"; A$,B
```

Wenn nun bei der Programmausführung der Computer auf dieses Statement stößt, dann spielt sich der folgende Dialog ab:

```
EINGABE KUNDENNAME, BETRAG? SCHMITT UND SCHOEPS, 2356.89
```

Der unterstrichene Teil zeigt die Antwort auf die Anforderung des Computers an. Bevor die weitere Programmausführung erfolgt, weist der Computer folgende Werte zu:

```
A$ = "SCHMITT UND SCHOEPS",  B = 2356.89
```

Testübung 3.2.5

Es ist ein Programm zu schreiben, durch das den Variablen A und B irgendwelche Werte mittels des Statements INPUT zugewiesen werden können. Durch dieses Programm soll A auf 12 und B auf 17 gesetzt werden.

Die nächsten beiden Beispiele illustrieren die Verwendung der INPUT-Anweisung und vermitteln außerdem weitere praktische Erfahrungen über die bedingte Verzweigungsanweisung IF.

Beispiel 4:

Während eines Lehrganges müssen die Teilnehmer vier Klausuren schreiben. Die Gesamtnote des Lehrganges ergibt sich aus dem Mittelwert aller vier Klausuren.

Die Klausuren werden mit Punkten bewertet. Der durchschnittliche Prozentsatz der erreichten Punkte wird wie folgt in die Zensuren umgesetzt:

90 bis 100	sehr gut
75 bis 89,9	gut
60 bis 74,9	befriedigend
45 bis 59,9	ausreichend
unter 45	ungenügend

Zur Lösung wollen wir für die Eingabe der vier Klausurenergebnisse ein INPUT-Statement vorsehen. Mittels einer Schleife sollen die Durchschnittsprozentzahlen aller Teilnehmer ermittelt werden. Aus diesen soll dann die einzelnen Zensuren gebildet werden. Das unter diesen Voraussetzungen geschriebene Programm ist in der Abb.3.17 aufgelistet.

Wir wollen uns nun einmal die Logik unseres Programmes ansehen. Nach der Eingabe der vier Klausurergebnisse wird der durchschnittliche Prozentsatz errechnet (Zeile 100). Im Statement auf der Zeile mit der Zeilennummer 120 fragen wir, ob dieser größer als oder gleich 90 ist. Wenn das der Fall ist, weisen wir die Zensur „sehr gut" zu und gehen zur nächsten Zeile über; die Zeile 120 benutzen wir, um den Computer wieder zum Anfang der Schleife, nämlich zur Zeile mit der Zeilennummer 10, zu schicken. Wenn aber der Durchschnittswert kleiner als 90 ist, dann führen wir durch die ELSE-Klausel den Computer auf die Zeile

```
10  PRINT "EINGABE DER 4 PROZENTZAHLEN EINES TEILNEHMERS"
20  PRINT "DIE PROZENTZAHLEN SIND DURCH KOMMAS ZU TRENNEN"
30  PRINT "NACH LETZTER PROZENTZAHL EINGABETASTE DRUECKEN"
40  PRINT "BEI EINGABE EINER NEGATIVEN PROZENTZAHL: PROGRAMMENDE"
50  INPUT  P1,P2,P3,P4
60  IF  P1<0  THEN 400
70  IF  P2<0  THEN 400
80  IF  P3<0  THEN 400
90  IF  P4<0  THEN 400
100 LET  D = (P1 + P2 + P3 + P4) / 4
110 PRINT "DURCHSCHNITTLICHE PROZENTZAHL AN ERREICHTEN PUNKTEN",D
120 IF  D>=90  THEN PRINT "GESAMTNOTE: SEHR GUT"  ELSE 140
130 GOTO 10
140 IF  D>=75  THEN PRINT "GESAMTNOTE: GUT"       ELSE 160
150 GOTO 10
160 IF  D>=60  THEN PRINT "GESAMTNOTE: BEFRIEDIGEND"  ELSE 180
170 GOTO 10
180 IF  D>=45  THEN PRINT "GESAMTNOTE: AUSREICHEND"   ELSE 200
190 GOTO 10
200 PRINT "GESAMTNOTE: UNGENUEGEND"
300 GOTO 10
400 END
```

Abb. 3.17 Ermittlung von Zensuren

mit der Zeilennummer 140. In dieser Zeile fragen wir durch das dort stehende Statement, ob die durchschnittliche Prozentzahl größer als oder gleich 75 ist. Wenn das der Fall ist, wird als Gesamtnote „gut" ausgegeben. (Das i-Tüpfelchen ist darin zu sehen, daß der einzige Weg, um zur Zeile mit der Zeilennummer 140 zu gelangen, darin besteht, daß D kleiner als 90 ist. Somit wird bei D größer als oder gleich 75 der Zensurenbereich „gut" angesteuert). Wenn andererseits D kleiner als 75 ist, geht das Programm über zur Zeile mit der Zeilennummer 160 usw. Anfangs scheint die Logik uns geringfügig aus der Fassung zu bringen, aber nach wiederholtem Gebrauch kommt sie uns recht natürlich vor. Das Programmende wird durch die Eingabe eines negativen Prozentsatzes angesteuert.

Beispiel 5:

Wir wollen ein Programm schreiben, mit dessen Hilfe wir unser Girokonto überwachen können. Das Programm soll von einem Anfangskontostand ausgehen und die Eingabe von Einzahlungen (Gutschriften usw.) und Auszahlungen (Überweisungen, Schecks usw.) ermöglichen. Bei Kontoüberziehungen soll eine Warnung ausgegeben werden.

Für die Lösung wollen wir zunächst einige Annahmen formulieren. Unter K wollen wir stets den aktuellen Kontostand verstehen. Das Programm soll so aufgebaut werden, daß nach der Eingabe des anfänglichen

```
10  INPUT "WIE HOCH IST DER ANFANGSKONTOSTAND"; K
20  INPUT "WELCHER TRANSAKTIONSCODE (E, A ODER S)"; T$
30  IF  T$="S"  THEN 1000:       'PROGRAMMENDE'
40  IF  T$="E"  THEN 100  ELSE 200
100 'BUCHUNG DER EINZAHLUNGEN
110 INPUT "BETRAG DER EINZAHLUNG"; B
120 LET  K = K + B :             'ADDITION DES BETRAGES AUF KONTOSTAND
130 PRINT "NEUER KONTOSTAND IST", K
140 GOTO 20
200 'BUCHUNG DER AUSZAHLUNGEN (ÜBERWEISUNGEN UND SCHECKS)
210 INPUT "BETRAG DER AUSZAHLUNG"; B
220 LET  K = K - B :             'SUBTRAKTION DES BETRAGES VOM KONTOSTAND
230 IF  K<0  THEN 300 :          'PRÜFUNG AUF ÜBERZIEHUNG DES KONTOS
240 PRINT "DER NEUE KONTOSTAND BETRÄGT", K
250 GOTO 20
300 'BEARBEITUNG EINER ÜBERZIEHUNG
310 PRINT "DIE AUSZAHLUNG (SCHECK USW.) FÜHRT ZUR ÜBERZIEHUNG
320 INPUT "SOLL DIE AUSZAHLUNG DURCHGEFÜHRT WERDEN? (J ODER N)"; A$
330 IF  A$="N"  THEN 400
340 PRINT "DER NEUE KONTOSTAND BETRÄGT", K
350 GOTO 20
400 'UNTERDRÜCKUNG DER AUSZAHLUNG
410 LET  K = K + B :             'STREICHUNG DER AUSZAHLUNG
420 GOTO 20
1000   END
```

Abb. 3.18 Programm zur persönlichen Wartung eines Kontos

Kontostandes zuerst nach einem Transaktionscode gefragt wird. Die Eingabe von „E" bedeutet, daß man eine Einzahlung verbuchen will, von „A", daß irgendeine Auszahlung (Überweisung oder Scheck) getätigt werden soll. Der Transaktionscode „S" bedeutet schließlich, daß mit den Buchungen Schluß gemacht werden soll, d.h. das Programm soll beendet werden. Nach der Eingabe des Betrages, der der entsprechenden Transaktion zugrundegelegt werden soll, ist der neue Kontostand zu bilden und auszugeben. Im Falle einer Überziehung soll diese Tatsache dem Benutzer mitgeteilt werden. Außerdem soll dem Benutzer gestattet sein, die Auszahlung (Scheck oder Überweisung) wieder hinfällig zu machen. In der Abb. 3.18 ist ein Programm dargestellt, das diesen Anforderungen gerecht wird.

Man sollte dieses Programm sorgfältig studieren, um sicher zu gehen, daß man jede bedingte Verzweigungsanweisung (IF) verstanden hat und den Programmablauf verfolgen kann. Außerdem befasse man sich mit einer eingehenden Analyse der INPUT-Statements. Zusätzlich sollte man ein Gefühl dafür entwickeln, wie der Dialog zwischen Mensch und Maschine aufgebaut werden sollte, wenn von INPUT-Anweisungen Gebrauch gemacht wird.

Man beachte ebenfalls, daß das Programm von Abb. 3.18 in *Abschnitte* aufgegliedert worden ist. Jeder Abschnitt beginnt mit einer Zeilennummer, die ein Vielfaches von 100 darstellt. Dar-

überhinaus beginnt jeder Abschnitt mit einem Kommentar, durch den die Funktion dieses Abschnittes kurz umrissen wird. Um ein umfangreiches Programm zu schreiben, ist es ratsam, dieses in handhabbare Abschnitte zu zerlegen. Man verlaufe sich nicht im Irrgarten der Komplexität! Man arbeite einen Abschnitt nach dem anderen sorgfältig aus und kommentiere ihn ausführlich. Dann erst stelle man die einzelnen Abschnitte zum Programm zusammen.

Beispiel 6:

Es ist ein Programm zu schreiben, mit dem die Fertigkeit beim Addieren zweier Zahlen überprüft werden kann. Der Benutzer dieses BASIC-Programmes (siehe Abb.3.19) sollte sich die Aufgaben selbst stellen können. Am Schluß des Programmes soll eine Nachricht ausgegeben werden, aus der ersichtlich ist, wieviel Aufgaben von insgesamt 10 Aufgaben der Benutzer richtig gelöst hat (erreichte Punktzahl).

Bei der Entwicklung der Lösung wollen wir davon ausgehen, daß der Benutzer die Summandenpaare über das INPUT-Statement eingeben kann. Die Summe wird ebenfalls durch eine solche Anweisung angefordert. Eine bedingte Verzweigungsanweisung (IF) soll dazu dienen, die Richtigkeit des vom Benutzer eingegebenen Ergebnisses zu prüfen. Die Variable R soll die Anzahl der richtigen Antworten festhalten. Da die Wiederholung insgesamt 10 Additionsaufgaben zu lösen sind, soll eine Schleife benutzt werden.

```
10  FOR  N=1 TO 10 :        'Schleife zur Stellung von 10 Aufgaben
20    INPUT "TIPPE ZWEI 2-STELLIGE ZAHLEN EIN"; A,B
30    INPUT "WIE GROSS IST IHRE SUMME"; C
40    IF  A + B = C  THEN 200
100   'Behandlung einer falschen Antwort
110   PRINT  "FALSCH ---> DIE RICHTIGE ANTWORT LAUTET", A+B
120   GOTO 300 :              'Übergang zur nächsten Aufgabe
200   'Behandlung einer richtigen Antwort
210   PRINT "DAS EINGEGEBENE ERGEBNIS IST RICHTIG! GRATULATION"
220   LET  R = R + 1 :        'Erhöhung der Zahl richtiger Ergebnisse
230   GOTO 300 :              'Übergang zur nächsten Aufgabe
300 NEXT N :                  'Schleifenende
400 'Ausgabe der erreichten Punktzahl (Anzahl der richtigen Ergebnisse)
410 PRINT  "VON 10 AUFGABEN WURDEN RICHTIG GELÖST: ", R
500 PRINT  "ZUR WIEDERHOLUNG IST DER BEFEHL RUN EINZUGEBEN"
600 END
```

Abb.3.19 Programm zur Stellung von Additionsaufgaben

Aufgabengruppe 7

1. Es ist ein Programm zu schreiben, das die Quadrate natürlicher Zahlen berechnet; die Quadrate sollen jedoch kleiner als 45000 sein.

2. Es ist ein Programm zu schreiben, das die Flächeninhalte aller Kreise berechnet, die kleiner als 5000 m^2 sind. Für die Radien sollen nur ganzzahlige Werte vorgegeben werden.

   Anmerkung: Der Flächeninhalt eines Kreises ergibt sich zu $A = \pi \cdot r^2$. Unter π ist der Näherungswert 3,14159 zu verstehen.

3. Es ist ein Programm zu schreiben, daß die Rauminhalte aller Kuben berechnet, deren Seitenlängen natürliche Zahlen sind. Das Programm soll dann enden, wenn ein errechnetes Volumen größer als oder gleich 175000 m^3 ist.

4. Das Programm von Beispiel 6 soll so abgeändert werden, daß mit seiner Hilfe die Fertigkeit des Multiplizierens überprüft werden kann.

5. Das Programm von Beispiel 6 soll so abgeändert werden, daß am Anfang eine Wahl bezüglich der Rechenart für die nachfolgenden 10 Aufgaben getroffen werden kann. Der Benutzer soll sich entweder für Additionen oder für Subtraktionen oder für Multiplikationen entscheiden können.

6. Es ist ein BASIC-Programm zu schreiben, das drei Zahlen mittels eines INPUT-Statements einliest und anschließend die größte dieser drei Zahlen bestimmt.

7. Es ist ein BASIC-Programm zu schreiben, das drei Zahlen mittels eines INPUT-Statements einliest und anschließend die kleinste dieser drei Zahlen bestimmt.

8. Es ist ein Programm zu schreiben, das mittels eines INPUT-Statements eine Reihe von Zahlen einliest und anschließend die größte derselben bestimmt.

9. Es ist ein Programm zu schreiben, das mittels eines INPUT-Statements eine Reihe von Zahlen einliest und anschließend die kleinste derselben bestimmt.

10. Die folgenden Daten wurden von einem Soziologen gesammelt. In sechs Städten wurden die folgenden Zahlen von Einbrüchen registriert:

Stadt	*Einbrüche 1980*	*Einbrüche 1981*
A	5782	6548
B	4811	6129
C	3865	4270
D	7950	8137
E	4781	4248
F	6598	7048

Für jede Stadt ist die Veränderung der Anzahl der Einbrüche zu berechnen (Anwachsen bzw. Verminderung). Außerdem ist zu ermitteln, welche der Städte ein Wachstum von mehr als 500 Einbrüchen zu verzeichnen hatte.

11. Es ist ein Programm zu schreiben, das die arithmetischen Funktionen einer Registrierkasse ausüben kann. Das Programm soll also die einzelnen Verkaufsbeträge mittels INPUT-Statements erfassen, den Gesamtbetrag errechnen, den Mehrwertsteuerbetrag (z. Z. 14%) hinzufügen und schließlich den Rechnungsbetrag bestimmen. Am Schluß soll das Programm den gezahlten Betrag erfragen und daraus das zurückzugebende Wechselgeld bestimmen.

12. Es ist ein Programm zu schreiben, das die Kassenführung analysiert. Das Programm soll erfragen:
    a) den Kassenbestand,
    b) die im nächsten Monat voraussichtlich eingehenden Rechnungsbeträge,
    c) die im nächsten Monat voraussichtlich zu zahlenden Beiträge.

    Durch Vergleich der Einnahmen und der Ausgaben soll der Kassenbestand des nächsten Monats errechnet werden. Außerdem soll in Form einer Meldung ausgegeben werden, ob ein Überschuß oder ein Defizit zu erwarten ist.

Antworten auf die Testübungen

3.2.1 a)
```
IF A<B THEN PRINT A+B ELSE END
```
b)
```
IF A2+B >= 5000 THEN 300 ELSE 500
```
c)
```
IF N>I+K THEN N=I+K ELSE N=K
```

3.2.2
```
10  INPUT "UNGEZAHLTER BETRAG"; B
20  IF  B > 500  THEN 100  ELSE 200
100 LET  C = B - 500
110 LET  ZI = .015*500 + .01*C
120 GOTO 300
200 LET  ZI = .015*B
300 PRINT "ZINSEN GLEICH",ZI
310 PRINT "NEUER KONTOSTAND GLEICH",B+ZI
400 END
```

3.2.3 100 → 110 → 120 → 400

3.2.4 *Üben macht Freude*

3.2.5
```
10 INPUT "DIE WERTE VON A UND B SIND"; A,B
20 END
```

3.3 Strukturierte Problemlösungen

Der Leser wird bemerkt haben, daß unsere Programme umfangreicher werden. Es führt kein Weg daran vorbei. Um den Computer zur Lösung von Problemen aus dem wirklichen Leben einzusetzen, müssen eben die Programme oft sehr lang sein und das volle Spektrum der Computereinrichtungen und -möglichkeiten benutzen. Damit werden jedoch eine Reihe von Problemen aufgeworfen:

1. Die Planung umfangreicher Programme ist schwierig.
2. Umfangreiche Programme sind schwierig zu schreiben und zu korrigieren.
3. Das Lesen umfangreicher Programme ist schwer.

Mit diesen drei Punkten wird ein Programmierer ständig konfrontiert. Wir wollen uns deshalb mit Methoden beschäftigen, die zur Arbeitserleichterung beitragen können.

Als Beispiel für den Programmentwurf wollen wir das letzte Programm des vorhergehenden Abschnitts heranziehen. Man erinnere sich: Es handelt sich dabei um das Programm zum Überprüfen der Fertigkeit zu addieren. Angenommen, man hat den Auftrag bekommen, ein solches Programm zu entwerfen. Wie sollte man vorgehen? In der ersten Eingebung möchte man am besten mit dem Niederschreiben von BASIC-Statements beginnen. Diesen Versuch gebe man um des Himmels willen gleich auf; man beginne erst nicht damit! Vielmehr fange man mit der Planung des Programmes an.

Der erste Schritt beim Planen eines Programmes ist die Bestimmung der erforderlichen Eingaben und Ausgaben. Welche Daten hat der Benutzer einzugeben und welche Antworten hat der Computer auszugeben? Man stelle sich zweckmäßigerweise dazu eine Übersicht auf:

Benutzereingabe: Antworten auf gestellte Fragen

Computerausgabe:
- Fragen, die vom Benutzer beantwortet werden müssen
- Erwiderungen auf die Antworten des Benutzers bezüglich der gestellten Fragen
  - Erwiderungen auf richtige Antworten
  - Erwiderungen auf falsche Antworten

- Bericht über die erzielte Punktzahl
- Schlußfrage hinsichtlich einer Wiederholung des Additionstestes

Im nächsten Schritt hat man diese Eingaben und Ausgaben in eine Folge von Schritten zu bringen, die logisch nacheinander angeordnet sind. Bis zu diesem Punkt kümmere man sich noch nicht um die einzelnen Anweisungen. Man beschreibe vielmehr die Schritte in allgemein verständlicher kompakter Form. Ein Schritt kann dabei durchaus mehrere Statements der BASIC-Sprache umfassen. Eine solche Beschreibung des Programmes zum Testen der Additionsfertigkeiten könnte beispielsweise wie folgt aussehen:

1. Fragenstellungen durch den Computer
2. Benutzerantworten auf die gestellten Fragen (Eingabe)
3. Analyse der Benutzerantworten und Erwiderungen
   a) Erwiderung, ob die Antworten korrekt sind
   b) Bei richtigen Antworten Erhöhung der Punktzahl (für richtige Antworten)
4. 10-malige Wiederholung der Schritte 1. bis 3.
5. Ausgabe der erreichten Punktzahl
6. Hinweis des Computers auf die Wiederholungsmöglichkeit des Tests (Ausgabe der Verhaltensweise bei gewünschter Wiederholung)

Der dritte Programmplanungsschritt besteht darin, die Struktur des Programmes zu skizzieren. Wir erkennen aus dem Pkt. 4., daß wir eine Schleife benötigen, um die Gesamtheit der vorgegebenen 10 Summationen verfolgen zu können. Darüberhinaus wissen wir, daß die Pkt. 1. und 2. in BASIC-Anweisungen sich niederschlagen, die nur eine Zeile umfassen. Diese Punkte sollten wir deshalb zusammen in einen Topf werfen, d.h. aus ihnen einen Programmabschnitt bilden. Andererseits ist die Behandlung richtiger Antworten anders vorzunehmen als die Behandlung falscher Antworten. Deshalb wollen wir auch diese beiden Antwortarten in getrennte Abschnitte hineinstellen. Ferner soll je ein gesonderter Programmabschnitt für die Pkt. 5. und 6. vorgesehen werden. Diese Gedanken sollte man unverzüglich zu Papier bringen (Abb. 3.20).

```
1Ø  FOR  N=1 TO 1Ø
:  (Die Zeilen 2Ø bis 9Ø bleiben für die Schritte 1. und 2. reserviert)
•
1ØØ   'Behandlung einer falschen Antwort
2ØØ   'Behandlung einer richtigen Antwort
3ØØ NEXT N
4ØØ 'Ausgabe der erreichten Punktzahl (Anzahl der richtigen Ergebnisse)
5ØØ 'Wiederholung der Programmausführung?
6ØØ END
```

Abb. 3.20 Entwurf einer Programmstruktur

Im vierten Schritt schließlich sollte man daran gehen, den geschaffenen Programmrahmen mit Statements zu füllen. Mit diesem Schritt also beginnt man erst, BASIC-Anweisungen niederzuschreiben, Variablen zu definieren usw. Jede unserer Programmschritte erfordert die Codierung nur weniger Anweisungen. Auf diese Weise läßt sich das Programm einfach niederschreiben. Unser Endprodukt sieht dann etwa so aus, wie es die Abb. 3.21 zeigt.

```
10  FOR  N=1 TO 10 :        'Schleife zur Stellung von 10 Aufgaben
20    INPUT "TIPPE ZWEI 2-STELLIGE ZAHLEN EIN"; A,B
30    INPUT "WIE GROSS IST IHRE SUMME"; C
40    IF  A + B = C  THEN 200
100   'Behandlung einer falschen Antwort
110   PRINT  "FALSCH ---> DIE RICHTIGE ANTWORT LAUTET", A+B
120   GOTO 300 :            'Übergang zur nächsten Aufgabe
200   'Behandlung einer richtigen Antwort
210   PRINT "DAS EINGEGEBENE ERGEBNIS IST RICHTIG! GRATULATION"
220   LET  R = R + 1 :      'Erhöhung der Zahl richtiger Ergebnisse
230   GOTO 300 :            'Übergang zur nächsten Aufgabe
300 NEXT N :                'Schleifenende
400 'Ausgabe der erreichten Punktzahl (Anzahl der richtigen Ergebnisse)
410 PRINT  "VON 10 AUFGABEN WURDEN RICHTIG GELÖST: ", R
500 PRINT  "ZUR WIEDERHOLUNG IST DER BEFEHL RUN EINZUGEBEN"
600 END
```

Abb. 3.21 Programm zum Testen der Fertigkeit des Addierens

Es ist denkbar, daß einige der Entwurfsschritte zu komplexen Operationsfolgen führt. Wenn das der Fall ist, so breche man diese Schritte in kleinere Schritte auf, so wie wir es für das gesamte Programm getan haben. Unter Umständen sollte man sein Programm auf eine wohlorganisierte Folge von einzelnen Programmteilen zurückführen; jeder Teil sollte nur in etwa ein Dutzend Statements münden. (Natürlich kann die tatsächliche Anzahl der Statements etwas größer oder kleiner sein, abhängig vom Komfort des Programmentwurfes. Man achte aber strikt darauf, daß die Anzahl nicht zu groß ist. Das nämlich wäre der sicherste Weg, auf dem man es erreichen kann, daß sich Fehler ins Programm einschleichen!). Bei der Überlegung hinsichtlich der Organisation eines Programmes kann man die verschiedenen Schritte oder Programmteile nicht isoliert voneinander betrachten. Vor einigen der sich daraus resultierenden Fehlerquellen sollte man sich unbedingt hüten:

1. Wenn eine Variable in zwei Schritten (Programmteilen) benötigt wird, dann muß sie in jedem Teil mit dem gleichen Namen benannt sein.
2. Wenn in einem Programmteil angenommen wird, daß in einem anderen Programmteil einer Variablen ein Wert zugewiesen wird, so sollte man das auf alle Fälle überprüfen.
3. Die gleiche Variable sollte nicht zur Bezeichnung verschiedener Größen benutzt werden; das führt nämlich oft zu einem fehlerhaften Gebrauch derselben. Nach mehreren Stunden Sitzungszeit an der Tastatur vergißt man nämlich allzu leicht, daß man einen Variablennamen bereits für irgendeine andere Größe vergeben hat. Selbstverständlich macht es nichts aus, wenn zwei Variablen, die in verschiedenen, voreinander isolierten Programmteilen angesprochen werden, den gleichen Namen erhalten. Jedoch sollte man bei solchen Variablen mit zwei Bedeutungen nicht einen Wert im Sinne der einen Bedeutung zuweisen und später von ihr Gebrauch im Sinne der anderen Bedeutung machen. Das kann dazu führen, daß Ergebnisse buchstäblich nur Schrottwert besitzen.
4. Man vergewissere sich, daß jeder Variablen der Anfangswert zugewiesen ist, den sie für den einwandfreien Ablauf des Programmes besitzen muß. (Man nennt diesen Vorgang auch „Initialisierung" der betreffenden Variablen.) In diesem Zusammenhang sei daran erinnert, daß bei fehlender Zuweisung eines Anfangswertes BASIC von sich aus den Anfangswert 0 zuweist.

Das soeben beschriebene Verfahren für den Entwurf und die Planung eines Programmes führt dazu, daß das betreffende Programm auch automatisch dokumentiert wird. Damit wird gleichzeitig das Programm leichter lesbar; als Folge davon kann man wiederum Fehlerquellen wesentlich rascher finden und die festgestellten Fehler schneller korrigieren, nicht nur zum Zeitpunkt der Erstellung des Programmes, sondern auch zu einem späteren Datum.

Die Diskussion, die wir in diesem Abschnitt aufgeworfen haben, kratzt gerade die Oberfläche des so außerordentlich wichtigen Themas der Planung und des Entwurfes von Programmen an. Trotzdem sind wir berechtigter Hoffnung, daß es für den Leser leicht sein wird, die Bürde des Schreibens und Verstehens von BASIC-Programmen zu tragen. Es sollte uns ferner auch gelingen, den Leser zu einem eigenen sinnvollen Vorgehen bei der Planung und dem Entwurf von Programmen zu bringen. Mehr über dieses Thema wird im Kapitel 5 ausgesagt.

3.4 Unterprogramme (Subroutinen)

Beim Niederschreiben von Programmen muß man oft die gleiche Anweisungsfolge mehr als einmal benutzen. Es ist erschwerend (und oft sogar fehleranfällig), die gleiche Statementfolge erneut einzutippen. Glücklicherweise bietet BASIC als Alternative einen weitaus bequemeren Weg an: das Unterprogramm, auch Subroutine genannt.

Unter einem Unterprogramm versteht man ein Programm, das in ein anderes, übergeordnetes Programm eingebettet ist. Das Unterprogramm kann vom übergeordneten Programm beliebig oft benutzt werden. Deshalb werden auch vielfach die Zeilen, die zum Unterprogramm gehören, isoliert an das Ende des übergeordneten Programmes verschoben. Diese Anordnung ist in der Abb. 3.22 veranschaulicht. Der Pfeil zum Unterprogramm markiert die Stelle im Hauptprogramm, von der aus das Unterprogramm benutzt wird. Man sagt, daß das Unterprogramm *aufgerufen* wird und nennt deshalb diese Stelle auch *Aufrufstelle*. Der Pfeil, der vom Unterprogramm wegführt, zeigt an, daß nach Vollendung der Ausführung des Unterprogrammes die Ausführung des Hauptprogrammes wieder aufgenommen wird, und zwar an dem Punkt, an dem es unterbrochen wurde. Man spricht in diesem Falle einfach von einer *Rückkehr* zum aufrufenden Programm.

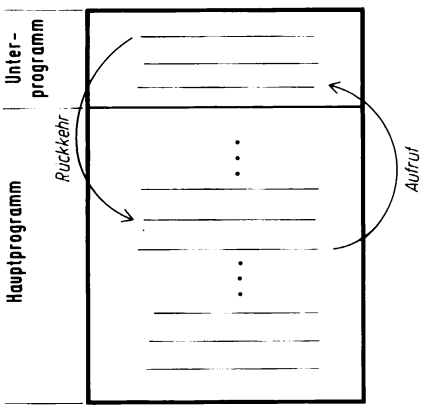


Abb. 3.22 Unterprogramm und Hauptprogramm

Unterprogramme werden mittels der beiden Statements

GOSUB und RETURN

gehandhabt. Das Statement

```
100 GOSUB 1000
```

z. B. führt den Computer zu dem Unterprogramm, das auf der Zeile mit der Zeilennummer 1000 beginnt. Der Computer führt danach ab dieser Zeile wie gewohnt Statement auf Statement nacheinander aus. Stößt er dabei auf ein RETURN-Statement, so kehrt er zum Hauptprogramm zurück und beginnt dort die weitere Ausführung auf der Zeile, die der Zeile mit der Zeilennummer 100 unmittelbar folgt.

Testübung 3.4.1

Man betrachte das folgende Programm:

```
10 GOSUB 40 :          'Aufruf des Unterprogrammes
20 PRINT "ZEILE 20"
30 END
40 PRINT "ZEILE 40"
50 RETURN   :          'Rückkehr ins Hauptprogramm
```

In welcher Reihenfolge werden die Statements ausgeführt?
Es sind die entsprechenden Zeilennummern zu nennen!

Unterprogramme können auch als sogenannte benutzerdefinierte Anweisungen benutzt werden, wie es das erste Beispiel zeigt.

Beispiel 1:

Man entwerfe ein Unterprogramm und schreibe es in BASIC nieder, das auf dem Bildschirm eine bestimmte Zeile l löscht und den Cursor auf die erste Stelle der gelöschten Zeile positioniert.

Das beschriebene Problem tritt sicher bei vielen Programmen auf. Es ist gewissermaßen das Vorspiel, wenn man auf die betreffende Zeile etwas aufzeichnen will. In der Tat kann es sogar mehrfach beim gleichen Programm erforderlich sein. Es wäre unsinnig, die gleiche Anweisungsfolge wiederholt niederzuschreiben. Deshalb laßt uns ein allgemeines Unterprogramm abfassen, das wir immer dann aufrufen können, wenn dies erforderlich sein sollte. Nehmen wir dazu zunächst an, daß die Zeile l gelöscht werden soll. Das Problem kann durch die folgenden Einzelschritte gelöst werden:

1. Positionieren des Cursors auf die erste Stelle der zu löschenden Zeile
2. Schreiben von 80 Leerzeichen (bzw. von 40, wenn WIDTH = 40 vorliegt)
3. Erneutes Positionieren des Cursors auf die erste Stelle der fraglichen Zeile

Um den Cursor auf die Spalte s der Zeile z zu positionieren, gebrauchen wir das Statement

```
LOCATE z,s
```

Zur Generierung der Leerzeichen könnten wir eine Schleife codieren. Es gibt jedoch einen einfacheren Weg. Unter SPACE$(n) wird eine n-stellige Kette aus Leerzeichen verstanden. Durch die Ausgabe die-

ser Kette können wir n Stellen mit Leerzeichen ausfüllen, also löschen, beginnend bei der augenblicklichen Stellung des Cursors. Das auf dieser Basis aufgebaute Unterprogramm ist in der Abb.3.23 aufgeführt.

```
5000 'Löschen der Zeile l
5010 LOCATE L,1 :  'Positionieren des Cursors auf die 1. Stelle der Zeile l
5020 PRINT SPACE$(80)
5030 LOCATE L,1
5040 RETURN
```

Abb.3.23 Unterprogramm zum Löschen von Zeilen

Immer, wenn wir dieses Unterprogramm verwenden wollen, müssen wir der Variablen L die Nummer der Bildschirmzeile zuweisen, deren Löschung wir beabsichtigen. Als nächstes Statement haben wir dann das Statement

```
GOSUB 5000
```

niederzuschreiben. Es ist unbedingt zu beachten, daß L vor diesem Statement ein Wert zugewiesen wird.

Testübung 3.4.2

Es ist ein Unterprogramm zu schreiben, mit dessen Hilfe die ersten M Spalten der Zeile L auf dem Bildschirm gelöscht werden können; ferner soll der Cursor in die linke obere Ecke des Bildschirmes gestellt werden.

Beispiel 2:

Es ist ein Programm zu schreiben, durch das der Computer in eine elektronische Registrierkasse verwandelt werden kann. Das Programm soll sowohl zu besteuernde als auch nicht zu besteuernde Beträge akzeptieren. Außerdem sollte es die jeweilige Summe beider Betragsarten aufbewahren. Auf Anforderung sollte es diese beiden Summen anzeigen sowie die Steuer und den Gesamtrechnungsbetrag berechnen. Eine Auflistung der Lösung findet man in Abb.3.25; dieses Programm illustriert einige der Tricks, die man bei der Planung benutzerfreundlicher Programme vorsieht. Wir ermöglichen nämlich dem Benutzer, daß er unter den folgenden Anforderungen des Programmes auswählen kann:

1. *Neuer Kunde (Neukunde)*
   Die Summen werden auf Null gesetzt, der Bildschirm wird gelöscht und daran anschließend werden auf dem Bildschirm Spaltenüberschriften ausgegeben (siehe Abb.3.24).
2. *Eingabe eines Betrages (Betragseingabe)*
   Es wird der Betrag für eine Ware akzeptiert. Das Programm fragt außerdem, ob dieser Betrag versteuert werden muß. Anschließend wird dieser Betrag unter der passenden Überschrift angezeigt und zur entsprechenden Summe (Summe der zu besteuerenden Beträge bzw. Summe der nicht zu besteuernden Beträge) addiert.
3. *Ermittlung des Rechnungsbetrages (Rechnungsbetrag)*
   Berechnung der Steuer und Anzeige der Summen
4. *Ende (Abschluß)*
   Ausgang aus dem Programm

```
 1   PC --- REGISTRIERKASSE
 2      1 - NEUKUNDE
 3      2 - BETRAGSEINGABE
 4      3 - RECHNUNGSBETRAG
 5      4 - ABSCHLUSS
 6              (Leerzeile)
 7                                          <------ Eingabezeile
 8              (Leerzeile)
 9   MIT STEUERN   OHNE STEUERN
10   ............  ............             <------ Ausgabe der Betrage
11              (Leerzeile)
12   SUMME MIT ST  SUMME OHNE ST
13   ............  ............             <------ Erste Gesamtsummenzeile
14              (Leerzeile)
15   STEUERN       RECHNUNGSBETRAG
16   ..............................        <------ Zweite Gesamtsummenzeile
17                 )
18                 |
19                 |
20                 >(Leerzeilen)
21                 |
22                 |
23                 |
24                 )
```

Abb. 3.24 Bildschirmanzeige bei einem neuen Kunden

Diese vier Auswahlmöglichkeiten werden in einer Liste angezeigt und damit dem Benutzer offeriert, die man „Menü" zu bezeichnen pflegt. Die Anweisungen, die die Ausgabe des Menüs bewirken, sind auf den Zeilen mit den Zeilennummern 1000 bis 1080 untergebracht.

Das Unterprogramm, das eine bestimmte Zeile löschen soll, beginnt auf der Zeile mit der Zeilennummer 6000. In vielen Programmteilen wollen wir auf eine bestimmte Bildschirmzeile etwas schreiben. Zu diesem Zweck benutzen wir dieses Unterprogramm, um zunächst einmal die fragliche Zeile zu löschen und den Cursor auf den Anfang der Zeile zu setzen.

In der Zeile mit der Zeilennummer 1100 wird der Benutzer aufgefordert, aus dem Menü diejenige Aktivität auszuwählen, die er anschließend durchführen will. Er erreicht diese Auswahl durch die Eingabe einer der Ziffern 1 bis 4. Basierend auf der Benutzerantwort, verzweigt das Programm daraufhin zu einem der Unterprogramme, die auf den Zeilen mit den Zeilennummern 2000, 3000 bzw. 4000 beginnen, oder im Fall der Eingabe einer 4, gar zum Programmabschluß (Zeile mit der Zeilennummer 5000). Obwohl auf der Zeile 5000 kein eigentliches Unterprogramm beginnt, haben wir dennoch diese Zeile mit einem GOSUB-Statement angesteuert; einige Bemerkungen hierzu folgen später. Nach der Ausführung des aufgerufenen Unterprogrammes erfolgt in den Fällen 1, 2 bzw. 3 stets die Rückkehr zu der Zeile, die der Zeile mit dem Aufrufstatement (GOSUB) unmittelbar folgt. Nach der Rückkehr wird in jedem Fall die Zeile mit der Zeilennummer 1150 angesteuert, deren Statement wiederum eine Rückverzweigung zur Zeile mit der Zeilennummer 1090 veranlaßt. Die Anweisungen, die auf dieser Zeile stehen, bewirken eine Löschung der Zeile 7 auf dem Bildschirm; danach wird vom Benutzer eine neue Aktivität angefordert. – Man kann dieses Programm alltäglich benutzen. Das Programmende wird durch Eingabe einer 4 angesteuert; ein Blick auf das Menü zeigt das dem Benutzer.

Die Wahlmöglichkeit 4 des Menüs auf dem Bildschirm verursacht den Übergang des Programmes zur Zeile mit der Zeilennummer 5000; der Bildschirm wird gelöscht und das Programm beendet (Zeile 5020). Hier könnten wir durchaus auf den Gebrauch eines Unterprogrammes verzichten, d.h. wir könnten auch

```
1000 'Anzeige des Menüs
1010    CLS
1020    LOCATE 1,1 :              'Ausgangsstellung des Cursors
1030    PRINT "PC --- REGISTRIERKASSE"
1040    PRINT "   1 - NEUKUNDE"
1050    PRINT "   2 - BETRAGSEINGABE"
1060    PRINT "   3 - RECHNUNGSBETRAG"
1070    PRINT "   4 - ABSCHLUSS"
1080    PRINT
1090    LET L=7:   GOSUB 6000:  'Löschen der Eingabezeile
1100    INPUT "BITTE CODE FÜR VORGANG EINGEBEN (1 BIS 4)"; ANTWORT$
1110    IF  ANTWORT$="1"  THEN GOSUB 2000
1120    IF  ANTWORT$="2"  THEN GOSUB 3000
1130    IF  ANTWORT$="3"  THEN GOSUB 4000
1140    IF  ANTWORT$="4"  THEN GOSUB 5000  ELSE 1090
1150    GOTO 1090
2000 'Unterprogramm zur Behandlung eines neuen Kunden
2010   'Nullsetzung der Summenfelder
2020    STSUMME=0:   NISTSUMME=0:     GESSUMME=0
2030   'Löschen der Zeilen 8 bis 24 auf dem Bildschirm
2040    FOR  L=8 TO 24
2050       LOCATE L,1
2060       PRINT SPACE$(40)
2070    NEXT L
2080   'Ausgabe der Überschriften
2090    LOCATE 9,1
2100    PRINT "MIT STEUERN","OHNE STEUERN"
2110    LOCATE 12,1
2120    PRINT "SUMME MIT ST","SUMME OHNE ST"
2130    LOCATE 15,1
2140    PRINT "STEUERN","RECHNUNGSBETRAG"
2150    RETURN :                  'Rückkehr
3000 'Unterprogramm zur Behandlung der Betragseingabe
3010    LET L=7:   GOSUB 6000:  'Löschen der Eingabezeile
3020    INPUT "BETRAG EINGEBEN (OHNE DM)"; BETRAG
3030    LET L=7:   GOSUB 6000:  'Löschen der Eingabezeile
3040    INPUT "MIT STEUERN - 1 --- OHNE STEUERN - 0";KENNUNG
3050    LET L=10:  GOSUB 6000
3060    IF  KENNUNG=1  THEN PRINT BETRAG,""
3070    IF  KENNUNG=0  THEN PRINT "",BETRAG
3080    IF  KENNUNG=1  THEN STSUMME=STSUMME+BETRAG
3090    IF  KENNUNG=0  THEN NISTSUMME=NISTSUMME+BETRAG
3100    RETURN :                  'Rückkehr
4000 'Unterprogramm zur Behandlung der Gesamtsumme (des Rechnungsbetrages)
4010    LET L=10:  GOSUB 6000:  'Löschen der Eingabezeile
4020    LET L=13:  GOSUB 6000:  'Löschen der ersten Gesamtsummenzeile
4030    PRINT STSUMME,NISTSUMME
4040    STEUERN = 0.05*STSUMME
4050    GESSUMME = STSUMME + STEUERN + NISTSUMME
4060    LET L=16:  GOSUB 6000:  'Löschen der zweiten Gesamtsummenzeile
4070    PRINT STEUERN,GESSUMME
4080    RETURN :                  'Rückkehr
5000 'Unterprogramm für den Abschluß
5010    CLS
5020    END :                     'Programmabschluß (keine Rückkehr)
6000 'Unterprogramm zum Löschen der Eingabezeile  L
6010    LOCATE L,1
6020    PRINT SPACE$(40) :        'Löschen der Eingabezeile
6030    LOCATE L,1
6040    RETURN :                  'Rückkehr
```

Anmerkungen:

1. *Der besseren Übersichtlichkeit wegen sind die einzelnen Programmeinheiten durch nachträgliche Hilfslinien am linken Rand markiert worden.*
2. *Die steuerabhängige Summenbildung (5% bzw. steuerfrei) entspricht nicht der deutschen Gesetzgebung. Die Berechnungen sind deshalb nur als Beispiel zu werten.*

Abb. 3.25 Programm zur Simulation einer Registrierkasse durch einen Personalcomputer (PC)

auf einem anderen Weg zu dieser Zeile gelangen, verzichten deshalb, weil wir keine Rückkehr mehr beabsichtigen. In diesem speziellen Fall kann jedoch

```
GOSUB 5000
```

ohne Schwierigkeiten benutzt werden. Da das Programm endet, schaut sich der Computer nicht nach einer Rückkehrstelle um. Eine gute Programmierpraxis sollte aber ein Unterprogramm dann und nur dann einsetzen, wenn garantiert ist, daß eine Rückkehrstelle angesteuert wird, d.h. daß das Unterprogramm auf ein RETURN-Statement stößt.

Testübung 3.4.3

Man sollte das Programm von Abb.3.25 so erweitern, daß als Teil des Unterprogrammes zur Bestimmung des Rechnungsbetrages nach dem gezahlten Betrag gefragt wird (10,– DM-Schein, 20,– DM-Schein usw.) und durch Vergleich mit dem Rechnungsbetrag das Wechselgeld ermittelt und angezeigt wird.

3.4.1 Ineinandergeschachtelte Unterprogramme

Im Beispiel 2 benutzten wir Unterprogramme, die von Unterprogrammen und nicht nur von Hauptprogrammen aufgerufen wurden. Beispielsweise bildeten die Zeilen mit den Zeilennummern von 4000 bis 4080 ein Unterprogramm. In diesem Unterprogramm wurde mehrere Male das Unterprogramm aufgerufen, das auf der Zeile mit der Zeilennummer 6000 beginnt (Löschen der Eingabezeile L). Solche Unterprogramme sind „ineinander verschachtelt"; sie bilden sogenannte *Unterprogrammnester.* BASIC ist in der Lage, mit solchen Unterprogrammnestern umzugehen. Man kann die Nester bis zu einer beliebigen Ebene verschachteln, also ein Unterprogramm in ein Unterprogramm in ein Unterprogramm usw. stecken. Man sollte jedoch sorgfältig darauf achten, daß sich eine RETURN-Anweisung immer auf das innerste Unterprogramm bezieht. Oder anders ausgedrückt, eine RETURN-Anweisung bezieht sich stets auf das Unterprogramm, das zuletzt aufgerufen wurde.

Trotz allem muß man Vorsicht walten lassen! Es ist nämlich zufällig oder gelegentlich möglich, daß man eine unbegrenzte Verschachtelung von Unterprogrammen erzeugt. Passieren kann das, wenn man wiederholt sich wiederholende GOSUB-Statements codiert. Ein simples Beispiel möge das illustrieren:

```
10 GOTO 20
20 GOSUB 10
```

Hier wird evtl. der Computer über den Speicher hinausgehen müssen, wenn er dieses Nest verfolgt. In der Konsequenz wird sich dann ein Fehler ergeben.

3.4.2 Die Anweisung GOSUB mit der ON-Klausel

Beim Beispiel 2 organisierten wir das Programm so, daß im Mittelpunkt vier Unterprogramme standen, entsprechend der vier Auswahlmöglichkeiten beim anfangs präsentierten Menü. Mehrere Anweisungen waren erforderlich, um das Programm so zu kanalisieren, daß es das richtige Unterprogramm ansteuerte. BASIC sieht für solche *Verteilungsaufgaben* eine andere,

bequeme Kurzform der Codierung vor, die Anweisung GOSUB mit der ON-Klausel. Diese Anweisung besitzt die Form:

```
ON <ausdruck> GOSUB <zeile1>, <zeile2>, ···
```

Wenn die Programmablaufsteuerung auf diese Anweisung stößt, wird zunächst der Wert von ausdruck ermittelt; dieser sollte ganzzahlig sein. Ergibt sich als Wert 1, so kommt es anschließend zur Ausführung eines GOSUB-Statements mit dem Verzweigungsziel <zeile 1>, ist der resultierende Wert gleich 2, so wird

```
GOSUB <zeile2>
```

ausgeführt usw. Ergibt sich als Wert 0 oder ist der sich ergebende Wert größer als die Anzahl der hinter GOSUB aufgeführten Zeilennummern, wird die GOSUB-Anweisung von der Programmablaufsteuerung nicht beachtet. Ein Fehler wegen eines ungültigen Funktionsaufrufes kommt jedoch dann zustande, wenn der Wert von ausdruck negativ ist oder wenn er größer als 255 ist.

Schauen wir noch einmal auf die Zeilen mit den Zeilennummern 1110 bis 1130 des Beispieles 2 (Abb. 3.25). Mit unseren jetzigen Kenntnissen könnten wir diese Zeilen durch eine einzige ersetzen, nämlich durch

```
1110    ON VAL(ANTWORT$)  GOSUB 2000,3000,4000
```

Der hier auftretende Ausdruck

```
VAL(ANTWORT$)
```

konvertiert die Kette von ANTWORT$ in ihr numerisches Äquivalent, d.h. "1" wird in 1 ungewandelt, "2" in 2 usw. Wenn das entstehende numerische Äquivalent gleich 1 ist, kommt es zur Ausführung von

```
GOSUB 2000
```

Bei einem Wert von 2 wird

```
GOSUB 3000
```

ausgeführt und beim Wert 3

```
GOSUB 4000
```

Aufgabengruppe 8

1. Es ist ein Unterprogramm zu schreiben, das 10 Sternzeichen, also * ausgibt, beginnend auf der ersten Stelle der Bildschirmzeile L.
2. Es ist ein Unterprogramm zu schreiben, das M Sternzeichen, also *, ausgibt, beginnend auf der ersten Stelle der Zeile L.
3. Es ist ein Unterprogramm zu schreiben, das M Sternzeichen, also *, ausgibt, beginnend auf der Stelle K der Zeile L.

4. Schreibe ein Programm, das das Unterprogramm der 3. Aufgabe dieser Aufgabengruppe aufruft, um Folgen von Sternzeichen auszugeben, und zwar gemäß der folgenden Vorgaben:
   a) K=5, L=3, M=30
   b) K=4, L=5, M=35
   c) K=8, L=7, M=12

5. Man betrachte die Anweisung

```
10 ON J-2  GOSUB 100,200,300,400,500
```

   Wohin erfolgt die Verzweigung, wenn die folgenden Vorgaben vorliegen:

   a) J=4, b) J=7, c) J=2, d) J=10, e) J=0

6. Man betrachte das folgende Programm:

```
10  Y = 5
20  J = 3
30  S = Y - J
40  ON S  GOSUB 100,200,300,400
50  CLS
60  END
100 RETURN
200 RETURN
300 RETURN
400 RETURN
```

Welche beiden Statements werden unmittelbar nach dem Statement auf der Zeile mit der Zeilennummer 40 ausgeführt?

Antworten auf die Testübungen

3.4.1 10 → 40 → 50 → 20 → 30

3.4.2 Das Unterprogramm könnte das folgende Aussehen haben:

```
5000 'Löschen der ersten M Spalten der Zeile L
5010  LOCATE L,1: 'Positionierung des Cursors auf die 1. Stelle
                   der Zeile L
5020  PRINT SPACE$(M)
5030  LOCATE 1,1
5040  RETURN
```

3.4.3 Man hätte die folgenden Zeilen hinzuzufügen:

```
4071    LOCATE 20,1
4072    PRINT "GEZAHLTER BETRAG"; ZAHLUNG
4073    PRINT "ZAHLUNG","WECHSELGELD"
4074    PRINT ZAHLUNG,ZAHLUNG-GESSUMME
```

4 Verarbeitung von Daten

4.1 Die Verarbeitung von tabellarischen Daten

Im vorhergehenden Kapitel befaßten wir uns mit der Schreibweise von Variablen und benutzten dabei Variablennamen, die beispielsweise wie folgt aussahen:

```
AA, B1, CZ, WO, ···
```

Mitunter reichen aber die uns bis jetzt zur Verfügung stehenden Variablen nicht aus, wenn wir uns der Bedürfnisse erinnern, die manche Programme an uns stellten. Wie wir deshalb in diesem Kapitel sehen werden, gibt es tatsächlich viele relativ harmlose Programme, die Hunderte oder sogar Tausende von Variablen erfordern. Um diese Notwendigkeiten abzudecken, gestattet BASIC den Gebrauch von sogenannten „indizierten Variablen". Solche Variablen werden auch häufig von Mathematikern verwendet; sie werden von ihnen durch Buchstaben identifiziert, an die tiefer gestellte Zahlen angehängt werden. Eine Liste von 1000 verschiedenen Variablen würde somit von den Mathematikern wie folgt geschrieben:

$$a_1,\ a_2,\ a_3,\ \cdots,\ a_{999},\ a_{1000}$$

Die Zahlen, die zur Unterscheidung der einzelnen Variablen dienen, heißen „Indizes". In ähnlicher Weise erlaubt auch BASIC die Definition von Variablen, die sich durch Indizes unterscheiden. Da sich jedoch bei Computern Schwierigkeiten ergeben würden, wenn man die Indizes auf die traditionelle Art an die Buchstaben anhängen würde, hat man sich zu der folgenden Schreibweise durchgerungen: Man schließt die Indizes einfach in Klammern ein, wodurch sie man in der gleichen Zeile wie die Buchstaben unterbringen kann. Die oben aufgeführte Liste von Variablennamen läßt sich somit in BASIC wie folgt schreiben:

```
A(1), A(2), A(3), ··· , A(999), A(1000)
```

Ausdrücklich sei an dieser Stelle noch einmal betont, daß die Variable A(1) nicht gleich der Variablen A ist; A(1) ist eine Variable, die Element eines Bereiches von Variablen ist, während A eine für sich allein stehende Variable ist.

Eine indizierte Variable stellt in Wirklichkeit eine Gruppe von Variablen dar, die einen gewöhnlichen Buchstaben als Identifizierung aufweist: die einzelnen zur Gruppe gehörenden Variablen werden durch „Indizes" unterschieden. Beispielsweise wird die weiter oben bereits erwähnte Variablengruppe die indizierte Variable A(···) bilden. Mitunter ist es recht sinnvoll, wenn man eine indizierte Variable als Tabelle oder Bereich betrachtet. Die soeben betrachtete indizierte Variable A(···) könnte also auch unter dem Blickwinkel einer Tabelle von Informationen angesehen werden.

```
A(1)
A(2)
A(3)
⋮
A(1000)
```

Wie hier gezeigt, definiert diese indizierte Variable eine aus 1000 Zeilen bestehende Tabelle.

In der Zeile J ist eine einzige Eintragung vorhanden, nämlich der Wert der Variablen A(J); die erste Zeile enthält also den Wert von A(1), die zweite den Wert von A(2) usw. Weil eine indizierte Variable als Tabelle (oder Bereich) angesehen werden kann, werden indizierte Variablen oft auch kurz einfach als Bereiche bezeichnet.

Der dargestellte Bereich ist, wie wir gesehen haben, eine aus 1000 Zeilen und einer Spalte bestehende Tabelle. Der IBM Personalcomputer ermöglicht jedoch auch die Berücksichtigung viel allgemeinerer Bereiche. Man betrachte dazu beispielsweise eine Einnahmentabelle, in der die monatlichen Einkünfte für die Monate Januar, Februar und März für jede der Reinigungen aufgezeichnet sind, die zu einer Kette von vier Reinigungen gehören:

	1. Geschäft	*2. Geschäft*	*3. Geschäft*	*4. Geschäft*
Jan.	1258.38	2437.46	4831.90	987.12
Febr.	1107.83	2045.68	3671.86	1129.47
März	1298.00	2136.88	4016.73	1206.34

Diese Tabelle weist drei Zeilen und vier Spalten auf. Die Eintragungen in diese Tabelle können im Computer in einer Menge von 12 Variablen gespeichert werden.

```
A(1,1)  A(1,2)  A(1,3)  A(1,4)
A(2,1)  A(2,2)  A(2,3)  A(2,4)
A(3,1)  A(3,2)  A(3,3)  A(3,4)
```

Dieser Variablenbereich besitzt eine gewisse Ähnlichkeit mit einer indizierten Variablen; allerdings sind jetzt zwei Indizes vorhanden. Der erste Index kennzeichnet die Zeilennummer und der zweite die Spaltennummer. So steht z.B. die Variable A(3,2) in der dritten Zeile auf der zweiten Spalte. Eine Ansammlung von Variablen wie die soeben dargestellte wird als zweidimensionaler Bereich oder als doppelt indizierte Variable bezeichnet.

Durch Zuweisungen von Werten zu einem solchen Bereich ist somit eine Wertetabelle definiert. Wenn wir z.B. die folgenden Zuweisungen vornehmen

```
A(1,1) = 1258.38,   A(1,2) = 2437.46,   A(1,3) = 4831.90
```

usw., so bekommen wir eine Tabelle der Einkünfte der Geschäfte der Reinigungskette.

Bisher haben wir nur numerische Bereiche betrachtet, d.h. Bereiche, deren Variablen nur numerische Werte annehmen können. Es ist jedoch auch möglich, Bereiche mit Variablen zu bilden, denen Kettenkonstanten als Werte zugewiesen werden könnnen. (Man erinnere sich, daß eine Kettenkonstante eine Folge von beliebigen Zeichen darstellt: Buchstaben, Ziffern und Sonderzeichen, wie Punkte, Kommas und andere druckbare Zeichen). Als Beispiel für einen Bereich, der Kettenkonstanten enthält, nennen wir

```
A$(1)
A$(2)
A$(3)
A$(4)
```

Hier signalisiert wie bisher das Währungszeichen (Dollarzeichen), daß jede der Variablen des Bereiches eine Kettenvariable ist. Wenn wir nun die folgenden Zuweisungen vornehmen

```
A$(1) = "LANGSAM",   A$(2) = "SCHNELL",
A$(3) = "SCHNELL",   A$(4) = "HALT",
```

so erhalten wir folgerichtig eine Tabelle von Wörtern, nämlich

```
LANGSAM
SCHNELL
SCHNELL
HALT
```

In gleicher Weise kann eine Tabelle von Sätzen mit den Daten von Arbeitnehmern

Versicherungs-nummer	*Alter*	*Geschlecht*	*Familien-stand*
130528G030	55	M	VERH.
090724S125	59	W	GESCH.
240243S103	40	W	GESCH.
091129E002	54	W	VERH.
210261W001	22	W	LED.

in einen Bereich gespeichert werden, der mit B$(I,J) bezeichnet ist. Hierbei kann I die Werte 1,2,3,4 und 5 annehmen, d.h. die Zeilen kennzeichnen, und J die Werte 1,2,3 und 4, d.h. die Spalten kennzeichnen. Beispielsweise hat also B$(1,1) den Wert "130528G030", B$(1,2) den Wert "55", B$(1,3) den Wert "M" und B(1,4) den Wert "VERH." usw.

Der IBM Personalcomputer gestattet sogar, Bereiche zu bilden, die drei, vier oder sogar noch mehr Indizes aufweisen. Ein Beispiel hierfür ist schnell gefunden. Wir brauchen nur die Kette von Reinigungsgeschäften zu betrachten, die wir bereits kennengelernt haben. Nehmen wir einmal an, wir hätten je einen solchen Bereich für 10 aufeinanderfolgende Jahre gebildet. Diese Datensammlung könnte dann ohne weiteres in einen dreidimensionalen Bereich der Form C(I,J,K) gespeichert werden. Die Indizes I (Zeilen) und J (Spalten) haben die gleiche Bedeutung wie zuvor; der Index K symbolisiert die Jahre. Der Index K durchläuft also die Werte von 1 bis 10.

Ein Bereich kann mit einer beliebigen Zahl von Dimensionen vorliegen; maximal sind 255 gestattet. Die Werte der Indizes, die zu jeder Dimension gehören, können von 0 bis 32767 reichen. Bei allen praktischen Anwendungen genügen diese Begrenzungen vollauf.

Der Computer muß über den Umfang der Bereiche informiert werden, die der Benutzer in seinem Programm anzusprechen gedenkt. Dadurch erst wird es dem Computer möglich, soviel Speicherplatz bereitzustellen, daß alle Werte untergebracht werden könnnen. Um den Umfang eines Bereiches festzulegen, muß man das Statement

```
DIM
```

verwenden. Mehr als diese Dimensionsvereinbarung braucht man nicht vorzusehen. Um beispielsweise den Umfang der indizierten Variablen A(J) mit $J = 1,2,3,\cdots,1000$ zu definieren, müssen wir die Vereinbarung

```
10 DIM A(1000)
```

in das betreffende Programm aufnehmen. Dieses Statement informiert den Computer, daß es im Programm die Variablen

```
A(0), A(1), A(2), ··· , A(1000)
```

erwarten muß und deshalb für die Werte dieser 1 001 Variablen Speicherplatz bereitzustellen hat. Man merke sich unbedingt, daß bei Fehlen weiterer Instruktionen von Seiten des Benutzers das BASIC alle Indizes mit 0 beginnen läßt. Wenn man A(0) benutzen will, um so besser, wenn nicht, dann läßt man einfach A(0) unbeachtet links liegen.

Man braucht nicht alle Variablen zu verwenden, die mittels eines DIM-Statements definiert wurden. Beispielsweise könnte man im Falle des obigen DIM-Statements tatsächlich nur die Variablen

```
A(1), ··· , A(900)
```

benötigen. Das schadet wirklich nichts! Man muß nur sicherstellen, daß genügend Variablen definiert sind, anderenfalls wird es Ärger geben. Wenn beispielsweise im Falle der obigen indizierten Variablen das Programm zu einer Variablen A(1 001) zugreifen will, wird eine Fehlersituation entstehen. Nehmen wir an, diese Variable wird zum ersten Mal in der Zeile mit der Zeilennummer 570 angesprochen. Versucht man nun, dieses Programm ablaufen zu lassen, so meldet sich der Computer mit der Nachricht

```
Subscript out of range in 570
```
(Index außerhalb seines Wertebereiches in Zeile 570)

Darüberhinaus wird die Ausführung des Programmes gestoppt. Zur Beseitigung dieses Fehlers braucht man bloß das DIM-Statement so neu zu formulieren, daß es auch den nicht definierten Indizes gerecht wird.

Um den Umfang eines zweidimensionalen Bereiches zu definieren, benutzt man eine DIM-Vereinbarung der Form

```
10 DIM A(4,5)
```

Durch dieses Statement ist ein Bereich A(I,J) festgelegt; hierbei kann I die Werte 0,1,2,3 und 4 annehmen, J die Werte 0,1,2,3,4 und 5. In ähnlicher Weise werden auch die Bereiche mit drei und mehr Dimensionen definiert.

Testübung 4.1.1

Gegeben sei ein Bereich mit den folgenden Werten:

12	645.80
148	489.75
589	12.89
487	14.50

a) Es ist eine indizierte Variable festzulegen, die diese Daten aufnehmen kann!

b) Es ist ein geeignetes DIM-Statement zu formulieren!

Es ist möglich, in einem DIM-Statement mehrere Bereiche zu dimensionieren. Beispielsweise werden durch die Dimensionsvereinbarung

```
10 DIM  A(1000), B$(5), C(5,4)
```

folgende Bereiche festgelegt:

- eindimensionaler numerischer Bereich A(0),A(1),···,A(1 000)
- eindimensionaler Kettenbereich B$(0),···,B$(5)
- zweidimensionaler numerischer Bereich C(I,J) mit I=0,···,5 und J=0,···,4

Wir wissen nun, wie wir Speicherplatz für die Variablen eines Bereiches reservieren können. Wir müssen nunmehr das Problem aufgreifen, wie wir diesen Variablen Werte zuweisen können. Natürlich könnten wir hintereinander individuelle LET-Statements vorsehen, aber bei einem Bereich mit beispielsweise 1 000 Variablen könnte das zu einer nicht mehr handhabbaren Anzahl von Statements führen. Viel sinnvoller ist einer Vorgehensweise, die sich der Schleifentechnik bedient. Die beiden nächsten Beispiele illustrieren zwei derartige Methoden.

Beispiel 1:

Es ist ein Bereich A(J) mit J=1,2,3,···,1 000 zu definieren. Anschließend sind den Variablen dieses Bereiches die folgenden Werte zuzuweisen:

```
A(1)=2,   A(2)=4,   A(3)=6,   A(4)=8,
```

Zur Lösung ist zunächst zu bemerken, daß wir jeder Variablen einen Wert zuzuweisen haben, der dem Zweifachen des Index der Variablen entspricht, d.h. der Variablen A(J) ist der Wert 2*J zuzuweisen. Zu diesem Zweck verwenden wir am zweckmäßigsten eine Schleife (Abb. 4.1).

```
10 DIM  A(1000)
20 FOR  J=1 TO 1000
30   A(J) = 2*J
40 NEXT J
50 END
```

Abb. 4.1 Wertzuweisungen zu den Variablen eines Bereiches

Man beachte, daß das Programm die Variable A(0) nicht berücksichtigt. Wie jede andere Variable, der kein Wert zugewiesen ist, weist sie damit den Wert 0 auf.

Testübung 4.1.2

Es ist ein Programm zu schreiben, das den Variablen

A(0),A(1),···,A(30)

eines Bereiches der Reihe nach die Werte 0,1,4,9,···,900 zuweist.

Wenn der Computer ausgeschaltet oder auf seinen Grundzustand zurückgesetzt wird, werden bekanntlich alle Variablen, inklusive der in Bereichen, gelöscht. Somit werden die numeri-

schen Variablen auf Null gesetzt, die Kettenvariablen hingegen auf die sogenannte Nullkette, d.h. auf eine Kette, die kein Zeichen enthält. Wenn man während der Programmausführung alle Variablen auf diesen Zustand zurückzuführen wünscht, kann man das durch die Anweisung CLEAR erreichen. Wenn also z.B. der Computer dem Statement

```
570 CLEAR
```

begegnet, so setzt er alle Variablen zurück, d.h. auf den oben beschriebenen Grundzustand. Die Anweisung CLEAR könnte z.B. sehr sinnvoll sein, wenn man einen Bereich zur Speicherung von zwei unterschiedlichen Informationsmengen zu verschiedenen Zeiten (Programmablaufphasen) zu benutzen beabsichtigt. Nach der ersten Verwendung des Bereiches könnte man ihn in einem solchen Falle für seinen zweiten Verwendungszweck präparieren, indem man die Anweisung CLEAR ausführen läßt.

Beispiel 2:

Es ist ein Bereich zu definieren, der die Tabelle mit den Daten von Arbeitnehmern (siehe oben) aufnehmen kann. Die Daten sind einzugeben und danach auf dem Bildschirm auszugeben.

Das von uns entwickelte Programm gibt zunächst Anforderungen nach den jeweiligen Daten aus, die daraufhin eingegeben werden müssen; die Eingabe wird immer für die gesamten Daten einer Zeile angefordert und vorgenommen. Die eingegebenen Daten werden im Bereich B$(I,J) gespeichert. Die Indizes durchlaufen dabei die folgenden Werte:

```
I=1,2,3,4,5      und      J=1,2,3,4
```

Infolgedessen ist der Bereich zu B$(5,4) zu dimensionieren. Das Programm ist in der Abb. 4.2 aufgelistet.

```
10  DIM  B$(5,4)
20  FOR  I=1 TO 5
30    INPUT "VERS.-NR.,ALTER,GESCHL.,FAM.-STD.";
            B$(I,1), B$(I,2),B$(I,3),B$(I,4)
40  NEXT I
50  CLS
60  PRINT "Vers.-Nr.","Alter","Geschl.","Fam.-Std."
70  FOR  I=1 TO 5
80    PRINT  B$(I,1),B$(I,2),B$(I,3),B$(I,4)
90  NEXT I
100 END
```

Abb. 4.2 Eingabe von Datensätzen in einen zweidimensionalen Bereich und anschließender Ausgabe derselben

Testübung 4.1.3

Angenommen, ein Programm benutzt folgende Bereiche:

- Bereich A$(I,J) mit 9 Zeilen und 2 Spalten
- Bereich B$(I,J) mit 9 Zeilen und 1 Spalte
- Bereich C(I,J) mit 9 Zeilen und 5 Spalten

Es ist das entsprechende DIM-Statement zu codieren (bzw. mehrere DIM-Statements)!

Wenn man die Dimensionierung eines Bereiches ins Auge faßt, muß man immer das DIM-Statement vor dem ersten Erscheinen einer zum Bereich gehörenden Variablen ins Programm einfügen. Anderenfalls würde nämlich BASIC die angestrebte Dimensionierung durchkreuzen. Wenn BASIC zuerst eine Bezugnahme auf einen Bereich entdeckt, ohne daß für denselben schon ein DIM-Statement vorgelegen hat, nimmt BASIC an, daß die Indizes von 0 bis 10 laufen. Wird dann in späteren Programmzeilen noch ein DIM-Statement für den betreffenden Bereich gefunden, so unterstellt BASIC, daß der Programmierer den Umfang eines Bereiches mitten im Programm noch ändern will; das aber ist nicht erlaubt. Als Resultat ergibt sich logischerweise eine Fehlersituation, über die der Benutzer durch die Nachricht

```
Duplicate Definition
```
(doppelte Definition)

informiert wird.

In unserer bisherigen Diskussion haben wir uns nur sehr beiläufig über die Ignorierung nicht benötigter Indizes geäußert, z.B. mehrfach über A(0). Bei einigen Programmen kann die Verwendung mehrerer sehr umfangreicher Bereiche notwendig sein, wodurch es zu einem Engpaß beim Speicherplatz kommen kann. Mitunter kann eine beträchtliche Menge von Speicherplatz durch sorgfältige Planung allein dadurch eingespart werden, indem man sich überlegt, welche Indizes tatsächlich benutzt werden müssen und deshalb nur die entsprechenden Variablen definiert. Dies kann dann durch das Statement OPTION BASE im Programm realisiert werden. Das Statement

```
10  OPTION BASE 1
```

z.B. sorgt dafür, daß alle vorgesehenen Bereiche Indizes aufweisen, die mit 1 beginnnen und nicht mit 0. Natürlich muß ein solches Statement in einem Programm vor allen Dimensionierungsvereinbarungen erscheinen.

Aufgabengruppe 9

Für jede der nachfolgenden Tabellen ist ein geeigneter Bereich anzugeben und das entsprechende DIM-Statement niederzuschreiben!

```
1.  5              2.  1.1   2.0   3.5
    2                  1.7   2.4   6.2
    1.7
    4.9
   11

3. JOHN            4.  1   2   3
   MARY
   SYDNEY

5. MIETE       575.00
   ERNÄHRUNG   249.78
   KLEIDUNG    174.98
   AUTO        348.70
```

6. Die Daten des folgenden Bereiches sind auf dem Bildschirm auszugeben:

	Wareneingänge in DM		
	Geschäft 1	*Geschäft 2*	*Geschäft 3*
01.1. bis 10.1.	57385.48	89485.45	38456.90
11.1. bis 20.1.	39485.98	76485.49	40387.86
21.1. bis 31.1.	45467.21	71494.25	37983.38

7. Es ist ein Programm zu schreiben, das den Bereich der Aufgabe 6. anzeigt; ferner ist die Gesamtsumme der Wareneingänge für jedes Geschäft zu berechnen und ebenfalls auszugeben.

8. Das Programm von Aufgabe 7. ist so zu erweitern, daß außerdem die Gesamtsummen der Wareneingänge pro Dekade errechnet und angezeigt werden.
   Die Gesamtsummen pro Dekade sind in gesonderten Zeilen, mit entsprechenden Überschriften versehen, anzubringen.

9. Es ist ein Programm zu entwerfen, das die Warenbestände einer Kette von vier Hausratgeschäften fortschreibt. Der jeweils ermittelte Warenbestand ist in einem Bereich zu speichern, der die folgende Form aufweist:

	Geschäft 1	*Geschäft 2*	*Geschäft 3*	*Geschäft 4*
Kühlschränke				
Öfen				
Waschautomaten				
Wäschetrockner				
Klimaanlagen				

Das Programm soll folgende Teilaufgaben erfüllen:

a) Eingabe der Warenbestände zu Tagesbeginn,
b) fortlaufende Aufforderung, die nächste Warenbewegung (Verkauf, Lieferung) einzugeben, und zwar unter Angabe des Geschäftes, der betreffenden Ware und der Menge (Zugänge positiv, Abgänge negativ),
c) auf Anforderung den augenblicklichen Warenbestand anzuzeigen.

Antworten zu den Testübungen

4.1.1 a) `A(I,J) mit I=1,2,3,4 und J=1,2`
b) `10 DIM A(4,2)`

4.1.2
```
10  DIM  A(30)
20  FOR  J=0 TO 30
30    A(J) = J^2
40  NEXT J
50  END
```

4.1.3
```
10 DIM A$(9,2),B$(9,1),C(9,5)
```

4.2 Dateneingabe

Im vorhergehenden Abschnitt befaßten wir uns mit Bereichen und besprachen mehrere Methoden, wie wir den Variablen eines Bereiches, den Bereichselementen also, Werte zuweisen können. Die flexibelste Methode war augenscheinlich die Benutzung des INPUT-Statements. Für umfangreiche Bereiche kann sie jedoch ziemlich langschweifig sein. BASIC ermöglicht deshalb auch den Einsatz alternativer Verfahren für die Eingabe von Daten.

Ein gegebenes Programm kann viele verschiedene Zahlen und Zeichenketten benötigen. Die hierzu erforderlichen Daten können in einem oder in mehreren DATA-Statements zur Verfügung gestellt werden. Ein typisches DATA-Statement besitzt die Form

```
10 DATA  3.457, 2.588, 11234, "SPANNWEITE"
```

Das hier vorgelegte DATA-Statement enthält vier Datenelemente, drei numerische und eine Zeichenkette. Die Datenelemente sind durch Kommas voneinander getrennt. Man kann in ein DATA-Statement soviele Datenelemente aufnehmen, wie die Zeilenlänge es zuläßt. Darüberhinaus kann man eine beliebige Anzahl von DATA-Statements in ein Programm einbauen, und man kann sie außerdem an einer beliebigen Stelle ins Programm einfügen. Gewöhnlich stellt man sie freilich der Übersicht wegen unmittelbar vor das END-Statement. Zu beachten ist ferner, daß wir die Zeichenkettenkonstante in Anführungszeichen gestellt haben, so wie wir es gelernt haben. Tatsächlich ist das hier nicht notwendig. Eine Zeichenkettenkonstante in einem DATA-Statement muß nur dann von Anführungszeichen eingeschlossen werden, wenn sie ein Komma oder ein Semikolon enthält bzw. wenn sie mit einem Leerzeichen[1] beginnt.

Das DATA-Statement kann zur Zuweisung von Werten zu Variablen, insbesondere zu Bereichen, eingesetzt werden. Wie das zu bewerkstelligen ist, soll nun geschildert werden. In Verbindung mit den DATA-Statements sind dazu ein oder mehrere READ-Statements zu benutzen. Man nehme beispielsweise einmal an, daß das obige DATA-Statement Teil eines Programmes ist. Wir wollen weiterhin annehmen, daß folgende Wertzuweisungen erfolgen sollen:

```
A = 3.457,   B = 2.588,   C = 11234,   Z$ = "SPANNWEITE"
```

Mit Hilfe des nachfolgenden READ-Statements läßt sich das auf einfache Art und Weise erreichen:

```
1000 READ A,B,C,Z$
```

Wir wollen nunmehr die Funktionsweise des READ-Statements beschreiben. Wenn der Computer ein READ-Statement feststellt, schaut er sich sofort nach einem DATA-Statement um. Anschließend beginnt er Werte den im READ-Statement aufgeführten Variablen zuzuweisen; die Werte werden der Reihe nach vom DATA-Statement genommen. Sind im ersten DATA-Statement nicht genügend viel Werte enthalten, um alle im READ-Statement vorgesehenen Wertzuweisungen zu befriedigen, so nimmt der Computer die Werte aus dem zweiten DATA-Statement und fährt mit den Wertzuweisungen fort. Wenn erforderlich, geht der Computer zum dritten DATA-Statement über usf.

---

[1] Auch als Zwischenraumzeichen oder Blank bezeichnet.

Testübung 4.2.1

Es sind die folgenden Wertzuweisungen durchzuführen:

```
A(1) = 5.1,     A(2) = 4.7,     A(3) = 5.8,
A(4) = 3.2,     A(5) = 7.9,     A(6) = 6.9
```

Der Computer verwaltet einen internen Zeiger, der jeweils zum nächsten Datenelement, das zu benutzen ist, zeigt. Wenn ein zweites READ-Statement auftaucht, so beginnt er das Lesen dort, wo der Zeiger stehen geblieben ist. Wenn wir z. B. anstelle des obigen READ-Statements die zwei Leseanweisungen

```
100 READ A,B
200 READ C,Z$
```

verwenden, so wird sich der Computer ständig nach der Stellung des Zeigers umschauen. Anfangs wird der Zeiger auf das erste Datenelement des ersten DATA-Statements gesetzt. Die erste Leseanweisung bewirkt, daß den Variablen A bzw. B die Werte 3.457 bzw. 2.588 zugewiesen werden. Der Zeiger wird darüberhinaus noch auf das dritte, im DATA-Statement stehende Datenelement geführt. Begegnet der Computer nun dem nächsten READ-Statement, so beginnt er mit der Wertzuweisung bei dem Datenelement, auf das der Zeiger augenblicklich verweist. Somit kommt es jetzt zur Zuweisung der Werte 11234 bzw. "SPANNWEITE" zu den Variablen C bzw. Y$.

Testübung 4.2.2

Welche Werte werden den Variablen A und B$ durch das nachfolgende Programm zugewiesen?

```
10 DATA 10,30,"MOTOR","GERAET"
20 READ A,B
30 READ C$,B$
40 END
```

Das nachfolgende Beispiel illustriert die Verwendung von DATA-Statements für die Zuweisung von Werten zu den Variablen in Bereichen, den Bereichselementen also.

Beispiel 1:

Angenommen, die monatlichen Stromkosten einer Familie betragen lt. Abrechnung des Elektrizitätswerkes:

Januar	89.74 DM	Juli	158.92 DM
Februar	95.84 DM	August	164.38 DM
März	79.42 DM	September	105.98 DM
April	78.93 DM	Oktober	90.44 DM
Mai	72.11 DM	November	89.15 DM
Juni	119.94 DM	Dezember	93.97 DM

Es ist ein Programm zu schreiben, das die durchschnittlichen monatlichen Stromkosten errechnet.

Die Erarbeitung der Lösung soll von folgenden Gedankengängen Gebrauch machen. Alle in der vorgelegten Tabelle enthaltenen Zahlen wollen wir formlos in DATA-Statements aufnehmen, die wir ans Ende des Programmes stellen wollen. Die DATA-Statements wollen wir willkürlich bei der Zeilennummer 1000 beginnen lassen, das END-Statement wollen wir auf die Zeile mit der Zeilennummer 2000 stellen. Damit steht uns hinsichtlich der Zeilennummern genügend Platz zur Verfügung, um die übrigen Statements unterbringen zu können. Zur Berechnung des Durchschnitts müssen wir bekanntlich alle monatlichen Stromkosten addieren und anschließend durch 12 dividieren. Um diese Operationen durchführen zu können, wollen wir zunächst einen Bereich A(J) mit $J=1,2,\cdots,12$ generieren. Danach müssen wir A(J) gleich den monatlichen Stromkosten des j-ten Monats setzen. Wir erledigen das durch eine Schleife, in die wir ein READ-Statement hineinstellen. Im Anschluß an eine zweite Schleife, die sich mit der Addition der monatlichen Stromkosten befaßt, nehmen wir die Division durch 12 vor und geben mittels der PRINT-Anweisung das Ergebnis aus. Das auf dieser Basis geschaffene Programm ist in der Abb. 4.3 dargestellt.

```
10    DIM A(12)
20    FOR J=1 TO 12
30      READ A(J)
40    NEXT J
50    FOR J=1 TO 12
60      S = S + A(J):        'S enthält die Summe der A(J)
70    NEXT J
80    S = S / 12:            'Division von S durch 12
90    PRINT "DIE DURCHSCHN. MONATL. STROMKOSTEN BETRAGEN",S
1000  DATA  89.74, 95.84, 79.42, 78.93, 72.11, 115.94
1010  DATA  158.92, 164.38, 105.98, 90.44, 89.15, 93.97
2000  END
```

Abb. 4.3 Programm zur Berechnung der durchschnittlichen monatlichen Stromkosten

Das Programm, das wir im Beispiel 2 entwickeln wollen, kann als Anhaltspunkt für die Erstellung der Lohnliste von kleinen Geschäften dienen.

Beispiel 2:

In einem kleinen Geschäft sind fünf Angestellte beschäftigt. Ihre Namen und ihre Stundenverdienste sind der nachstehenden Tabelle zu entnehmen.

Name	*Stundenverdienst*
Adolf Dziadul	7.75
Stefan Hagen	8.50
Paul Friedens	8.50
Klaus Becker	6.00
Peter Fliege	6.00

Es ist ein Programm zu schreiben, das den wöchentlichen Verdienst der Arbeitnehmer errechnet. Zur Eingabe gelangen die wöchentlichen Arbeitsstunden jedes Arbeitnehmers. Es sollen bestimmt werden:

a) Wöchentlicher Bruttoverdienst jedes Arbeitnehmers
b) Wöchentlicher Sozialversicherungsbetrag jedes Arbeitnehmers (6.7% des Bruttoverdienstes)

Anmerkung: Diese Annahmen entsprechen nicht der deutschen Arbeitsgesetzgebung.

Wenden wir uns nun der Entwicklung der Lösung zu. Die Namen und die Stundenverdienste wollen wir in zwei Bereichen, die wir B$(J) bzw. A(J) mit J=1,2,3,4,5 nennen wollen, festhalten. Leider können wir die Daten nicht in einen einzigen zweidimensionalen Bereich hineinstellen, da die Namen Zeichenketten darstellen, wohingegen die Stundenverdienste numerischer Natur sind. Erinnern wir uns: BASIC gestattet nicht die Hineinstellung der beiden Datenarten in den gleichen Bereich. Der erste Programmteil befaßt sich mit der Zuweisung der Werte zu den Bereichselementen der beiden Bereiche. Die nächsten Schritte bestehen darin, daß das Programm nacheinander die Namen der Arbeitnehmer ausgibt und nach ihren wöchentlichen Arbeitsstunden fragt. Die eingegebenen Werte werden in den Bereichselementen des Bereiches C(J) mit J=1,2,3,4,5 gespeichert. Im Anschluß an die Eingabe werden die Bruttoarbeitsverdienste nach der Formel

```
D(J) = A(J) * C(J)
```

errechnet, d.h. sie sind gleich dem Produkt aus den wöchentlichen Arbeitsstunden und dem Stundenverdienst. Wie bereits aus der Formel ersichtlich, werden die Bruttoverdienste in den Variablen (Bereichselementen) des Bereiches, den wir mit D(J) bezeichnet haben, gespeichert. Nun kann die Berechnung des Sozialversicherungsbeitrages vorgenommen werden; diese Beiträge wollen wir im Bereich E(J) mit J=1,2,3,4,5 speichern. Zur Berechnung ziehen wir die Formel

```
E(J) = 0.067 * D(J)
```

heran. Nach Durchführung aller Berechnungen lassen wir die Resultate auf dem Bildschirm ausgeben. Unter diesen Überlegungen nimmt unser Programm die Gestalt an, die die Auflistung in Abb. 4.4 zeigt.

```
10    DIM A(5),B$(5),C(5),D(5),E(5)
20    FOR J=1 TO 5
30      READ B$(J),A(J)
40    NEXT J
50    FOR J=1 TO 5
60      PRINT "EINGABE DER ARBEITSSTUNDEN VON",B$(J)
70      INPUT C(J)
80      D(J) = A(J) * C(J)
90      E(J) = 0.067 * D(J)
100   NEXT J
110   PRINT "ARBEITNEHMER","BRUTTOVERD.","SOZIALVERS."
120   FOR J=1 TO 5
130     PRINT B$(J),D(J),E(J)
140   NEXT J
200   DATA  ADOLF DZIADUL,7.75, STEFAN HAGEN,8.50
210   DATA  PAUL FRIEDENS,8.50, KLAUS BECKER,6.00
220   DATA  PETER FLIEGE,6.00
1000  END
```

Abb. 4.4 Berechnung von Verdiensten

Bei manchen Anwendungen will man die in den DATA-Statements enthaltenen Daten mehr als einmal lesen. Um das zu bewerkstelligen, muß der Zeiger mittels des RESTORE-Statements wieder auf seinen Ausgangspunkt zurückgesetzt werden. Hinsichtlich dieser Technik betrachte man das Programm in Abb. 4.5.

```
10 DATA  2.3, 5.7, 4.5, 7.3
20 READ  A,B
30 RESTORE
40 READ  C,D
50 END
```

Abb. 4.5 Zurücksetzung des Zeigers

Durch das READ-Statement auf der Zeile mit der Zeilennummer 20 wird A auf 2.3 und B auf 5.7 gesetzt. Durch das RESTORE-Statement auf der Zeile mit der Zeilennummer 30 wird der Zeiger zurück auf das erste Datenelement, nämlich auf 2.3, geführt. Somit wird durch das nachfolgende READ-Statement (Zeile 40) der Variablen C der Wert 2.3 und der Variablen D der Wert 5.7 zugewiesen. Ohne das RESTORE-Statement würde C auf 4.5 und D auf 7.3 gesetzt.

Bei Verwendung der DATA- und READ-Statements werden mitunter zwei Fehler gemacht. Der erste Fehler zeigt sich darin, daß man durch das Programm mehr Daten lesen lassen will, als in den DATA-Statements aufgeführt sind. Hierzu betrachte man das Programm in Abb. 4.6.

```
10 DATA  1,2,3,4
10 FOR  J=1 TO 5
10   READ A(J)
10 NEXT J
10 END
```

Abb. 4.6 Fehler beim Lesen von Daten

Das in Abb. 4.6 aufgelistete Programm will fünf Datenelemente lesen, im DATA-Statement sind aber nur vier enthalten. In diesem Fall wird die Fehlernachricht

```
Out of data in 30
```
(Keine Daten mehr in Zeile 30)

Ein zweiter, häufig auftretender Fehler besteht darin, daß man eine Zeichenkettenkonstante einer Variablen des numerischen Typs zuweisen will und umgekehrt. Ein solcher Versuch führt zu einer Fehlernachricht, die auf die verwendeten ungleichen Datenarten hinweist.

Aufgabengruppe 10

Jedes der folgenden Programme weist Werte den Variablen eines Bereiches, d.h. den Bereichselementen, zu. Diese Werte sind zu ermitteln!

1.
```
10  DIM A(10)
20  FOR  J=1 TO 10
30    READ A(J)
40  NEXT J
50  DATA  2,4,6,8,10,12,14,16,18,20
100 END
```

2.
```
10  DIM A(3),B(3)
20  FOR  J=0 TO 3
30    READ A(J),B(J)
40  NEXT J
50  DATA  1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9
100 END
```

3.
```
10  DIM A(3),B$(3)
20  FOR  J=0 TO 3
30    READ A(J)
40  NEXT J
50  FOR  J=0 TO 3
60    READ B$(J)
70  NEXT J
80  DATA  1,2,3,4, A,B,C,D
100 END
```

4.
```
10  DIM A(3),B$(3)
20  READ A(0),B(0)
30  READ A(1),B(1)
40  RESTORE
50  READ A(2),B(2)
60  READ A(3),B(3)
70  DATA  1,2,3,4,5,6,7,8
100 END
```

5.
```
10  DIM A(3,4)
20  FOR  I=1 TO 3
30    FOR  J=1 TO 4
40      READ A(I,J)
50    NEXT J
60  NEXT I
70  DATA  1,2,3,4,5,6,7,8,9,10,11,12
100 END
```

6.
```
10  DIM A(3,4)
20  FOR  J=1 TO 4
30    FOR  I=1 TO 3
40      READ A(I,J)
50    NEXT I
60  NEXT J
70  DATA  1,2,3,4,5,6,7,8,9,10,11,12
100 END
```

Jedes der folgenden Programme enthält einen Fehler; dieser ist zu suchen!

7.
```
10  DIM A(5)
20  FOR  J=1 TO 5
30    READ A(J)
40  NEXT J
50  DATA  1,2,3,4
100 END
```

8.
```
10  DIM A(5)
20  FOR  J=1 TO 5
30    READ A(J)
40  NEXT J
50  DATA  1,A,2,B
100 END
```

9. Im Staate X gilt für Arbeitnehmer mit wöchentlicher Verdienstabrechnung die folgende Lohnsteuertabelle:

Wochenverdienst (brutto)	*Einzubehaltende Steuer*
200.00 bis 209.99	29.10
210.00 bis 219.99	31.20
220.00 bis 229.99	33.80
230.00 bis 239.99	36.40
240.00 bis 249.99	39.00
250.00 bis 259.99	41.60
260.00 bis 269.99	44.20
270.00 bis 279.99	46.80
280.00 bis 289,99	49.40
290.00 bis 299.99	52.10
300.00 bis 309.99	55.10
310.00 bis 319.99	58.10
320.00 bis 329.99	61.10
330.00 bis 339.99	64.10
340.00 bis 349.99	67.10

Das Programm von Abb. 4.4 ist so zu erweitern, daß zusätzlich auch die einzubehaltende Lohnsteuer ermittelt und der Nettoverdienst errechnet wird, d.h. vom Bruttoverdienst sind Sozialversicherungsbeitrag und Lohnsteuer zu subtrahieren.

10. Vom Nationalen Wetterdienst eines Staates wird von jedem Tag eine Temperaturtabelle veröffentlicht; die einzelnen Temperaturen werden dabei für jede volle Stunde angegeben. Es ist ein Programm zu schreiben, das die Durchschnittstemperatur eines Tages errechnet. Die Tagestemperaturen sind DATA-Statements zu entnehmen. Dem Benutzer soll weiterhin die Möglichkeit gegeben werden, die Temperatur abzufragen, die zu einer bestimmten Stunde des Tages herrschte.

Uhrzeit	*Temperatur in Grad C*	*Uhrzeit*	*Temperatur in Grad C*
0.00	10	12.00	38
1.00	10	13.00	39
2.00	9	14.00	40
3.00	9	15.00	40
4.00	8	16.00	42
5.00	11	17.00	38
6.00	15	18.00	33
7.00	18	19.00	27
8.00	20	20.00	22
9.00	25	21.00	18
10.00	31	22.00	15
11.00	35	23.00	12

Antworten zu den Testübungen

4.2.1
```
10 DATA  5.1, 4.7, 5.8, 3.2, 7.9, 6.9
20 FOR  J=1 TO 6
30    READ A(J)
40 NEXT J
50 END
```

4.2.2 A = 10 und B$ = „GERAET"

4.3 Die Formatierung von Ausgaben

In diesem Abschnitt wollen wir über die verschiedenen Möglichkeiten sprechen, die zur Formatierung von Ausgaben auf dem Bildschirm bzw. über den Drucker zur Verfügung stehen. Das BASIC des IBM Personalcomputers ist nämlich ziemlich flexibel hinsichtlich der Form, in der der Benutzer seine Ausgaben gestalten kann. Man kann u.a. bezüglich der Ausgaben folgende Fakten beeinflussen:

- Größe der Buchstaben auf dem Bildschirm
- Plazierung der auszugebenden Zeichen auf den Zeilen
- Genauigkeit der auszugebenden Ergebnisse von Berechnungen

usw. Wir wollen bei den Besprechungen stets bei den Erkenntnissen beginnen, die wir uns bei früheren Behandlungen der Ausgabe bereits angeeignet hatten.

4.3.1 Semikolons in PRINT-Statements

Beim IBM Personalcomputer kann der Bildschirm mit verschiedenen Zeilenlängen, nämlich 40 oder 80, gebraucht werden; die Einrichtung der Zeilenlängen geschieht mit dem Statement WIDTH. Entsprechend der gesetzten Zeilenlänge können nun 40 oder 80 Zeichen pro Zeile ausgegeben werden. Die Zeilen selbst sind dabei in sogenannte Ausgabezonen unterteilt; jede Ausgabezone umfaßt 14 Stellen. Um die Ausgabe am Anfang der nächsten Ausgabezone beginnen zu lassen, muß ein Komma zwischen die auszugebenden Datenelemente eingeschoben werden.

Bei vielen Anwendungen will man nun mehr Spalten ausgeben lassen als Ausgabezonen zur Verfügung stehen. Es kann auch sein, daß wir die Ausgabe als viel übersichtlicher empfinden, wenn die Ausgabespalten nicht so breit wie die Ausgabezonen sind. Um unnötigen Zwischenraum zwischen den auszugebenden Werten zu vermeiden, muß man die entsprechenden Datenelemente im PRINT-Statement durch ein Semikolon voneinander trennen. Man betrachte dazu die folgende Anweisung:

```
10 PRINT "PERSÖN";"LICHER COMPUTER"
```

In diesem Fall kommt es zu der folgenden Ausgabe:

```
PERSÖNLICHER COMPUTER
```

Das Semikolon unterdrückt also jeden Zwischenraum zwischen der Anzeige der Zeichenkette PERSÖN und der Anzeige der Zeichenkette LICHER COMPUTER.

Hinsichtlich der Ausgabe von Zahlen erinnere man sich daran, daß positive Zahlen immer mit einem Leerzeichen beginnen, das an die Stelle des unterdrückten Pluszeichens tritt. Die Anzeige negativer Zahlen beginnt hingegen selbstverständlich mit dem Minuszeichen und nicht mit einem Leerzeichen. Man betrachte dazu beispielsweise das Statement:

```
20 PRINT "DER WERT VON A IST GLEICH";2.36
```

In diesem Fall wird auf dem Bildschirm die folgende Anzeige erscheinen:

```
DER WERT VON A IST GLEICH 2.36
```

Die Leerstelle zwischen dem H und der 2 rührt von dem Leerzeichen her, das als Bestandteil der Zahl 2.36 gilt. Andererseits führt das Statement

```
30 PRINT "DER WERT VON A IST GLEICH";-2.36
```

zu der folgenden Ausgabe:

```
DER WERT VON A IST GLEICH-2.36
```

Um einen Zwischenraum zwischen dem H und dem Minuszeichen zu bekommen, müssen wir ein Leerzeichen in die auszugebende Zeichenkette einschießen. Somit würde das entsprechende Statement wie folgt umzuformen sein:

```
30 PRINT "DER WERT VON A IST GLEICH ";-2.36
```

Testübung 4.3.1

Es ist ein Programm zu schreiben, das die Eingabe zweier Zahlen zuläßt. Das Programm soll sie anschließend in der Form einer Summe (mit Ergebnis) anzeigen, als beispielsweise bei Eingabe von 5 und 7 in der Form

```
5 + 7 = 12
```

Nach Vollendung der Ausführung eines PRINT-Statements unterstellt BASIC automatisch eine Betätigung der Eingabetaste, so daß der Cursor auf den Anfang der nächsten Zeile gestellt wird. Diese Cursorbewegung kann man unterdrücken, indem man das betreffende PRINT-Statement mit einem Semikolon abschließt. Die beiden Statements

```
40 PRINT "DER WERT VON A IST GLEICH";
50 PRINT 2.36
```

führen demzufolge ebenfalls zu der bereits bekannten Ausgabe:

```
DER WERT VON A IST GLEICH 2.36
```

Testübung 4.3.2

Es ist die Form der Ausgabe zu beschreiben, die sich aus dem folgenden Programm ergibt:

```
10 A=5: B=3: C=8
20 PRINT "DER WERT VON A BETRAEGT",A
30 PRINT "DER WERT VON B";
40 PRINT "BETRAEGT";B
50 PRINT "DER WERT VON C BETRAEGT";-C
60 END
```

Unsere obige Diskussion orientierte sich an der Ausgabe auf Bildschirme. Man kann die Semikolons jedoch auch in den LPRINT-Statements benutzen und damit die Ausgabe über Drucker steuern.

4.3.2 Horizontales Tabulieren

Man kann mit der Ausgabe eines Datenelementes auf jeder Stelle beginnen, also nicht nur am Anfang der Druckzonen. Um das zu erreichen, muß man sich der TAB-Angabe bedienen. Die Druckstellen sind fortlaufend von links nach rechts numeriert, beginnend bei 1 und endend bei 255. Eine logische Zeile kann nämlich bis zu 255 Stellen lang sein. Wenn, wie bei Bildschirmen, die physische Zeile kleiner als die logische Zeile ist, führt die Ausgabe einer überlangen logischen Zeile dazu, daß die überzähligen Zeichen der logischen Zeile auf die nächste physische Zeile gestellt werden. Natürlich wird eine logische Zeile korrekt auf eine physische Zeile gestellt, wenn der entsprechende Drucker eine genügend große Zeilenbreite besitzt. Die Angabe TAB(7) bedeutet, daß zur Druckstelle 7 übergegangen wird. Die TAB-Angabe wird immer in Verbindung mit dem PRINT-Statement gebraucht. So wird z. B. das PRINT-Statement

```
50 PRINT TAB(7) A
```

dazu führen, daß der Wert der Variablen A ab Stelle 7 ausgegeben wird. Man kann mehr als eine TAB-Angabe in ein PRINT-Statement hineinstellen. Die Anweisung

```
150 PRINT TAB(5) A; TAB(15) B
```

z. B. gibt den Wert der Variablen A ab Stelle 5 und den Wert der Variablen B ab Stelle 15 aus. Man beachte hierbei das Auftreten eines Semikolons zwischen den beiden TAB-Angaben.

Testübung 4.3.3

Es ist eine Anweisung niederzuschreiben, durch die der Wert von A ab Stelle 25 und der Wert von B sieben Stellen rechts davon ausgegeben wird.

Bei manchen Anwendungen wünscht man, eine bestimmte Anzahl von Leerzeichen zwischen zwei Datenelemente zu stellen (im Gegensatz zur TAB-Angabe, bei der die Ausgabe auf einer bestimmten Stelle begonnen werden soll). Das kann man durch die Heranziehung der SPC-Angabe bewerkstelligen, die ähnlich wie die TAB-Angabe funktioniert. Wenn beispielsweise

die Werte der Variablen A und B mit 5 zwischen ihnen erscheinenden Leerzeichen ausgegeben werden sollen, so kann man das folgende Statement niederschreiben:

```
110 PRINT  A; SPC(5) B
```

4.3.3 Formatieren von Zahlen

Das BASIC des IBM Personalcomputers weist ziemlich umfangreiche Vorkehrungen auf, durch die die Ausgabe numerischer Werte formatiert werden kann. Einige dieser Vorkehrungen, die bei der Ausgabe von Zahlen berücksichtigt werden können, seien hier nunmehr aufgezählt:

- Festlegung der Stellenanzahl für die Genauigkeit der Zahl (Runden)
- Ausrichtung der Ausgabe auf die Stellenwerte der Zahl (Einerstelle, Zehnerstelle, Hunderterstelle usw.)
- Festlegung der Positionierung eines einleitenden Währungszeichens (Dollarzeichen)
- Festlegung von Kommastellen zur besseren Lesbarkeit großer Zahlen (wie z.B. bei 1,000,000.87)
- Festlegung der Positionierung von Vorzeichen, d.h. von + bzw. –

Diese Möglichkeiten zur Formatierung der auszugebenden Zahlen können mittels des Statements PRINT mit der USING-Option angefordert werden. Grob gesagt, man teilt durch die Spezifizierung eines „Prototyps" dem Computer mit, wie die auszugebenden Zahlen aussehen sollen. Angenommen z.B., man will den Wert der Variablen A mit 4 Stellen links und 2 Stellen rechts vom Dezimalpunkt ausgeben lassen, so kann das mit Hilfe der Anweisung

```
10 PRINT USING "####.##"; A
```

erfolgen. Im Prototyp der USING-Option steht jedes der Sonderzeichen # für eine Ziffer und der Punkt steht für den Dezimalpunkt. Wenn also z.B. die Variable A den Wert 5432.381 besitzt, so wird durch diese Angabe der Wert auf zwei Stellen nach dem Dezimalpunkt gerundet und zu 5432.38 ausgegeben. Wenn andererseits der Wert von A gleich 932.547 ist, so wird dementsprechend 932.55 ausgegeben. In diesem Fall geht der Ziffernfolge vor dem Dezimalpunkt ein Leerzeichen voraus, weil das Format mit vier Ziffern vor dem Dezimalpunkt spezifiziert ist. Diese Art der Ausgabe ist besonders bei der Ausrichtung von Zahlentabellen sinnvoll und nützlich; das folgende Beispiel soll das vor Augen führen:

```
  367.1
 1567.2
29573.3
    2.4
```

Diese Zahlentabelle kann bei Benutzung des Programmes ausgegeben werden, das in der Abb.4.7 aufgelistet ist.

Testübung 4.3.4

Es ist eine Anweisung niederzuschreiben, mit der die Zahl 456.75387 auf zwei Dezimalstellen gerundet ausgegeben werden kann.

```
10  DATA  367.1, 1567.2, 29573.3, 2.4
20  FOR  J=1 TO 4
30    READ A(J)
40    PRINT USING "#####.#"; A(J)
50  NEXT J
60  END
```

Abb. 4.7 Ausgabe einer Zahlentabelle

Man kann ein einziges Statement PRINT USING dazu benutzen, mehrere Zahlen auf einer Zeile auszugeben. Beispielsweise werden durch die Anweisung

```
10 PRINT USING "##.##"; A,B,C
```

die Werte der Variablen A, B und C auf die gleiche Zeile ausgegeben, alle im Format ##.##. Es erscheint dabei nur ein Leerzeichen zwischen den einzelnen Zahlen. Zusätzliche Leerzeichen müssen durch gesonderte # im Prototyp der USING-Klausel spezifiziert werden. Wenn man auf eine Zeile Zahlen in zwei verschiedenen Formaten ausgeben lassen will, so muß man zwei verschiedene PRINT-Statements mit USING-Optionen niederschreiben. Das erste muß dabei mit einem Semikolon enden, auf diese Weise anzeigend, daß die Ausgabe auf derselben Zeile fortgesetzt werden soll.

Wenn man versucht, eine Zahl auszugeben, deren Wert den Wertebereich des unterstellten Prototyps übersteigt, so wird die Zahl mit einem vorausgehenden Prozentzeichen angezeigt. Man betrachte z. B. das nachfolgende Statement:

```
10 PRINT USING "###"; A
```

Wenn der Wert von A gleich 5000 ist, so kommt es zu folgender Ausgabe:

```
%5000
```

Testübung 4.3.5

Es ist ein Programm zu schreiben, das die Werte 2^J mit J = 1,2,3,···,15 berechnet und anzeigt. Die ausgegebenen Werte sollen in geeigneter Weise rechtsbündig ausgerichtet sein, d. h. ihre Einerstellen sollen untereinander stehen.

Man kann den Computer auch veranlassen, ein Währungszeichen (Dollarzeichen) zu einer auszugebenden Zahl hinzuzufügen. Die folgenden zwei Anweisungen illustrieren dieses Vorgehen:

```
10 PRINT "$####.##"; A
20 PRINT "$$####.##"; A
```

Angenommen, der Wert von A betrage 34.78. Die Statements in den Zeilen mit den Zeilennummern 10 und 20 führen dann zu der folgenden Ausgabe:

```
$  34.78
 $34.78
```

Man beachte hier den Unterschied der beiden von den genannten Anweisungen produzierten Ausgaben. Das einzelne Dollarzeichen in der 5. Stelle links vom Dezimalpunkt beim Prototyp erzeugt bei der Ausgabe ebenfalls ein Dollarzeichen, und zwar in der Stelle, die beim Prototyp vorgesehen ist. Die Aufeinanderfolge von zwei Dollarzeichen ($$ beim Prototyp des Statements in der Zeile mit der Zeilennummer 20) bedeutet dagegen ein sogenanntes „gleitendes Dollarzeichen". Hier wird das Dollarzeichen auf der Stelle ausgegeben, die der Zahl unmittelbar vorausgeht. Zwischen Zahl und Dollarzeichen verbleibt dabei keine Leerstelle.

Beispiel 1:

Nachstehende Schecks wurden von einer in den USA weilenden Familie im Verlauf des Monats März ausgeschrieben:

```
15.32, 387.00, 57.98, 3.47, 15.88
```

Die Schecks wurden stets in Dollar ausgestellt. Die Liste der ausgeschriebenen Schecks soll auf dem Bildschirm in Form einer Zahlentabelle ausgegeben werden. Die Beträge sollen dabei entsprechend ausgerichtet sein, d.h. die Dezimalpunkte sollen untereinander stehen. Die Scheckbeträge sollen außerdem addiert werden. Der sich ergebende Gesamtbetrag ist ebenfalls anzuzeigen.

Bei der Lösung des Problems gehen wir wie folgt vor: Zunächst werden die Scheckbeträge nacheinander in einen Bereich A(J) mit J = 1,2,3,4,5 eingelesen; die eingelesenen Beträge werden sofort auf die Summe addiert, die in der Variablen B gebildet wird. Im Anschluß an die Addition werden die Scheckbeträge ausgegeben. Nach Beendigung der Schleife wird die Summe, d.h. der akkumulierte Wert der Variablen B ausgegeben. Das unter diesen Überlegungen entstandene Programm ist in der Abb. 4.8 aufgelistet.

```
10 DATA  15.32, 387.00, 57.98, 3.47, 15.88
20 FOR  J=1 TO 5
30   READ A(J)
40   LET  B= B + A(J)
50   PRINT USING "$####.##"; A(J)
60 NEXT J
70 PRINT "_______"
80 PRINT USING "$####.##"; B
85 PRINT "========"
90 END
```

Abb. 4.8 Programm zur Ausgabe einer Scheckliste

Die Programmzeilen mit den Zeilennummern 70 und 85 enthalten Statements, die die Ausgabe übersichtlicher gestalten. Die auf dem Bildschirm ausgegebene Tabelle ist in der Abb. 4.9 aufgeführt.

```
$ 15.32
$387.00
$ 57.98
$  3.47
$ 15.88
_______
$479.65
=======
```

Abb. 4.9 Ausgabe des Programmes von Abb. 4.8

Wenn die Beträge auf DM gelautet hätten, so könnten wir die Währungseinheit ebenfalls in den Prototyp der USING-Option aufnehmen, allerdings nur ein einziges Mal. Von der Möglichkeit des Gleitens kann hier also nicht Gebrauch gemacht werden. Ein auf DM abgeändertes Programm und die von ihm hervorgerufene Ausgabe ist in der Abb. 4.10 zu finden.

```
          P R O G R A M M                          |  A U S G A B E
==========================================         |  =============
                                                   |
10 DATA  15.32, 387.00, 57.98, 3.47, 15.88         |    DM  15.32
20 FOR  J=1 TO 5                                   |    DM 387.00
30   READ A(J)                                     |    DM  57.98
40   LET  B= B + A(J)                              |    DM   3.47
50   PRINT USING "DM ###.##"; A(J)                 |    DM  15.88
60 NEXT J                                          |    ___________
70 PRINT "__________"                              |    DM 479.65
80 PRINT USING "DM ###.##"; B                      |    =========
85 PRINT "=========="                              |
90 END                                             |
```

Abb. 4.10 Abgeändertes Programm von Abb. 4.8 und seine Ausgabe

Das Statement PRINT mit der Option USING weist noch eine Reihe anderer Variationen auf. Wenn man z. B. Kommas in große Zahlen einfügen will, um sie besser lesbar zu machen, so braucht man nur in dem entsprechenden Prototyp der USING-Option an geeigneten Stellen links vom Dezimalpunkt Kommas anzugeben. Man sehe sich hierzu das folgende Statement an:

```
10 PRINT USING "###,###"; A
```

Wenn die Variable A den Wert 123756 aufweist, so wird in diesem Fall die Zeichenfolge

```
123,756
```

ausgegeben.

Das Statement PRINT mit der USING-Option kann auch zur Positionierung der Vorzeichen bei den auszugebenden Zahlen verwendet werden. Ein Pluszeichen zu Beginn oder Ende des Prototyps verursacht, daß das entsprechende Vorzeichen in die angegebene Position gestellt wird. Man betrachte hierzu das nachfolgende Statement:

```
10 PRINT USING "+####.###"; A
```

Angenommen, A besitze den Wert −458.73. In diesem Fall kommt es zur Ausgabe der Zeichenfolge

```
- 458.730
```

Wir wollen uns noch ein zweites Beispiel ansehen. Die Anweisung lautet:

```
10 PRINT USING "+###.##"; A
```

Wenn nun A den Wert 0.05873 aufweist, dann kommt es jetzt zur Ausgabe der Zeichenfolge

```
+   .06
```

Wir wollen die Diskussion mit einem äußerst wichtigen Hinweis abschließen. Unsere Ausführungen betrafen nämlich nur die Ausgabe auf den Bildschirm. Alle Möglichkeiten, die wir erwähnt haben, können selbstverständlich auch auf die Ausgabe über den Drucker angewendet werden. Natürlich müssen wir hierfür das Statement LPRINT mit der USING-Option niederschreiben. Es sei ferner darauf hingewiesen, daß die physische Zeilenlänge der Drucker größer ist und daher mehr Zeichen auf einer Zeile ausgegeben werden können als auf dem Bildschirm. Insbesondere gibt es mehr 14-stellige Druckzonen (wieviel es gibt, hängt vom angeschlossenen Druckertyp ab). Als weitere Folgerung ist zu bemerken, daß TAB mit einer höheren Stellennummer benutzt werden kann.

Man erinnere sich, daß BASIC zwei verschiedene Darstellungsarten für Zahlen benutzt, einmal die gewöhnliche Dezimaldarstellung und andererseits die wissenschaftliche Darstellung (Darstellung mit einem Potenzfaktor). Man kann in Prototypen die Zeichenfolge ^^^^ verwenden, um das Potenzfeld der wissenschaftlichen Darstellung zu markieren. Damit kann man bei der Ausgabe die wissenschaftliche Darstellung einer Zahl erzwingen. Ein Beispiel soll dieses angeschnittene Thema abrunden. Dazu nehmen wir an, daß eine Zahl in wissenschaftlicher Darstellung mit zwei Stellen links vom Dezimalpunkt und zwei Stellen rechts vom Dezimalpunkt ausgegeben werden soll. Der Prototyp in der USING-Option muß dann wie folgt lauten:

```
"##.##^^^^"
```

Bei dieser Formatvorgabe würde die Zahl 100 wie folgt ausgegeben:

```
10.00E+01
```

4.3.4 Andere Varianten des PRINT-Statements mit der Option USING

Das Statement PRINT mit der USING-Option bietet dem Benutzer noch eine Reihe weiterer Möglichkeiten. Unsere Beispiele haben wir bisher aus dem Rechnungswesen ausgewählt. Die Möglichkeiten, mit denen wir uns jetzt befassen wollen, finden ihre Anwendung vorzugsweise im Finanzwesen und dort vor allem bei der Aufbereitung von Belegen und Dokumenten.

Wenn der Prototyp in der USING-Option mit der Zeichenfolge ** beginnt, werden bei der Ausgabe die unbenutzten Ziffernpositionen einer Zahl mit Sternzeichen aufgefüllt. Man betrachte dazu beispielsweise das Statement:

```
10 PRINT  USING "**####.#"; A
```

Wenn A der Wert 34.86 zugewiesen ist, dann wird dieser Wert zu

```
**34.9
```

ausgegeben.

Man beachte, daß hier zwei Sternzeichen ausgegeben werden, da vier Ziffernpositionen links vom Dezimalpunkt im Prototyp vorgesehen sind; der Wert von A weist aber nur zwei Ziffern in

seinem ganzzahligen Anteil auf. Deswegen werden die restlichen zwei Ziffernpositionen mit Sternzeichen aufgefüllt.

Man kann im Prototyp ** und $ kombinieren. Man sollte, als Anregung unsererseits gedacht, mit dieser Möglichkeit selbst experimentieren, da sie sich als besonders nützlich bei der Ausgabe von Dollarbeträgen in der Form

```
$*******387.98
```

erweist. Eine solche Form verhindert sinnvoll unerlaubte Änderungen von Beträgen beispielsweise bei Schecks (Schutzsternschreibung).

Wenn ein Minuszeichen an das Ende des Prototyps gestellt wird, so wird die entsprechende Zahl mit einem abschließenden Minuszeichen ausgegeben, falls sie negativ ist. Bei positiven Zahlen entfällt hingegen das Vorzeichen. Das Statement

```
10 PRINT USING "####.##-"; A
```

mit A = −57.89 führt somit zur Ausgabe von

```
57.89-
```

Ist jedoch A gleich 57.89, so sieht die Ausgabe wie folgt aus:

```
57.89
```

Diese Form der Ausgabe wird oft in der Buchführung bei der Erstellung von Berichten und Zusammenstellungen benutzt.

Aufgabengruppe 11

Es sind Programme zu schreiben, die zu den nachstehend beschriebenen Anzeigen auf dem Bildschirm führen. Die darunter bzw. darüber stehenden punktierten Zeilen sollen nicht angezeigt werden; sie dienen hier nur zur Orientierung hinsichtlich des Zeilenaufbaus der Ausgaben, insbesonders zur Verteilung der Leerstellen.

1.
```
DER WERT VON X BETRAEGT 5.378
.............................
```
(zur Orientierung)

2.
```
DER WERT VON X BETRAEGT     5.378
.................................
```
(zur Orientierung)

3.
```
DATUM  MENGE   E.-PREIS  GES.-PREIS  RABATT   NETTO
.................................................
```
(zur Orientierung)

4.
```
   6.753
  15.111
 111.850
   6.702
 -------
 xxx.xxx
 =======
```
(Errechnete Summe)

5.
```
$ 12.82
$117.58
$  5.87
$  0.99
$  0.99
$xxx.xx      (Errechnete Summe)
=======
```

6.
```
.......................................................      (zur Orientierung)
                                        DATUM: 18.03.1981

ZAHLUNG GEMAESS RECHNUNG DER FA. JAENECKE & CO.

BETRAG:                                *********$89,385.00
```

7.
```
  5,787
    387
127,486
 38,531
xxx,xxx      (Errechnete Summe)
=======
```

8.
```
 $385.41
-$17.85
$xxx.xx      (Errechnete Differenz)
=======
```

9. Es ist ein Programm zu schreiben, durch das eine beliebige positive Zahl auf die nächste ganze Zahl gerundet wird. Wenn z.B. die eingegebene Zahl gleich 11.7 ist, so muß die auszugebende gleich 12 sein; bei der Eingabe von 158.2 muß die Ausgabe von 158 erfolgen. Die zu rundende Zahl soll dabei mittels des INPUT-Statements eingegeben werden können.

10. Es ist ein Programm zu schreiben, das die Benutzung des Computers als Registrierkasse erlaubt. Die Kaufbeträge sollen dabei mittels INPUT-Statements eingegeben werden können. Der Kassierer soll selbst dem Computer mitteilen, wenn er keine Kaufbeträge mehr eintippen will. Nach Beendigung der Eingabe soll das Programm die Kaufbeträge, versehen mit DM-Vermerken, spaltengerecht, d.h. Dezimalpunkt unter Dezimalpunkt, ausgeben. Außerdem soll natürlich die Summe der Kaufbeträge ermittelt und ebenfalls ausgegeben werden. Zum Schluß soll schließlich noch die 14%-ige Mehrwertsteuer errechnet und zur Kaufsumme addiert werden. Nach der Ausgabe von Mehrwertsteuerbetrag und Rechnungsbetrag soll vom Programm nach dem gezahlten Betrag gefragt und daraus das auszuzahlende Wechselgeld bestimmt werden.

11. Die Ausgabe von Aufgabe 6. ist auf eine Zeilenlänge von 40 Stellen umzustellen.

Antworten zu den Testübungen

4.3.1
```
10 INPUT A,B
20 PRINT A;" +";B;" =";A+B
30 END
```

4.3.2
```
DER WERT VON A BETRAEGT     5
DER WERT VON A BETRAEGT 3
DER WERT VON A BETRAEGT-8
```

4.3.3
```
10 PRINT  TAB(25) A; TAB(32) B
```

4.3.4
```
10 PRINT USING ###.##; 456.7387
```

4.3.5
```
10 FOR J=1 TO 15
20   PRINT USING ######; 2^J
30 NEXT J
40 END
```

4.4 Computerspiele

Als eine der interessantesten Besonderheiten eines Computers erweist sich seine Fähigkeit, Vorfälle zu erzeugen, deren Erscheinungsbilder auf einer Zufallsbasis beruhen. Beispielsweise kann man den Computer instruieren, daß er „zwei Würfel wirft"; er produziert in diesem Fall ein Zufallspaar zweier ganzer Zahlen zwischen 1 und 6. Oder man kann den Computer anweisen, daß er „aus einem aus 52 Karten bestehenden Kartenpaket eine Karte auf Zufallsbasis herauszieht". Ebenso kann man den Computer programmieren, daß er auf Zufallsbasis eine zweistellige Zahl zwischen 1 und 99 generiert. Es lassen sich noch viele andere Beispiele nennen. Der Ursprung aller solcher zufälligen Auswahlen ist der sogenannte Zufallszahlengenerator, der Teil von BASIC ist. Damit bietet sich an, daß wir mit der Erklärung des Zufallszahlengenerators beginnen und wie wir uns seiner bedienen können. Anschließend wollen wir eine Reihe interessanter Anwendungen besprechen, in die computerunterstützte Anleitungen und Glücksspiele verwickelt sind.

Man kann Zufallszahlen dadurch erzeugen, daß man die Funktion RND von BASIC benutzt. Um zu erklären, wie diese Funktion arbeitet, wollen wir uns ein einfaches Programm (Abb. 4.11) ansehen.

```
10 FOR X = 1 TO 500
20   PRINT RND
30 NEXT X
40 END
```

Abb. 4.11 Benutzung des Zufallszahlengenerators von BASIC

Dieses Programm enthält eine Schleife, die 500 Zahlen ausgibt; jede Zahl wird dabei durch RND angesprochen. Alle Zahlen liegen dabei im Wertebereich von 0.000000 (inklusiv) bis 1.000000 (exklusiv). Immer, wenn RND aufgerufen wird (wie hier in der Zeile mit der Zeilennummer 20), erfolgt durch den Computer eine Zufallsauswahl einer Zahl aus dem genannten Wertebereich. Diese Zahl wird vom Programm ausgegeben.

Um eine bessere Vorstellung über das Thema zu gewinnen, über das wir reden, sollte man zuerst durch ein ähnliches Programm wie das von Abb. 4.11 einige Zufallszahlen erzeugen. Freilich sind 500 Zahlen zuviel, um sie mit einem Blick überschauen zu können, es sei denn, man hat einen Drucker am Computer angeschlossen. Wenn als Ausgabemedium nur der Bildschirm zur Verfügung steht, sollte man sich auf die Ausgabe von 25 Zeilen beschränken. Pro Zeile könnte man vier Zufallszahlen ausgeben, eine in jeder Druckzone. Einen Ausschnitt aus einer solchen Ausgabe finden wir in der Abb. 4.12.

```
.245121        .305003        .311866        .515163
.984546        .901159        .727313        6.83401E-03
.896609        .660212        .554489        .818675
.583931        .448163        .86774         .0331043
.137119        .226544        .215274        .876763

  . . .
  . . .
  . . .
```

Abb. 4.12 Erzeugte Zufallszahlen

Wodurch kommt es nun, daß diese Zahlen „zufällige" Zahlen sind? Das vom Computer zu ihrer Auswahl benutzte Verfahren arbeitet nämlich „unbeeinflußt"; für alle Zahlen besteht die gleiche Wahrscheinlichkeit, ausgewählt zu werden. Darüberhinaus verdient noch folgendes festgehalten zu werden. Wenn man eine sehr große Anzahl Zufallszahlen generieren läßt, dann wird man feststellen, daß nahezu 10% der Zahlen zwischen 0 und 0.1 liegen, nahezu 50% zwischen 0.5 und 1 usw. In gewisser Weise liefert also der Zufallszahlengenerator eine gleichförmige Verteilung der Zahlen zwischen 0 und 1 ab.

Testübung 4.4.1

Angenommen, RND wird zur Generierung von 1 000 Zufallszahlen benutzt. Von wievielen dieser Zahlen kann man erwarten, daß sie zwischen 0.6 und 0.9 liegen?

Der Zufallszahlengenerator wird von einer sogenannten „Keimzahl" oder „Ausgangszahl" gesteuert. Von diesem Anfangswert hängt die Folge der generierten Zufallszahlen ab, d. h. wenn einmal ein bestimmter Anfangswert festgelegt ist, ist die Folge der Zufallszahlen unveränderlich, also fest. Dadurch aber würden Glücksspiele auf Computerbasis ziemlich uninteressant, da stets die gleiche Spielfolge eintreten würde. Davor kann man sich schützen, indem man die Keimzahl mittels der Anweisung RANDOMIZE ändert. Eine Anweisung der Form

```
10 RANDOMIZE
```

verursacht, daß der Computer die Meldung

```
Random Number Seed (-32768 to 32767)?
```
(Zufallskeimzahl (−32768 bis 32767)?)

Auf diese Meldung hin muß man mit einer Zahl antworten, die in dem angezeigten Intervall liegt. Angenommen, wir wählen z. B. 129. Der Computer wird dann den Zufallszahlengenerator mit der Keimzahl 129 aufsetzen. Die Folge der Zufallszahlen wird damit entsprechend dieser Keimzahl erzeugt. Eine andere Möglichkeit zur Festlegung einer Keimzahl besteht darin, daß man ein Statement der Form

```
20 RANDOMIZE 129
```

formuliert. Diese Anweisung setzt die Keimzahl auf 129, ohne daß eine Rückfrage erfolgt. Im Kapitel 8 werden wir sehen, wie wir den internen Zeitgeber des Computers benutzen können,

uns eine Keimzahl zu liefern. Diese Methode der Vorgabe einer Keimzahl kann niemand steuern und kontrollieren.

Die Funktion RND generiert bekanntlich Zufallszahlen, die zwischen 0 und 1 liegen. Bei vielen Anwendungen benötigen wir jedoch zufällig ausgewählte ganze Zahlen, die zu einem bestimmten Wertebereich gehören. Nehmen wir z. B. einmal an, daß wir ganze Zufallszahlen aus der Zahlenfolge 1,2,3,4,5,6 auswählen wollen. Wenn wir RND mit 6 multiplizieren, also 6*RND bilden, bekommen wir Werte zwischen 0.00000 und 5.99999. Addieren wir nun eine 1 auf diese Zufallszahlen, bilden als 6*RND + 1, so transformieren wir diese Zufallszahlen auf den Wertebereich von 1.00000 bis 6.99999. Um schließlich die gewünschten ganzen Zahlen aus [1,6] zu erhalten, müssen wir die Nachkommastellen von 6*RND + 1 „abhacken". Um das zu erreichen, benutzen wir die Funktion INT. Sei x irgendeine reelle Zahl, dann ist INT(x) die größte ganze Zahl, die kleiner als oder gleich x ist. Die nachfolgenden Beispiele mögen das soeben besprochene untermauern:

```
INT(5.23) = 5,  INT(7.99) = 7,   INT(100.01) = 100
```

Bei Verwendung der Funktion INT mit negativen Zahlen als Argument muß man achtgeben. Die von uns gegebene Definition ist korrekt, aber ohne genaues Überdenken der Sachlage macht man leicht Fehler. So ist z. B.

```
INT(-7.4) = -8,
```

da die größte ganze Zahl, die kleiner als oder gleich −7.4 ist, nun einmal −8 ist. (Man zeichne notfalls eine Zahlengerade und trage auf ihr −7.4 und −8 auf, dann erkennt man die Zusammenhänge recht anschaulich.) Wir wollen nun zu unserem Zufallszahlenproblem zurückkehren. Um den gebrochenen Anteil von 6*RND + 1 zu entfernen, heißt es, den Ausdruck

```
INT(6*RND+1)
```

zu bilden. Dieser Ausdruck liefert uns die gewünschte Zufallszahl im Bereich von 1 bis 6. In gleicher Weise würde uns z. B. der Ausdruck

```
INT(100*RND+1)
```

eine ganze Zufallszahl aus der Folge 1,2,3,· · ·,99,100 liefern.

Testübung 4.4.2

Es sind ganze Zufallszahlen 0 oder 1 zu generieren. (Das entspricht dem Werfen einer Münze; beim Computer könnte die Vorderseite der Münze der 0 entsprechen, die Rückseite der 1.) Man lasse ein Programm ablaufen, das 50-mal das Hochwerfen einer Münze simuliert. Wieviel Mal ist zu erwarten, daß nach dem Wurf die Vorderseite (Avers) der Münze oben liegt?

Beispiel 1:

Es ist ein Programm zu schreiben, das den Computer in ein Paar Spielwürfel verwandelt. Das Programm soll sowohl die gewürfelte Zahl jedes einzelnen Würfels als auch die Summe der gewürfelten Augen ausgeben.

Für die Lösung wollen wir festlegen, daß die gewürfelten Augen des ersten Würfels in der Variablen X, die des zweiten Würfels in der Variablen Y festgehalten werden. Im Programm werden die Werte von X und Y errechnet und anschließend ausgegeben; dasselbe geschieht mit der Summe der gewürfelten Augen (Gewinnsumme). Das unter diesen Voraussetzungen entwickelte Programm ist in der Abb. 4.13 abgebildet.

```
5   RANDOMIZE
10  CLS
20  LET  X = INT(6*RND+1)
30  LET  Y = INT(6*RND+1)
40  PRINT  "MEINE DAMEN UND HERREN, BITTE WETTEN!"
50  INPUT  "SIND ALLE WETTEN ERFOLGT (J/N)"; A$
60  IF  A$ = "J"  THEN 100  ELSE 40
100 PRINT  "ES WURDEN GEWUERFELT:     "; X,Y
110 PRINT  "DIE GEWINNSUMME BETRAEGT: "; X+Y
120 INPUT  "EIN NEUES SPIEL (J/N)"; B$
130 IF  B$ = "J"  THEN 10  ELSE 200
200 PRINT  "DAS KASINO IST GESCHLOSSEN --- LEIDER!"
210 END
```

Abb. 4.13 Programm zur Simulierung von Spielwürfeln

Zu diesem Programm wäre noch zu bemerken, daß wir hier reichlich von einer durch den Computer initiierten Konversation über den Bildschirm Gebrauch machen. Man beachte ebenfalls, daß die Statements auf den Zeilen mit den Zeilennummern 120 und 130 dem Spieler Gelegenheit geben, selbst zu entscheiden, wieviele Male er zu würfeln wünscht. Zum Schluß erwähnen wir nochmals die Verwendung der Funktion RANDOMIZE in der Zeile mit der Nummer 5. Durch sie wird eine Frage an den Spieler gestellt, die ihm erlaubt, die Keimzahl nach seinem Ermessen selbst vorzugeben; darüber haben wir bereits im einzelnen gesprochen.

Testübung 4.4.3

Es ist ein Programm zu schreiben, durch das der Wurf einer Münze simuliert wird. Das Ergebnis des Wurfes soll dabei beeinflußt werden; zu einem Drittel nämlich soll die Münzvorderseite (Avers) oben zu liegen kommen und zu zwei Dritteln die Rückseite.

Man kann den Realismus bei einem Spielprogramm erhöhen, wenn man den Computer die getätigten Wetten verfolgen läßt, wie es im nachfolgenden Beispiel 2 geschieht.

Beispiel 2:

Ein Roulette weist 38 verschiedene Positionen auf: 1 bis 36, 0 und 00. In unserem Programm wollen wir diese Positionen durch die Zahlen 1 bis 38 repräsentieren. Der Null des Rouletts soll die Zahl 37 entsprechen, der 00 die Zahl 38. Dem Zurruhekommen der Scheibe nach vorhergehendem Rotieren ist die Auswahl einer ganzen Zufallszahl zwischen 1 und 38 gleichzusetzen. Vor dem Beginn des Rotierens der Scheibe wird jeder Spieler nach seiner Wette gefragt, d. h. nach der Nummer, auf die er setzen will, und nach der Höhe seines Einsatzes. Das Programm ermittelt nach Feststellung der Gewinnzahl die Gewinner und die Verlierer. Die Auszahlung für einen Gewinn beträgt das 32-fache des Einsatzes eingesetzten Betrages. Jeder Spieler besitzt natürlich ein Spielkonto, das A(J) mit J = 1,2,3,4,5 genannt ist. Nach jedem Spiel werden die Konten auf den neuesten Stand gebracht und angezeigt. Wie beim Beispiel 1 wird vom Programm gefragt, ob ein neues Spiel gewünscht wird. Das unter diesen Voraussetzungen geschriebene Programm ist in der Abb. 4.14 aufgelistet.

```
5   RANDOMIZE
8   CLS
10  INPUT  "ANZAHL DER SPIELER"; N
20  DIM  A(5), B(5), C(5):     'höchstens 5 Spieler sind zugelassen
30  FOR  J=1 TO N:             'Ankauf von Chips zu Spielbeginn
40    PRINT  "SPIELER "; J
50    INPUT  "WIEVIELE CHIPS"; A(J)
60  NEXT J
100 PRINT  "MEINE DAMEN UND HERREN! IHRE EINSAETZE BITTE"
110 FOR  J=1 TO N:             'Plazierung der Spielerwetten
120   PRINT  "SPIELER "; J
130   INPUT  "NUMMER  BETRAG"; B(J),C(J):   'Eingabe der Wette
140 NEXT J
200 X = INT(38*RND+1):         'Rotation der Scheibe
220 PRINT  "DIE GEWINNZAHL IST"; X
300 'Ermittlung der Gewinne und Verluste
310 FOR  J=1 TO N
320   IF  X = B(J)  THEN 400  ELSE 330
330   A(J) = A(J) - C(J):      'Verlust des Spielers J
340   PRINT  "SPIELER"; J; " HAT VERLOREN"
350   GOTO 420
400   A(J) = A(J) + 32*C(J):  'Gewinn des Spielers J
410   PRINT  "SPIELER"; J; " HAT"; 32*C(J); " DM GEWONNEN"
420 NEXT J
421 PRINT
422 PRINT
430 PRINT  "SPIELSTAND"
440 PRINT
450 PRINT  "SPIELER", "CHIPS"
460 FOR  J=1 TO N
470   PRINT  J,A(J)
480 NEXT J
500 INPUT  "WOLLEN SIE EIN WEITERES SPIEL SPIELEN (J/N)"; S$
510 CLS
520 IF  S$ = "J"  THEN 100:   'Wiederholung des Spiels
530 PRINT  "DAS KASINO IST GESCHLOSSEN! LEIDER..."
600 END
```

Abb. 4.14 Roulett mit dem Computer

Man sollte unbedingt einige Runden mit dem Programm spielen. Das Programm erweist sich als recht instruktiv und bereitet vor allem auch Freude. Beachtenswert ist, daß man einen höheren Wetteinsatz tätigen kann als man Chips auf seinem Konto hat. Hier ist auch der Ansatzpunkt für eine Erweiterung des Programmes zu sehen. Wir wollen es der eigenen Initiative überlassen, eine Prüfung einzubauen, durch die festgestellt werden kann, ob ein Spieler noch genügend Chips besitzt, um einen evtl. Spielverlust ausgleichen zu können. Auch läßt sich u.a. durch entsprechende Statements dafür Sorge tragen, daß ein Spieler einen Kredit aufnehmen kann usw. Wir jedenfalls wollen in unserem nächsten Beispiel das Programm um die Wetten auf Geradzahligkeit (pair) und Ungeradzahligkeit (impair) ergänzen.

Bevor wir aber zum nächsten Beispiel übergehen, wollen wir uns mit einer weiteren Schwäche des Programms beschäftigen. Die Zeile mit der Nummer 5 enthält das RANDOMIZE-Statement. Dadurch kommt es vom Programm her zu einer Rückfrage nach der Keimzahl für den Zufallszahlengenerator. Diejenige Person, die die Keimzahl bestimmt, kontrolliert damit die Folge der Zufallszahlen und infolgedessen auch das Roulett. Wenn er die bei einer bestimm-

ten Keimzahl auftretende Folge der Zufallszahlen auswendig lernt, sind seine Gewinnchanchen ungleich höher als die seiner Mitspieler. Eine unbefriedigende Tatsache, die es auf möglichst simplen Wege auszumerzen gilt.

Der IBM Personalcomputer besitzt nun einen internen Zeitgeber, der immer dann aufgesetzt wird, wenn man den Computer einschaltet. Dieser Zeitgeber kommt einer Uhr gleich, die die Zeit in Stunden, Minuten und Sekunden mißt. Zu der augenblicklichen Zeit kann man mittels der Funktion TIME$ zugreifen. Mit näheren Einzelheiten über diese Zeitmessung werden wir uns im Kapitel 8 auseinandersetzen. Für den jetzigen Zeitpunkt wollen wir einfach einmal ein Ergebnis der zukünftigen Diskussion ausleihen. Die momentan vom Zeitgeber abzulesenden Sekunden können durch

```
VAL(RIGHT$(TIME$,2))
```

ausgedrückt werden. Diese Zahl nun können wir sinnvoll als Keimzahl für den Zufallszahlengenerator einsetzen. Es ist geradezu unwahrscheinlich, daß irgendjemand die exakte Zahl der Sekunden zu Spielbeginn nennen kann.

Beispiel 3:

Das ein Roulett simulierende Programm vom Beispiel 2 soll so erweitert werden, daß es auch Wetten auf gerade bzw. ungerade Zahlen zuläßt. Der Einsatz von einem Chip auf solche Gewinnchancen soll dabei zu einem Gewinn von einem Chip führen. Wir wollen wie bisher annehmen, daß ein Chip dem Betrag von einer DM gleichkommt.

Wir wollen zunächst einige Überlegungen hinsichtlich der Lösung anstellen. Es werden jetzt also drei Wettarten zugelassen:

- Wette auf eine bestimmte Zahl
- Wette auf eine beliebige gerade Zahl (Wette auf „pair")
- Wette auf eine beliebige ungerade Zahl (Wette auf „impair")

Es liegt nahe, daß wir für jede dieser drei Wettarten Unterprogramme vorsehen. Diese Unterprogramme sollen entscheiden, ob der Spieler J gewonnen oder verloren hat. Für jedes Unterprogramm soll die Variable X mit ihrem Wertevorrat von 1 bis einschließlich 38 die Zahl enthalten, „auf der die Scheibe nach der Rotation zum Stillstand gekommen ist". Im vorhergehenden Programm wurde die Wette des Spielers J in den entsprechenden Elementen von zwei Bereichen festgehalten und damit auch beschrieben: B(J) enthielt die Zahl, auf die der Spieler J wettete, und C(J) den gewetteten Betrag. Nun aber benötigen wir für die Beschreibung einer Wette einen dritten Bereich, den wir mit D bezeichnen wollen. Den Elementen dieses Bereiches wollen wir die folgenden Werte zuweisen:

- D(J)=1, wenn der Spieler J auf eine bestimmte Zahl wettet,
- D(J)=2, wenn der Spieler J auf Geradzahligkeit wettet,
- D(J)=3, wenn der Spieler J auf Ungeradzahligkeit wettet.

```
1000 'Wette auf Zahl
1010      IF  B(J) = X  THEN 1050  ELSE 1020
1020      PRINT  "SPIELER"; J; " HAT VERLOREN"
1030      A(J) = A(J) - C(J)
1040      GOTO 1070            'Ankauf von Chips zu Spielbeginn
1050      PRINT  "SPIELER"; J; " HAT"; 32*C(J); " DM GEWONNEN"
1060      A(J) = A(J) + 32*C(J)
1070 RETURN
```

Abb. 4.15 Unterprogramm zur Ermittlung der Gewinner von Wetten auf bestimmte Zahlen (Subroutine 1)

Im Falle, daß D(J) gleich 2 oder gleich 3 ist, wollen wir B(J) gleich Null setzen; C(J) enthält natürlich wie bisher den Wetteinsatz. Das Unterprogramm, das die Gewinner von Wetten auf Zahlen bestimmen soll, können wir durch geringfügige Änderungen des entsprechenden Programmteiles des bisherigen, in Abb. 4.14 dargestellten Programmes erhalten; es ist in der Abb. 4.15 aufgelistet.

Das zur Ermittlung der Gewinner von Wetten auf gerade Zahlen geschriebene Unterprogramm ist in der Abb. 4.16 dargestellt.

```
2000 'Wette auf gerade Zahl (Wette auf pair)
2010      LET  K = 0
2020      IF  X = 2*K  THEN 2070  ELSE 2020
2030      LET  K = K + 1:   IF  K >= 19  THEN 2040  ELSE 2020
2040      PRINT  "SPIELER"; J; " HAT VERLOREN"
2050      A(J) = A(J) - C(J)
2060      GOTO 2090
2070      PRINT  "SPIELER"; J; " HAT"; C(J); " DM GEWONNEN"
2080      A(J) = A(J) + C(J)
2090 RETURN
```

Abb. 4.16 Unterprogramm zur Ermittlung der Gewinner von Wetten auf gerade Zahlen, also Wetten auf pair (Subroutine 2)

Das zur Ermittlung der Gewinner von Wetten auf ungerade Zahlen abgefaßte Unterprogramm ist schließlich in der Abb. 4.17 dargestellt; es ähnelt in gewisser Hinsicht dem Unterprogramm von Abb. 4.16.

```
3000 'Wette auf ungerade Zahl (Wette auf impair)
3010      LET  K = 0
3020      IF  X = 2*K+1  THEN 3070  ELSE 3020
3030      LET  K = K + 1:   IF  K >= 19  THEN 3040  ELSE 3020
3040      PRINT  "SPIELER"; J; " HAT VERLOREN"
3050      A(J) = A(J) - C(J)
3060      GOTO 3090
3070      PRINT  "SPIELER"; J; " HAT"; C(J); " DM GEWONNEN"
3080      A(J) = A(J) + C(J)
3090 RETURN
```

Abb. 4.17 Unterprogramm zur Ermittlung der Gewinner von Wetten auf ungerade Zahlen, also Wetten auf impair (Subroutine 3)

Nachdem wir alle notwendigen Unterprogramme geschrieben haben, können wir diese mit dem Hauptteil, der nahezu gleich dem bisherigen Programm (siehe Abb. 4.14) ist, zusammenfügen. Der einzige wesentliche Unterschied besteht darin, daß wir für jeden Spieler festhalten müssen, welche Art von Wette er getätigt hat. Das auf diese Weise entstandene Gesamtprogramm findet man in der Abb. 4.18.

Zweifelsfrei ist bei diesem Programm erkennbar, wie hilfreich Subroutinen bei der Einrichtung und dem Aufbau eines Programmes sein können. Jede Subroutine ist leicht zu schreiben; ihre Abfassung erweist sich als einfach zu bewältigende Aufgabe, über die man nur noch wenig nachzudenken hat, wenn man das Gesamtprogramm betrachtet. Es kann deshalb jedem Programmierer nur angeraten werden, umfangreiche Programme in eine Anzahl von Subroutinen aufzubrechen. Es ist nicht nur leichter, ein in Unterprogrammen organisiertes Programm zu schreiben, sondern auch leichter, ein solches Programm auf seine Richtigkeit zu überprüfen

```
10   CLS
20   RANDOMIZE  VAL(RIGHT$(TIME$,2))
30   INPUT  "ANZAHL DER SPIELER"; N
40   DIM  A(5), B(5), C(5), D(5):   'höchstens 5 Spieler sind zugelassen
50   FOR  J=1 TO N:             'Ankauf von Cips zu Spielbeginn
60     PRINT  "SPIELER "; J
70     INPUT  "WIEVIELE CHIPS"; A(J)
80   NEXT J
90   PRINT  "MEINE DAMEN UND HERREN! IHRE EINSAETZE BITTE"
100  FOR  J=1 TO N:             'Plazierung der Spielerwetten
110    PRINT  "SPIELER "; J
120    PRINT  "WETTART:  1=ZAHL, 2=GERADE ZAHL, 3=UNGERADE ZAHL"
130    INPUT  "WETTART (1, 2 ODER 3)"; D(J)
140    IF  D(J) = 1  THEN 170  ELSE 150
150    INPUT  "EINSATZ"; C(J)
160    GOTO 180
170    INPUT  "NUMMER  EINSATZ"; B(J),C(J)
180  NEXT J
190  X = INT(38*RND+1):         'Rotation der Scheibe
200  CLS
210  PRINT  "DIE GEWINNZAHL IST"; X
220  FOR J=1 TO N               'Ermittlung der Gewinne und Verluste
230    ON D(J)   GOSUB 1000,2000,3000
240  NEXT J
248  PRINT
249  PRINT
250  PRINT  "SPIELSTAND"
251  PRINT
260  PRINT  "SPIELER", "CHIPS"
270  FOR  J=1 TO N
280    PRINT  J,A(J)
290  NEXT J
300  INPUT  "WOLLEN SIE EIN WEITERES SPIEL SPIELEN (J/N)"; S$
310  CLS
320  IF  S$="J" OR S$="j"  THEN 90:   'Wiederholung des Spiels
330  PRINT  "DAS KASINO IST GESCHLOSSEN! LEIDER..."
340  END
999  '-----------------------------------------------------
1000 'Wette auf Zahl                                  Subroutine 1
...
...
...
1070 RETURN
1999 '-----------------------------------------------------
2000 'Wette auf gerade Zahl (Wette auf pair)          Subroutine 2
...
...
...
2090 RETURN
2999 '-----------------------------------------------------
3000 'Wette auf ungerade Zahl (Wette auf impair)      Subroutine 3
...
...
...
3090 RETURN
3999 '-----------------------------------------------------
4000 END
```

Abb. 4.18 Gesamtprogramm zur Simulation eines Rouletts

(zu testen) und dabei die auftretenden Fehler zu lokalisieren. Unterprogramme können nämlich auch einzeln getestet werden.

Man kann die Ausgabe des Zufallszahlengenerators genau so behandeln wie jede andere Zahl. Insbesonders können die erzeugten Zufallszahlen in arithmetische Operationen eingehen. Der Ausdruck

```
5*RND
```

bedeutet beispielsweise, daß die vom Zufallszahlengenerator ausgegebene Zahl mit 5 multipliziert wird,

```
RND + 2
```

hingegen, daß auf die generierte Zufallszahl 2 addiert wird. Derartige Rechenoperationen werden eingesetzt, um Zufallszahlen zu erzeugen, die einem anderen Wertebereich angehören als dem Intervall von 0 bis 1. So kann man den letzten Ausdruck dazu benutzen, um Zufallszahlen im Intervall von 2 bis 3 zu bekommen.

Beispiel 4:

Es ist ein Programm zu schreiben, das 10 Zufallszahlen erzeugt, die im Intervall 5 bis 8 liegen.

Die für die Lösung benötigte Formel wollen wir in zwei Schritten zusammenstellen. Wir gehen von der Funktion RND aus, die bekanntlich für Zufallszahlen zwischen 0 und 1 sorgt. – Zuerst kümmern wir uns um die Länge des vorgesehenen Intervalls, 5 von 8 subtrahiert, ergibt 3 Längeneinheiten. Infolgedessen multiplizieren wir zunächst RND mit 3; der Ausdruck 3*RND liefert uns Zufallszahlen zwischen 0 und 3. Nunmehr haben wir den Startpunkt des gewünschten Intervall, d.h. also 5, zu berücksichtigen. Durch Addition von 5 erhalten wir Zufallszahlen, die zwischen 0+5 und 3+5, also zwischen 5 und 8 liegen. Somit erzeugt der Ausdruck

```
3*RND + 5
```

Zufallszahlen im Intervall von 5 bis 8. Das unter diesen Überlegungen entstandene Programm ist in der Abb. 4.19 dargestellt.

```
10 FOR  J=1 TO 10
20   PRINT  3*RND + 5
30 NEXT J
40 END
```

Abb. 4.19 Generierung von Zufallszahlen, die im Intervall von 5 bis 8 liegen

Beispiel 5:

Es ist der Ausdruck niederzuschreiben, durch den ganze Zufallszahlen z mit z = 5,6, · · ·,11,12 erzeugt werden.

Es gibt also acht aufeinanderfolgende mögliche ganze Zahlen, die die Aufgabenstellung befriedigen. Deshalb bilden wir fürs erste den Ausdruck 8*RND, der uns Zufallszahlen zwischen 0 und 8 abliefert. Da die von uns gewünschten Zufallszahlen mit 5 beginnen sollen, müssen wir auf diesen Ausdruck 5 addieren; der so erhaltene Ausdruck erzeugt aber reelle Zufallszahlen zwischen 5.00000 und 12.9999. Durch Benut-

zung der Funktion INT können wir nun den gebrochenen Anteil der Zufallszahlen kappen; die Lösung unseres Problems stellt also der Ausdruck

```
INT(8*RND+5)
```

dar.

Aufgabengruppe 12

Es sind BASIC-Ausdrücke niederzuschreiben, die die folgenden Zufallszahlen erzeugen:

1. Zufallszahlen zwischen 0 und 100
2. Zufallszahlen zwischen 100 und 101
3. Ganze Zufallszahlen zwischen 1 und 50
4. Ganze Zufallszahlen zwischen 4 und 80
5. Gerade ganze Zufallszahlen zwischen 2 und 50
6. Zufallszahlen zwischen 50 und 100
7. Ganze, durch 3 teilbare Zufallszahlen zwischen 3 und 27
8. Ganze Zufallszahlen aus der Folge 4,7,10,13,16,19 und 22
9. Das Würfelprogramm (siehe Abb. 4.13) ist so zu erweitern, daß ähnlich wie beim Rouletteprogramm (Beispiel 2) eine Gewinn- und Verlustabrechnung sowie eine Spielkontenführung erfolgt. In der nachstehenden Tabelle sind die Gewinnausschüttungen aufgeführt, die bei einem Wetteinsatz von 1 DM auf das betreffende Würfelaugenpaar ausgezahlt werden sollen.

Gewürfelte Summe	*Gewinnauszahlung*
2	35
3	17
4	11
5	8
6	6.20
7	5
8	6.20
9	8
10	11
11	17
12	35

10. Das Roulettprogramm von Beispiel 2 ist so zu erweitern, daß geprüft wird, ob ein Spieler einen genügend hohen Chipbestand besitzt, um die Wette abzudecken.
11. Das Roulettprogramm von Beispiel 2 ist so zu erweitern, daß jedem Spieler gestattet ist, einen bis zu 100.– DM limitierten Kredit aufzunehmen.
12. Es ist ein Programm zu entwerfen, das die vom Benutzer eingegebenen Ergebnisse zufällig ausgewählter arithmetischer Aufgaben mit einstelligen Zahlen überprüft!
13. Es ist eine Liste mit 10 Namen aufzustellen. Anschließend ist ein Programm zu schreiben, das aus dieser Liste vier Namen auf Zufallsbasis herauspickt! Auf diesen Weg lassen sich ungeliebte Arbeiten unparteiisch zuweisen.

Antworten zu den Testübungen

4.4.1 300 bzw. 30%

4.4.2
```
10 FOR  J=1 TO 50
20   PRINT  INT(2*RND+1)
30 NEXT J
40 END
```

Nach der Wahrscheinlichkeit 25-mal.

4.4.3
```
10 LET  X = INT(3*RND+1)
20 IF  X = 1  THEN PRINT "VORDERSEITE"  ELSE PRINT "RÜCKSEITE"
30 END
```

Durch die benutzte Formel werden als Zufallszahlen nur 1, 2 und 3 erzeugt. Es ist wahrscheinlich, daß die Zufallszahlen zu einem Drittel eine 1 sind, zwei Drittel sind 2 oder 3.

5 Enttäuschungen bei der Programmierung und ihre Verhütung

Wie man wahrscheinlich bis hierhin ausfindig gemacht hat, erweist sich die Programmierung als eine verzwickte, vielfach enttäuschende Beschäftigung. Zuerst muß man sich die Anweisungen ausdenken, die man dem Computer geben will. Danach muß man sie in den RAM eingeben. Schließlich muß man das Programm ablaufen lassen. Nach dem ersten Programmablauf hat man gewöhnlich erneut über das Programm nachzudenken, nachzudenken darüber, warum das Programm nicht so arbeitet, wie man es vorgesehen hatte. Diese Vorgehensweise kann ermüdend und enttäuschend zugleich sein, vor allem dann, wenn man sich mit langen oder komplexen Programmen beschäftigen muß. Wir sollten freilich an dieser Stelle noch einmal nachdrücklich betonen, daß die bei der Programmierung vorkommenden Enttäuschungen sich oft als das Resultat der Begrenzungen und der Starrheit der Computer herausstellen, die dadurch nicht exakt verstehen können, was man ihnen sagen will. Bei Gesprächen mit anderen Personen siebt man gewöhnlich die belanglosen Informationen heraus, korrigiert unbedeutende Fehler und kann somit ohne weiteres den Gesprächsfluß aufrecht erhalten. Bei einem Computer jedoch muß man zuerst alle Unklarheiten aus dem Weg räumen, bevor die Konversation überhaupt beginnen kann.

Glücklicherweise besitzt der Computer Besonderheiten, die geschaffen wurden, um die Bürde der Programmierung zu erleichtern und bei der Aufspürung und der Korrektur von Fehlern Hilfestellung zu leisten. Mit diesen Besonderheiten nun wollen wir uns in diesem Kapitel auseinandersetzen. Darüberhinaus wollen wir einige weitere Tips präsentieren, die dazu beitragen sollten, Programme schneller und mit weniger Fehlern zu entwerfen.

5.1 Programmablaufpläne

Die Programme in den letzten beiden Kapiteln waren leidlich einfach. Vom Ende des 3. Kapitels an erkannten wir, daß sie allmählich verwickelter wurden. Es gibt jedoch viele Programme, die sehr viel länger und komplexer als die vorgestellten sind. Vielleicht fragt sich der eine oder andere Leser nun, wie es möglich ist, derartige Programme zu planen und auszuführen. Die Schlüsselidee hierzu besteht darin, umfassende Programme auf eine Folge kleinerer Programme, die separat geschrieben und getestet werden können, zurückzuführen.

Der altbekannte Ausspruch „Ein Bild sagt mehr als tausend Wörter", bewahrheitet sich auch für die Programmierung der Computer. Beim Entwerfen eines Programmes, speziell eines langen Programmes, ist es äußerst nützlich, ein Bild zu zeichnen, das die Anweisungen eines Programmes und ihre Beziehungen zueinander veranschaulicht. Ein solches Bild wird *Programmablaufplan*, kurz *PAP* genannt.

Ein Programmablaufplan besteht aus einer Folge von Sinnbildern oder Symbolen, die untereinander mit Pfeilen verbunden sind. In ein Sinnbild trägt man eine oder mehrere Computeranweisungen ein; ein auf diese Weise ausgefülltes Sinnbild bezeichnet man meist als Block. Die Pfeile dienen dazu, den logischen Zusammenhang, den Fluß der Anweisungen also, darzustellen. Der in Abb. 5.1 präsentierte Programmablaufplan zeigt ein Programm zur Berechnung der Summe

$$1 + 2 + 3 + \cdots + 100$$

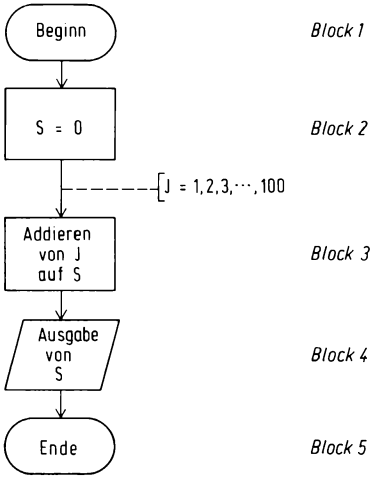


Abb. 5.1 PAP zur Summenberechnung

Die Pfeile im PAP zeigen die Reihenfolge der Operationen an. Man beachte, daß vor dem 3. Block der Vermerk

$$I = 1,2,3,\cdots,100$$

erscheint. Diese Schreibweise bedeutet, daß eine Schleife mit der Laufvariablen I vorliegt. Die Operation, die im nachfolgenden Sinnbild notiert ist, ist demzufolge 100-mal zu wiederholen, und zwar beginnend mit I = 1 und endend mit I = 100, wobei eine Schrittweite von 1 zugrundezulegen ist. Es ist augenscheinlich, daß es außerordentlich leicht ist, den PAP als Vorlage für die Niederschrift des entsprechenden BASIC-Programmes zu benutzen; in der Abb. 5.2 ist es aufgelistet.

```
1Ø LET  S = Ø              (Block 2)
2Ø FOR  J = 1 TO 1ØØ
3Ø   S = S + J             (Block 3)
4Ø NEXT J
5Ø PRINT S                 (Block 4)
6Ø END                     (Block 5)
```

Abb. 5.2 Programm für die Summenberechnung

Die Gestaltung von Programmablaufplänen ist nach DIN 66001 genormt. Die in dieser Norm festgelegten verschiedenen Sinnbilder deuten auf gewisse Programmoperationen hin. Wir wollen uns zu eigen machen, daß wir uns vorerst nur der einfachsten Sinnbilder bedienen (Abb. 5.3).

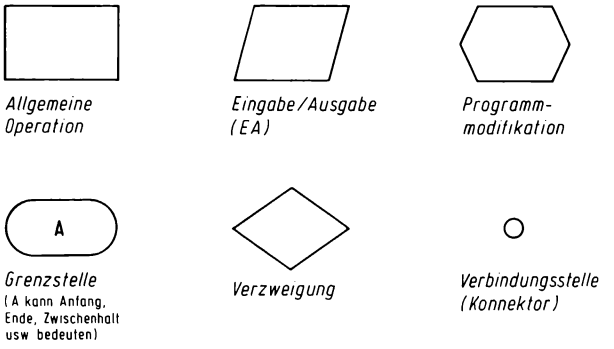


Abb. 5.3 Sinnbilder für Programmabläufe

Programmabläufe, die Verzweigungen beinhalten, verdienen gesondert hervorgehoben zu werden. Der in der Abb. 5.4 gezeigte PAP zeigt die graphische Darstellung eines Programmes, in das eine Entscheidung und damit eine Verzweigung aufgenommen ist. Diese Entscheidung betrifft die Tatsache, ob ein Kreditlimit überschritten ist oder nicht.

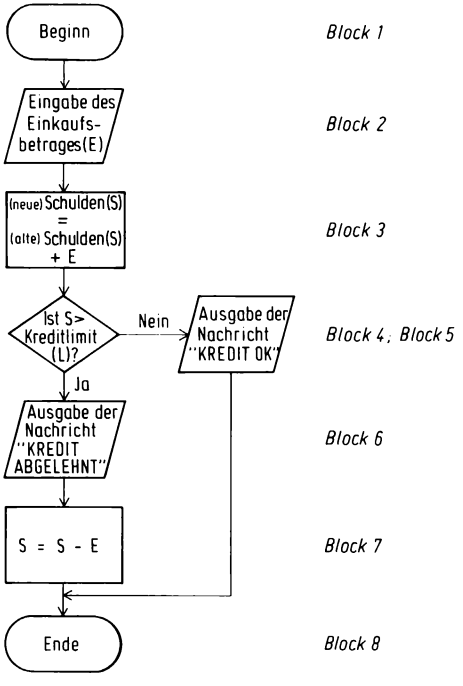


Abb. 5.4 Entscheidung bezüglich Überschreitung des Kreditlimits (PAP)

In der Abb. 5.4 beachte man besonders den Rhombus. Er enthält die Frage

Ist S>Kreditlimit (L)?,

aufgrund der die Entscheidung gefällt werden muß. Bedingt durch die zwei möglichen Antworten auf diese Frage führen auch zwei Pfeile aus diesem Sinnbild heraus; sie sind entsprechend mit „Ja" bzw. „Nein" gekennzeichnet. - Es sei ferner darauf hingewiesen, wie wir gleichzeitig in den verschiedenen Blöcken die Bezeichnungen festgelegt haben, die wir den einzelnen Variablen geben wollen. Wir haben sie nämlich einfach in Klammern notiert. Nachdem der Programmablaufplan erstellt worden ist, ist es leicht, diesen durch eine Programmiersprache zu beschreiben. Durch diese Umformung entsteht das Programm von Abb. 5.5.

```
1Ø  INPUT E                          (Blöcke 1 und 2)
2Ø  INPUT #1, S,L
3Ø  S = S + E                        (Block 3)
4Ø  IF  S > L  THEN 1ØØ  ELSE 2ØØ    (Block 4)
1ØØ PRINT "KREDIT ABGELEHNT"         (Block 6)
11Ø S = S - E                        (Block 7)
12Ø GOTO 3ØØ                         ——→ zur Zusammenführung mit dem „Nein-Zweig"
2ØØ PRINT "KREDIT OK"                (Block 5)
3ØØ END                              (Block 8)
```

Anmerkung: Das Statement in der Zeile mit der Zeilennummer 2Ø bewirkt das Einlesen der Werte von S und L von einer auf einer Diskette gespeicherten Datei; Näheres darüber wird im Kap. 6 besprochen.

Abb. 5.5 Entscheidung bezüglich Überschreitung des Kreditlimits (Programm in BASIC)

Schon nach diesen beiden vorgestellten Beispielen wird man unserer Behauptung zustimmen können, daß graphische Darstellungen nützliche Werkzeuge beim Nachdenken über die einzelnen Operationen sein können, die notwendig für den geplanten Ablauf eines Programmes sind. Man sollte hierzu ruhig einen persönlichen Arbeitsstil entwickeln; für den Eigengebrauch ist es nicht unbedingt erforderlich, daß man sich der genormten Sinnbilder für die Operationsarten bedient. Im Grunde genommen sollte man alle persönlichen Arbeitsweisen unterstützen und demzufolge auch akzeptieren, so lange sie den einzelnen beim Schreiben seiner Programme nützlich sind; freilich sollten sie im Interesse eines Gedankenaustausches für andere verständlich sein.

Aufgabengruppe 13

Es sind Programmablaufpläne (PAP) für Computerprogramme zu zeichnen, die die nachfolgend beschriebenen Aufgabenstellungen lösen!

1. Es ist die Summe der Quadratzahlen von 1 bis 100 zu berechnen, das Ergebnis auszugeben und zu untersuchen, ob die Summe größer als oder kleiner als oder gleich 4873 ist.
2. Es ist die Zeit zu berechnen, die seit dem Augenblick verstrichen ist, in dem der Computer eingeschaltet worden ist.
3. Roulette-Spiel (siehe Abschn. 4.4)
4. Erstellung einer Lohnliste (siehe Beispiel 2, Abschn. 4.2)

5.2 Fehler und Fehlersuche

Im Computerjargon wird ein Fehler mitunter auch als „Laus" (engl.:bug) bezeichnet. Fehler oder „Läuse" in einem Programm zu finden und zu beseitigen wird Fehlersuche (engl.: debugging) genannt, was sich oftmals als eine recht knifflige Angelegenheit erweist. Die Hersteller kommerzieller Software müssen regelmäßig Fehler beheben, die sie in ihren eigenen Programmen entdecken! Der IBM Personalcomputer ist mit einer Anzahl von Einrichtungen ausgerüstet, die bei der Auffindung von Fehlern helfen können.

5.2.1 Programmablaufverfolgung

Oft erweist sich der erste Versuch, ein Programm ausführen zu lassen, als Mißerfolg; eine Andeutung darüber, warum der Programmablauf fehlgeschlagen ist, wird jedoch nicht gegeben. Beispielsweise könnte ein Programm unbegrenzt ablaufen, ohne daß der Benutzer durch einen Fingerzeig erfährt, welche Operation gerade durchgeführt wird. Wie kann man sich da ausmalen, was falsch ist. Eine Möglichkeit besteht darin, sich der Einrichtung zur Programmablaufverfolgung zu bedienen. Wir wollen den Gebrauch dieser Einrichtung anhand des Programmbeispieles in Abb. 5.6 demonstrieren. Das fragliche Programm, in dem wir uns um die Auffindung der Fehler bemühen wollen, soll zur Berechnung der Summe der natürlichen Zahlen von 1 bis 100, d.h. von

$$1 + 2 + 3 + \cdots + 100$$

dienen.

```
1Ø   LET   S = Ø
2Ø   LET   J = Ø
3Ø   LET   S = S + J
4Ø   IF   J=1ØØ   THEN 1ØØ   ELSE 2ØØ
1ØØ LET   J = J + 1
11Ø GOTO 2Ø
2ØØ PRINT   S
3ØØ END
```

Abb. 5.6 Programm zur Berechnung der Summe der natürlichen Zahlen von 1 bis 100

Das in Abb. 5.6 vorgelegte Programm weist zwei Fehler auf. (Kann man sie auf Anhieb richtig erkennen?). Alles, was man anfangs erkennt, ist bloß die Tatsache, daß das Programm nicht richtig funktioniert. Das Programm läuft zwar ab, aber gibt als Summe den Wert 0 aus, welchen wir als unsinnig ansehen müssen. Wie können wir nun die Fehler lokalisieren? Dazu wollen wir die Funktion „Programmablaufverfolgung" (engl.: trace) wirksam machen, indem wir TRON (Trace ON) eintippen. Der Computer reagiert darauf mit der Antwort Ok. Daraufhin tippen wir RUN ein. Dadurch wird in üblicher Weise der Computer veranlaßt, das Programm ablaufen zu lassen. Bedingt durch die Eingabe von TRON werden darüberhinaus jetzt auch die Nummern der Zeilen mit den ausgeführten Anweisungen ausgegeben. Wir erhalten auf dem Bildschirm die Anzeigen, die in Abb. 5.7 dargestellt sind.

```
TRON
Ok

RUN
(10) (20) (30) (40) (200) 0
(300)
```

Abb. 5.7 Anzeigen über den Ablauf des Programmes von Abb. 5.6

Die in den eckigen Klammern stehenden Zahlen bedeuten die Nummern der Zeilen, mit den ausgeführten Statements. Somit hat in unserem Fall der Computer der Reihe nach die Statements ausgeführt, die auf den Zeilen mit den Zeilennummern 10, 20, 30, 40, 200 und 300 enthalten sind. Bei der nicht in eckigen Klammern stehenden 0 handelt es sich um die Programmausgabe der Summe S, hervorgerufen durch das PRINT-Statement auf der Zeile mit der Zeilennummer 200. Diese angezeigte Zeilennummernliste entspricht nicht der von uns erwarteten. Unser Programm war so entworfen worden (zumindest dachten wir uns das so), daß wir die Ausführung des Statements auf der Zeile 100 nach der Ausführung des Statements auf der Zeile 40 erhofften. Außerdem erkennen wir auf der Zeilennummernliste, daß keine Schleife durchlaufen wurde. Wie konnte es also geschehen, daß bei der Programmausführung die Zeile 200 unmittelbar nach der Zeile 40 angesteuert wurde? Dies bringt uns auf den Gedanken, den Inhalt der Zeile 40 zu untersuchen: Und siehe da! Wir entdecken einen Fehler. Die Bezugnahmen auf die Zeilen 100 und 200 sind, wie sie in der Zeile 40 auftreten, vertauscht worden, ein Fehler übrigens, der allzuleicht gemacht werden kann. Nachdem wir den Fehler erkannt haben, wollen wir ihn durch Neueintippen der Zeile 40 sofort korrigieren.

```
40 IF I=100 THEN 200 ELSE 100
```

Triumphierend über unseren Erfolg lassen wir nunmehr das Programm erneut ablaufen. Die jetzt auf dem Bildschirm erscheinende Ausgabe ist in der Abb. 5.8 dargestellt.

```
(10) (20) (30) (40) (100) (110) (20) (30)
(40) (100) (110) (20) (30) (40) (100) (110)
(20) (30) (40) (100) ···
⋮
(20) (30) (40) (100)
Break in 110
```

5.8 Verfolgung des Ablaufes des Programmes von Abb. 5.6 nach der Korrektur des ersten festgestellten Fehlers

Tatsächlich rauscht die in Abb. 5.8 vorgestellte Ausgabe nur so an uns vorüber, da der Computer mit irrsinnigem Tempo die einzelnen Anweisungen ausführt. Nach spätestens 30 Sekunden sollten wir spüren, daß irgendetwas nicht in Ordnung ist, weil es einfach unwahrscheinlich ist, daß unser Programm so lange zu seiner Ausführung braucht. Wir stoppen deshalb die Ausführung mittels der Unterbrechungstaste (Tastenkombination *Ctrl-Break*) ab. Die letzte Zeile (Abb. 5.8) zeigt an, daß wir den Computer in dem Augenblick unterbrochen haben, als er gerade das Statement auf der Zeile mit der Zeilennummer 110 ausführte. Tatsächlich erscheint

nicht nur diese Zeile auf dem Bildschirm. Vielmehr ist der Bildschirm gefüllt mit Ausgabezeilen, die denen von der Abb. 5.8 ähneln. Wir können daraus ohne weiteres den Schluß ziehen, daß sich der Computer in einer Schleife befindet. Jedes Mal nämlich, wenn die Zeile 110 erreicht worden ist, geht die Ablaufsteuerung wieder zur Zeile 20 zurück. Warum endet nun die Schleife niemals? Diese nur allzu berechtigte Frage gehen wir an, indem wir uns daran erinnern, daß die Schleife dann beendet werden sollte, wenn der Wert von I gleich 100 ist. Aber, kann denn I jemals den Wert 100 annehmen? Natürlich nicht, da jedes Mal, wenn der Computer das Statement auf Zeile 20 ausführt, der Wert von I erneut auf 0 zurückgesetzt wird. Somit kann die Variable I niemals den Wert 100 annehmen und das Statement auf Zeile 40 bewirkt konsequenterweise über die Zeilen 100 und 110 eine Rückverzweigung zur Zeile 20. Wir wollen verständlicherweise keine ständige Rücksetzung des Wertes von I auf 0. Nach Erhöhung des Wertes von I um 1, was durch das Statement auf Zeile 100 geschieht, wollen wir vielmehr den neuen Wert von I zur Summe S addieren lassen. Wir wollen also zur Zeile 30 und nicht zur Zeile 20 übergehen. Deshalb korrigieren wir die Zeile 110:

```
110  GOTO 30
```

Nunmehr lassen wir durch Eingabe des Befehles RUN das Programm erneut ablaufen. Jetzt erscheinen auf dem Bildschirm eine ganze Reihe von in eckigen Klammern eingeschlossene Zeilennummern rasch hintereinander, gefolgt von der Zahl 5050, dem korrekten Wert der Summe S (Ausgabestatement auf Zeile 200). Unser Programm wird jetzt also richtig ausgeführt. Wenn wir die Programmablaufverfolgung nicht mehr benötigen, wie es nun der Fall ist, können wir sie durch die Eingabe des Befehles

```
TROFF
```

(Trace OFF) wieder ausschalten. Zuguterletzt lassen wir das Programm ein letztes Mal ohne Programmablaufverfolgung ablaufen. Die Anzeigen auf dem Bildschirm, die wir durch die soeben beschriebene Vorgehensweise nach der Neueingabe der Zeile 110 erhalten, sehen so aus, wie es in der Abb. 5.9 dargestellt ist.

```
⋮
(40) (200) 5050
(300)
Ok

TROFF
Ok

RUN
5050
Ok
```

Abb. 5.9 Programmablaufverfolgung und Ausführung des Programmes von Abb. 5.6 nach der Endkorrektur

Bei dem soeben besprochenen Beispiel veranlaßten wir, daß die Zeilennummern aller ausgeführten Statements auf dem Bildschirm angezeigt wurden. Bei „langen" Programmen kann

das zu einer ungeheuer großen Liste mit Zeilennummern führen. Man kann aber selektiv vorgehen, indem man die Befehle TRON und TROFF als Statements in ein Programm aufnimmt. Man muß sie dann natürlich wie die anderen BASIC-Anweisungen mit Zeilennummern versehen. Wenn die Programmablaufsteuerung bei einem so präparierten Programm auf TRON stößt, beginnt die Anzeige der Nummern der Zeilen, die die ausgeführten Anweisungen enthalten. Stößt die Programmablaufsteuerung hingegen auf TROFF, so wird die Anzeige der Zeilennummern eingestellt. Um Fehler in einem Programm zu suchen, kann man also zeitweilig an ausgewählten Stellen TRON- und TROFF-Anweisungen hinzufügen. Wenn man die Fehler lokalisiert und behoben hat, kann man die entsprechenden Anweisungen zur Programmablaufverfolgung wieder entfernen.

5.2.2 Fehlernachrichten

Beim im Unterabschnitt 5.2.1 behandelten Beispiel lief das Programm tatsächlich ab und kam letzten Endes zum Ergebnis. Eine wahrscheinlich häufigere Fehlerart macht sich dadurch bemerkbar, daß eine oder mehrere Programmzeilen vorliegen, deren Inhalt der Computer infolge eines Fehlers oder infolge eines anderen Problems nicht verstehen kann. In solchen Fällen endet die Programmausführung schon sehr zeitig. Oft kann hierbei der Computer selbst helfen, da er auf die Erkennung allgemein vorkommender Fehler eingestellt ist. Durch Ausgabe einer auf die Fehlerart hinweisenden Fehlernachricht und der Nummer der Zeile, in der der Fehler entdeckt wurde, wird die Arbeit des Programmierers erheblich erleichtert. Man sollte unmittelbar nach der Ausgabe der Fehlernachricht die betreffende Zeile mittels Eingabe des LIST-Befehles auflisten lassen und dann versuchen, die Fehlerursache zu ermitteln.

Angenommen, wir lesen die Fehlernachricht

```
Syntax Error in 530
```

(Syntaxfehler in 530)

Um die Fehlerursache zu ergründen, geben wir den Befehl

```
LIST 530
```

ein, der in der Ausgabe von

```
530  LET  Y = (X + 2(X^2-2)
```

mündet. Beim Studieren dieser Zeile bemerken wir, daß zu einer geöffneten Klammer die paarige geschlossene Klammer fehlt. Das genügt für den Computer, einen Fehler auszuweisen. Wir modifizieren nun die Zeile 530:

```
530  LET  Y = X + 2(X^2-2)
```

Nach Berichtigung dieses Fehlers geben wir erneut den Befehl RUN ein und entdecken, daß nach wie vor ein Syntaxfehler in der Zeile 530 existiert. Das ist für uns wahrlich frustrierend! Es sind eben nicht alle Fehler leicht zu erkennen. Wenn wir uns jedoch eingehender als bisher um den Ausdruck auf der rechten Seite des Ergibtzeichens (Gleichheitszeichens) kümmern, werden wir auch bemerken, daß wir das Sternzeichen zur Kenntlichmachung des Produktes der beiden Faktoren 2 und (X^2 – 2) ausgelassen haben. Dieser Fehler unterläuft speziell den-

jenigen, die mit den gewöhnlichen Regeln der Algebra vertraut sind (bekanntlich wird in der Algebra ein Produkt gewöhnlich ohne Operationszeichen niedergeschrieben, wenn ein Mißverständnis ausgeschlossen werden kann). Wir korrigieren erneut die Zeile 530, entweder durch komplette Neueingabe derselben oder Verwendung des Zeileneditors

```
530  LET  Y = X + 2*(X^2-2)
```

Jetzt endlich finden wir, daß in der Zeile 530 kein Syntaxfehler mehr vorhanden ist.

Der nächste Abschnitt dieses Kapitels enthält eine Aufstellung der am häufigsten vorkommenden Fehlernachrichten. Wir haben also nicht alle Fehlernachrichten in diese Liste aufgenommen, insbesonders fehlen die, die für die Diskettenversion von BASIC gelten. Wenn der Leser eine Zusammenstellung aller Fehlernachrichten benötigt, so wird er von uns auf folgende Literatur verwiesen:

IBM Personal Computer
BASIC Handbuch
Herausgegeben von IBM Deutschland,
VU Literatur 0426

Aufgabengruppe 14

1. Die vom Computer herausgegebenen Fehlernachrichten sind zur Fehlersuche im nachstehenden Programm zu verwenden.

```
1Ø  LET  S = "Ø"
2Ø  FOR  J=1 TO 1ØØ
3Ø    S = S + J(2
4Ø  NEXT K
5Ø  LET  T = Ø
6Ø  FOR  J=1 TO 2Ø
7Ø    LET  T = T + J^3
8Ø  NXT T
9Ø  NEXT T
1ØØ LET  A = ST
11Ø PRINT ALS LOESUNG ERBIT SICH, A
12Ø END
```

Das vorliegende Programm soll zur Berechnung des Produktes

$$(1^2 + 2^2 + \quad + 50^2)\ (1^3 + 2^3 + \quad + 20^3)$$

dienen.

2. Die Einrichtung zur Programmablaufverfolgung soll zur Fehlersuche bei dem folgenden Programm herangezogen werden:

```
10  LET  N = 0
20  IF  N^2 > 175263  THEN 100
30  PRINT  "DAS KLEINSTE N IST GLEICH "
100 N = N + 1
110 GOTO 10
200 END
```

Das Programm soll die kleinste natürliche Zahl 12 ermitteln, für die gilt, daß ihr Quadrat größer als oder gleich 175263 ist.

5.3 Zusammenstellung wichtiger Fehlernachrichten

Wenn BASIC einen Fehler im Programm entdeckt, wird bekanntlich eine Fehlernachricht angezeigt. Diese Zusammenstellung enthält nur die wichtigsten Fehlernachrichten.

① **Syntax error**
(Syntaxfehler)
Zur Anzeige dieser Fehlernachricht kommt es in folgenden Fällen:
a) Unverständliches Statement (falsche Schreibweise eines Schlüsselwortes usw.)
b) Unpaarige Klammern; geöffnete und geschlossene Klammern müssen bekanntlich in einem Ausdruck in gleicher Anzahl auftreten.
c) Falsche Zeichensetzung
d) Verwendung ungültiger Zeichen
e) Fehlende Operatoren
f) Ungültige Variablennamen

② **Undefined Line number**
(Nicht definierte Zeilennummer)
Im Programm wurde eine Zeilennummer benutzt, die nicht zu einer Zeile mit einem Statement gehört. Das kann z.B. leicht dadurch geschehen, daß man Programmzeilen entfernt hat, auf die man sich im Programm irgendwie bezieht (IF, GOTO usw.). Ebenso kann es dazu kommen, wenn man einen Programmteil testet, in dem eine Bezugnahme auf eine Zeile enthalten ist, die man noch nicht geschrieben hat.

③ **Overflow**
(Überlauf)
Im Programm trat eine Zahl auf, die, absolut genommen, größer als die zulässige Maximalzahl ist.

④ **Division by zero**
(Division durch Null)
Es wurde der Versuch unternommen, durch Null zu dividieren. Dieser Fehler wird mitunter nur sehr schwer zu erkennen sein. Der Computer rundet nämlich auf 0 jede Zahl, die, absolut genommen, kleiner als die zulässige Minimalzahl ist. Wird nun eine solche Zahl in nachfolgenden Rechenoperationen als Divisor benutzt, so kommt es zur Division durch Null.

⑤ **Illegal function call**
(Ungültiger Funktionsaufruf)
Es wurde versucht, eine Funktion mit Argumenten auszuwerten, deren Werte außerhalb ihres definierten Wertebereichs liegen. Beispielsweise ist die Quadratwurzelfunktion nur für nichtnegative Zahlen definiert,

die Logarithmusfunktion nur für positive Zahlen, arctan nur für Zahlen zwischen −1 und +1. Jeder Versuch, eine Funktion mit einem Wert außerhalb der jeweiligen Gültigkeitsbereiche auszuwerten, mündet in die Fehlernachricht „ungültiger Funktionsaufruf".
– Diese Fehlernachricht bezieht sich nur auf mathematische Funktionen.

⑥ **Missing operand**
(Fehlender Operand)
Es wurde der Versuch unternommen, eine Anweisung auszuführen, in der erforderliche Daten fehlen, d.h. ein Ausdruck enthält einen Operator (z.B. +, − oder), dem kein Operand folgt.

⑦ **Subscript out of range**
(Index außerhalb des Bereiches)
Es wurde der Versuch unternommen, ein Bereichselement mit einem oder mit mehreren Indizes anzusprechen, die außerhalb der Dimensionen des Bereiches liegen. Die Dimensionen gehen bekanntlich aus dem DIM-Statement hervor.

⑧ **String too long**
(Zeichenkette zu lang)
Es wurde der Versuch unternommen, eine Zeichenkette zu erstellen, die mehr als 255 Zeichen enthält.

⑨ **Out of memory**
(Hauptspeicher reicht nicht aus)
Das Programm ist so umfangreich, daß es nicht in den Hauptspeicher (RAM) des Computers hineingeht. Die Ursache hierfür könnte sein, daß zu große Bereiche definiert sind, oder daß das Programm zu viele Anweisungen enthält, oder daß eine Kombination der beiden soeben erwähnten Gründe vorliegt.

⑩ **String formula too complex**
(Formel für Zeichenkette zu komplex)
Herrührend von der internen Verarbeitung der Formel ergibt sich ein Zeichenkettenausdruck, der entweder zu lang oder zu komplex ist. Dieser Fehler kann dadurch behoben werden, daß man den fraglichen Zeichenkettenausdruck in eine Folge einfacherer Ausdrücke zerlegt.

⑪ **Type mismatch**
(Keine Typenübereinstimmung)
Es wurde der Versuch unternommen, einer Zeichenkettenvariablen einen numerischen Wert, z.B. eine numerische Konstante, zuzuweisen bzw. einer numerischen Variablen einen Zeichenkettenwert, z.B. eine Zeichenkettenkonstante.

⑫ **Duplicate definition**
(Doppelte Definition)
Es wurde der Versuch unternommen, einen Bereich, der bereits dimensioniert wurde, noch einmal zu dimensionieren. Dabei ist zu beachten, daß ein Bereich standardmäßig mit der Dimension 10 angenommen wird, sobald er einmal in einem Programm angesprochen ist und das DIM-Statement fehlt; ein nachfolgendes DIM-Statement verursacht demzufolge auch die Ausgabe dieser Fehlernachricht.

⑬ **NEXT without FOR**
(NEXT ohne FOR)
Im Programm liegt ein NEXT-Statement vor; das vorausgehende FOR-Statement fehlt jedoch.

⑭ **FOR without NEXT**
(FOR ohne NEXT)
Im Programm liegt ein FOR-Statement vor, dem kein NEXT-Statement zugehört, d.h. eine FOR-Schleife war aktiviert worden, ihre Beendigung erfolgte jedoch erst durch das physische Programmende.

⑮ **RETURN without GOSUB**
(RETURN ohne GOSUB)
Es wurde ein RETURN-Statement gefunden, ohne daß ein Unterprogramm (eine Subroutine) vorliegt.

⑯ **Out of data**
(Zu wenig Daten)
Es wurde der Versuch unternommen, durch ein READ-Statement mehr Daten zu lesen, als in DATA-Statements vorhanden sind. Dieselbe Fehlernachricht kann auch auftreten, wenn sich das Lesen von Daten auf Kassetten oder Disketten bezieht und auf dem entsprechenden Medium keine Daten mehr vorliegen.

⑰ **Can't continue**
(Programmausführung kann nicht fortgesetzt werden)
Es wurde versucht, durch einen CONT-Befehl die Ausführung eines Programmes fortzusetzen, obwohl das Programm bereits beendet ist. Dieselbe Fehlernachricht wird auch angezeigt wenn der Befehl CONT eingegeben wurde, ohne daß zuvor der Befehl RUN eingetastet wurde, beispielsweise unmittelbar nach dem Edieren.

5.4 Weitere Hinweise für die Fehlersuche

Die Fehlersuche ist irgendwie zwischen einer „schwarzen Kunst" und einer Wissenschaft einzuordnen. Programmfehlern nachzuspüren, kann in eine sehr verzwickte Beschäftigung ausarten; will man auf diesem Gebiet Hervorragendes leisten, muß man ein guter „Detektiv" sein. Im vorhergehenden Abschnitt 5.3 faßten wir in einer Übersicht einige der Fehlernachrichten zusammen, die uns als Anhaltspunkte bei der Fehlersuche dienen können. Mitunter versagte jedoch das System der Fehlernachrichten bei der Diagnostizierung von Fehlern. Beispielsweise könnte das Programm ausgeführt werden, ohne daß es zur Ausgabe einer Fehlernachricht kommt; es liefert dabei allerdings nicht die Ergebnisse ab, die wir erwartet haben („es tut nicht so, wie es soll"). Wenn derartige Umstände vorliegen, muß man versuchen, ein eigenes Gespür für die Untersuchung eines Programmes zu entwickeln. Einige der hierbei am meisten eingesetzten Techniken sollen nunmehr beschrieben werden.

5.4.1 Einfügung gesonderter PRINT-Statements

Man kann temporär in ein Programm gesonderte PRINT-Statements einfügen. Der Einsatz dieser Technik ermöglicht es, die Wertänderung einer oder mehrerer Variablen während der Programmausführung zu verfolgen.

5.4.2 Einfügung von STOP-Statements

Es ist durchaus möglich, daß sich bereits bei der Planung bzw. beim Entwurf eines Programmes ein logischer Irrtum einschleicht. In diesem Fall wird man bei Zugrundelegung des Planes, z.B. eines Programmablaufplanes, ein Programm niederschreiben, das formal fehlerfrei ist, das also ohne Ausgabe von Fehlernachrichten abläuft. Das große Erwachen kommt jedoch dann beim Anschauen der Ergebnisse: Das Programm hat nicht so gearbeitet, wie man es erwartet hat. Um diese Ungereimtheiten in den Griff zu bekommen, ist es hier ratsam, zeitweilig ins Programm STOP-Anweisungen einzufügen, um durch sie Stopps nach Durchlaufen bestimmter Programmteile zu erzwingen.

Diese Fehlersuchtechnik kann auf verschiedene Weisen benutzt werden:

1. Wenn der Programmablauf auf eine STOP-Anweisung stößt, wird bekanntlich die Ausführung angehalten und auf dem Bildschirm die Nummer der Zeile angezeigt, bei der der Programmablauf gestoppt wurde. Läuft nun das Programm tatsächlich auf den vorgesehenen Stopp auf, so weiß man, daß damit auch die Anweisungen, die unmittelbar vor der STOP-Anweisung liegen, ausgeführt wurden. Wenn andererseits der Programmablauf nicht stoppt, kann man annehmen, daß das Programm einen unvorhergesehenen Weg eingeschlagen hat, d.h. die der STOP-Anweisung vorausgehende Anweisung nicht durchläuft. Ermittelt man den Grund für dieses nicht vorausschaubare Verhalten, wird man wahrscheinlich den Fehler finden und darauf aufbauend den fraglichen Programmabschnitt auch verbessern können.
2. Wenn der Programmablauf anhält, bleiben die Werte, die den Variablen zugewiesen wurden, erhalten. Man kann diese nur untersuchen bzw. prüfen, um festzustellen, ob sie auch dem bisherigen Programmverhalten entsprechen. Mehr darüber sagen wir im Abschnitt 5.4.3 aus.
3. Man kann in ein Programm natürlich mehrere STOP-Anweisungen einfügen. Bei jedem Halt kann man sich den gegenwärtigen Zustand des Programmes notieren, z.B. die Zeilennummer der erreichten STOP-Anweisung, die Werte von für den Programmablauf maßgebenden Variablen usw. Man kann anschließend die Wiederaufnahme des Programmablaufes veranlassen, indem man den Befehl CONT eingibt und die Eingabetaste anschlägt. Es sei jedoch der folgende Hinweis gestattet: Wenn man während eines Halts eine oder mehrere Programmzeilen ändert, kann man nicht die Wiederaufnahme des Programmablaufes veranlassen, sondern man muß in diesem Fall den Programmablauf neu starten, indem man den Befehl RUN eingibt und anschließend die Eingabetaste betätigt.

5.4.3 Untersuchung und Prüfung von Variablen im Sofortmodus

Wenn BASIC den Ablauf eines Programmes stoppt, werden die augenblicklichen Werte der Variablen nicht zerstört. Sie verbleiben vielmehr im Hauptspeicher (RAM) und können daher untersucht und geprüft werden, um daraus Rückschlüsse auf den bisherigen Programmablauf zu ziehen. Das gilt sowohl, wenn der Programmablauf aufgrund einer STOP-Anweisung gestoppt wurde, als auch, wenn es zu einem Halt aufgrund einer von außen einwirkenden Unterbrechung (gleichzeitiges Drücken der beiden Tasten *Ctrl* und *Break*) gekommen ist.

Nehmen wir einmal an, daß ein Programm in seinem Ablauf angehalten und die Nachricht Ok angezeigt wurde. Um die augenblicklichen Werte der im Programm vorkommenden Variablen BETRAG und DATEINAME zu bestimmen, brauchen wir nur den Befehl

```
PRINT BETRAG,DATEINAME$
```

einzugeben und die Eingabetaste anzuschlagen. Eine Zeilennummer ist bei Befehlen bekanntlich nicht einzugeben, sie fehlt hier also auch. Da in diesem Fall etwas sofort getan werden soll, spricht man vom S o f o r t m o d u s . BASIC veranlaßt, daß die augenblicklichen Werte der beiden Variablen angezeigt werden, und zwar genau so, wie wenn ein PRINT-Statement im Programm vorhanden gewesen wäre.

```
145.83        RECHN.MRZ
```

Wir haben jedoch noch eine Warnung auszusprechen. Sobald man irgendeine Änderung im Programm (Verbesserung einer vorhandenen Zeile, Hinzufügung einer neuen Zeile usw.) vor-

nimmt, setzt BASIC alle Variablen wieder auf den definierten Grundwert zurück. Den numerischen Variablen wird also der Wert 0 zugewiesen, den Zeichenkettenvariablen die Nullkette. Wenn man also einen akkuraten Stand der Werte der Variablen, so wie sie sich aus dem bisherigen Programmablauf ergeben haben, bekommen will, dann muß man sicherstellen, daß man die Werte anfordert, bevor man irgendeine Programmänderung vornimmt.

5.4.4 Ausführung von Programmteilen

Mitunter nützt es, wenn man nur einen Teil des Programmes ausführen läßt. Man kann den Programmablauf bei jeder beliebigen Zeile starten, indem man eine Erweiterung des Befehles RUN benutzt. Wenn man beispielsweise die Ausführung bei der Zeile 500 beginnen lassen will, dann tippt man einfach

```
RUN 500
```

ein und drückt anschließend die Eingabetaste. Hier muß allerdings darauf hingewiesen werden, daß durch den Befehl RUN alle Variablen auf ihren definierten Grundwert zurückgesetzt werden. Wenn also in einigen vor dem Startpunkt liegenden Programmteilen bestimmten Variablen Werte zugewiesen werden, so spiegelt ein Ablaufbeginn mitten in einem Programm kein korrektes Abbild der Programmoperationen wider. Um dieses Problem zu umgehen, kann man solchen Variablen Werte im Sofortmodus zuweisen. Nehmen wir beispielsweise einmal an, daß in einem vor dem Startpunkt liegenden Programmabschnitt die Variable BETRAG auf 145.83 und die Variable DATEINAME auf RECHN.MRZ gesetzt wird. Um einen einwandfreien Ablauf des gewünschten Programmteiles mit diesen Variablenwerten zu erhalten, muß man zunächst

```
BETRAG=145.83: DATEINAME="RECHN.MRZ"
```

eingeben und anschließend die Eingabetaste betätigen. Natürlich kann man die beiden Wertzuweisungen auch nacheinander auf zwei getrennten Zeilen eingeben: man muß dann selbstverständlich nach Eingabe jeder Zeile die Eingabetaste drücken. Um das Programm nun bei der Zeile 500 zu starten, hat man

```
GOSUB 500
```

einzugeben und anschließend die Eingabetaste zu betätigen. Es genügt hier nicht die Verwendung des Befehles

```
RUN 500
```

Dieser würde nämlich automatisch den Variablen wieder die für sie gültigen Grundwerte (0 bzw. Nullkette) zuweisen.

6 Sammlungen von Dateien

In diesem Kapitel wollen wir über die Methoden sprechen, wie die Computer zur Speicherung und Wiederauffindung von Informationen benutzt werden können.

6.1 Was sind Dateien (Files)?

Unter einer Datei oder einem File versteht man eine Sammlung von Informationen, die auf einem Massenspeicher (Diskette, Kassette oder Hartplattenspeicher) gespeichert ist. Man unterscheidet gewöhnlich zwei Arten von Dateien (Files):

- Dateien mit Programmen (*Programmdateien* oder Programm Files)
- Dateien mit Daten (*Datendateien* oder Daten Files)

a) *Programmdateien (Programm Files)*
Wenn ein Programm auf einer Diskette gespeichert ist, ist es als Programmdatei gespeichert. Einige derartige Programmdateien haben wir selbst bereits erzeugt, indem wir nämlich unsere BASIC-Programme auf einer Diskette sichergestellt haben. Zusätzlich zu den von uns erzeugten Programmdateien enthält die DOS-Diskette auch Programmdateien, die notwendig für den Betrieb des Computers sind; die in sie aufgenommenen Programme gehören zum DOS bzw. zur Programmiersprache BASIC.

b) *Datendateien (Daten Files)*
Die in der Industrie und im Handel benutzten Computerprogramme beziehen sich gewöhnlich auf Dateien mit Informationen, die auf Massenspeichern gespeichert sind. Beispielsweise würde sich die Personalabteilung eine Datei mit den persönlichen Daten jedes Beschäftigten zulegen; in dieser würden sicher der Name, der Vorname, die Adresse, das Alter die Sozialversicherungsnummer, das Eintrittsdatum, die Stellung, das Gehalt bzw. der Lohn und andere persönliche Angaben verzeichnet sein. Ein Warenhaus wird sich natürlich auch eine Artikelbestandsdatei anlegen und warten; diese wird in der Regel u.a. die folgenden Informationen aufweisen: Artikelbezeichnung, Lieferant, auf eine Einheit (Anzahl, kg, l usw.) bezogener gegenwärtiger Lagerbestand, verkaufte Einheiten in der letzten Berichtsperiode, letztes Lieferdatum, letzte Liefermenge, verkaufte Einheiten im letzten Jahr usw. Derartige Dateien werden Datendateien genannt.

Sowohl die Programmdateien als auch die Datendateien werden normalerweise nebeneinander auf dem gleichen Speichermedium untergebracht. Von vielen Blickwinkeln aus gesehen, sind sie einander gleich, zumindest vom Standpunkt des Computers aus, d.h. sie können z.B. kopiert, gelöscht und überprüft werden; für die beiden Dateiarten sind die hierfür benötigten Verfahren gleich. In diesem Kapitel wollen wir über die Verfahren zur Handhabung der Dateien im allgemeinen und zur Handhabung der Datendateien im besonderen sprechen.

Der Computer zeichnet auf einer Diskette die Daten in einer Form auf, die es gestattet, die Daten auch wieder zu lesen. Wir wollen uns hier nicht um die Form kümmern, in der die Programmdateien auf dem Speichermedium stehen; das würde uns zu weit vom eigentlichen Thema entfernen. Für das Verständnis der Dateien mit Daten ist es jedoch wichtig, die Speicherungsformen derselben kennenzulernen.

Wir wollen ein erstes Beispiel betrachten. Angenommen, ein Dozent speichert die von seinen Studenten in den Klausuren erreichten prozentualen Punktzahlen in einer Datendatei. Jeder

Student seiner Seminargruppen muß an vier Klausuren teilnehmen. Eine typische Eintragung[1] in einer solchen Datendatei könnte die folgenden Datenelemente enthalten:

- Name des Studenten (der Studentin)
- Erreichte prozentuale Punktzahl in der 1. Klausur
- Erreichte prozentuale Punktzahl in der 2. Klausur
- Erreichte prozentuale Punktzahl in der 3. Klausur
- Erreichte prozentuale Punktzahl in der 4. Klausur

In einer Datendatei können die einzelnen Datenelemente lückenlos aufeinander folgen; man sagt dann, die Datei ist sequentiell organisiert. Somit könnte der Anfang der obigen Datei etwa so aussehen:

```
"Hannelore Adam", 98, 98, 87, 76, "Walter Adolfi", 56, 87, 96, 91, "Fritz Bu-
cher", 88, 97, 86, 98, "Hans Doller", 78, 85, 83, 97, ···
```

Die Datendatei setzt sich also aus einer Folge von entweder Zeichenkettenkonstanten (hier: die Namen) oder numerischen Konstanten (hier: die erreichten prozentualen Punktzahlen) zusammen; die verschiedenartigen Datenelemente innerhalb einer Datenreihe sind dabei nach einem bestimmten Muster angeordnet (hier: Name, gefolgt von vier Punktzahlen). Durch den Entwurf einer solchen speziellen Anordnung erreicht man, daß die Datei auch wieder gelesen und ihr Inhalt verstanden werden kann. Beim Lesen der obigen Datenelemente z.B. wissen wir im voraus, daß die Datenelemente in Gruppen zu fünf vorliegen; das erste Element der Gruppe stellt dabei einen Namen dar, die anschließenden vier die zugehörigen Punktzahlen.

In diesem Kapitel wollen wir lernen, wie man Datendateien erzeugt, die solche Informationen enthalten, wie wir sie bei dem obigen Beispiel kennengelernt haben. Wie wir später sehen werden, können die Daten in zwei Arten von Datendateien gespeichert werden:

- Dateien mit sequentiellem Zugriff (sequential access)
- Dateien mit wahlfreiem Zugriff (random access), auch Direktzugriffsdateien genannt

Für jede der beiden Arten von Datendateien wollen wir uns mit den folgenden in der Praxis erforderlichen Grundoperationen beschäftigen:

1. Erzeugung einer Datendatei
2. Schreiben (Aufzeichen) von Datenelementen als Datenreihe in eine Datei
3. Lesen von Datenelementen aus einer Datei
4. Änderung von Datenelementen in einer Datei
5. Durchsuchen einer Datei nach bestimmten Datenelementen (Wiedergewinnung von Informationen)

---

[1] Eine solche, aus logisch zusammengehörenden Datenelementen bestehende Eintragung in einer Datei wollen wir, dem üblichen Sprachgebrauch folgend, als Datenreihe oder als Datensatz oder kurz als Satz bezeichnen

6.2. Bezeichnung von Dateien und Geräten

Dateien und Geräte müssen, um sie unterscheiden und ansprechen zu können, bezeichnet werden.

a) *Dateinamen*
Jede Datei ist durch einen Dateinamen gekennzeichnet, der aus bis zu acht Zeichen bestehen darf. Die Zeichen, aus denen Dateinamen gebildet werden können, schließen die Buchstaben A bis Z und die Ziffern 0 bis 9 ein. Es können auch einige Sonderzeichen verwendet werden, doch zu diesem Zeitpunkt wollen wir vorläufig noch an Dateinamen festhalten, die ausschließlich aus Buchstaben und Ziffern gebildet werden. Zusätzlich können einem Dateinamen drei weitere Zeichen angehängt werden, die von der ersten Zeichengruppe durch einen Punkt getrennt werden müssen. Beispiele für Dateinamen ohne Erweiterung sind:

LOHNLIST
ZENSUREN
LAGER

Beispiele für Dateinamen mit Erweiterung sind:

LOHNLIST. MAI
ZENSUREN.KLI
LAGER.002

Man kann die Dateinamen mit Kleinbuchstaben eingeben, doch werden diese grundsätzlich vom Computer als Großbuchstaben ausgelegt. Die nachfolgenden Dateinamen beziehen sich deshalb auf die gleiche Datei:

LOHNLIST
Lohnlist
lohnlist
LohnList

Innerhalb der Dateinamen (einschließlich der Erweiterung) dürfen keine Leerzeichen auftreten. Dateinamenserweiterungen werden auch als Dateinamenszusätze bezeichnet.

b) *Gerätenamen*
Den verschiedenen Computerteilen sind Namen gegeben worden, so daß man sich in Statements und Befehlen auf sie beziehen kann. So sind die Diskettenlaufwerke beispielsweise mit A: und B: benannt; wenn drei oder vier Laufwerke angeschlossen sind, heißen die weiteren entsprechend C: und D: . Zusätzlich gibt es einige weitere Gerätenamen, so z.B.:

CAS1:	Kassettenrecorder
CON:	Konsole bzw. Computertastatur
LPT1:	Drucker
COM1:	Kommunikationskanal (auch bekannt als „asynchroner Kommunikationsadapter" oder „RS232-C Interface")

Um eine Datei komplett zu spezifizieren, ist es im allgemeinen notwendig, sowohl den Dateinamen als auch den Gerätenamen des Gerätes anzugeben, auf dem die Datei untergebracht ist. Die Kombination aus Gerätenamen und Dateinamen wird als Dateispezifikation

bezeichnet. Wenn z.B. die Datei mit dem Namen LOHNLIST auf der Kassette gespeichert ist, so lautet für diese Datei die Dateispezifikation:

```
CAS1:LOHNLIST
```

Analog lautet für eine Datei mit dem Namen BELEGE, die auf der Diskette im Laufwerk B: gespeichert ist, die Dateispezifikation:

```
B:BELEGE
```

Wenn man das Disketten-BASIC oder das erweiterte BASIC benutzt, kann man die Gerätebezeichnung weglassen, sofern man sich auf eine Datei bezieht, die auf der Diskette im Standardlaufwerk gespeichert ist.

Testübung 6.2.1

Es ist die Dateispezifikation für die Datei RECHN niederzuschreiben; sie befindet sich auf der Diskette im Laufwerk B:

Testübung 6.2.2

Welche der folgenden Zeichenketten stellen gültige Dateinamen dar?

a) DUMP.001
b) ORGANISATION2
c) .INS
d) A.0002

6.2.1 Kopieren von Dateien

Man kann eine Datei von einem Platz innerhalb des Computers auf einen anderen Platz übertragen, indem man den Befehl COPY des Betriebssystems DOS benutzt. Hierzu muß man der Reihe nach wie folgt vorgehen:

1. Man muß zunächst die DOS-Systemanfrage zu erlangen versuchen bzw. sie abwarten, d.h. wenn man in BASIC ist, muß zunächst SYSTEM eingegeben und die Eingabetaste gedrückt werden.
2. Zur eigentlichen Kopierung der Datei ist nunmehr der Befehl

   ```
   COPY <ds1> <ds2>
   ```

   einzugeben und die Eingabetaste zu drücken. Hierbei bedeuten:

<ds1>	Dateispezifikation der zu kopierenden Datei
<ds2>	Dateispezifikation der neuen, durch das Kopieren entstehenden Datei

   Im Befehl muß zwischen den beiden Dateispezifikationen mindestens eine Leerstelle gelassen werden.

Um beispielsweise A: BASIC.COM (d.h. also die Datei BASIC.COM mit der Programmiersprache BASIC, auf der Diskette im Laufwerk A: befindlich) in eine Datei gleichen Namens auf eine ins Laufwerk B: eingeführte Diskette zu kopieren, hat man somit den Befehl

```
COPY A:BASIC.COM B:BASIC.COM
```

einzugeben und die Eingabetaste anzuschlagen. Dadurch wird, wie gewünscht, eine Kopie der Originaldatei BASIC.COM auf der Diskette im Laufwerk B: erstellt.

Wenn wir, wie hier, wollen, daß der Dateiname der Kopie der gleichen ist wie der der Originaldatei, dann brauchen wir beim COPY-Befehl anstelle der kompletten Dateispezifikation der Zieldatei nur den Gerätenamen anzugeben. Somit kann also die obige Kopieroperation auch einfach durch die Eingabe des Befehles

```
COPY A:BASIC.COM B:
```

bewerkstelligt werden; natürlich ist nach der Eingabe dieses Befehles wie üblich die Eingabetaste zu betätigen.

Der COPY-Befehl ist einer der nützlichsten Befehle des Betriebssystemes DOS. Mit seiner Hilfe können wir z.B. auch eine Datei erstellen. Setzen wir uns also an den Computer und warten wir zunächst die DOS-Systemanfrage A> ab. Danach tippen wir den Befehl

```
COPY CON: A:TEST
```

ein und drücken wie üblich die Eingabetaste.

Dadurch haben wir vom Betriebssystem DOS mitgeteilt, daß wir eine Datei von der Konsole (der Tastatur also) auf das Laufwerk A: kopieren wollen; die Datei soll dabei TEST genannt werden. Nunmehr können wir beispielsweise eintippen:

```
Das ist eine Uebung zur Erstellung einer Datei.
Wir erzeugen dabei die Datei TEST auf dem Laufwerk A:.
```

Nach dem Eintippen jeder Zeile haben wir wie bei der Erstellung von BASIC-Programmen die Eingabetaste zu betätigen. Man wird bemerken, daß die eingetippten Zeilen auf dem Bildschirm angezeigt werden. Nach der Eingabe der Zeilen, die zu der zu erstellenden Datei gehören sollen, drücken wir die Funktionstaste *F6* und danach die Eingabetaste. Man wird jetzt das Aufleuchten der Anzeigelampe am Laufwerk A: feststellen. Der Computer wird mit der Antwort reagieren:

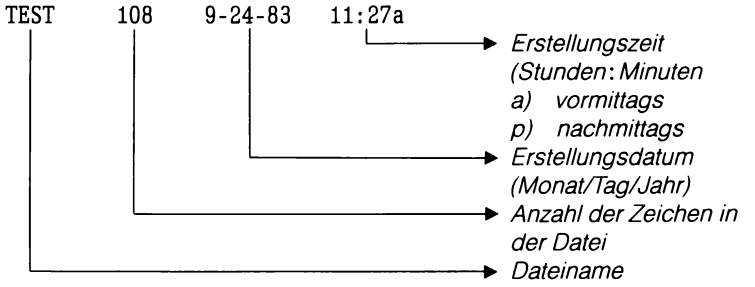
```
1 File(s) copied
```

(1 Datei(en) kopiert)

Nunmehr haben wir die Datei TEST erzeugt. Sollten wir davon noch nicht überzeugt sein, so können wir uns das *Verzeichnis* (Directory) der Dateien auf der ins Laufwerk A: eingeschobenen Diskette ansehen. Hierzu brauchen wir nur den Befehl

```
DIR A:
```

einzugeben, gefolgt vom Anschlagen der Eingabetaste. Unter den auf jetzt auf dem Bildschirm angezeigten Zeilen befindet sich auch eine Zeile mit dem Inhalt:



Diese Eintragung ins Verzeichnis besagt uns, daß der Dateiname gleich TEST ist und daß die Datei insgesamt 108 Zeichen enthält; hierbei sind die Buchstaben, die Ziffern, die Sonderzeichen (Leerzeichen usw.) und die Gerätesteuerzeichen (Drücken der Eingabetaste) gezählt. Zu den anderen Angaben ist zu bemerken, daß die Datei am 24.9. 1983 um 11.27 Uhr erstellt wurde. Datum und Zeit der Erstellung werden dabei aus den Daten errechnet, die beim Einschalten des Computers eingegeben wurden.

Wenn man nun immer noch nicht davon überzeugt ist, daß man eine Datei erstellt hat, sollte man den Inhalt der soeben erstellten sequentiellen Datei auf dem Bildschirm anzeigen lassen. Man braucht dazu nur den Befehl

```
COPY TEST CON:
```

einzugeben und die Eingabetaste zu drücken. Durch diesen Befehl haben wir das Betriebssystem DOS aufgefordert, den Inhalt der Datei TEST , die auf der Diskette gespeichert ist, die augenblicklich in das Laufwerk A: eingelegt ist, auf die Konsole kopieren soll. Vom Standpunkt des Computers aus gesehen, besteht die Konsole aus der Tastatur und dem Bildschirm. Daher wird jetzt der Dateiinhalt auf dem Bildschirm angezeigt.

Schließlich können wir den Inhalt der Datei TEST auch über den Drucker kopieren. Hierzu müssen wir den Befehl

```
COPY TEST LPT1:
```

eingeben und die Eingabetaste betätigen. Durch diesen Befehl werden die Sätze der im Befehl spezifizierten Datei gedruckt.

Testübung 6.2.3

Eine Datei mit dem Namen TEST2 ist auf der ins Laufwerk B: eingelegten Diskette zu erstellen. Die Datei soll folgende Daten enthalten, d.h. einen einzigen Satz:

```
Diese Zeile ist Teil der Datei TEST2 auf der Diskette im Laufwerk B:
```

Testübung 6.2.4

Der Inhalt der in der Testübung 6.2.3. erstellten Datei soll auf dem Bildschirm wieder angezeigt werden.

6.2.2 Bezugnahmen auf Gesamtheiten von Dateien

Oft will man sich auf mehrere Dateien gleichzeitig beziehen können. Bei solchen Bezugnahmen ist es deshalb sicher angebracht, wenn man sich gewisser Mehrdeutigkeiten bedienen kann. Man kann das durch die Benutzung der Sonderzeichen * und ? erreichen. Das Sonderzeichen * steht anstelle einer beliebigen Folge von Zeichen innerhalb von Dateinamen oder der Erweiterung von Dateinamen. So sind z. B. mit der Zeichenfolge

```
*.001
```

alle möglichen Dateinamen gemeint, die die Erweiterung 001 aufweisen. Wenn auf der Diskette im Laufwerk A: beispielsweise die Dateien mit den Dateinamen

```
CHAR.001   DAG.001   GELD.001   MAR.001   TIN.001
```

vorhanden sind, dann wird durch *.001 diese Gesamtheit angesprochen. Ein Befehl der Gestalt

```
COPY  A:*.001 B:
```

wird infolgedessen die Kopierung aller Dateien auf der Diskette im Laufwerk A: bewirken, die einen beliebigen Dateinamen mit der Erweiterung .001 besitzen.

Denken wir die ganze Sache logisch zu Ende! Um alle auf der ins Laufwerk A: eingelegten Diskette gespeicherten Dateien auf die ins Laufwerk B: eingelegte Diskette zu kopieren, genügt also der Befehl

```
COPY  A:*.* B:
```

Das Sonderzeichen ? arbeitet ähnlich wie das Sonderzeichen * , durch ? wird jedoch nur ein einziges beliebiges Zeichen angesprochen. Somit steht z. B. die Zeichenfolge

```
??ME
```

für alle Dateinamen, die aus vier Zeichen bestehen; die ersten beiden umfassen dabei alle möglichen Zeichen, die bei Dateinamen verwendet werden dürfen, das 3. und 4. Zeichen sind M und E. Sind auf der Diskette A: beispielsweise Dateien mit den Dateinamen

```
OTME   WIME   RUME   JOME   AAME   BAME   GRME   MAME
```

vorhanden, so ist durch ??ME diese Gesamtheit gemeint.

Testübung 6.2.5

Es ist der Befehl zu nennen, durch den mit einem Mal alle Dateien, die beliebige, jedoch mit der Erweiterung .BAS versehene Dateinamen aufweisen und auf der ins Laufwerk B: eingelegten Diskette gespeichert sind, auf die ins Laufwerk A: eingelegte Diskette kopiert werden können.

6.2.3 Löschen von Dateien

Man kann Dateien mittels des DOS-Befehles ERASE löschen. Dieser Befehl besitzt die folgende allgemeine Form:

```
ERASE  <ds>
```

Hierbei bedeutet <ds> eine Dateispezifikation. Nach der Eingabe dieses Befehles muß selbstverständlich die Eingabetaste betätigt werden. Um beispielsweise die Datei TEST von der ins Laufwerk B: eingelegten Diskette zu löschen, ist der Befehl

```
ERASE  B:TEST
```

einzutippen. Analog bewirkt die Eingabe des Befehles

```
ERASE  B:*.*
```

die Löschung aller Dateien von der ins Laufwerk B: eingelegten Diskette. Gerade beim ERASE-Befehl sollte man sich die Verwendung dieser umfassenden Sonderzeichen sorgfältig überlegen. Wie leicht sind nach der Befehlsausführung mehr Dateien gelöscht, als man eigentlich beabsichtigt hatte!

Aufgabengruppe 15

Es sind Dateispezifikationen für die folgenden Dateien niederzuschreiben!

1. KOSTEN auf der Kassette
2. EDITH auf der ins Laufwerk A: eingelegten Diskette
3. NIEDER.001 auf der ins Laufwerk B: eingelegten Diskette
4. FORM1040.82 auf der Kassette

Es ist zu untersuchen, welche der folgenden Zeichenketten gültige Dateinamen darstellen!

5. AXLE$
6. 1234567890.1
7. ASD#%§.ASC
8. A.GRAPH.001

Es sind DOS-Befehle niederzuschreiben, die die folgenden Aufgabenstellungen lösen!

9. Drucken des Inhalts der Datei B:TEST
10. Kopieren aller Dateien, deren Dateinamen mit der Erweiterung COM versehen sind, von der Diskette A: auf die Diskette B: .

11. Löschen der Datei A:TEST .

12. Kopieren der Datei A:TEST auf B: , wobei die neue Datei den Namen TEST3 erhalten soll.

13. Kopieren aller Dateien, deren Dateiname mit D beginnt und bis zu vier Zeichen umfaßt, von A: auf B: .

Antworten auf die Testübungen

6.2.1 B:RECHN

6.2.2 Nur die unter a) aufgeführte Zeichenkette stellt einen gültigen Dateinamen dar. Ansonsten liegen die folgenden Verstöße gegen die Regeln für die Bildung von Dateinamen vor:
b) Zeichenkette enthält zu viele Zeichen.
c) Beginn mit einem Sonderzeichen, könnte höchstens als Erweiterung eines Dateinamens gebraucht werden.
d) Erweiterung enthält zu viele Zeichen.

6.2.3 Zunächst muß die DOS-Systemanfrage A> abgewartet werden. Nach Eingabe des Befehles

```
COPY CON: B:TESTS3
```

und nachfolgendem Drücken der Eingabetaste kann danach die Zeile eingegeben werden. Am Schluß ist die Funktionstaste F6, gefolgt von der Eingabetaste, zu drücken.

6.2.4 Nach Ausgabe der DOS-Systemanfrage ist der Befehl

```
COPY A:TEST2 CON:
```

einzugeben und anschließend die Eingabetaste zu drücken.

6.2.5 Nach Ausgabe der DOS-Systemanfrage ist der Befehl

```
COPY B:*.BAS A:
```

einzugeben und anschließend die Eingabetaste zu drücken.

6.3 Sequentielle Dateien

Im vorhergehenden Abschnitt erstellten wir Dateien, wobei wir die direkte Tastatureingabe, gekoppelt an den DOS-Befehl COPY, benutzten. Diese Methode kann selbstverständlich nicht in BASIC-Programmen verwendet werden. Wenn Dateien wirklich nützlich sein sollen, muß der Computer auch in der Lage sein, Datenreihen ohne unsere Eingriffe zu schreiben und zu lesen. In diesem Abschnitt wollen wir uns deshalb mit den BASIC-Anweisungen befassen, die das Schreiben von Datenreihen in und das Lesen von Datenreihen aus Dateien bewirken.

Unter einer sequentiellen Datei wollen wir eine Datendatei verstehen, bei der zu den Datensätzen (d.h. zu den logischen Zusammenfassungen von Datenelementen) in ihrer Reihen-

folge, also gemäß ihrer Anordnung in der Datei, zugegriffen wird. In solchen Dateien sind also die Datenelemente in starr aufeinanderfolgender Reihenfolge aufgezeichnet worden. Beim Lesen wird die gleiche Reihenfolge wie beim Schreiben eingehalten. Man kann Datensätze nur am Ende einer sequentiellen Datei hinzufügen. Wenn man also einen Datensatz irgendwo in der Mitte einer Datei einfügen will, so bleibt nur das eine übrig, die gesamte Datei neu zu schreiben. Wenn man andererseits einen Datensatz lesen will, der am Ende der Datei liegt, so muß man in gleicher Weise zuvor erst alle Datensätze lesen, die vor dem gewünschten Satz in der Datei liegen; die nicht gewünschten Sätze sind hierbei gewissermaßen wegzuwerfen.

6.3.1 Eröffnen und Abschließen sequentieller Dateien

Bevor irgendwelche Operationen auf eine Datei einwirken können, muß die betreffende Datei zunächst eröffnet werden. Um sich diesen Prozeß klarzumachen, sollte man sich einmal vorstellen, daß sich die Datei in der „Schublade" eines „Dateischrankes" befindet (Diskette). Um die Datei lesen zu können, müssen wir daher erst die Schublade öffnen. Diese Operation wird durch die BASIC-Anweisung OPEN bewerkstelligt. Wenn man eine Datei eröffnen will, muß man dabei die zu eröffnende Datei durch Angabe ihrer Dateispezifikation benennen und außerdem sagen, ob die Datei gelesen oder geschrieben werden soll. Beim Eröffnen der Datei

```
B:LOHNLIST
```

für die Eingabe (zum Lesen also), verwenden wir beispielsweise das OPEN-Statement in der Form

```
10  OPEN "B:LOHNLIST" FOR INPUT AS #1
```

Die Zeichenfolge #1 dient als sogenannte Referenznummer, die wir einer Datei beim Eröffnungsprozeß zuweisen. Die Referenznummer einer Datei wird oft auch kurz Dateinummer genannt. Solange nun eine Datei eröffnet bleibt, können wir uns auf diese Datei mittels der Referenznummer beziehen und brauchen uns nicht mehr der schwerfälligen, langatmigen Dateispezifikation, hier also

```
B:LOHNLIST,
```

bedienen. Eine alternative Form der OPEN-Anweisung zum Eröffnen einer Eingabedatei lautet:

```
10  OPEN "I",#1,"CAS1:LOHNLIST"
```

Um die Datei B:ZENSUREN.AUG für die Ausgabe, d.h. um die Datei zu schreiben, zu eröffnen, verwenden wir die Anweisung

```
20  OPEN "B:ZENSUREN.AUG" FOR OUTPUT AS #2
```

In diesem Fall lautet die alternative Form der OPEN-Anweisung:

```
20  OPEN "O",#2,"B:ZENSUREN.AUG"
```

Bei beiden Formen der OPEN-Anweisung kann die Dateinummer ohne das Nummernzeichen # aufgeführt werden.

Beim Disketten-BASIC und beim erweiterten BASIC kann man mit drei gleichzeitig eröffneten, auf Disketten liegenden Dateien arbeiten. Diese Anzahl kann erhöht werden, indem man den hierfür geschaffenen Befehl eingibt. Über dieses Verfahren werden wir später Näheres aussagen.

Die Aufrechterhaltung und Pflege von Ordnungssystemen (Akten usw.) erfordert, daß diese ordentlich und organisiert unterhalten werden. Dasselbe gilt erst recht für Computerdateien. Eine Datei kann für die Eingabe o d e r die Ausgabe eröffnet werden, aber nicht gleichzeitig für beide Zugriffsrichtungen. Solange eine Datei eröffnet ist, können auf sie Anweisungen (Eingabe- oder Ausgabeanweisungen) einwirken, selbstverständlich nur von der Art, wie sie beim Eröffnen vorgesehen wurden. Wenn man die Art der auf eine Datei einwirkenden Operationen ändern muß oder will, so hat man zuvor die Datei a b z u s c h l i e ß e n. Beabsichtigt man z.B. die Datei B:LOHNLIST abzuschließen (oben im Beispielstatement 10 eröffnet), so verwenden wir die Anweisung

```
40  CLOSE #1
```

Wir haben hier willkürlich die Zeilennummer 40 verwendet. Nach dem Abschließen können wir die Datei für die Ausgabe wiedereröffnen, indem wir ein Statement der Art benutzen, wie wir es oben im Beispielstatement 20 gezeigt haben, also etwa

```
20  OPEN "B:LOHNLIST" FOR OUTPUT AS #2
```

Es ist gestattet, mehrere Dateien auf einmal abzuschließen. Durch das Statement

```
50  CLOSE #5,#6
```

werden beispielsweise die Dateien mit den Referenznummer 5 und 6 gleichzeitig abgeschlossen. Durch die einfache Anweisung

```
60  CLOSE
```

können *alle* in diesem Augenblick noch eröffneten Dateien mit einem Male abgeschlossen werden. Bei den CLOSE-Anweisungen können die Dateinummern auch ohne Nummernzeichen # niedergeschrieben werden.

Eine gute, praxisbewährte Programmierung diktiert geradezu, daß nach ihrer Benutzung jede Datei auch wieder abgeschlossen werden sollte. In jedem Fall aber sorgen die Befehle NEW und RUN dafür, daß Dateien, die versehentlich in einem früheren Programm nicht abgeschlossen wurden, automatisch abgeschlossen werden.

6.3.2 Schreiben von Datenelementen in sequentielle Dateien

Angenommen, wir wollen eine sequentielle Datei erstellen, die RECHNUNG.001 heißen und den folgenden, aus fünf Datenelementen bestehende Datenreihe enthalten soll:

```
DJ VERKAUF  51357 4  $358.79   4/5/81
```

Wir beabsichtigen also in die Datei eine logische Gesamtheit (Datenreihe oder Satz) zu schreiben, die aus den folgenden fünf Datenelementen bestehen soll:[1)]

- *Zeichenkettenkonstante* "DJ VERKAUF"
- *numerische Konstante* 51357
- *numerische Konstante* 4
- *Zeichenkettenkonstante* "$358.79"
- *Zeichenkettenkonstante* "4/5/81"

In Abb. 6.1 haben wir den Programmabschnitt aufgelistet, der diese Aufgabenstellung löst.

```
100  OPEN "A:QUADRATE" FOR OUTPUT AS #1
110  WRITE #1,  "DJ VERKAUF", 51357, 4, "$358.79", "4/5/81"
120  CLOSE #1
```

Abb. 6.1 Programmabschnitt zum Schreiben eines Satzes in eine sequentielle Datei

Der auf das Schlüsselwort WRITE folgende Teil #1 der Anweisung auf Zeile 110 weist auf die Referenznummer hin, die der Datei

```
RECHNUNG.001
```

in der OPEN-Anweisung auf Zeile 100 gegeben wurde, nämlich 1. Der Referenznummer muß ein Komma folgen. Wir sollten noch bemerken, daß die WRITE-Anweisung in vieler Hinsicht wie eine PRINT-Anweisung arbeitet; die Werte der Datenelemente werden jedoch in eine Datei ausgegeben und nicht auf dem Bildschirm.

Solange eine Datei eröffnet ist, können beliebig viele WRITE-Anweisungen zum Einfügen von Datenreihen in die Datei aufgesetzt werden. Darüberhinaus können selbstverständlich in den WRITE-Statements auch Variable als Datenelemente aufgeführt werden. Hierfür wollen wir ebenfalls ein Beispielstatement bringen:

```
200 WRITE#1, A, A$
```

Durch diese Anweisung werden die augenblicklichen Werte der beiden Variablen A und A$ in die Datei mit der Referenznummer 1 geschrieben.

Beispiel 1:

Es ist ein Programm niederzuschreiben, durch das eine Datei erzeugt wird, in die hintereinander Datenreihen aufgenommen werden sollen, die jeweils aus einer natürlichen Zahl und ihrem Quadrat bestehen sollen. Die natürlichen Zahlen sollen von 1 bis 100 laufen.

[1)] Die Datenreihe, die nacheinander in eine sequentielle Datei geschrieben werden, müssen nicht die gleiche Zusammensetzung aufweisen, d.h. sie können sogar aus unterschiedlichen Datenelementen gebildet werden. Beim Lesen einer solchen Datei muß man natürlich wissen, aus welchen Datenreihen in welchen Anordnung die Datei geschrieben wurde.
Aus diesen Ausführungen ergibt sich, daß die Datenreihen (Datensätze) einer sequentiellen Datei unterschiedliche Längen, d.h. variabel lange Sätze, aufweisen können.

Zur Lösung wollen wir davon ausgehen, daß die zu erstellende Datei QUADRATE genannt werden soll. Außerdem soll sie auf der Diskette im Laufwerk A: gespeichert werden (siehe Abb. 6.2).

```
10  OPEN "A:QUADRATE" FOR OUTPUT AS #1
20  FOR  J = 1  TO  100
30    WRITE #1,  J, J^2
40  NEXT J
50  CLOSE #1
60  END
```

Abb. 6.2 Programm zur Erstellung einer sequentielle Datei

Beispiel 2:

Es ist eine Datei zu erstellen, deren einzelne Datenreihen sich aus den folgenden Datenelementen zusammensetzen sollen:

- *Name*
- *Adreßangaben*
  - *Straße*
  - *Postleitzahl*
  - *Ort*
  - *Ortszusatz*
- *Telefonnummer*

Durch die in die Datei aufzunehmenden Sätze soll ein persönliches Telefonverzeichnis entstehen. Wir wollen die einzelnen Datenelemente über die Tastatur eingeben. Bei der Eingabe soll außerdem die Möglichkeit bestehen, dem Computer mitzuteilen, wann die letzte Telefonnummer eingetippt wurde.

Um die verschiedenen Daten einzugeben, bedienen wir uns des INPUT-Statements. Wir führen folgende Namen für die Variablen ein:

- A$ *für den Namen*
- B$ *für die Straße*
- C$ *für die Postleitzahl*
- D$ *für den Ort*
- E$ *für den Ortszusatz*
- F$ *für die Telefonnummer*

```
5   OPEN "TELEFON" FOR OUTPUT AS #1
10  INPUT "NAME"; A$
20  INPUT "STRASSE"; B$
30  INPUT "POSTLEITZAHL"; C$
40  INPUT "ORT"; D$
50  INPUT "ORTSZUSATZ"; E$
60  INPUT "TELEFONNUMMER"; F$
70  WRITE #1, A$, B$, C$, D$, E$, F$
80  INPUT "IST EINE WEITERE EINGABE ERFORDERLICH (J/N)"; G$
90  IF  G$ = "J"  THEN 10  ELSE 100
100 CLOSE #1
110 END
```

Abb. 6.3 Programm zur Erstellung eines privaten Telefonverzeichnisses

Für die Eingabe der einzelnen, den Variablen zuzuweisenden Werten wird je ein INPUT-Statement vorgesehen. Nach der Eingabe gehen wir zum Schreibstatement über, durch das die logische Gesamtheit der eingegebenen Daten als Satz in der Datei, die wir TELEFON nennen wollen, aufgezeichnet wird (siehe Abb. 6.3).

Man sollte sich wirklich ein solches privates, durch den Computer unterstütztes Telefonverzeichnis einrichten. Es erweist sich als recht lehrreich und instruktiv. Wenn es darüberhinaus noch mit einem Suchprogramm, das wir später entwickeln werden, gekoppelt wird, können wir Adressen und Telefonnummern nachschauen, indem wir dafür den Computer einsetzen.

Testübung 6.3.1

Man benutzte das Programm von Abb. 6.3, um die folgenden Adressen in die Datei TELEFON aufzunehmen:

Marianne Schmitt
Bockenheimer Str. 17
8031 Gilching
Ortsteil Geisenbrunn
08116-6711

Martin Sperber
Rundplatz 34/II
8031 Gilching
Ortsteil Argelsried
08116-4711

6.3.3 Lesen von Datenelementen

Damit wir aus einer Datei die in ihr gespeicherten Daten lesen können, muß diese Datei zuvor für die Eingabe (INPUT) eröffnet werden. Wir wollen uns das am, im Beispiel 1 (Unterabschnitt 6.3.2) erstellten Telefonverzeichnisses ansehen. Die Eröffnung der Datei TELEFON besorgt das Statement

```
300  OPEN "TELEFON" FOR INPUT AS #2
```

Alternativ können wir auch die folgende Form der OPEN-Anweisung benutzen:

```
300  OPEN "I",#2,"TELEFON"
```

Bei dieser Form steht "I" für INPUT (durchs Programm angeforderte Eingabe). Durch die Zahl 2 ist die Referenznummer für die Datei TELEFON bei beiden Formen im Programm festgelegt.

Nach der Eröffnung können nun die in der Datei vorhandenen Daten mittels des Statements

```
400  INPUT#2, A$,B$,C$,D$,E$,F$
```

gelesen werden.[1] Durch diese Anweisung werden sechs Datenelemente aus der Datei gelesen, die mit den sechs Angaben einer Eintragung im Telefonverzeichnis übereinstimmen; dabei wird das erste Datenelement der Variablen A$ zugewiesen, das zweite B$ usw.

[1] Es ist nicht erforderlich, daß beim Lesen stets die gleichen Datenreihen aufgeführt werden wie beim Schreiben, d. h. eine durch ein WRITE-Statement in eine Datei geschriebene Datenreihe kann durchaus in mehreren aufeinanderfolgenden Statements gelesen werden, beispielsweise das erste Datenelement der geschriebenen Datenreihe durch das erste INPUT-Statement, die restlichen durch das zweite INPUT-Statement.

Um die Daten aus einer Datei korrekt lesen zu können, muß man das Format genau kennen, in dem die Daten in der Datei gespeichert sind. Die Form des o.a. INPUT-Statements wurde z.B. von der Tatsache bestimmt, daß jede Eintragung ins Telefonverzeichnis, in die Datei also, sich aus sechs aufeinanderfolgenden Zeichenkettenkonstanten zusammensetzt. Das INPUT-Statement arbeitet wie jedes andere Eingabestatement auch: Versehen mit einer Liste von durch Kommas getrennten Variablennamen weist es den aufgeführten Variablen Werte zu, und zwar in der Reihenfolge, in der die Datenelemente präsentiert werden. Immer, wenn man dabei versuchen sollte, eine Zeichenkettenkonstante einer numerischen Variablen oder umgekehrt eine numerische Konstante einer Zeichenkettenvariablen zuzuweisen, wird sich ein Fehler ergeben.

Solange eine Datei eröffnet ist, kann man mit der Eingabe der in ihr enthaltenen Datenelemente fortfahren, d.h. INPUT-Statements auf sie beziehen; man kann dabei soviele INPUT-Statements verwenden, wie man will. Diese wiederum können mit anderen Statements vermischt sein, die nichts mit dem Lesen von Datenelementen aus der betreffenden Datei zu tun haben. Jedes INPUT-Statement beginnt das Lesen aus der Datei an der Stelle, an der das vorhergehende INPUT-Statement mit dem Lesen aufgehört hatte.

Daraus erhebt sich die Frage, wie man bestimmen kann, ob alle Datenelemente gelesen wurden, d.h. die Frage nach dem D a t e i e n d e (engl.: end of file, abgekürzt EOF). In BASIC werden deshalb, um diese Frage entscheiden zu können, für jede eröffnete Datei Variablen mitgeführt; diese werden der Reihe nach

```
EOF(1), EOF(2), ···
```

genannt. Diese Variablen sind l o g i s c h e Variablen, sie können nur die Werte TRUE oder FALSE annehmen, d.h. „wahr" oder „falsch". Man kann diese Variablen für Prüfungen auf das Eintreten der Dateiendebedingungen heranziehen, indem man Statements der Form

```
IF        THEN ···
```

benutzt. Man betrachte hierzu beispielsweise die nachfolgende Anweisung:

```
100  IF  EOF(1)  THEN 2000  ELSE 10
```

Dieses Statement bewirkt, daß BASIC untersucht, ob in dem betreffenden Augenblick das Dateiende der Datei mit der Referenznummer 1 erreicht ist. Wenn das der Fall ist, verzweigt das Programm zum Statement auf der Zeile mit der Zeilennummer 2000, anderenfalls zur Zeile mit der Zeilennummer 10. Es sollte freilich dazu bemerkt werden, daß das Dateiende erst dann erreicht ist, wenn das letzte Datenelement gelesen worden ist. Man muß sich strikt einprägen, daß eine Prüfung auf das Dateiende unbedingt erforderlich ist, da jeder Versuch, nach Eintreten des Dateiendes weiter aus der betreffenden Datei lesen zu wollen, in einen Fehler einmünden wird.

Beispiel 3:

Es ist ein Programm zu schreiben, mit dem ermittelt werden kann, wieviele numerischen Werte in einer Datei mit Namen NUMMERN verzeichnet sind.

Um die Anzahl der Eintragungen bestimmen zu können, müssen wir einen Zähler in das Programm einführen; diesen wollen wir mit ZAEHL bezeichnen. Mit ihm können wir bei passender Programmierung die

Anzahl der bereits gelesenen Datenreihen verfolgen. Unser Lösungsprogramm wird also im Prinzip so aufzubauen sein, daß nach dem Lesen eines numerischen Wertes der Zähler zu erhöhen ist; anschließend ist eine Prüfung auf Dateiende vorzunehmen; man beachte die die Reihenfolge der Statements in der Lösung (siehe Abb. 6.4).

```
10  ZAEHL = 0
20  OPEN "NUMMERN" FOR INPUT AS #1
30  IF  EOF(1)  THEN 100
40  INPUT #1, A
50  ZAEHL = ZAEHL + 1
60  GOTO 30
100 PRINT "ANZAHL DER IN DER DATEI ENTHALTENEN ZAHLEN: ",ZAEHL
110 CLOSE
110 END
```

Abb. 6.4 Bestimmung der Anzahl der in einer Datei enthaltenen Eintragungen (Sätze)

Beispiel 4:

Es ist ein Programm zu schreiben, mit dem das von uns erstellte private Telefonverzeichnis (siehe das Beispiel 2 dieses Unterabschnittes) nach einer bestimmten Eintragung durchsucht werden kann.

Um das Durchsuchen einleiten zu können, müssen wir zunächst den Suchbegriff eingeben, was wir durch ein INPUT-Statement erreichen können. Als Suchbegriff benutzen wir den Namen. Nach der Eingabe des Namens soll das Programm solange die Datenelemente in der Datei nacheinander lesen, bis die Eintragung mit dem vorgegebenen Namen gefunden wird. Durch einen geeigneten Vergleich kann dieses Ziel erreicht werden. Das Lösungsprogramm ist in der Abb. 6.5 aufgelistet.

```
5    OPEN "TELEFON" FOR INPUT AS #1
10   INPUT  "BITTE DEN ZU SUCHENDEN NAMEN EINGEBEN:"; Z$
20   INPUT #1, A$, B$, C$, D$, E$, F$
30   IF  A$ = Z$  THEN 100  ELSE 40
40   IF  EOF(1)  THEN 200
50   GOTO 20
100  CLS
110  PRINT A$
120  PRINT B$
130  PRINT C$,D$
140  PRINT E$
150  PRINT F$
160  GOTO 1000
200  CLS
210  PRINT "DER GESUCHTE NAME BEFINDET SICH NICHT IN DER DATEI"
1000 CLOSE #1
1010 END
```

Abb. 6.5 Durchsuchen des privaten Telefonverzeichnisses nach einer bestimmten Eintragung (siehe auch Abb. 6.3)

Testübung 6.3.2

Das in der Abb. 6.5 dargestellte Programm soll dazu benutzt werden, um die Eintragung für „Martin Sperber" wiederaufzufinden.

Beispiel 5:

Bei diesem Beispiel wollen wir Adreßetiketten drucken lassen. Wir setzen hierbei voraus, daß wir das private Telefonverzeichnis mit dem Programm von Beispiel 2 (Abb. 6.3) erstellt haben und daß wir die entstandene Datei TELEFON genannt haben; sie befinde sich auf der Diskette im Laufwerk A: . Es ist nun ein Programm zu verfassen, das die Sätze dieser Datei fortlaufend liest und mit den Daten dieser Sätze Adreßetiketten druckt; die Telefonnummer soll auf den Adreßetiketten weggelassen werden.
Zur Entwicklung der Lösung wollen wir annehmen, daß die Adressen so zahlreich vorliegen, daß es sich lohnt, sie fortlaufend auf Endlospapier zu drucken. Im Handel kann man endloses Druckpapier erwerben, auf dem bereits sechszeilige Etiketten vorgestanzt sind (gummiert auf Folie). Da wir auf jedes Adreßetikett vier Zeilen, nämlich Name, Ortszusatz, Straße und Postleitzahl/Ort) zu drucken haben, müssen wir nach der Ortszeile zwei Leerzeilen erzeugen. Das unter diesen Aspekten entwickelte Programm finden wir in der Abb. 6.6.

```
10   OPEN "TELEFON" FOR INPUT AS #1
20   IF  EOF(1)  THEN 1000
30   INPUT #1, A$, B$, C$, D$, E$, F$
40   LPRINT A$              :'DRUCKEN DES NAMENS
50   LPRINT E$              :'DRUCKEN DES ORTSZUSATZES
60   LPRINT B$              :'DRUCKEN DER STRASSE
70   LPRINT C$              :'DRUCKEN DER POSTLEITZAHL (ORTSZEILE)
80   LPRINT TAB(8) D$       :'DRUCKEN DES ORTES (ORTSZEILE)
90   LPRINT:LPRINT:         :'ERZEUGUNG VON ZWEI LEERZEILEN
100  GOTO 20                :'UEBERGANG ZUM NAECHSTEN ADRESSETIKETT
1000 CLOSE 1
1010 END
```

Anmerkung: Die Telefonnummer, die beim Einlesen der Variablen F$ zugewiesen wurde, wird hier, entsprechend der Aufgabenstellung, nicht gedruckt.

Abb. 6.6 Programm zum Drucken von Adreßetiketten

6.3.4 Erweiterung von Dateien

Eine wichtige Tatsache vom Schreiben von Dateien darf nicht unerwähnt bleiben. Das Schreiben in eine Datei zerstört nämlich den bisherigen Inhalt der betreffenden Datei. Im Gegensatz dazu kann eine Datei beliebig oft gelesen werden, ohne daß ihr Inhalt verloren geht. Betrachten wir zur näheren Erläuterung wiederum die Datei TELEFON, für deren Erstellung wir im Beispiel 2 (Abb. 6.3.) ein Programm vorgelegt haben. Nehmen wir nun einmal an, daß wir ein Pro-

gramm schreiben, das diese Datei für die Ausgabe (OUTPUT) eröffnet. Anschließend wollen wir zusätzliche Eintragungen, von denen wir vermuten, daß sie noch nicht in das private Telefonverzeichnis aufgenommen sind, in die Datei einfügen. Nach Ablauf eines solchen Programmes werden wir feststellen müssen, daß in der Datei TELEFON nur die zusätzlichen Eintragungen enthalten sind; die ursprünglichen Eintragungen sind sämtlich verschwunden. Damit erhebt sich logischerweise von ganz allein die Frage, wie man zu einer bereits existierenden Datei Sätze hinzufügen kann. – Die Frage ist einfach zu klären. In der BASIC-Sprache für den IBM Personalcomputer gibt es eine spezielle Anweisung, die dieses Problem zu bewältigen hilft. Wir kennen diese Anweisung bereits im Prinzip, es ist die OPEN-Anweisung. Wir brauchen nur die Datei nicht mit OUTPUT zu eröffnen, sondern mit APPEND. Dadurch sagen wir aus, daß eine bestehende Datei erweitert werden soll. Für die Datei TELEFON könnte ein entsprechendes Statement wie folgt lauten:

```
500  OPEN "TELEFON" FOR APPEND AS #1
```

Durch dieses Statement erreicht man, daß der Computer die Datei auf ihr gegenwärtiges Ende einstellt. Alle zusätzlich in die Datei aufzunehmenden Eintragungen werden nun im Anschluß an diesen Einstellpunkt geschrieben; die zuvor in die Datei aufgenommenen Sätze bleiben dabei selbstverständlich ungeändert erhalten.

Beispiel 6:

Es ist ein Programm zu schreiben, mit dem zusätzliche Eintragungen in die Datei TELEFON aufgenommen werden können. Die Eingabe der zusätzlichen Daten soll über die Tastatur erfolgen, und zwar mittels von INPUT-Statements. Beim Schreiben des Programmes kann man davon ausgehen, daß sich die Datei TELEFON auf der Diskette im Laufwerk A: befindet.

Um zu einer bestehenden Datei Datenreihen hinzufügen zu können, muß diese Datei durch ein OPEN-Statement mit APPEND eröffnet werden. Danach kann durch eine entsprechende Aufforderung der Benutzer veranlaßt werden, die Daten der neuen Eintragung einzutippen; nach der Eingabe ist die neue Eintragung in die Datei zu schreiben. In der Abb. 6.7 ist das unter diesen Vorbedingungen geschriebene Programm aufgelistet.

```
10  OPEN "TELEFON" FOR APPEND AS #1
210 PRINT "BITTE EINGEBEN: NAME,STRASSE,POSTLEITZAHL,ORT,"
220 PRINT "                ORTSZUSATZ,TELEFONNUMMER"
230 INPUT A$, B$, C$, D$, E$, F$
240 WRITE #1, A$, B$, C$, D$, E$, F$
250 INPUT "SIND WEITERE EINTRAGUNGEN AUFZUNEHMEN? (J/N)"; Z$
260 IF  Z$ = "J"  THEN 300  ELSE 500
300 CLS
310 GOTO 210
500 CLOSE 1
510 END
```

Abb. 6.7 Programm zur Erweiterung einer bestehenden Datei

Im nächsten Beispiel stellen wir ein Programm vor, das für diejenigen Eltern von Nutzen ist, die ihren Kindern organisatorische Kenntnisse beibringen wollen. Die meisten Kinder lieben es,

mit dem Computer zu spielen. Das von uns präsentierte Programm kann als „elektronisches" Aufgabenheft dienen; es überwacht die Durchführung der Hausaufgaben. Es wurde von *Jonathan Goldstein,* einem 9-jährigen Computerfan, geschrieben und von *Horst Elmar* auf deutsche Verhältnisse übertragen.

Beispiel 7:

Es ist ein Programm zu schreiben, das eine Datei erstellt, in der die Hausaufgaben verzeichnet sind (Aufgabenheft). Das Kind sollte nach seiner Heimkehr von der Schule in diese Datei die ihm aufgegebenen Hausaufgaben fachbezogen eingeben können. Nach der Eintragung der neuen Hausaufgaben sollten die erledigten Aufgaben markiert werden können. Außerdem sollte das Programm dem Kind mitteilen können, inwieweit die Hausaufgaben durchgeführt worden sind.

Es ist sicher angebracht, vor der eigentlichen Programmierung einige Bemerkungen betreffs des Lösungsweges zu machen. Unser Programm wird zuerst fragen, ob die Hausaufgaben bereits in der Datei aufgezeichnet sind. Ist das der Fall, so braucht die Eingabe der Hausaufgaben nicht mehr zu erfolgen. Sind sie dagegen in der Datei noch nicht aufgezeichnet, so wird das Programm das Kind auffordern, die Hausaufgaben fachbezogen einzugeben, indem Fragen der folgenden Art an das Kind gestellt werden:

```
WELCHE HAUSAUFGABE IST IN MATHEMATIK ZU TUN?
```

Wenn keine Hausaufgabe für das betreffende Unterrichtsfach zu erledigen ist, braucht nur die Eingabetaste betätigt zu werden. Wenn keine Hausaufgaben mehr einzugeben sind, fragt der Computer beim Kind an, ob es sich die zu erledigenden Aufgaben ansehen will. Wird diese Frage vom Kind bejaht, erzeugt der Computer auf dem Bildschirm eine Liste der Fächer und der für diese Fächer erteilten Aufgaben (siehe Abb. 6.8).

```
FACH                ERTEILTE AUFGABE

DEUTSCH             AUFSATZGLIEDERUNG
MATHEMATIK          S. 345, AUFG. 2, 3, 4A
MUSIK               TEXT -HAENSCHEN KLEIN ...- LERNEN
```

Abb. 6.8 Aufgabenliste

Nunmehr gibt der Computer dem Kind die Chance, die von ihm bereits durchgeführten Aufgaben als erledigt zu markieren. Dabei läßt der Computer keine Vergeßlichkeit zu, indem er wie folgt vorgeht: Er fragt zunächst nach dem Fach, zeigt danach die für dieses Fach vorliegende Aufgabe an und fragt abschließend, ob diese Aufgabe auch wirklich erledigt ist. Wenn das der Fall ist, werden beim nächsten Anzeigen der Aufgabenliste die erledigten Aufgaben in einer gesonderten Spalte durch den Buchstaben X markiert. Zum Schluß überprüft der Computer die Aufgabenliste, ob alle Aufgaben durchgeführt worden sind. Wenn das der Fall ist, wird als Schlußnachricht der Text

```
HAUSAUFGABEN ERLEDIGT
```

ausgegeben, anderenfalls der Text

```
HAUSAUFGABEN NICHT ERLEDIGT !!!
```

Das unter diesen Überlegungen entstandene Programm ist in der Abb. 6.9 aufgelistet.

```
20  DIM  B$(20),C$(20),D$(20)
30  CLS
40  PRINT  "HAST DU DIE HAUSAUFGABEN BEREITS EINGEGEBEN? (J/N):"
50  INPUT A$            :'Eingabe der Antwort
60  IF  A$ = "J"  THEN 70  ELSE 140
70  OPEN "AUFGABEN" FOR INPUT AS #1
80  PRINT  "FACH"; TAB(20) "ERTEILTE AUFGABE"
90  FOR  J = 1  TO  7
100   INPUT #1, B$(J),C$(J),D$(J)
110   PRINT  B$(J); TAB(20) C$(J); TAB(60) D$(J)
120 NEXT J
130 CLOSE
135 GOTO 370
140 CLS
150 DATA  "DEUTSCH","ERDKUNDE","MATHEMATIK","BIOLOGIE","GESCHICHTE",
160 DATA  "MUSIK","ZEICHNEN","SONSTIGES"
170 FOR  J = 1  TO  7
180   READ B$(J)
190   PRINT  "HAST DU FUER DIESES FACH HAUSAUFGABEN ZU MACHEN? (J/N):"
200   INPUT A$          :'Eingabe der Antwort
210   IF  A$ = "J"  THEN 220  ELSE 250
220   PRINT  "WELCHE HAUSAUFGABE IST IN "; B$(J); " ZU TUN?"
230   INPUT C$(J)
250 NEXT J
260 PRINT  "WILLST DU DEINE HAUSAUFGABEN ANSEHEN? (J/N):"
270 INPUT A$            :'Eingabe der Antwort
280 IF  A$ = "J"  THEN 300  ELSE 370
300 CLS
310 PRINT  "FACH","TAB(20) "ERTEILTE AUFGABE"
320 PRINT
330 FOR  J = 1  TO  7
340   PRINT  B$(J); TAB(20) C$(J)
350 NEXT J
370 PRINT  "WILLST DU PRUEFEN, OB EINE AUFGABE ERLEDIGT IST? (J/N):"
380 INPUT A$            :'Eingabe der Antwort
400 IF  A$ = "J"  THEN 410  ELSE 520
410 INPUT  "FACH"; B$
420 FOR  J = 1  TO  7
440   PRINT  "HAST DU DIESE AUFGABE ERLEDIGT? (J/N):"
450   PRINT  C$(J)
460   INPUT  A$
470   IF  A$ = "J"  THEN  D$(J) = "X"
475   IF  B$ = B$(J)  THEN PRINT B$(J)  ELSE NEXT J
480 CLS
490 FOR  J = 1  TO  7
500   PRINT  B$(J); TAB(20) C$(J); TAB(60) D$(J)
510 NEXT J
520 FOR  J = 1  TO  7
530   IF   D$(J) <> "X"  AND  C$(J) <> ""   THEN E$="W"
540 NEXT J
550 IF  E$ = "W"  THEN PRINT "HAUSAUFGABEN NICHT ERLEDIGT !!!"  ELSE 572
560 PRINT  "WILLST DU DIE ERLEDIGUNG EINER ANDEREN AUFGABE PRUEFEN? (J/N):"; A$
570 IF  A$ = "J"  THEN 380  ELSE 580
572 PRINT  "HAUSAUFGABEN ERLEDIGT"
574 GOTO 630
580 OPEN "AUFGABEN" FOR OUTPUT AS #1
590 FOR  J = 1  TO  7
600   WRITE #1,  B$(J), C$(J), D$(J)
610 NEXT J
620 CLOSE #1
630 PRINT  "PROGRAMMENDE"
700 END
```

Abb. 6.9 Programm zur Überwachung der Hausaufgaben

Aufgabengruppe 16

1. Es ist ein Programm zu schreiben, das eine Datei auf einer Diskette erstellt. Die Datei soll die folgenden Zahlen enthalten:
   5.7, −11.4, 123, 485, 49

2. Es ist ein Programm zu schreiben, das die Datei liest, die in der Aufgabe 1. erstellt wurde, und die gelesenen Zahlen auf dem Bildschirm anzeigt.

3. Es ist ein Programm zu schreiben, mit dem die Datei, die in der Aufgabe 1. erstellt wurde, erweitert werden kann. Es sollen die folgenden Zahlen zusätzlich in die Datei aufgenommen werden:
   5, 78, 4.79, −1.27

4. Es ist ein Programm zu schreiben, das die Datei liest, die in der Aufgabe 1. erstellt wurde, und die gelesenen Zahlen auf dem Bildschirm anzeigt.

5. Es ist ein Programm zu schreiben, mit dem die Inhalte der in den Scheckbüchern verbliebenen Abschnitte in eine Datei aufgezeichnet werden können. In die Datei sind die folgenden Daten aufzunehmen:

   Schecknummer, Ausstellungsdatum, Zahlungsempfänger, Betrag, Verwendungszweck

   Mit Hilfe des Programms ist anschließend eine Datei zu erstellen, in die die vom Benutzer ausgestellten Schecks des vergangenen Monats aufgenommen werden sollen.

6. Es ist ein Programm zu schreiben, mit dem der Inhalt der in der Aufgabe 5 erstellten Datei gelesen werden kann. Es ist die Gesamtsumme aller in der Datei verzeichneten Schecks zu ermitteln und auf dem Bildschirm anzuzeigen.
   Anmerkung: Der letzte Satz in der Datei SCHECKS enthält die „künstliche Schecknummer" −999; dieser Satz ist selbstverständlich nicht zu verarbeiten, er signalisiert bloß das Dateiende.

7. Für ein Einzelhandelsgeschäft ist ein Programm für die Lagerverwaltung zu schreiben. Die Lagerbewegungen sollen dabei in einer Datei fortgeschrieben werden; jede einzelne Eintragung in dieser Datei soll die folgenden Daten enthalten:

   Artikelbezeichnung, Einzelpreis, Menge

   Das Programm soll drei verschiedene Aktivitäten ermöglichen:
   — Anzeige der in der Datei gespeicherten Daten für einen bestimmten Artikel
   — Aufzeichnung des Wareneingangsdaten für einen bestimmten Artikel
   — Aufzeichnung des Verkaufsdaten für einen bestimmten Artikel
   Anmerkung: Der letzte Satz in der Datei LAGER enthalte als Artikelbezeichnung "ENDE"; die Abfrage nach dieser Bezeichnung signalisiert im bejahenden Falle also das Dateiende.

8. Es ist ein Programm zu schreiben, durch das eine Datei erstellt werden kann, in der die Lieblingsrezepte für Getränke verzeichnet sind.

9. Es ist ein Programm zu schreiben, durch das eine Schülerdatei erstellt werden kann, in der die für jeden Schüler Name, Klasse, Unterrichtsfach, Fehltage und Zensuren (max. 8) verzeichnet sind. Anfangs sind die Zensuren auf 0 zu setzen.
   (Aufgabe nur für Lehrer)

10. Es ist eine Datei zu erstellen, in der die folgenden Kreditkartendaten verzeichnet sind:
    - Kreditkartennummer
    - Angaben über die Person oder die Kreditanstalt, die bei Verlust oder Diebstahl zu benachrichtigen sind:
      — Name
      — Ortsteil
      — Straße
      — Postleitzahl
      — Ort

Antworten auf die Testübungen

6.3.1 Bei der Eingabe der Daten vergesse man nicht, nach jedem Eintippen die Eingabetaste zu drücken.

6.3.2 Nach der geforderten Eingabe ist die Eingabetaste zu betätigen.

6.4 Ergänzende Betrachtungen über sequentielle Dateien

Wenn Datenelemente in eine sequentielle Datei ausgegeben werden, fügt BASIC automatisch gewisse Interpunktionszeichen ein; dadurch ist es möglich, die Daten wieder zu lesen. Im einzelnen handelt es sich um die folgenden Interpunktionen:

1. Zeichenketten werden von Anführungszeichen umgeben.
2. Datenelemente werden voneinander durch Kommas getrennt.
3. Dem letzten Datenelement im WRITE-Statement folgt bekanntlich das Drücken der Eingabetaste. Für den Computer bedeutet das Drücken dieser Taste die Eingabe eines Zeichens; dieses Zeichen wollen wir als ENTER-Zeichen bezeichnen oder einfach kurz als ENTER. ENTER, das sei noch einmal betont, ist für den Computer ein Zeichen wie jedes andere auch, z.B. "A" oder ";". Das ENTER-Zeichen sagt dem Computer, daß das Ende der laufenden Zeile vorliegt; es veranlaßt den Positionsanzeiger oder Cursor, sich auf den Anfang der nächsten Zeile zu begeben. Das ENTER-Zeichen kann wie jedes andere Zeichen in eine Datei aufgenommen werden.
4. Positive Zahlen werden in eine Datei ohne führendes Leerzeichen eingefügt.

Wir wollen z.B. einmal annehmen, daß die folgenden Datenelemente vorliegen:

```
A$="JOHN", B$="SCHMITT", C=1234, D=-14
```

Durch das WRITE-Statement

```
100  WRITE #1, A$,B$,C,D
```

werden diese Daten wie folgt in der Datei mit der Dateinummer 1 aufgezeichnet:

```
„JOHN",„SCHMITT",1234,-14<E>
```

Unter <E> wollen wir von jetzt ab das ENTER-Zeichen verstehen.

Wenn diese Gesamtheit von Daten durch ein mit einer Dateinummer versehenem INPUT-Statement gelesen werden, bewirken die Anführungszeichen, die Kommas und das Zeichen <E>, daß BASIC in die Lage versetzt wird, die verschiedenen Datenelemente voneinander zu trennen. Aus diesem Grunde werden die Interpunktionszeichen auch Trennungszeichen oder Trennzeichen (Separatoren) genannt.

Wenn man das WRITE-Statement benutzt, braucht man sich nicht um Trennzeichen zu kümmern. Bei anderen, auf sequentielle Dateien bezogene Anweisungen ist man jedoch nicht so glücklich.

Man betrachte z.B. das mit einer Dateinummer versehene PRINT-Statement. Diese Anweisung kann zur Druckausgabe von Daten in eine Datei benutzt werden, gerade so, als ob man die Daten auf den Bildschirm ausgibt. Alle Besonderheiten, die für die auf Bildschirme bezogenen PRINT-Statements wie TAB, SPC und Semikolons, sind auch bei der PRINT-Ausgabe in Dateien aktiv. Allerdings schließt das mit einer Dateinummer versehene PRINT-Statement keine Trennzeichen (Separatoren) ein. Um das näher zu erläutern, wollen wir das obige Beispiel mit den Variablen A$, B$, C und D wieder aufgreifen.

Das Statement

```
20  PRINT #1, A$;B$;C;D
```

schreibt die folgende Datenreihe in die Datei mit der Dateinummer 1:

```
JOHNSCHMITT 1234-14<E>
```

Es ist also folgendes zu beachten:

1. Das Leerzeichen vor der positiven Zahl 1234 wird in die Datenreihe aufgenommen,
2. Zwischen den einzelnen Datenelementen gibt es keine Trennungen,
3. Die Zeichenkettenkonstanten sind nicht von Anführungszeichen umgeben.

Damit die einzelnen Datenelemente korrekt gelesen werden können, muß man in der Datenliste des mit einer Dateinummer versehenen PRINT-Statements Trennzeichen mit aufnehmen. An unserem obigen Beispiel wollen wir das demonstrieren (auszugebende Kommas müssen dabei selbstverständlich erscheinen):

```
20  PRINT #1, A$;",";B$;",";C:","D
```

Nunmehr sieht die ausgegebene Datenreihe wie folgt aus:

```
JOHN,SCHMITT, 1234,-14<E>
```

Jetzt können die einzelnen Datenelemente in gewohnter Weise gelesen werden.

Noch sind wir mit unserer Besprechung nicht am Ende, beileibe nicht. Man wird es schon bemerkt haben: Die Zeichenkettenkonstanten sind nicht von Anführungszeichen umgeben. Hier wird uns kein Kummer daraus erwachsen. Um das einzusehen, wollen wir etwas über die Wirkungsweise des mit einer Dateinummer versehenen INPUT-Statements sprechen.

Die INPUT-Anweisung erkennt die folgenden Zeichen als Trennzeichen:

- Kommas
- <E>
- Formularvorschub (über dieses Steuerzeichen werden wir uns später unterhalten)

Wenn sich ein, eine Dateinummer enthaltendes INPUT-Statement einer Datenreihe gegenübersieht, werden die folgenden Operationen ausgeführt:

1. Die Zeichen werden untersucht und ins augenblickliche Datenelement weggelegt, es sei denn, es liegt ein Trennungszeichen vor. Ein solches zeigt nämlich nun das Ende des au-

genblicklichen Datenelementes an; das Trennungszeichen selbst wird nicht ins Datenelement aufgenommen.

2. Wenn ein numerisches Datenelement in der Datenliste von INPUT angefordert wird, erfolgt eine Prüfung, die sicherstellt, daß das Datenelement eine Zahl ist (es dürfen keine für Zahlen ungültige Zeichen, wie A oder ; auftreten). Wenn ungültige Zeichen festgestellt werden, so wird der Fehler „ungeeigneter Datentyp" signalisiert.
3. Wenn ein Zeichenkettendatenelement in der Datenliste von INPUT angefordert wird, wird geprüft, ob das Datenelement von Anführungszeichen umgeben ist. Ist das der Fall, so werden diese entfernt.

Das Verständnis der soeben beschriebenen Prüfreihenfolge kann vor peinlichen Fehlern bewahren. Zu einem solchen Fehler kommt es, wenn beispielsweise ein Komma in ein Datenelement eingeschlossen ist. Nehmen wir einmal an, daß die folgenden Wertzuweisungen vorliegen:

```
A$="SCHMITT,JOHN"
B$="ZIMMERMANN"
```

Das PRINT-Statement

```
30  PRINT #1, A$;",";B$
```

schreibt die folgende Datenreihe in die Datei 1:

```
SCHMITT,JOHN,ZIMMERMANN<E>
```

Das nachfolgende, auf eine Datei bezogene INPUT-Statement

```
40  INPUT #2, A$,B$
```

führt zu den Zuweisungen

```
A$="SCHMITT"
B$="JOHN"
```

Die herausgeschriebene Zeichenfolge

```
SCHMITT,JOHN
```

ist also nach dem Einlesen zwei Zeichenkettenvariablen zugewiesen und nicht, wie sicher beabsichtigt, nur einer. Um dieses Problem zu umgehen, müssen explizit Anführungszeichen um die Zeichenkettenkonstanten gestellt werden, die Kommas enthalten. Eine Zeichenkettenkonstante, die aus dem Anführungszeichen selbst besteht, ist durch CHR$(34) auszudrücken; mehr darüber wird später (Kapitel 7) gesagt. Um Anführungszeichen um die Zeichenkettenkonstante

```
SCHMITT,JOHN
```

zu stellen, muß also das fragliche Ausgabestatement zu

```
50  PRINT #1, CHR$(34);A$;CHR$(34);",";B$
```

codiert werden.

In der Datei wird nunmehr die Datenreihe

```
"SCHMITT,JOHN",ZIMMERMANN<E>
```

eingetragen.

Fassen wir zusammen: Zeichenkettenkonstanten, die Kommas, Semikolons, Leerzeichen am Anfang, Leerzeichen am Ende und/oder <E> enthalten, müssen stets von Anführungszeichen umgeben sein.

Wie wir aus den soeben gemachten Ausführungen erkennen können, sind die auf Dateinummern bezogenen PRINT-Statements weitaus weniger bequem zu gebrauchen, als die entsprechenden WRITE-Statements. In vielen Fällen ist es eben einfacher, WRITE-Statements einzusetzen. Jedoch besitzen auch die mit einer Dateinummer versehenen PRINT-Anweisungen ihre Vorteile. In diese kann man nämlich die USING-Option aufnehmen und damit die Daten formatieren. Um beispielsweise den Wert der Variablen A in die Datei 1 im Format ##.# auszugeben, können wir codieren

```
60  PRINT #1, USING ##.#; A
```

Das auf Dateien bezogene INPUT-Statement liest immer nur ein einzelnes Datenelement zu einer Zeit. Bei einer Reihe von Anwendungen will man aber gleichzeitig eine ganze Zeile aus einer Datei lesen, d.h. man will alle Daten bis zum Zeichen <E> lesen. Man kann das erreichen, indem man das Statement LINE INPUT verwendet. Angenommen, in der Datei 1 sei die folgende Eintragung enthalten:

```
SCHMITT,JOHN,ZIMMERMANN<E>
```

Die Anweisung

```
70  LINE INPUT #1, A$
```

bewirkt, daß A$ die Zeichenkettenkonstante

```
"SCHMITT,JOHN,ZIMMERMANN"
```

zugewiesen wird. Man beachte hierbei jedoch den nachstehend beschriebenen kuriosen Zwiespalt. Wenn man Zeichenkettendaten mit umgebenden Anführungszeichen sichergestellt, d.h. in eine Datei geschrieben hat, werden diese durch das Statement

```
z  LINE INPUT #n, ···
```

als Teil der Zeichenkettenkonstanten gelesen.
z bedeutet hier die Zeilennummer und n die Dateinummer. Wenn man also plant, ganze Zeilen mittels des Statements LINE INPUT zu lesen, ist es sicher ratsam, bei der vorhergehenden Sicherstellung die auf eine Datei bezogene PRINT-Anweisung zu benutzen; dadurch werden keine gesonderten Anführungszeichen generiert. Wir selbst werden das mit einer Dateinummer versehene Statement LINE INPUT dann benutzen, wenn wir unseren Textprozessor in Kapitel 9 erarbeiten.

6.4.1 Dateipuffer

Nach kurzer Zeit wird man festgestellt haben, daß die Anzeigelampe des angesprochenen Diskettenlaufwerkes nicht immer aufleuchtet, wenn eine Datei geschrieben wird. Man verfolge z.B. dieses Experiment: Man eröffne eine Datendatei und schreibe ein einzelnes Datenelement in diese Datei, schließe diese Datei jedoch nicht ab. Am Diskettenlaufwerk geschieht nichts. Wenn man jedoch dieses simple Programm ein zweites Mal ablaufen läßt, wird die Anzeigelampe aufleuchten. Das kommt dem Benutzer ziemlich befremdend vor. Doch ist diese Erscheinung leicht zu erklären, wenn man weiß, wie BASIC Dateien auf Disketten schreibt bzw. von Disketten liest.

Diskettenlaufwerke sind im Vergleich zu den von BASIC ausgeführten, nicht auf Disketten bezogenen Operationen, sehr langsam. Um Diskettenoperationen zu beschleunigen, werden beim Schreiben auf Disketten sogenannte Dateipuffer zwischengeschaltet. Ein Dateipuffer, kurz Puffer genannt, ist weiter nichts als ein Gebiet in RAM, in das BASIC vorübergehend die in eine Datei zu schreibenden Daten speichert. Zu jeder eröffneten Datei gehört ein Puffer. BASIC reserviert den dafür benötigten Speicherplatz während der Eröffnungsoperationen, die durch das OPEN-Statement veranlaßt werden. Wenn nun eine auf eine Datei bezogene Schreiboperation ausgeführt wird, stellt BASIC die entsprechenden Informationen in den Puffer der betreffenden Datei. Erst wenn der Puffer gefüllt ist, werden die Daten von BASIC in die Datei geschrieben.

Die durch das CLOSE-Statement ausgelösten Operationen veranlassen u.a., daß die Puffer (voll oder nicht voll) in die ihnen entsprechenden Dateien geschrieben werden. Wenn eine Datei nun, wie beim oben beschriebenen Experiment, in einem Programm nicht abgeschlossen wird, bleibt der zu ihr gehörende Puffer erhalten und enthält u.U. einige in ihn bereits hineingestellte Datenelemente, die also noch nicht auf die Diskette übertragen worden sind. In diesem Fall verursachen RUN oder END, daß die Pufferinhalte, die zu den nicht abgeschlossenen Dateien gehören, in die entsprechenden Dateien übertragen werden. Sobald man ein Programm im RAM ändert, werden ebenfalls die Pufferinhalte auf die Diskette(n) geschrieben. Bei unserem Experiment verursachte der RUN-Befehl, daß die Anzeigelampe aufleuchtete; die Ergebnisse der vorhergehenden Programmausführung werden zu diesem Zeitpunkt auf Diskette geschrieben.

Aufgabengruppe 17

Es werde vorausgesetzt, daß folgende Wertzuweisungen zu Variablen vorliegen:

```
A$="MY"
B$="DOG"
C$="SAM"
D=1234
```

In welchem Format werden diese Datenelemente in die Datei 1 durch die nachfolgenden Statements geschrieben?

```
1.  WRITE #1, A$,B$,C$,D
2.  PRINT #1, A$,B$,C$,D
3.  PRINT #1, A$;",";B$;",";C$;",";D
4.  PRINT #1, CHR$(34);A$;" ";B$;", ";CHR$(34);",";C$;",";D
```

Man betrachte die Datei, die durch das Statement von Aufgabe 1. geschrieben wurde. Wie sieht die Ausgabe auf dem Bildschirm aus, die durch die nachstehenden Anweisungen hervorgerufen wird?

```
5.  INPUT #1,E$:      PRINT E$
6.  LINE INPUT #1, E$:      PRINT E$
```

Man betrachte die Dateien, die durch die Statements der Aufgabe 2. bis 4. geschrieben wurden. Wie sieht jeweils die durch die nachfolgenden Anweisungen hervorgerufene Ausgabe aus?

```
7.  INPUT #1,E$:      PRINT E$
```

Man betrachte die Datei, die durch das Statement von Aufgabe 4. geschrieben wurde. Es ist ein Programm zu schreiben, durch das auf dem Bildschirm der folgende Text angezeigt wird:

```
8.  MY DOG, SAM
    1234
```

6.5 Dateien mit wahlfreiem oder direktem Zugriff (Direktzugriffsdateien)

Die bisher in diesem Kapitel behandelten Dateien waren Beispiele für sequentielle Dateien. Im Klartext bedeutete das, daß die Dateien vom Anfang bis zum Ende hintereinander mit Daten beschrieben wurden. Diese Art von Dateien läßt sich zwar sehr einfach erstellen, sie sind aber für viele Anwendungen nur umständlich und beschwerlich zu handhaben, da die in ihnen enthaltenen Datenelemente nur sequentiell gelesen werden können. Wenn aber z. B. nur einige Daten braucht, die am Ende einer solchen sequentiellen Datei stehen, muß man alle Datenelemente vom Anfang der Datei bis zur fraglichen Stelle lesen, bevor man endlich zu den gewünschten Datenelementen zugreifen kann. Dateien mit direktem Zugriff, kurz Direktzugriffsdateien, umgehen diese Schwierigkeiten. Bei der Benutzung von Direktzugriffsdateien ist es möglich, exakt zu den Datenelementen zuzugreifen, die man für die Bearbeitung benötigt. Natürlich muß man für die Bequemlichkeit einen angemessenen Preis bezahlen. Kein lukullisches Mahl ist umsonst zu haben! Wir müssen schon ein wenig härter arbeiten, um den Gebrauch von Direktzugriffsdateien zu erlernen.

Eine Direktzugriffsdatei ist in Segmente fester Länge unterteilt, die wir als Regionen oder Satzbereiche bezeichnen wollen (siehe Abb. 6.10.). Die Länge eines Satzbereiches wird in Stellen oder Bytes angegeben. Bei einer Zeichenkettenkonstanten, einschließlich des Leerzeichens und der Interpunktionszeichen, belegt jedes Zeichen gerade ein Byte. So hat z. B. die Zeichenkette

```
RECHNUNG-62345
```

eine Länge von 14 Stellen (Bytes).

Um ein Datenelement in eine Direktzugriffsdatei speichern zu können, muß es stets in Zeichenform (Kettenform) vorliegen, d. h. erforderlichenfalls in diese Form umgewandelt werden. Die Umwandlung ist also bei numerischen Konstanten und den Werten numerischer Variablen vorzunehmen. Zur Ausführung dieser Umwandlungen stehen spezielle Anweisungen zur Verfügung. Eine Zahl, genauer gesagt, eine Zahl einfacher Genauigkeit wird in eine vier Bytes lange Kette überführt, gleichgültig, wieviel Ziffernstellen die Zahl selbst besitzt. Ein Satz möge aus den vier Datenelementen "RECHNSTELL", 5000, 0.735 und 7876 bestehen. Diese vier Daten werden lückenlos aufeinanderfolgend in der Datei gespeichert. Die Länge dieses Satzes beträgt demnach 22 Bytes (10 Stellen für die Zeichenkettenkonstante und 3 mal 4 Bytes für jede numerische Konstante); man betrachte hierzu die Abbildung 6.11.

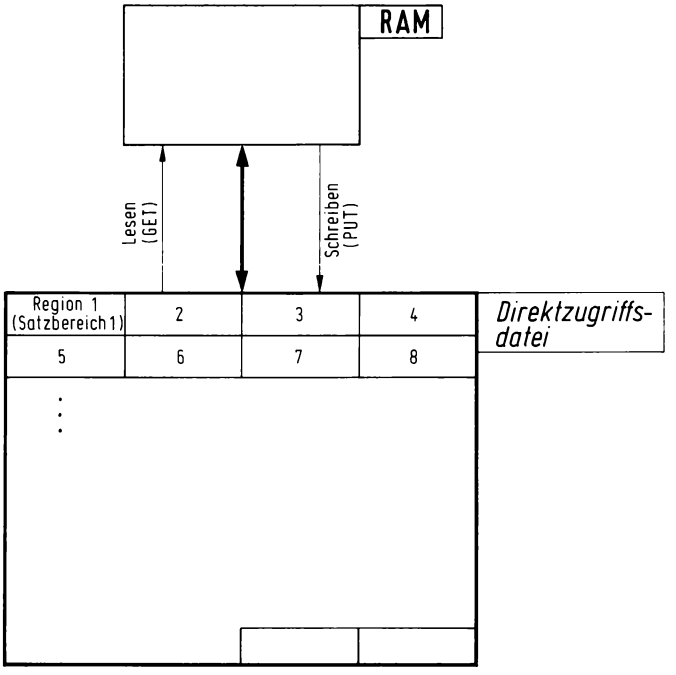


Abb. 6.10 Schematische Darstellung einer Direktzugriffsdatei

Feld 1										Feld 2				Feld 3				Feld 4				
Abteilungsbezeichnung										*Einnahmen*				*Rationalisierungsgrad*				*Unkosten*				
(*Feld:* ABTBEZ$)										(*Feld:* EINNAHME$)				(*Feld:* RATIONGRD$)				(*Feld:* UNKOST$)				
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	*Byte*

Abb. 6.11 Typischer Satz einer Direktzugriffsdatei (Beispiel)

Um Daten in eine Direktzugriffsdatei schreiben zu können, muß diese in gewohnter Weise zunächst eröffnet werden. Um eine Datei zu eröffnen, die "ABTEIL" heißen und von der Organisationsform her eine Direktzugriffsdatei mit Sätzen von 22 Bytes Länge sein soll, gebrauchen wir die Anweisung

```
10 OPEN "ABTEIL" AS #1, LEN=22
```

Als nächstes müssen wir den Aufbau der Sätze beschreiben, die zur Datei gehören sollen. Nehmen wir z. B. an, daß jeder Satz der Datei 1 mit einer 10-stelligen Zeichenkette beginnt; auf diese Kette folgen drei, in Zeichenform umgewandelte Zahlen. Nehmen wir weiterhin an, daß die Zeichenkette die Abteilungsbezeichnung darstellt, die erste Zahl die Einnahmen der be-

treffenden Abteilung, die zweite Zahl den Rationalisierungsgrad und die dritte Zahl die laufenden Unkosten. Diese Sachlage beschreiben wir durch das Statement

```
20  FIELD #1, 10 AS ABTBEZ$, 4 AS EINNAHME$,
              4 AS RATIONGRD$, 4 AS UNKOST$
```

Dieses Statement ist der Datei zugeordnet, die durch die Dateinummer identifiziert wird, die beim Eröffnen vergeben wurde. Jeder Satzteil wird als Feld (engl.: field) bezeichnet. Jedes Feld wird durch den Namen einer Zeichenkettenvariablen und durch seine in Bytes angegebene Länge bestimmt.

Um einen Satz in eine Direktzugriffsdatei zu schreiben, muß man zunächst die entsprechenden Daten in die einzelnen Felder hineinstellen. Dies erfolgt mittels der Anweisungen LSET und RSET. Um beispielsweise das Feld ABTBEZ$ mit der Zeichenkettenkonstanten "RECHNSTELL" zu versorgen, verwenden wir die Anweisung

```
30  LSET ABTBEZ$="RECHNSTELL"
```

Wenn wir dieses Feld mit dem Wert versorgen wollen, der den Zeichenkettenvariablen N$ zugewiesen ist, haben wir

```
40  LSET ABTBEZ$=N$
```

zu codieren. Wenn hierbei der Wert von N$ weniger als zehn Zeichen umfaßt, wird das Feld rechtsbündig mit Leerzeichen aufgefüllt. Man spricht deshalb von einer „linksbündigen Ausrichtung" des Feldes. Enthält der Wert von N$ dagegen mehr als 10 Zeichen, so wird das Feld nur mit den 10 ersten Zeichen von N$ gefüllt.

Die Anweisung RSET wirkt ähnlich wie die Anweisung LSET; die Leerzeichen erscheinen hier jedoch am linken Ende des Feldes. Man spricht hier deshalb von einer „rechtsbündigen Ausrichtung" des Feldes.

Um Zahlen (numerische Konstanten) in Zeichenfolgen (Ketten) zwecks Aufnahme derselben in Direktzugriffsdateien umzuwandeln, benutzen wir die Funktion MKS$. Um z.B. die Zahl 0.735 in das mit RATIONGRD$ bezeichnete Feld hineinzustellen, verwenden wir diese Konstante als Argument der Funktion MKS$, d.h. wir schreiben MKS$(0.735). Um den Wert der numerischen Variablen EINN in eine Kette umzuwandeln, genügt die Codierung von MKS$(EINN). Nach der Umwandlung bedienen wir uns der Anweisung LSET (bzw. RSET), um die fraglichen Felder des definierten, zur Datei gehörenden Satzes zu füllen. Für die beiden soeben aufgeführten Beispiele lauten infolgedessen die Statements wie folgt:

```
50  LSET RATIONGRD$=MKS$(0.735)
60  LSET EINNAHME$=MKS$(EINN)
```

Hier eine Warnung auszusprechen, ist sicher äußerst sinnvoll. Man versuche niemals, Felder eines Satzes mithilfe der LET-Anweisung zu füllen. Die Anweisung

```
70  LET RATIONGRD$="0.735"
```

z.B. wird katastrophale Folgen für das betreffende Programm zeitigen.

Nachdem die Felder eines bestimmten Satzes mittels der Anweisungen LSET bzw. RSET gesetzt worden sind, kann man den Satz mittels der Anweisung PUT in die zugehörige Datei schreiben. Die zu einer Direktzugriffsdatei gehörenden Sätze sind fortlaufend numeriert, beginnend mit 1. Diese Nummern wollen wir als Satznummern oder Regionsnummern bezeichnen. Als wesentlichste Eigenschaft einer Direktzugriffsdatei erweist sich nun, daß man Informationen als Sätze gezielt in eine bestimmte Region aufzeichnen bzw. gezielt von einer bestimmten Region lesen kann. Um z. B. die gegenwärtig den Feldern des Satzes zugewiesenen Daten in den Satzbereich (in die Region) 38 der Datei 1 zu schreiben, ist das Statement

```
80  PUT #1, 38
```

zu codieren.

Testübung 6.5.1

Es ist ein Programm zu schreiben, das eine Direktzugriffsdatei mit vier Sätzen erstellt. Die vier einzelnen Sätze sollen sich aus den folgenden Daten zusammensetzen:

RECHNSTELL	5000	0.735	7876
KONSTRUKT	3500	0.872	2200
VERWALTUNG1	4338	0.381	5130
WERBUNG	10832	0.95	12500

Wenn man Sätze aus einer Direktzugriffsdatei lesen will, muß man die betreffende Datei zunächst durch eine Anweisung der Form

```
90  OPEN "ABTEIL" AS #1, LEN=22
```

eröffnen. Man erkennt ohne weiteres, daß diese Form für uns nicht neu ist. Die OPEN-Anweisung sieht also immer gleich aus, es spielt keine Rolle, ob die Direktzugriffsdatei geschrieben oder gelesen werden soll. In dieser Hinsicht unterscheiden sich die Direktzugriffsdateien von den sequentiellen Dateien. Das Eröffnen einer Direktzugriffsdatei präpariert diese sowohl für das Schreiben als auch für das Lesen. Vor dem Abschließen einer solchen Datei kann man also sowohl Sätze schreiben als auch Sätze lesen.

Der nächste Schritt stellt auch hier die Definition des Satzaufbaus dar, wozu man sich das FIELD-Statements bedient. Ein Beispiel möge das ganze noch einmal untermauern:

```
100 FIELD #1, 10 AS ABTBEZ$, 4 AS EINNAHME$,
              4 AS RATIONGRD$, 4 AS UNKOST$
```

Auch dieses Statement gleicht völlig dem Statement, das wir beim Schreiben in eine Datei verwendet haben. Es sei denn, daß dieses FIELD-Statement durch ein anderes außer Kraft gesetzt wird, gilt es also sowohl für das Schreiben als auch für das Lesen, wobei als Bezugspunkt die Datei gilt, der durch das OPEN-Statement die Dateinummer 1 zugeordnet wurde.

Um eine Leseoperation aufzusetzen, gebrauchen wir die Anweisung

```
110 GET #1, 4
```

Den Variablen ABTBEZ$, EINNAHME$, RATIONGRD$ und UNKOST$ werden dadurch die Werte zugewiesen, die im Satz mit der Satznummer 4 der Datei 1 stehen. Wir könnten nun diese Werte auf dem Bildschirme anzeigen lassen, beispielsweise den Wert von ABTBEZ$ durch das Statement

```
120 PRINT ABTBEZ$
```

Wenn wir aber beispielsweise den Wert von RATIONGRD$ in eine arithmetische Anweisung oder in eine PRINT-Anweisung eingehen lassen wollen, müssen wir ihn zuallererst wieder in die numerische Form zurück umwandeln. Um dies zu bewerkstelligen, bedienen wir uns der CVS-Funktion. Diese Funktion können wir in jedes Statement hineinstellen. Das Statement

```
130 PRINT CVS(RATIONGRD$)
```

bewirkt somit die Anzeige des Wertes von RATIONGRD$ auf dem Bildschirm, während die Anweisung

```
140 LET  N = 100*CVS(RATIONGRD$)
```

der numerischen Variablen N den 100-fachen Betrag des numerischen Äquivalents von RATIONGRD$ zuweist.

Es sollte unbedingt bemerkt werden, daß die Variablen, mit denen die Felder benannt werden, wie hier u.a. ABTBEZ$ und RATIONGRD$, stets die Werte enthalten, die durch das letzte GET-Statement aus der Datei gelesen wurden. Wenn man deshalb gleichzeitig die Daten aus mehreren Sätzen handhaben, beispielsweise miteinander vergleichen muß, ist es notwendig, die Werte, die durch das vorhergehende GET-Statement gelesen wurden, zuvor anderen Variablen zuzuweisen, bevor man ein neues GET-Statement hinausschickt.

Testübung 6.5.2

Wir setzen voraus, daß die in der Testübung 6.5.1 geschriebene Datei existiert. Es ist nun ein Programm zu schreiben, das den in der Region 3 dieser Datei gespeicherten Satz liest und die in ihm enthaltenen vier Daten auf dem Bildschirm anzeigt!

Zur Umwandlung numerischer Daten in die Zeichenform oder Kettenform bzw. in umgekehrter Richtung haben wir bisher nur die Funktionen MKS$ und CVS benutzt. Diese beiden Funktionen sind nur auf die Darstellungsart der numerischen Daten beschränkt, die wir als „einfache Genauigkeit" bezeichnet haben. Wie wir später im Kap. 12 sehen werden, gibt es für die numerischen Daten, für die Zahlen also, beim Personalcomputer der IBM verschiedene Darstellungsarten; wir werden über sie dort sprechen. Nur soviel sei vorerst gesagt: Zusätzlich zu den Zahlen einfacher Genauigkeit gibt es noch

- *Zahlen doppelter Genauigkeit*
  Die Zahlen können bei doppelter Genauigkeit bis zu 17 Ziffern aufweisen
- *ganze Zahlen*
  Ganze Zahlen sind auf den Wertebereich von −32768 bis 32767 beschränkt

Um Zahlen doppelter Genauigkeit in die Kettenform umzuwandeln, gebrauchen wir die Funktion MKD$, zur Rückumwandlung die Funktion CVD. Die Umwandlung ganzer Zahlen in die Zeichen- oder Kettenform erfordert den Aufruf der Funktion MKI$, die Rückumwandlung den Aufruf von CVI.

Sowohl in der numerischen Form als auch in der Kettenform belegt eine ganze Zahl zwei Bytes, eine Zahl einfacher Genauigkeit vier Bytes und eine doppelt genaue Zahl acht Bytes. Im einzelnen bedeutet das z.B., daß die Funktion MKI$ eine zwei Bytes umfassende Kette produziert, MKS$ eine vier Bytes lange und MKD$ eine acht Bytes lange.

BASIC weist mehrere Funktionen auf, mit denen die Verwaltung (Wartung und Pflege) von Direktzugriffsdateien beachtlich erleichtert wird. Die Funktion LOF (Länge der Datei, engl.: length of file) liefert die Anzahl der in der Bezugsdatei enthaltenen Bytes, aufgerundet auf das nächste ganzzahlige Vielfache von 128. Wenn also die Datei 2 insgesamt 140 Datenbytes enthält, wird sich der Funktionswert von LOF(2) die Zahl 256 ergeben. Falls die Länge der in die Datei aufgenommenen Sätze 128 Bytes beträgt, kann man aus dem Funktionswert von LOF und der Satzlänge sehr einfach die Anzahl der in der Datei vorhandenen Sätze bestimmen. Wenn nichts anderes vorliegt, ist 128 die voreingestellte Satzlänge.

Es wäre noch zu bemerken, daß Direktzugriffsdateien keine „Löcher" aufweisen können. Das heißt, daß beim Schreiben des für die Region 150 bestimmten Satzes BASIC den für die Sätze in den Satzbereichen 1 bis 149 benötigten Speicherplatz berücksichtigt, selbst wenn in diese Satzbereiche noch nichts hineingeschrieben worden ist.

Der Funktionswert der Funktion LOC ist gleich der Nummer des zuletzt angesprochenen, d.h. geschriebenen oder gelesenen Satzes derjenigen Datei, auf die im Funktionsargument verwiesen wird. Wenn also der zuletzt in die Datei 1 geschriebene oder aus ihr gelesene Satz der Satz 58 gewesen ist, liefert der Funktionsaufruf LOC(1) den Funktionswert 58.

Die Funktion EOF (Dateiende, engl.: end of file) arbeitet genau so wie bei sequentiellen Dateien, wo wir sie auch besprochen haben. Diese logische Funktion liefert bekanntlich den Funktionswert TRUE, d.h. wahr, wenn das Ende der Datei erreicht worden ist, auf die bezug genommen wurde, anderenfalls den Wert FALSE, d.h. falsch.

Wenn BASIC zu arbeiten beginnt, wird ein Teil des RAM für die Unterstützung des Lesens und des Schreibens von Direktzugriffsdateien bereitgestellt. Dieser Teil des RAM wird als Direktdateipuffer bezeichnet. Seine Größe bestimmt eine Grenze für die Satzlänge der Direktzugriffsdateien. Man kann, wie wir weiter unten sehen werden, jede beliebige Satzlänge festlegen. Wenn man jedoch den Umfang des Direktdateipuffers berücksichtigt, so kann man nur bis zu einem Wert von 128 gehen. Wie kann man diese Beschränkung nun umgehen?

Nehmen wir zu diesem Zwecke einmal an, daß wir in einem Programm Direktzugriffsdateien benötigen, deren Sätze eine Länge von 200 Bytes besitzen. Man kann das arrangieren, indem man BASIC wie folgt startet:

1. Man sorge zunächst dafür, daß man die DOS-Systemanfrage A> bekommt.

2. Man tippe den Befehl

```
BASIC /S:200
```

ein und drücke die Eingabetaste.

BASIC zeigt nun die Meldung Ok an und man kann wie gewöhnlich programmieren.

Wenn man versucht, ein FIELD-Statement zu benutzen, das mehr Bytes anfordert als im Direktdateipuffer vorhanden sind, wird eine Fehlernachricht angezeigt, die auf einen Feldüberlauffehler (engl.: field overflow error) hinweist.

Aufgabengruppe 18

1. Es ist ein Programm zu schreiben, das das Telefonverzeichnis (siehe Beispiele im Abschnitt 6.3.2) als Direktzugriffsdatei erstellt. Diese Datei soll mit TELEFON bezeichnet werden.
   Für die in diese Datei aufzunehmenden Sätze sind vorzusehen:
   - Name: 20-stellige Zeichenkette
   - Ortszusatz: 10-stellige Zeichenkette
   - Straße: 20-stellige Zeichenkette
   - Postleitzahl: 4-stellige Zeichenkette
   - Ort: 20-stellige Zeichenkette
   - Telefonnummer: 10-stellige Zeichenkette

2. Für eine Personaldatei ist der folgende Satz bestimmt:

   SCHMITTƀƀƀƀƀ HANSƀƀƀƀƀƀ PROGRAMMIERERƀƀ 17.11.1982VERDIENST25.00

   Es ist ein FIELD-Statement niederzuschreiben, das die Bestandteile dieses Satzes korrekt voneinander trennt.
   Anmerkung: Leerzeichen sind hier der besseren Lesbarkeit wegen im Satz durch ƀ gekennzeichnet worden.

3. Angenommen, eine Datei namens "VERKAUF" besteht aus 20 Sätzen. Jeder Satz wiederum setzt sich aus vier Zahlen einfacher Genauigkeit zusammen. Es ist ein Programm zu schreiben, das die Sätze dieser Datei nacheinander liest und die in ihnen enthaltenen Zahlen in vier Spalten auf dem Bildschirm auflistet.

4. Es ist ein Programm zu schreiben, das aufgrund eines vorgegebenen Namens die zugehörigen Daten in der Datei TELEFON (Telefonverzeichnis) aufsucht. Die gefundenen Adreßangaben sind auf dem Bildschirm anzuzeigen. Die Datei TELEFON wurde durch das Programm von Aufgabe 1. dieser Aufgabengruppe erstellt.

Antworten auf die Testübungen

6.5.1
```
10  OPEN "ABTEIL" AS #1, LEN=23
20  FIELD #1, 11 AS ABTBEZ$, 4 AS EINNAHME$,
              4 AS RATIONGRD$, 4 AS UNKOST$
30  FOR  J=1 TO 4
40    READ A$,B,C,D
50    LSET ABTBEZ$ = A$
60    LSET EINNAHME$ = MKS$(B)
70    LSET RATIONGRD$ = MKS$(C)
80    LSET UNKOST$ = MKS$(D)
90    PUT #1,J
100 NEXT J
110 DATA "RECHNSTELL",5000,0.735,7876
120 DATA "KONSTRUKT",3500,0.872,2200
130 DATA "VERWALTUNG1",4338,0.381,5130
140 DATA "WERBUNG",10832,0.95,12500
150 CLOSE #1
160 END
```

6.5.2
```
10  OPEN "ABTEIL" AS #1, LEN=23
20  FIELD #1, 11 AS ABTBEZ$, 4 AS EINNAHME$,
              4 AS RATIONGRD$, 4 AS UNKOST$
30  GET #1, 3
40  PRINT  "ABTEILUNG","EINNAHMEN","RATIONALISIERUNGSGRAD","UNKOSTEN"
50  PRINT  ABTBEZ$,CVS(EINNAHME$),CVS(RATIONGRD$),CVS(UNKOST$)
60  CLOSE #1
70  END
```

6.6 Beispiel für eine Anwendung von Dateien mit wahlfreiem Zugriff (Direktzugriffsdateien)

In diesem Abschnitt wollen wir detailliert ein Beispiel ausarbeiten, durch das die Anwendung von Direktzugriffsdateien einleuchtend illustriert wird. Wir wollen ein Programm entwerfen und codieren, das wir als „Listenmanager" bezeichnen wollen. Es soll uns ermöglichen, Listen zu handhaben. Ein Programm dieser Art wird mitunter als Datenbankmanagement-Programm bezeichnet; freilich liegt hier nur ein bescheidenes dieser Art vor.

Unser Programm soll ganz allgemeine Listen handhaben können. Eine typische Liste besteht aus einer Reihe von Eintragungen, die selbst wieder in eine Reihe von Datenelementen zerfallen. Wir lassen zu, daß jede Eintragung in einer Liste bis zu fünf Zeichenkettenelemente und bis zu fünf numerische Elemente enthalten darf. Die Zeichenkettenelemente sollen zuerst aufgelistet werden. Eine charakteristische Listeneintragung sieht also wie folgt aus:

Element 1 → Zeichenkettenelement
Element 2 → Zeichenkettenelement
Element 3 → Zeichenkettenelement
Element 4 → Zeichenkettenelement
Element 5 → Zeichenkettenelement
Element 6 → numerisches Element
Element 7 → numerisches Element
Element 8 → numerisches Element
Element 9 → numerisches Element
Element 10 → numerisches Element

Es ist nicht notwendig, alle 10 Elemente zu benutzen. So können z.B. die Eintragungen in einer speziellen Liste aus nur drei Zeichenkettenelementen, gefolgt von zwei numerischen Elementen bestehen. Das Programm fragt deshalb zu Beginn nach der Struktur der Eintragungen in eine Liste, d.h. nach der Anzahl der Zeichenkettenelemente und der Anzahl der numerischen Elemente. Danach nimmt das Programm an, daß alle Eintragungen in die betreffende Liste die erfragte Anzahl von Datenelementen der genannten Arten erhält.

Der Listenmanager soll dem Benutzer die Ausübung der folgenden Aktivitäten ermöglichen:

1. *Namensgebung für die Liste und Erstellung einer die Liste enthaltenden entsprechenden Direktzugriffsdatei*

2. *Betitelung der einzelnen, zu jeder Eintragung gehörenden Datenelemente (Beispiele: "NAME", "ADRESSE", "VERDIENST", "NR." usw.)*
   Jede Elementbenennung kann aus bis zu 12 beliebigen Zeichen bestehen.

3. *Eingabe von Datenelementen, die zu Eintragungen der betreffenden Liste gehören*
   Das Programm soll zu diesem Zweck die verschiedenen Benennungen der Datenelemente anzeigen und anschließend die Eingabe der Elemente, die zu einer bestimmten Eintragung gehören sollen, ermöglichen. Diese Eingabeoperationen sollen so oft wiederholt werden können, wie man es beabsichtigt. Auf diese Weise sollen Listen beliebigen Umfangs aufgebaut werden können.

4. *Änderung von Listeneintragungen*
   Die Listeneintragungen sollen durch Neueingabe von Datenelementen geändert werden können.

```
10   'Programm:  Listenmanager
15   '=========================
18   '
20   'Hauptprogramm
25   '-------------
30   GOSUB 4200:          'Vorgabe des Dateinamens und Dateieroeffnung
40   GOSUB 1010:          'Anzeige des Hauptmenues
50   ON REPLY     GOSUB 2000,3000,4100,4140
60   GOTO 40
61   '-----------------------------------------------------------------
1000 'Anzeige des Hauptmenues
1010     CLS
1020     PRINT  "MANAGEMENT VON LISTEN"
1030     PRINT:   PRINT
1040     PRINT  "PROGRAMM-AKTIVITAETEN"
1050     PRINT
1060     PRINT  "1. ZUWEISUNG VON BENENNUNGEN FUER DIE DATENELEMENTE"
1070     PRINT  "2. SPEZIFIZIERUNG VON LISTENEINTRAGUNGEN"
1080     PRINT  "3. DURCHSUCHEN UND ANZEIGEN VON LISTEN"
1090     PRINT  "4. AUSGANG AUS DEM LISTENMANAGEMENT"
1100     PRINT
1110     INPUT  "GEWUENSCHTE AKTIVITAET (1 BIS 4) EINGEBEN "; REPLY
1120 RETURN
1121 '-----------------------------------------------------------------
2000 'Zuweisung von Benennungen fuer die Datenelemente
2010     CLS
2020     IF  LOF(1) = 1  THEN 2040:       'Neue Datei?
2030     GOSUB 4400:                      'Holen der alten Benennungen
2040     FOR  ELEMENTNUMMER = 1  TO  10
2050       PRINT  "DATENELEMENT-NR. "; ELEMENTNUMMER;
                  TAB(28) "LAUFENDE BENENNUNG "
2060       PRINT  A$(ELEMENTNUMMER)
2070       INPUT  "NEUE BENENNUNG "; TITEL$(ELEMENTNUMMER)
2080       LSET  A$(ELEMENTNUMMER) = TITEL$(ELEMENTNUMMER)
2090     NEXT ELEMENTNUMMER
2100     PUT #1, 1:                       'Aufzeichnung der neuen Benennungen
2110 RETURN
2111 '-----------------------------------------------------------------
3000 'Spezifizierung von Listeneintragungen
3010     CLS
3020     INPUT "NUMMER DER LISTENEINTRAGUNG (0=NEUE EINTRAGUNG) "; ETRNUMMER
3030     IF  ETRNUMMER = 0  THEN ETRNUMMER=LOC(1)+1
                            ELSE ETRNUMMER=ETRNUMMER+1
3040     GOSUB 4400:                      'Holen der Benennungen
3050     PRINT  "LISTENEINTRAGUNG-NR. "; ETRNUMMER-1;
                TAB(28) "SPEZIFIZIERUNG DER DATENELEMENTE DER EINTRAGUNG "
3060     FOR  ELEMENTNUMMER = 1  TO  KETTENFELDER
3070       PRINT  "BENENNUNG DES DATENELEMENTES: "; TITEL$(ELEMENTNUMMER)
3080       INPUT "EINGABE EINER ZEICHENKETTE: "; EINGABE$
3090       LSET  A$(ELEMENTNUMMER) = EINGABE$
3100     NEXT ELEMENTNUMMER
3110     FOR  ELEMENTNUMMER = 6  TO  5+NUMERFELDER
3120       IF  TITEL$ = ""  THEN 3160
3130       PRINT  "BENENNUNG DES DATENELEMENTES: "; TITEL$(ELEMENTNUMMER)
3140       INPUT "EINGABE EINER ZAHL: "; EINGABE
3150       LSET  A$(ELEMENTNUMMER) = MKS$(EINGABE)
3160     NEXT ELEMENTNUMMER
3170     PUT #1, ETRNUMMER
3180 RETURN
3181 '-----------------------------------------------------------------
```

Abb. 6.12 Beispielprogramm unter Benutzung einer Direktzugriffsdatei (1. Teil)

```
4000 'Verschiedene Unterroutinen (Unterprogramme)
4001 '-----------------------------------------
4100   'Durchsuchen und Anzeigen von Listen
4110    GOSUB 4700:                    'Menue fuer Durchsuchen und Anzeigen
4120    ON REPLY     GOSUB 4800,4900,5100
4130   RETURN
4131 '-----------------------------------------------------------------
4140   'Ausgang aus dem Listenmanagement
4150    CLS
4160    CLOSE #1
4170    END
4180   RETURN
4181 '-----------------------------------------------------------------
4200   'Erlangung des Dateinamens und Eroeffnen der Datei
4210    CLS
4220    CLOSE
4230    PRINT  "LISTENMANAGEMENT"
4240    INPUT "DATEINAME "; DATEINAME$
4250    INPUT "ANZAHL DER KETTENFELDER (1 BIS 5) "; KETTENFELDER
4260    INPUT "ANZAHL DER NUMERISCHEN FELDER (1 BIS 5) "; NUMERFELDER
4270    OPEN DATEINAME$ AS #1
4280    FIELD #1,  12 AS A$(1), 12 AS A$(2), 12 AS A$(3), 12 AS A$(4),
                   12 AS A$(5), 12 AS A$(6), 12 AS A$(7), 12 AS A$(8),
                   12 AS A$(9), 12 AS A$(10)
4290    GOSUB 4400:                    'Holen der alten Benennungen
4300   RETURN
4301 '-----------------------------------------------------------------
4400   'Holen der alten Benennungen
4410    GET #1, 1
4420    FOR  J = 1  TO  10
4430      TITEL$(J) = A$(J)
4440    NEXT J
4450   RETURN
4451 '-----------------------------------------------------------------
4500   'Anzeige einer Eintragung
4510    CLS
4520    PRINT:                         'Uebergang zur 2. Zeile
4530    GOSUB 4400:                    'Holen (Lesen) der Benennungen
4540    IF  ANZEIGENUMMER > LOF(1)/128  THEN 4680:
                                       'Satz existiert nicht
4550    GET #1, ANZEIGENUMMER
4560    FOR  ELEMENTNUMMER = 1  TO  KETTENFELDER
4570      EINGABE$(ELEMENTNUMMER) = A$(ELEMENTNUMMER)
4580      PRINT  TITEL$(ELEMENTNUMMER); TAB(21) EINGABE$(ELEMENTNUMMER)
4590    NEXT ELEMENTNUMMER
4600    FOR  ELEMENTNUMMER = 6  TO  5+NUMERFELDER
4610      IF  A$(ELEMENTNUMMER) = ""  THEN 4620  ELSE 4640
4620      PRINT  TITEL$(ELEMENTNUMMER)
4630      GOTO 4660
4640      EINGABE(ELEMENTNUMMER) = CVS(A$(ELEMENTNUMMER))
4650      PRINT  TITEL$(ELEMENTNUMMER); TAB(21) EINGABE(ELEMENTNUMMER)
4660    NEXT ELEMENTNUMMER
4670    LOCATE 1, 1
4680   RETURN
4681 '-----------------------------------------------------------------
```

Abb. 6.12 Beispielprogramm unter Benutzung einer Direktzugriffsdatei (2. Teil)

```
4700   'Menue fuer das Anzeigen und Suchen
4710    CLS
4720    PRINT  "MENUE FUER DAS ANZEIGEN UND SUCHEN"
4730    PRINT:     PRINT
4740    PRINT  "1. ANZEIGE EINER EINTRAGUNG MIT GEGEBENER NUMMER"
4750    PRINT  "2. ANZEIGE AUFEINANDERFOLGENDER EINTRAGUNGEN"
4760    PRINT  "3. SUCHEN"
4770    PRINT
4780    INPUT  "GEWUENSCHTE AKTIVITAET (1 BIS 3) EINGEBEN "; REPLY
4790   RETURN
4791 '-------------------------------------------------------------------
4800   'Anzeige einer Eintragung mit gegebener Nummer
4810    CLS
4820    PRINT:                          'Uebergang zur 2. Zeile
4830    INPUT "NUMMER DER ANZUZEIGENDEN EINTRAGUNG EINGEBEN "; ANZEIGENUMMER
4840    ANZEIGENUMMER = ANZEIGENUMMER + 1
4850    IF  ANZEIGENUMMER > LOF(1)/128  THEN 4890
4860    GOSUB 4500
4870    INPUT "ZUR FORTSETZUNG IST DIE EINGABETASTE ZU DRUECKEN "; REPLY$
4880    IF  REPLY$ = ""  THEN 4790 ELSE 4670
4890   RETURN
4891 '-------------------------------------------------------------------
4900   'Anzeige aufeinanderfolgender Eintragungen
4910    CLS
4920    PRINT:                          'Uebergang zur 2. Zeile
4930    INPUT "NUMMER DER ERSTEN ANZUZEIGENDEN EINTRAGUNG EINGEBEN ";
              ANZEIGENUMMER
4940    ANZEIGENUMMER = ANZEIGENUMMER + 1
4950    IF  ANZEIGENUMMER > LOF(1)/128  THEN 5020
4960    GOSUB 4500
4970    LOCATE 1, 1
4980    INPUT "ZUR ANZEIGE DER NAECHSTEN EINTRAGUNG 0,
               BEI RUECKKEHR ZUM HAUPTMENUE 1 EINGEBEN "; REPLY
4490    IF  REPLY = 1  THEN 5020
5000    ANZEIGENUMMER = ANZEIGENUMMER + 1
5010    GOTO 4950
5020   RETURN
5021 '-------------------------------------------------------------------
5100   'Suchen
5110    CLS
5120    INPUT "ZU UEBERPRUEFENDE ELEMENTNUMMER EINGEBEN "; ENR
5130    PRINT  "SUCHEN NACH DEM ELEMENT MIT DER ELEMENTNUMMER "; ENR;
               " GLEICH";
5140    IF  ENR < 6  THEN INPUT SUCHBEGRIFF$
5150    IF  ENR > 5  THEN INPUT SUCHBEGRIFF
5160    L = LEN(SUCHBEGRIFF$):          'Laenge der Suchkette
5170    SUCHBEGRIFF$ = SUCHBEGRIFF$ + SPACE$(12-L):
                                        'Auffuellen mit Leerzeichen
5180    LAENGE = LOF(1)/128
5190    FOR  J = 2  TO  LAENGE
5200      GET #1, J
5210      IF  ENR > 5  THEN A=CVS(A$(ENR))
                       ELSE A$=A$(ENR)
5220      IF   ENR < 6  AND  A$=SUCHBEGRIFF$   THEN GOSUB 5300
5230      IF   ENR > 5  AND  A=SUCHBEGRIFF   THEN GOSUB 5300
5240    NEXT J
5250   RETURN
5251 '-------------------------------------------------------------------
```

Abb. 6.12 Beispielprogramm unter Benutzung einer Direktzugriffsdatei (3. Teil)

```
5300   'Behandlung einer Gleichheit beim Suchen
5310    CLS
5320    LOCATE 1,1
5330    PRINT  "GLEICHHEIT BEI DER EINTRAGUNG "; J-1
5340    INPUT  "SOLL DIESE EINTRAGUNG ANGEZEIGT WERDEN? (J/N): "; REPLY$
5350    IF  REPLY$ = "J"  THEN 5360  ELSE 5400
5360    ANZEIGENUMMER = J
5370    GOSUB 4500
5380    INPUT "ZUR FORTSETZUNG IST DIE EINGABETASTE ZU DRUECKEN "; REPLY$
5390    IF  REPLY$ = ""  THEN 5400  ELSE 5380
5400   RETURN
5401 '-------------------------------------------------------------------
```

```
Anmerkungen:  1. Die bei der Ausführung des Programmes anfallenden Benutzer-
                 antworten werden stets den  folgenden Variablen zugewiesen:
                 a)  REPLY   bei numerischen Antworten,
                     z. B.  0  oder  1  oder  ...
                 b)  REPLY$  bei alphanumerischen Antworten,
                     z. B.  J  oder  N  (für JA oder NEIN stehend)

              2. Bei  Zeichenkettenausdrücken  bedeutet das  Pluszeichen die
                 Verkettung von  Zeichenketten,  d. h. zwei Ketten werden zu
                 einer Kette verbunden  (siehe dazu die Zeile 5170 im Teil 3
                 dieser Abbildung).
```

Abb. 6.12 Beispielprogramm unter Benutzung einer Direktzugriffsdatei (4. Teil)

5. *Anzeige von Listeneintragungen*
   Es sollen sowohl einzelne Listeneintragungen als auch eine Reihe aufeinanderfolgender Listeneintragungen angezeigt werden können.

6. *Durchsuchen von Listen*
   Die Liste soll nach Eintragungen durchsucht werden können, in denen ein bestimmtes Datenelement einen bestimmten Wert aufweist, d.h., daß z.B. die Eintragungen aufgesucht werden sollen, in denen das Datenelement "POSTLEITZAHL" den Wert 8031 besitzt. Das Programm soll den Benutzer informieren, bei welcher Eintragung die Vorgabe zutrifft und zu diesem Zweck dem Benutzer die Nummer der entsprechenden Eintragung nennen. Anschließend soll es den Benutzer fragen, ob er die genannte Eintragung ansehen will. Wenn das der Fall ist, soll die betreffende Eintragung angezeigt werden. Nachdem das geschehen ist, sollte man die Anzeige genügend lange betrachten können. Erst nach Drücken der Eingabetaste soll das Programm weiter ausgeführt werden; es soll dann, falls vorhanden, die nächste Eintragung gesucht werden, für die die vorgegebene Bedingung zutrifft.

Das in der Abb. 6.12 dargestellte Programm ist im beachtlichem Maße strukturiert, denn ein Hauptprogramm verkehrt mit Unterprogrammen. Durch Kommentare und Bemerkungen ist das Programm weitgehend selbsterklärend, so daß wir uns nur auf einige grundlegende Bemerkungen zu beschränken brauchen.

a) Die Bennennung, die in der Liste gebraucht werden, werden im Satz 1 der Direktzugriffsdatei gespeichert (Kopfsatz).

b) Die zur Liste gehörenden Eintragungen werden ab Satz 2 gespeichert. Infolgedessen ist die Nummer der Eintragung stets um 1 geringer als die Nummer des entsprechenden Satzes in der Datei, d.h. beispielsweise ist die Eintragung 5 der Liste im Satz 6 der Datei untergebracht.

c) Es gibt zwei Menüs. Das Haupt- oder Primärmenü erlaubt die Auswahl der folgenden Aktivitäten:
   - Spezifizierung der Benennungen (Titel)
   - Spezifizierung von Listeneintragungen
   - Suchen und Anzeigen
   - Ausgang, d.h. Verlassen (Beendigung) des Programmes

   Das zweite (Folge-)Menü wird nur dann auf dem Bildschirm angezeigt, wenn im Primärmenü die Option „Suchen und Anzeigen" ausgewählt wurde. Im zweiten Menü bestehen die folgenden Wahlmöglichkeiten:
   - Anzeige einer einzelnen Eintragung in der Liste
   - Anzeige aufeinanderfolgender Listeneintragungen
   - Durchsuchen der Liste nach einer bestimmten Eintragung

d) Die zu einer Eintragung gehörenden Datenelemente werden fortlaufend mit 1 bis 10 bezeichnet, die Zeichenketten mit 1 bis 5 und die Zahlen von 6 bis 10. Diese Numerierung wird selbst dann beibehalten, wenn einige Datenelemente in der jeweiligen Liste nicht existieren. Das erste numerische Datenelement hat konsequenterweise also stets die Nummer 6.

Wir wollen dies Beispielprogramm mit einigen Bemerkungen abschließen. Zunächst einmal haben wir jedes Feld der in die Datei aufgenommenen Sätze 12 Stellen lang gemacht. Das rührt daher, daß wir alle Sätze dem Aufbau der Kopfsatzes mit seinen Benennungen anpassen wollten. Da wir die Satzlänge nicht gesondert spezifiziert haben, nimmt BASIC an, daß jeder Satz 128 Stellen umfaßt. Davon gebrauchen wir nur 120 Stellen, denn unser Satz enthält gerade 10 Felder zu je 12 Stellen. Natürlich kann man, wenn man will, dieses Programm neu entwerfen, um damit weitergehende Bedürfnisse, wie man Datenelemente pro Eintragung oder längere Zeichenketten und Benennungen, befriedigen zu können. Wenn man jedoch die Sätze länger macht als 128 Stellen, muß man BASIC neu initialisieren; nur so läßt BASIC einen ausreichend großen Puffer für Direktzugriffsdateien zu.

Aufgabengruppe 19

1. Das Programm „Listenmanager" ist einzutippen.
2. Das Programm „Listenmanager" ist zur Erzeugung einer Liste über die verschickten Weihnachts- und Neujahrsgrüße einzusetzen.
3. Die Möglichkeit zum Durchsuchen der Direktzugriffsdatei ist zum Auffinden (Lokalisieren) mehrerer Eintragungen in der Grußliste (siehe Aufgabe 2.) zu praktizieren.

6.7 Sortiertechniken

In den vorhergehenden Abschnitten lernten wir die Verfahren kennen, durch die wir in der Lage waren, Dateien zu erstellen, zu schreiben und zu lesen. In diesem Abschnitt wollen wir nun über die Organisation der Daten innerhalb der Dateien sprechen.

Wenn eine Datei öfters benutzt wird, müssen wir in der Lage sein, zu den in ihr gespeicherten Daten leicht zuzugreifen. Zuerst mag das wie eine einfache Erfordernis erscheinen. Schließlich

können wir immer eine Datei durchsuchen und dabei die Sätze solange nacheinander überprüfen, bis wir denjenigen Satz gefunden haben, der uns interessiert. Unglücklicherweise ist diese Vorgehensweise nicht immer möglich. Bis jetzt haben wir nämlich nur mit ziemlich kleinen Dateien gearbeitet. Viele Applikationen erfordern jedoch den Umgang mit Dateien, die Tausende, ja sogar Zehntausende von Sätzen enthalten. Wenn nun eine Datei einen solch gewaltigen Umfang besitzt, reicht selbst die größte Geschwindigkeit von Großcomputern nicht mehr aus, um ein erträglich rasches Durchsuchen einer Datei nach einem bestimmten Satz zu garantieren. In der Tat könnte es passieren, daß es beim Suchen nach einem speziellen Datenelement erforderlich ist, eine ganze Datei zu durchgehen, und das kann durchaus Stunden dauern. Aus diesem und auch aus anderen Gründen organisieren wir den Inhalt einer Datei gewöhnlich in einer Weise, die den Zugriff zu den in ihr enthaltenen Daten wesentlich verbessert. Einige Beispiele für allgemein vorgenommene Dateiorganisationen sollen nun folgen:

1. Die Datensätze in einer Kundendatei könnten so angeordnet werden, daß sie alphabetisch nach den Kundennamen aufeinander folgen.
2. Eine Adreßliste könnte so angeordnet werden, daß die einzelnen Adressen nach den Postleitzahlen aufeinander folgen.
3. Ein Inventarverzeichnis könnte nach den Inventarnummern angeordnet sein.
4. Eine Kreditkarten ausgebende Gesellschaft könnte ihre Kundenkontodatei entsprechend der Kreditkartennummern geordnet haben.

Bei jedem der soeben aufgeführten Beispiele liegen die Sätze in der betreffenden Datei in einer bestimmten Reihenfolge vor. Die Reihenfolge basiert in jedem Fall auf dem Wert, der in einem besonderen Feld des Satzes vorliegt, bei unserem Beispielen sind es der Reihe nach die Felder, die den Namen, die Postleitzahl, die Inventarnummer und die Kreditkartennummer enthalten. Für die Wartung und Pflege solcher Dateien ist es außerordentlich wichtig, daß man dazu imstande ist, die Sätze auch in der gewünschten Reihenfolge anzuordnen. Den Prozeß, eine Menge von Datensätzen in eine gewünschte Reihenfolge zu bringen, bezeichnet man als S o r t i e r e n . Tatsächlich ist das Sortieren ein extrem wichtiges Thema der Programmierung und ist infolgedessen schon in zahlreichen Veröffentlichungen (Aufsätzen wie auch Büchern) behandelt worden; viele wissentschaftliche Untersuchungen sind diesem Gegenstand bereits gewidmet worden. In diesem Abschnitt wollen wir eine Einführung in das Sortieren geben, indem wir eine der elementaren Sortiertechniken, nämlich das Blasensortieren (engl.: bubble sort) beschreiben.

Wir wollen damit beginnen, daß wir unser Problem mit einfachen Worten auszudrücken versuchen. Wir wollen annehmen, daß wir die Sätze in einer Datei so anordnen wollen, daß sie nach dem Inhalt eines bestimmten Feldes, etwa des Feldes 1, angeordnet sind. Präzis gesprochen, unser Problem soll darin bestehen, daß wir die Sätze einer Datei nach aufsteigenden Werten des Feldes 1 anordnen wollen. Um unsere einleitende Betrachtung des Themas zu erleichtern, wollen wir weiterhin voraussetzen, daß die Werte, die in diesem Feld enthalten sind, numerisch sind. Später wollen wir uns auch mit Sortierfeldern beschäftigen, die Zeichenketten darstellen.

Zunächst wollen wir Bereiche A( ) und B( ) wie folgt einrichten: Die Werte der Felder 1 in den Sätzen sollen in die Elemente des Bereiches A eingelesen werden, d.h.

A(1) ← Wert des Feldes 1 im Satz 1
A(2) ← Wert des Feldes 1 im Satz 2
A(3) ← Wert des Feldes 1 im Satz 3
usw.

Wir wollen bekanntlich die Sätze aufgrund gewisser Regeln in eine neue Reihenfolge bringen. Da die in der Datei gespeicherten Sätze ziemlich lang sein können, wollen wir uns nur mit dem Inhalt der ersten Felder beschäftigen. Damit wir uns aber ständig auf dem Laufenden halten können, zu welchem Satz jeweils ein bestimmter, im Feld 1 stehender Wert gehört, benutzen wir den Bereich B. Dieser soll deshalb stets die folgenden Werte enthalten:

B(1) ← Nummer des Satzes, zu dem der in A(1) stehende Wert des Feldes 1 gehört
B(2) ← Nummer des Satzes, zu dem der in A(2) stehende Wert des Feldes 1 gehört
B(3) ← Nummer des Satzes, zu dem der in A(3) stehende Wert des Feldes 1 gehört
usw.

Angenommen, wir lesen zu Anfang die Werte des ersten Feldes fortlaufend in die Elemente des Bereiches A, entsprechend der aufsteigenden Satznummern also. Damit müssen wir den Elementen des Bereiches B anfangs die folgenden Werte zuweisen:

B(1) = 1, B(2) = 2, B(3) = 3,

6.7.1 Das Blasensortierverfahren (Bubble Sort)

Das Blasensortierverfahren ermöglicht es, eine Zahlenmenge nach aufsteigenden Werten anzuordnen. Es beinhaltet eine Reihe von Vertauschungen zwischen aufeinanderfolgenden Zahlen. Wir wollen dieses Verfahren anhand der folgenden Zahlenfolge beschreiben:

90, 38, 15, 48, 80, 1

Diese Zahlen sollen nach aufsteigenden Werten geordnet werden. Das Verfahren zerfällt in einzelne Schritte.

a) *Schritt 1*
Wir beginnen beim rechten Ende der Zahlenfolge und vergleichen die zwei nebeneinander stehenden Zahlen. Wenn sie sich nicht in der gewünschten Reihenfolge befinden, sind sie umzustellen, anderenfalls bleiben sie an ihrer Stelle. Diese Vorgehensweise ist von rechts nach links mit allen Paaren nebeneinander liegender Zahlen fortzusetzen. Es ergeben sich die folgenden Resultate:

90, 38, 15, 48, **1, 80** → *Da 1 kleiner als 80 ist, wurden diese beiden Zahlen vertauscht*
90, 38, 15, **1, 48,** 80 → *Da 1 kleiner als 48 ist, wurden diese beiden Zahlen vertauscht*
90, 38, **1, 15,** 48, 80 → *Da 1 kleiner als 15 ist, wurden diese beiden Zahlen vertauscht*
90, **1, 38,** 15, 48, 80 → *Da 1 kleiner als 38 ist, wurden diese beiden Zahlen vertauscht*
1, 90, 38, 15, 48, 80 → *Da 1 kleiner als 90 ist, wurden diese beiden Zahlen vertauscht*

Mit diesem letzten Vergleich ist der 1. Schritt abgeschlossen. Durch ihn hat die Zahl 1 bereits ihren korrekten Platz in der Zahlenfolge gefunden.

b) *Schritt 2*
Die im Schritt 1 praktizierte Vergleichsfolge ist erneut auf die jetzt vorliegende Zahlenfolge anzuwenden, wiederum von rechts nach links vorgehend; jetzt allerdings nur noch fünf Mal, da die erste Zahl schon auf ihrem richtigen Platz steht.

1, 90, 38, 14, **48, 80** → *Da 80 nicht kleiner als 48 ist, wurden diese beiden Zahlen nicht vertauscht*
1, 90, 38, **14, 48,** 80 → *kein Vertauschen*
1, 90, **14, 38,** 48, 80 → *Vertauschen*
1, **14, 90,** 38, 48, 80 → *Vertauschen*

Als Resultat des zweiten Schrittes hat jetzt die Zahl 14 den ihr zukommenden Platz in der Zahlenfolge gefunden.

c) *Schritt 3*
Die Vorgehensweise von Schritt 1 ist nunmehr auf die verbleibenden unsortierten vier rechten Zahlen der Zahlenfolge anzuwenden.

1, 14, 90, 38, **48, 80** → *kein Vertauschen*
1, 14, 90, **38, 48,** 80 → *kein Vertauschen*
1, 14, **38, 90,** 48, 80 → *Vertauschen*

d) Schritt 4
Die Vorgehensweise von Schritt 1 ist nunmehr auf die rechten drei Zahlen der im Schritt 3 entstandenen Zahlenfolge anzuwenden.

1, 14, 38, 90, **48, 80**
1, 14, 38, **48, 90,** 80

e) *Schritt 5*
Die Vorgehensweise von Schritt 1 ist abschließend auf die verbleibenden zwei rechten Zahlen der in Schritt 4 entstandene Folge anzuwenden; nur diese beiden Zahlen sind noch zu vergleichen und evtl. zu vertauschen

1, 14, 38, 48, **80, 90**

Wir erkennen, daß jetzt die richtige Sortierfolge der Zahlen vorliegt; ein weiterer Schritt erübrigt sich also.

Nach Durchführung des Blasensortierverfahrens an einem einfachen und übersichtlichen Beispiel ist es wohl angebracht, seine Charakteristik hervorzuheben. Bei jedem Schritt wird die verbliebene kleinste Zahl an die Stelle der Zahlenfolge übertragen, die sie einnehmen muß. Angenommen, wir betrachten einmal die ursprüngliche Zahlenfolge vertikal geschrieben, also so, wie sie in der Abb. 6.13. dargestellt ist.

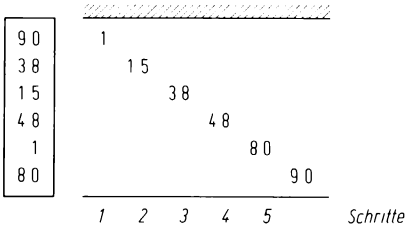


Abb. 6.13 Vertikale Darstellung einer zu sortierenden Zahlenfolge

Dann wird bei jedem Schritt die verbliebene kleinste Zahl auf die entsprechende Ebene der Folge bewegt. Man denke sich jede Zahl als Blase unter Wasser, deren Gewicht durch ihren Wert bestimmt ist. Dann wird bei jedem Schritt des Verfahrens so weit nach oben bewegt, wie sie bis zur Oberfläche steigen kann. Aus diesem Grund spricht man von diesem Verfahren als dem „Blasensortierverfahren".

Bei unserem Beispiel haben wir die einzelnen Schritte bis ins kleinste Detail ausgeführt. Auf diese Weise hofften wir, daß es uns nunmehr keine besonderen Schwierigkeiten mehr bietet, dieses Blasensortierverfahren durch ein Programm zu erfassen. Wir nehmen dazu an, daß die zu sortierenden Elemente im Bereich A mit der Dimension N gespeichert sind. In der Abb.6.14 ist das Programm aufgelistet, mit dem das Blasensortierverfahren verwirklicht wird.

```
200 'Subroutine (Unterprogramm) fuer das Blasensortierverfahren
210 FOR  I = 2  TO  N
220   FOR  J = N  TO  I  STEP -1
230     IF  A(J-1) > A(J)  THEN SWAP A(J-1),A(J)
240   NEXT J
250 NEXT I
260 RETURN
```

Abb.6.14 Programm für das Blasensortierverfahren

Wir haben also, die Abb.6.14 zeigt uns das, das Blasensortierverfahren durch ein Unterprogramm realisiert; damit können wir es in ein größeres Programm aufnehmen. Zu bemerken ist in diesem Fall noch, daß das DIM-Statement für den Bereich A in das größere, d.h. das umfassende Programm aufgenommen werden muß. Ebenso muß in dem größeren Programm der Variablen N der Wert zugewiesen werden, der die Anzahl der zu sortierenden Zahlen symbolisiert. Man kann das Unterprogramm mit der Zahlenfolge

100, 99, 98, 97, 96, ···, 3, 2, 1

testen. Hierzu erweitern wir das Unterprogramm so, wie es die Abb.6.15. zeigt.

```
5   'Programm zum Testen des Blasensortierverfahrens
10  DIM  A(100)
20  N = 100
30  FOR  J = 1  TO  N
40    A(J) = 101 - J
50  NEXT J
60  GOSUB 200:    'Aufruf des Sortierunterprogrammes
70  FOR  J = 1  TO  N
80    PRINT  J,A(J)
90  NEXT J
100 END
```

Abb.6.15 Testprogramm für das Sortieren von 100 Zahlen

In der Abb.6.14, bei unserem Unterprogramm für das Blasensortierverfahren als, haben wir den Austausch der beiden Werte A (J-1) und A(J) durch die Anweisung

```
SWAP  A(J-1),A(J)
```

bewerkstelligt. Diese Anweisung dient, allgemein gesehen, dazu, die Werte zweier Variabler des gleichen Datentyps auszutauschen. Sei z.B. A = 5 und B = 12, dann ergibt sich nach Ausführung von

```
SWAP  A,B
```

nunmehr, daß A = 12, und B = 5 ist.

Wir können die Routine für das Blasensortierverfahren dazu benutzen, um aus ihr einige interessante Schlußfolgerungen für die Sortiertechniken zu ziehen. Zu diesem Zweck betrachten wir einmal die von uns ermittelten Ausführungszeiten für das Blasensortierverfahren, bezogen auf die Zahlenfolge

N, N-1, N-2, ···, 1

Hinsichtlich der Ausführungszeiten gesehen, handelt es sich hier um den ungünstigsten Fall, da bei jedem Schritt ständig der Austausch zweier Werte erfolgen muß (Abb. 6.16).

Wert von N	*Ausführungszeit für das Blasensortierverfahren in Sekunden*
100	67
50	17
20	4
10	1

Abb. 6.16 Tabelle gemessener Ausführungszeiten für das Blasensortierverfahren für den ungünstigsten Fall

Zu der Tabelle von Abb. 6.16 ist zunächst zu bemerken, daß bereits bei nur 100 zu sortierenden Zahlen die Ausführungszeit spürbar groß ist. Zweitens erkennen wir sofort, daß mit steigender Zahl der zu sortierenden Zahlen auch die Ausführungszeit anwächst. Es scheint sogar, daß bei Verdoppelung der Anzahl der zu sortierenden Zahlen die Ausführungszeit sich vervierfacht. In gleicher Weise wird die Ausführungszeit um den Faktor 9 größer, wenn sich die Anzahl der zu sortierenden Zahlen verdreifacht. Im allgemeinen ergibt sich für den von uns untersuchten ungünstigsten Fall, daß sich bei Vervielfachung der zu sortierenden Zahlen um den Faktor k die Ausführungszeit um den Faktor k^2 vervielfacht. Im Durchschnitt sind freilich die Ausführungszeiten nicht so schlecht. Vergessen wir nicht, daß wir für unsere Betrachtung den ungünstigsten Fall ausgewählt haben, um zu demonstrieren, wie Sortierzeiten schnell recht unhandluch werden können.

Beim Blasensortierverfahren besteht ein prinzipielles Problem darin, daß es für den BASIC-Interpretierer geschrieben wurde. Um wirklich ernsthaft, d.h. mit vertretbarem Zeitaufwand, sortieren zu können, muß das in BASIC geschriebene Programm durch einen Kompilierer (engl.: Compiler) zuvor in ein Programm in der Maschinensprache umgewandelt werden, d.h. die BASIC-Sprache muß in die Maschinensprache übersetzt werden. Dafür gibt es den BASIC-Compiler. Die Benutzung dieses Werkzeuges wird in unserem Buch

„Advanced BASIC and beyond: Techniques for the IBM PC" Robert J. Brady Co., 1984

beschrieben.

Wir wollen zu unserer ursprünglichen Aufgabenstellung zurückkehren, nämlich zum Sortieren der Sätze einer Direktzugriffsdatei. Zum Sortieren der Elemente des Bereiches A wollen wir nach wie vor das Blasensortierverfahren benutzen. Bei jedem Austausch von Bereichselementen wollen wir zusätzlich auch die entsprechenden Elemente des Bereiches B austauschen. Am Ende der Subroutine befinden sich die Elemente von A wertmäßig in aufsteigender Reihenfolge. Die Elemente von B enthalten dann die Nummern der Sätze, zu denen die in den A(J) vorliegenden Werte gehören. Das unter diesem Aspekt erweiterte Unterprogramm von Abb. 6.14 ist in der Abb. 6.17 dargestellt.

```
200 'Subroutine fuer das Blasensortierverfahren (fuer Saetze von Dateien)
210 FOR  I = 2  TO  N
220   FOR  J = N  TO  I  STEP -1
230     IF  A(J-1) > A(J)  THEN SWAP A(J-1),A(J):
                                SWAP B(J-1),B(J)
240   NEXT J
250 NEXT I
260 RETURN
```

Abb. 6.17 Unterprogramm für das Blasensortierverfahren von Sätzen

Der Bereich B kann nun in einer zusätzlichen Datei gespeichert und zum Lesen der Sätze aus der Direktzugriffsdatei herangezogen werden (in Übereinstimmung mit der aufsteigenden Reihenfolge des der Sortierung zugrundeliegenden Feldes).

Um die Elemente eines Zeichenkettenbereiches in die alphabetische Reihenfolge zu bringen, verwenden wir ein Blasensortierverfahren, das ähnlich demjenigen arbeitet, das wir zur Sortierung numerischer Daten entwickelt haben. Natürlich müssen wir jetzt aufeinanderfolgende Paare von Zeichenketten, d.h. A$(J-1) und A$(J) , miteinander vergleichen und erforderlichenfalls vertauschen. Wir verweisen hierzu auf die Einzelheiten, die wir bei den Aufgaben der Aufgabengruppe 20 finden.

In diesem Abschnitt haben wir gerade die Oberfläche des unerschöpflichen Themas „Sortieren" angekratzt. Für ein intensives Studium dieses Themas empfiehlt sich das Buch:

Niklaus Wirth
„Algorithms + Data Structures = Programs"
Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1976

Aufgabengruppe 20

1. Es ist ein Programm zu schreiben, mit dem die Elemente eines Bereiches A$ in die alphabetische Reihenfolge gebracht werden können. Dazu ist das Blasensortierverfahren einzusetzen.

2. Das zur Lösung der Aufgabe 1. entwickelte Programm ist mit dem Bereich zu testen, der aus den Elementen

```
A$(1) = "Z"
A$(2) = "Y"
A$(3) = "X"
 :
A$(26) = "A"
```

besteht.

3. Es ist ein Programm zu schreiben, durch das ein Bereich mit N Zufallszahlen erzeugt werden kann; der Wert der Variablen N ist dabei über ein INPUT-Statement einzulesen. Anschließend sind die Elemente dieses Bereiches nach aufsteigenden Werten zu ordnen. Der Zeitgeber ist im Programm zu verwenden, um die Operation zur Bestimmung der Zufallszahlen auszulösen.
   Mit Hilfe des Programmes soll eine Tabelle der Sortierzeiten für die verschiedenen Werte von N aufgestellt werden.
   Anmerkung: Natürlich ist RANDOMIZE einzusetzen, damit sich nicht wiederholende Bereiche erzeugt werden.

4. Es ist ein Programm zu schreiben, durch das der kleinste Wert des ersten Feldes der Sätze der Datei "TEST" ermittelt werden kann.
   a) Die Datei besitzt einen Gesamtumfang von 25600 Bytes.
   b) Die Länge des ersten Feldes beträgt vier Bytes.

6.8 Auf Dateien bezogene Befehle in BASIC

Die BASIC-Sprache für die Personalcomputer der IBM weist eine Reihe nützlicher Befehle auf, die zur Manipulation von Dateien eingesetzt werden können.

6.8.1 Das Dateiverzeichnis (Directory)

Innerhalb von BASIC kann ein Verzeichnis aller Dateien angefordert werden, die auf einer bestimmten Diskette liegen. Diese Anforderung kann durch den Befehl FILES bewerkstelligt werden. Um z. B. alle Dateien, die auf der Standarddiskette gespeichert sind, kennenzulernen, muß man den Befehl

```
FILES
```

eingeben und anschließend die Eingabetaste drücken. Dieser Befehl ähnelt dem Befehl DIR im Betriebssystem DOS; er kann jedoch unter BASIC benutzt werden, d. h. die sonst notwendige Rückkehr zum DOS kann entfallen.

Der FILES-Befehl ist sehr wandelbar. Man kann ihn u. a. dazu verwenden, um sich eine Übersicht über alle Dateien zu verschaffen, die sich durch eine bestimmte Dateispezifikation auszeichnen. Wenn man sich z. B. alle Dateinamen ansehen will, die die Erweiterung .BAS besitzen, und die zu Dateien gehören, die auf der Diskette B: gespeichert sind, so braucht man nur den Befehl

```
FILES „B:*.BAS"
```

einzugeben und die Eingabetaste zu drücken.

Um alle Dateien auf der Diskette A: zu eruieren, hat man einfach den Befehl

```
FILES „A:*.*"
```

einzugeben und anschließend die Eingabetaste zu betätigen.

Testübung 6.8.1

Es ist der Befehl niederzuschreiben, durch den alle Dateinamen aufgelistet werden, die mit einer mit dem Buchstaben B beginnenden Erweiterung versehen sind. Es sollen dabei nur die Dateinamen aufgeführt werden, die Dateien kennzeichnen, die auf der im augenblicklichen Standardlaufwerk eingelegten Diskette gespeichert sind.

6.8.2 Löschen von Dateien

Zum Löschen von Dateien kann man sich des BASIC-Befehles KILL bedienen. Dieser Befehl besitzt die folgende allgemeine Form:

```
KILL <ds>
```

Unter <ds> ist eine Dateispezifikation aufzuführen. Um beispielsweise die Datei mit dem Dateinamen BEISP.TXT auf der Diskette A: zu löschen, gebraucht man den Befehl

```
KILL "BEISP.TXT"
```

Um alle Dateien auf der ins Laufwerk B: eingelegten Diskette zu löschen, hat man einfach den Befehl

```
KILL "*.*"
```

einzugeben und (selbstverständlich wie immer) anschließend die Eingabetaste zu betätigen.

Das letzte Beispiel für den KILL-Befehl ist sehr gefährlich: Wie leicht löscht man dadurch auch Dateien, deren Löschung gar nicht beabsichtigt war! Deshalb sollte man diese Abart des KILL-Befehles nur mit äußerster Vorsicht anwenden.

Es wäre noch zu bemerken, daß der KILL-Befehl sowohl für Programmdateien als auch für Datendateien angewendet werden kann. Man erinnere sich in diesem Zusammenhang jedoch daran, daß BASIC automatisch die Erweiterung .BAS bei Programmdateien hinzufügt. Man vergesse als nicht, diese Erweiterung in den Befehl einzubeziehen. Zum Schluß sei noch einmal auf den Löschbefehl ERASE hingewiesen, der unter dem Betriebssystem DOS gilt; dieser Befehl wurde bereits besprochen.

Testübung 6.8.2

Es sind BASIC-Befehle zum Löschen der folgenden Dateien niederzuschreiben:

a) BASIC-Programmdatei "FARBEN"
b) BASIC-Programmdatei "RECHN.001"

6.8.3 Umbenennung von Dateien

Man kann Dateien umbenennen, wenn man den BASIC-Befehl NAME benutzt. Um die Datei ROULETTE in SPIELE umzubenennen, muß man den Befehl

```
NAME  ROULETTE AS SPIELE
```

eingeben und die Eingabetaste drücken.

Hinzuweisen ist in diesem Fall, daß die alten Namen zuerst genannt werden müssen; die neuen Namen folgen auf die alten. Weiterhin sollte man bei Programmdateien nicht vergessen, daß deshalb u.U. auch die Erweiterung des Dateinamens aufzuführen ist. Diese ist gleich .BAS dann, wenn keine andere Erweiterung vergeben wurde.

6.8.4 Sicherstellung von Dateien

Wie wir bereits im Kapitel 2 gelernt haben, kann das gerade in Bearbeitung befindliche Programm auf einer Diskette sichergestellt werden, wenn man den Befehl SAVE benutzt. Nehmen wir die Gelegenheit wahr, um einige Ergänzungen zu besprechen, die diesen Befehl betreffen. BASIC erlaubt nämlich, daß ein Programm in einem von drei verschiedenen Formaten sichergestellt werden kann. Folgende Formate sind möglich:

- verdichtetes Format
- ASCII-Format[1)]
- Binärformat (geschütztes Format)

a) *Verdichtetes Format*
Bis zu diesem Zeitpunkt haben wir bei der Sicherstellung ausschließlich dieses Format benutzt. In diesem Format werden die verschiedenen BASIC-Wörter, wie IF, THEN, ELSE, GET, LET, PRINT usw., auf eine numerische Kurzform reduziert. Dadurch kann das Programm auf weniger Speicherplatz untergebracht werden.

b) *ASCII-Format*
Bei diesem Format wird das Programm Zeichen für Zeichen, d.h. Buchstabe für Buchstabe, Ziffer für Ziffer usw., so gespeichert, wie es eingegeben wurde. Dadurch wird natürlich auf der Diskette mehr Speicherplatz als bei der verdichteten Form belegt. Andererseits ist es aber dadurch möglich, daß ein Programm gemischt (MERGE) und verkettet (CHAIN) wird. Über die Operation MERGE werden wir im Unterabschnitt 6.8.5 sprechen. Um ein Programm im ASCII-Format sicherzustellen, muß man sich des Befehles

```
SAVE <ds>,A
```

bedienen. Hierbei bedeutet <ds> wie üblich die Dateispezifikation der sicherzustellenden Datei. Um z.B. ein Programm , das sich im RAM befindet, in der Datei STEUERN auf der Diskette im Laufwerk B: im ASCII-Format sicherzustellen, könnte man den Befehl

```
SAVE "B:STEUERN",A
```

eingeben.

c) *Binärformat* (geschütztes Format)
Bei diesem Format handelt es sich um ein Schutzformat. Wenn ein Programm in diesem Format sichergestellt worden ist, kann es nicht mehr aufgelistet werden. Dadurch kann ein gewisser, freilich nur bescheidener Grad von Schutz gegenüber Neugierigen und Schnüff-

---

[1)] ASCII ist die Abkürzung von „American standard code for information interchange" (dtsch.: Amerikanischer Standard-Code für Informationsaustausch)

lern erzielt werden. Um das Programm in der Datei STEUERN auf der Diskette im Laufwerk B: im Binärformat sicherzustellen, ist es erforderlich, den Befehl

```
SAVE "B:STEUERN",P
```

einzugeben.
BASIC sieht keine Möglichkeit vor, im Binärformat gespeicherte Programme wieder in ein Format umzusetzen, das die Auflistung derselben gestattet. Aus diesem Grunde sollte das Binärformat nur nach reiflicher Überlegung benutzt werden.

6.8.5 Mischen von Programmen

BASIC besitzt die Fähigkeit, das gegenwärtig im RAM befindliche Programm mit jedem anderen, auf Diskette gespeicherten Programm zu mischen. Diese Möglichkeit erweist sich dann als besonders nützlich, wenn man auf Disketten gespeicherte Standard-Unterprogramme in ein Programm einfügen muß, das man gerade entwickelt. Das Mischen kann man mit dem Befehl MERGE bewerkstelligen. Um z.B. das im RAM vorliegende Programm mit dem in der Datei LOHNLIST gespeicherten Programm zu mischen, verwenden wir den Befehl:

```
MERGE "LOHNLIST"
```

Wie erfolgt nun das Mischen? Um diese Frage zu beantworten, nehmen wir einmal an, daß das im RAM befindliche Programm die Zeilen 10, 20, 30 und 100 aufweist, während das Programm in der Datei LOHNLIST die Zeilen 40, 50, 60, 70, 80, 90 und 100 enthält. Aus diesen beiden Ursprungsprogramm entsteht nun durch den MERGE-Befehl im RAM ein Mischprogramm, das die Zeilen 10, 20, 30, 40, 50, 60, 70, 80, 90 und 100 aufweist; die Zeile mit der Zeilennummer 100 wird dabei vom Programm in der Datei LOHNLIST genommen. Es gilt nämlich grundsätzlich, daß die Zeilen des auf der Diskette gespeicherten, hineingemischten Programmes, hier: LOHNLIST, im Falle doppelter Zeilennummern die Zeilen der im RAM befindlichen Programmes ersetzen. Um die Möglichkeit des Mischens auszunutzen, muß aber das in der Datei auf der Diskette befindliche Programm im ASCII-Format sichergestellt sein. Bei unserem Beispiel muß also zuvor einmal der Befehl

```
SAVE "LOHNLIST",A
```

ausgeführt worden sein. Wenn das nicht der Fall sein sollte, so muß man das Programm in der Datei LOHNLIST zuerst noch einmal durch den Befehl

```
LOAD "LOHNLIST"
```

wieder in den RAM laden und durch den o.a. Befehl sicherstellen.

Beim Mischen sollte man jedoch auf eins achtgeben. Wenn man gerade ein Programm eingegeben hat, mit dem man das Programm in LOHNLIST mischen will, so sollte man es unbedingt vor dem Eingeben des MERGE-Befehles sicherstellen, denn sonst würde das Programm in der eingegebenen Form verloren gehen.

Testübung 6.8.3

a) Das nachstehende Programm ist im ASCII-Format in der Datei GEISTVOL sicherzustellen!

```
10  PRINT 5+7
100 END
```

b) Das nachstehende Programm ist einzugeben.

```
30  PRINT 7+9
40  PRINT 7-9
```

c) Die Programme der Testübungen a) und b) sind miteinander zu mischen. Das aus diesem Prozeß entstandene Mischprogramm ist in der Datei "ZIEL.BAS" sicherzustellen.

Aufgabengruppe 21

1. a) Es ist ein Programm zu schreiben, das die Summe

$$1^2 + 2^2 + 3^2 + 4^2 + \quad + 50^2$$

berechnet!

b) Das in a) erstellte Programm ist in der Datei QUADRATE im ASCII-Format sicherzustellen.

2. a) Es ist ein Programm zu schreiben, das die Summe

$$1^3 + 2^3 + 3^3 + 4^3 + \quad + 30^3$$

berechnet! Die Zeilennummern dieses Programmes sind dabei so zu wählen, daß sie nicht mit den Zeilennummer übereinstimmen, die für das Programm der Aufgabe 1. vergeben wurden.

b) Die in den Teilaufgabe a) der Aufgaben 1. und 2. erstellten Programme sind miteinander zu mischen (Befehl MERGE).

c) Das in b) entstandene Mischprogramm ist aufzulisten (Befehl LIST).

d) Das in b) entstandene Mischprogramm ist auszuführen (Befehl RUN).

e) Das in b) entstandene Mischprogramm ist in der Datei KOMBIN sicherzustellen (Befehl SAVE).

3. Das in der Teilaufgabe a) der Aufgabe 2. erstellte Programm ist wiederherzustellen, ohne daß es dabei neu eingegeben wird.

4. Das in der Teilaufgabe a) der Aufgabe 1. erstellte und in der Datei QUADRATE sichergestellte Programm ist zu löschen.

Antworten auf die Testübungen

6.8.1 `FILES "*.B??"`

6.8.2

a) `KILL "FARBEN.BAS"`

b) `KILL "RECHN.001"`

6.8.3

a) `SAVE "GEISTVOL",A`

b) Es ist zunächst der Befehl NEW einzugeben. Danach ist das Programm einzutippen.

c)
```
MERGE "GEISTVOL"
SAVE "ZIEL",A
```

7 Handhabung von Zeichenketten

In diesem Kapitel wollen wir einige Feinheiten über die Handhabung von Zeichenketten herausarbeiten. Man sollte diese Diskussion als Vorspiel für die Textverarbeitung ansehen, über die wir in Kapitel 9 sprechen wollen.

7.1 ASCII-Zeichencodes

Jedem Zeichen ist eine Dezimalzahl zwischen 0 und 255 zugewiesen. Die zugewiesene Codezahl wird als ASCII-Code des betreffenden Zeichens bezeichnet. Beispielsweise hat der Buchstabe A den ASCII-Code 97. Natürlich sind in diesen Code auch die Sonderzeichen (Satzzeichen usw.) aufgenommen worden. Z.B. stellt der ASCII-Code 40 die Verschlüsselung der geöffneten runden Klammer (und 62 die Verschlüsselung des Größer-als- Zeichens > dar. Sogar die Tasten auf der Tastatur, die nicht zu den schreib- bzw. druckbaren Zeichen gehören, besitzen ASCII-Codes. So entspricht z.B. dem Anschlag der Leerzeichentaste (Zwischenraumtaste) der ASCII-Code 32 und dem Anschlag der Eingabetaste u.a. der ASCII-Code 13. Die druckbaren Zeichen weisen ASCII-Codes von 32 bis 126 auf; diese Zeichen sind mit ihren Codes in der Abb. 7.1 dargestellt.

ASCII-Code	Zeichen	ASCII-Code	Zeichen	ASCII-Code	Zeichen
32	(Leerz.)[1]	64	§	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	–	77	M	109	m
46		78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	Ä	123	ä
60	<	92	Ö	124	ö
61	=	93	Ü	125	ü
62	>	94	^	126	ß
63	?	95	_	127	

[1] Unter Leerz. ist das Leerzeichen (Zwischenraumzeichen) zu verstehen

Abb. 7.1 Druckbare Zeichen und ihre ASCII-Codes (siehe dazu auch Anhang C)

Über die ASCII-Codes der Steuerzeichen insgesamt werden wir im Abschnitt 7.3 sprechen; es handelt sich hierbei um die Codes 0 bis 31. Augenblicklich sollten wir unser Augenmerk nur auf zwei dieser Codes richten (Abb. 7.2).

ASCII-Code	*Zeichen für*	*Aktion in bezug auf den Positionsanzeiger*
10	Zeilenvorschub	Bewegung des Positionsanzeigers (Cursors) um eine Zeile nach unten, d.h. Einstellung auf neue Zeile.
13	Cursorrücklauf (Wagen- oder Schreibkopfrücklauf)	Bewegung des Positionsanzeigers (Cursors) auf die erste Stelle der laufenden Zeile

Abb. 7.2 Generierte Steuerzeichen beim Anschlagen der Eingabetaste

Das Drücken der Eingabetaste generiert sowohl ein Schreibkopfrücklaufzeichen (Cursorrücklaufzeichen) als auch ein Zeilenvorschubzeichen, d.h. die beiden ASCII-Codes 13 und 10.

Der Computer benutzt ASCII-Codes, um sowohl Zeichen als auch Steueroperationen anzusprechen. Jede Datei, sowohl Programmdateien als auch Datendateien, kann auf eine Folge von ASCII-Codes zurückgeführt werden. Man betrachte z.B. die in der Abb. 7.3 dargestellte dreizeilige Adresse.

```
Adreßzeilen
-----------
| Charlotte Schmidt    ---> 1. Adreßzeile
| Ahornweg 4           ---> 2. Adreßzeile
| 8031 Gilching        ---> 3. Adreßzeile

ASCII-Code für die Adreßzeilen
------------------------------
| 67,104,97,114,108,111,116,116,101,32,83,99,104,109,105,100,116,13,10 ---> 1. Adreßzeile
| 65,104,111,114,110,119,101,103,32,52,13,10                           ---> 2. Adreßzeile
| 56,48,51,49,32,71,105,108,99,104,105,110,103,13,10                   ---> 3. Adreßzeile

Anmerkung:  Der leichteren Lesbarkeit wegen sind die einzelnen
            ASCII-Codes durch Kommas voneinander getrennt.
```

Abb. 7.3 Dreizeilige Adresse und ihre Verschlüsselung nach dem ASCII-Code

Wir sehen aus der Abb. 7.3, wie die dreizeilige Adresse als Folge von ASCII-Codes gespeichert wird. Man beachte dabei, daß die Zwischenraum- oder Leerzeichen (Code 32) genau so wie die Cursorrücklaufzeichen (Code 13) und die Zeilenvorschubzeichen (Code 10), die am Ende jeder Zeile durch Drücken der Eingabetaste entstanden, in die Codefolge aufgenommen sind.

Die ASCII-Codes ermöglichen uns, jeden Text, der über die Tastatur eingegeben wurde, als Folge von Dezimalzahlen darzustellen. Solche Folgen schließen selbstverständlich alle Formatsteuerungen, wie Leerzeichen, Schreibkopfrückläufe (Cursorrückläufe), Groß- bzw. Kleinschreibungen usw. ein. Ein Textstück kann weiterhin, auch wenn es in eine Folge von ASCII-Codes umgewandelt worden ist, wahrheitsgetreu auf dem Bildschirm oder auf dem Drucker wiedergegeben werden.

Testübung 7.1.1

Es ist die Folge von ASCII-Codes niederzuschreiben, durch die die nachstehende Anzeige abgebildet wird.

ZUM VERKAUF: Junge Beagles mit Stammbaum,
8 Wochen. 250 DM.

Man kann sich auf die Zeichen mittels ihres ASCII-Codes beziehen, wenn man die Funktion CHR$ benutzt. So ist z.B. CHR$(74) das Zeichen, das dem ASCII-Code 74 entspricht, d.h. der Großbuchstabe J. Um bei einem weiteren Beispiel zu bleiben: Hinter CHR$(32) verbirgt sich das Zwischenraum- oder Leerzeichen. Selbstverständlich kann die Funktion CHR$ auch in Verbindung mit den Anweisungen PRINT und LPRINT verwendet werden. Z.B. wird durch die Anweisung

```
10 PRINT CHR$(74)
```

auf dem Bildschirm der Großbuchstabe J in der ersten Position der ersten Ausgabezone angezeigt.

Testübung 7.1.2

Es ist ein Programm zu schreiben, durch das die in der Testübung 7.1.1 vorgestellte Anzeige unter Zugrundelegung der ASCII-Codes angezeigt werden kann.

Die Funktion ASC steht zur Verfügung, um den ASCII-Code eines Zeichen zu bekommen. Beispielsweise wird durch die Anweisung

```
20 PRINT ASC("B")
```

der ASCII-Code des Großbuchstabens B angezeigt, nämlich 66. Anstelle von B kann natürlich Bezug auf jede beliebige Zeichenkette genommen werden. Die Funktion liefert dann den ASCII-Code des ersten Zeichens dieser Zeichenkette ab; beispielsweise wird durch die Anweisung

```
30 PRINT ASC(A$)
```

der ASCII-Code des ersten Zeichens der der alphanumerischen Variablen zugewiesenen Zeichenkettenkonstanten angezeigt.

Testübung 7.1.3

Es ist ein Programm zu schreiben, durch das die ASCII-Codes der Zeichen $, g, X und + angezeigt werden können; auf das Ansehen der Tabelle der ASCII-Codes soll dabei verzichtet werden.

ASCII-Codes werden selbst beim Schreiben höchst einfacher Programme viel verwendet. Nehmen wir z. B. einmal an, daß wir auf dem Bildschirm ein Anführungszeichen anzeigen lassen wollen. Um das zu erreichen, müssen wir eine Zeichenkettenkonstante erzeugen, die aus einem Anführungszeichen besteht. Die übliche Methode, eine Zeichenkettenkonstante zu definieren, besteht darin, sie in Anführungszeichen einzuschließen. Versucht man das jedoch in unserem speziellen Fall, so müßte man wie folgt codieren:

```
"""
```

Unglücklicherweise betrachtet BASIC diese Kette wie folgt: Das erste Anführungszeichen teilt mit, daß eine Kette beginnen soll, das zweite, daß eine Kette zu beenden ist; das dritte Anführungszeichen wird ignoriert. Somit wird also durch die Anweisung

```
40 PRINT """
```

auf dem Bildschirm nichts angezeigt.

Die ASCII-Codes weisen uns einen Weg aus diesem Dilemma. Der ASCII-Code des Anführungszeichens ist 34. Der Funktionsaufruf CHR$(34) liefert uns demzufolge einen Funktionswert ab, der aus einer einstelligen Zeichenkette, nämlich dem Anführungszeichen, besteht. Das Anführungszeichen läßt sich also durch die Anweisung

```
50 PRINT CHR$(34)
```

auf dem Bildschirm anzeigen.

In ähnlicher Weise können die ASCII-Codes benutzt werden, um in eine Zeichenkette Gerätesteuerzeichen, wie Cursorrücklaufzeichen oder Zeilenvorschubzeichen, in eine Zeichenkette aufzunehmen. Zu bemerken wäre hierzu nämlich, daß man keine Zeichenfolge eintippen kann, die diese Zeichen enthält, denn das Drücken der Eingabetaste ist für BASIC nur das Signal, die soeben eingebene Zeichenfolge als Zeile anzunehmen. Diese beiden Zeichen werden aber nicht in die Zeichenkette selbst aufgenommen. Sollen also die beiden erwähnten Zeichen selbst Bestandteil einer Zeichenkette sein, können wir das nur unter Benutzung der ASCII-Codes erreichen. Näheres darüber werden wir im folgenden Abschnitt aussagen.

Aufgabengruppe 22

1. Es sind die ASCII-Codes der folgenden Zeichen zu bestimmen, ohne dabei auf eine Tabelle zu schauen!
   A,a,B,b,C,c,D,d
2. Von der Aufgabe 1. ausgehend, sind die Beziehungen zwischen den ASCII-Codes der Großbuchstaben und den ASCII-Codes der entsprechenden Kleinbuchstaben zu verallgemeinern, d. h. beispielsweise zwischen A und a, B und b usw.
3. Es ist die Folge der ASCII-Codes anzuzeigen, die zu dem folgenden Satz gehört:
   Er sagt mir, daß der IBM PC ein ausgezeichneter Computer sei

Antworten auf die Testübungen

7.1.1 Die Anzeige führt auf die nachstehende Folge von ASCII-Codes:

```
90,85,77,32,86,69,82,75,65,85,70,58,32,74,117,110,103,101,32,
66,101,97,103,108,101,115,32,109,105,116,32,83,116,97,109,109,
98,97,117,109,44,13,10,
56,32,87,111,99,104,101,110,46,32,50,53,48,32,68,77,46,13,10
```

7.1.2 Das folgende Programm übt die vorgesehene Funktion aus:

```
10  DATA 90,86,...............
11  DATA .....................   siehe die ASCII-Codes
12  DATA .....................   aus der Testübung 7.1.1.
13  DATA .....................
20  FOR  J=1 TO 62
30    READ A
40    PRINT CHR$(A)
50  NEXT J
60  END
```

7.1.3 Das folgende Programm übt die vorgesehene Funktion aus:

```
10  DATA $,g,X,+
20  FOR  J=1 TO 4
30    READ A$
40    B = ASC(A$)
50    PRINT A$,B
60  NEXT J
70  END
```

7.2 Operationen mit Ketten

In den vorangegangenen Kapiteln haben wir uns bisher nur mit solchen Zeichenketten beschäftigt, die druckbare Zeichen enthielten. Nunmehr wollen wir diese Einschränkung fallen lassen und den Kettenbegriff auf alle Zeichen des ASCII-Codes audehnen. So kann z.B. eine Kette auch Zeilenvorschubzeichen, Schreibkopfrücklaufzeichen und jedes der anderen Steuerzeichen, die wir demnächst definieren werden, enthalten. Die in eine Kette aufgenommenen Steuerzeichen werden genau so betrachtet wie jedes andere Zeichen.

In BASIC können mit Ketten eine ganze Reihe verschiedener Operationen ausgeführt werden. Als fundamentalste Operation ist wohl die sogenannte „Kettenaddition" anzusehen, oft auch als „Verkettung" bezeichnet. Angenommen, es liegen die beiden Kettenvariablen A$ und B$ vor. Als Werte sind ihnen zugewiesen:

A$ = "TEXT" und B$ = "VERARBEITUNG"

Dann gilt als „Summe" von A$ und B$, geschrieben zu A$+B$, die Kette, die durch das Aneinanderreihen der A$ bzw. B$ zugewiesenen Ketten erhalten wird, also

"TEXTVERARBEITUNG"

Beim Aneinanderreihen wird kein Leerzeichen zwischen die beiden ursprünglichen Ketten gestellt. Um ein Leerzeichen in der Zielkette zu erhalten, nehmen wir an, daß C$ = " " ist, d.h. der Variablen C$ ist eine Kette zugewiesen, die gerade ein Leerzeichen enthält. Unter diesen Voraussetzungen ergibt sich die Summe A$ + B$ + C$ zu

```
"TEXT VERARBEITUNG"
```

Testübung 7.2.1

Was verbirgt sich hinter A$ + B$, wenn A$ = "4" und B$ = "7" ist?

Testübung 7.2.2

Der Variablen A$ ist als Wert die Zeichenkette

Er sagte: "Nein".<W><Z>

zuzuweisen.

Anmerkung: Unter <W> ist das Schreibkopfrücklaufzeichen (Cursorrücklaufzeichen) und unter <Z> das Zeilenvorschubzeichen zu verstehen (siehe Abb. 7.2).

Bei Benutzung der Funktion LEN kann man die Länge einer Kette ermitteln. Z.B. ergibt

```
LEN("KAUFEN")
```

den Wert 6, da das Wort KAUFEN aus sechs Buchstaben besteht. In ähnlicher Weise ergibt sich, wenn

```
A$ = "Einkommen der Familie"
```

ist, für LEN(A$) der Wert 21. Da die Leerzeichen zwischen den Wörtern vollgültige Zeichen darstellen, sind sie auch mitzuzählen. Das gleiche gilt, wie wir oben schon sagten, auch für die verschiedenen Steuerzeichen, wie Schreibkopfrücklaufzeichen, Zeilenvorschubzeichen usw.

Doch sehen wir uns zur Demonstration der Funktion LEN einmal ein Beispiel an.

Beispiel 1

Es ist ein Programm zu schreiben, das einen Wert für die Zeichenkette A$ einliest. Der eingegebene Wert ist anschließend zentriert auf einer 80-stelligen Bildschirmzeile auszugeben.

Es liegt also gemäß Aufgabenstellung eine 80-stellige Zeile vor, deren Stellen fortlaufend von 1 bis 80 numeriert sind. Die A$ durch das Einlesen zugewiesene Zeichenkette nimmt LEN(A$) dieser Stellen für sich in Anspruch, so daß 80-LEN(A$) Leerstellen auf der Bildschirmzeile verbleiben, die gleichmäßig links und rechts von A$ zu verteilen sind. Die Bildschirmzeile beginnt also mit der Hälfte von 80-LEN(A$) Leerzeichen, d.h. mit (80-LEN(A$))/2 Leerzeichen. Die Zeichenkette muß also bei der Tabulatorstellung

```
TAB((80-LEN(A$))/2+1)
```

beginnen. Somit ergibt sich das in Abb. 7.4 dargestellte Programm.

```
10 INPUT A$
20 CLS
30 PRINT  TAB((80-LEN(A$))/2+1) A$
40 END
```

Abb. 7.4 Anwendung der Funktion LEN

Testübung 7.2.3

Das für das Beispiel 1 (Abb. 7.4) entwickelte Programm ist zur zentrierten Anzeige der Zeichenkette

DER IBM PERSONALCOMPUTER

auf dem Bildschirm zu benutzen.

Es ist ebenfalls möglich, Zeichenketten zu zerlegen; hierzu stehen die Funktionen LEFT$, RIGHT$ und MID$ zu Verfügung. Die Funktionen gestatten die Bildung einer Teilkette, die in der im Funktionsaufruf genannten Länge von links aus, von rechts aus bzw. aus der Mitte der ursprünglichen Kette extrahiert wird. Man betrachte z. B. die Anweisung

```
10  A$ = LEFT$("LIEBE",2)
```

Durch dieses Statement wird der Zeichenkettenvariablen A$ die Teilkette zugewiesen, die aus den beiden linksbündigen Zeichen von "LIEBE" besteht, also "LI". In gleicher Weise führen die beiden Anweisungen

```
20  B$ = "Tennis"
30  C$ = RIGHT$(B$,3)
```

dazu, daß der Zeichenkettenvariablen C$ die Teilkette "nis" zugewiesen wird, d. h. die drei rechtsbündigen Zeichen der Ursprungskette B$. Und noch ein letztes Beispiel: Der Variablen C$ sei die Zeichenkette "Republikaner" zugewiesen. Das Statement

```
50 D$ = MID$(C$,5,3)
```

sorgt dafür, daß der Variablen D$ die Teilkette "bli" zugewiesen wird, d. h. die Teilkette, die beim 5. Zeichen von C$ beginnt und drei Zeichen lang ist.

Testübung 7.2.4

Es ist die Zeichenkette zu nennen, die durch die Funktion

```
RIGHT$(LEFT$("Computer",4),3)
```

bestimmt ist.

Für die Manipulation von Zeichenketten ist es wichtig, daß man den Unterschied zwischen numerischen Daten und Kettendaten erkennt. Die Zahl 14 wird zu 14 geschrieben. Die Zeichenkette dagegen, die aus den beiden Zeichen 14 bestehen soll, ist zu "14" zu schreiben. Im ersten Fall haben wir es mit einer numerischen Konstanten, im zweiten Fall mit einer Zeichenkettenkonstanten, kurz Kettenkonstanten, zu tun. In die arithmetischen Operationen können nur numerische Konstanten eingehen, in den Kettenoperationen, wie LEFT$, RIGHT$ und MID$, haben sie dagegen nichts zu suchen. In diese Operationen können nur Kettenkonstanten eingehen. Wie können wir nun Kettenoperationen auf numerische Konstanten einwirken lassen? Diese Frage stellen, heißt sie natürlich zu beantworten. BASIC sieht hierfür eine sehr einfache Methode vor. Zunächst haben wir die numerischen Konstanten in Kettenkonstanten umzuwandeln, wofür die Funktion STR$ zur Verfügung steht. Unter Zuhilfenahme von ihr können wir z. B. die numerische Konstante 14 in die Kettenkonstante " 14" umwandeln, indem wir die Anweisung

```
60  A$ = STR$(14)
```

formulieren. Als Resultat dieser Operation wird " 14" der Kettenvariablen A$ zugewiesen. Man beachte das Leerzeichen vor der Ziffernfolge 14. Das kommt daher, weil BASIC stets Platz für ein Vorzeichen läßt. Wenn die Zahl positiv ist, wird das Pluszeichen immer durch das Leerzeichen ersetzt. Bei einer negativen Zahl erscheint dagegen stets das Minuszeichen –. Nehmen wir für ein zweites Beispiel nun an, daß die numerische Variable B den Wert 1.457 besitzt. Der Funktionsaufruf STR$(B) liefert dann die Kettenkonstante " 1.457" ab.

Um Kettenkonstanten in numerische Werte umzuwandeln, bedienen wir uns der in BASIC eingebauten Funktion VAL. Das Statement

```
70  B = VAL("3.78")
```

weist z. B. der numerischen Variablen B den Wert 3.78 zu. Man kann sogar VAL für Ketten benutzen, bei denen auf eine Ziffernfolge andere Zeichen folgen. Hier pickt die Funktion VAL selbständig den numerischen Anteil aus der Kettenkonstanten heraus und beachtet den Teil der Kette nicht, der mit dem ersten nichtnumerischen Zeichen beginnt. So ist z. B. VAL("12.5 cm") gleich 12.5

Testübung 7.2.5

Angenommen, der Variablen A$ ist die Kettenkonstante "5 Prozent" und der Variablen B$ die Kettenkonstante "758.45 DM" zugewiesen. Es ist ein Programm zu schreiben, das aus diesen beiden Ausgangswerten 5 % von 758.45 DM berechnet und anzeigt.

7.2.1 Das Statement INSTR[1)]

Bei manchen Anwendungen ist es erforderlich, eine Kette nach einer bestimmten Zeichenfolge zu durchsuchen. Zur Veranschaulichung wollen wir dafür einige Beispiele folgen lassen:

- Man bestimme die Stelle, auf der in der der Kettenvariablen A$ zugewiesenen Kettenkonstanten zuerst der Buchstabe A erscheint.

[1)] INSTR kommt von „IN STRing" (dtsch.: in der Kette)

- Man bestimme die Stelle, auf der in der der Kettenvariablen B$ zugewiesenen Kettenkonstanten zuerst ein Punkt erscheint.
- Man bestimme die Stelle, auf der in der der Kettenvariablen A$ zugewiesenen Kettenkonstanten nach den ersten acht Stellen zuerst die Ziffer 1 erscheint.
- Erscheint die Zeichenfolge „ABS" irgendwo in der der Variablen A$ zugewiesenen Kettenkonstanten?

Solche und ähnliche Suchfragen lassen sich sehr einfach lösen, wenn man die in BASIC eingebaute Funktion INSTR benutzt. Für diese Funktion existieren zwei Formate. Das einfachere stellen wir an einem Beispiel vor:

```
10  P = INSTR(A$,B$)
```

Als Ergebnis der Ausführung dieses Statements wird der (numerischen) Variablen P die Nummer der Stelle zugewiesen, auf der in A$ die Kettenkonstante zum ersten Mal auftritt, die B$ zugewiesen ist. Nehmen wir zu diesem Zweck einmal an, daß der obigen Anweisung die Anweisungen

```
5   A$ = "Hier handelt es sich um einen Test der INSTR-Anweisung"
8   B$ = "Te"
```

In diesem Fall tritt die B$ zugewiesene Kettenkonstante "Te" zum ersten Mal in der A$ zugewiesenen Kettenkonstanten zu Beginn des Wortes "Test" auf. Da dieses Wort auf der 31. Stelle beginnt, wird der Variablen P der Wert 31 zugewiesen.

Wenn die B$ zugewiesene Kettenkonstante nicht in der A$ zugewiesenen enthalten ist, ist der Funktionswert von INSTR gleich Null. Um also zu bestimmen, ob die Kettenkonstante "ABS" in der A$ zugewiesenen Kettenkonstante auftritt, könnte man in ein Programm den Programmabschnitt einfügen, der in der Abb. 7.5 gezeigt ist.

```
...
...
...
10  P = INSTR(A$,"ABS")
20  IF  P = 0  THEN PRINT "ABS tritt nicht auf"
30  IF  P > 0  THEN PRINT "ABS tritt auf"
...
...
...
```

Abb. 7.5 Auftreten von Ketten

Wenn das Suchen nach B$ in A$ erst ab einer bestimmten Stelle m beginnen soll, muß man sich des zweiten Formats der Funktion INSTR bedienen. Dieses lautet:

```
10  P = INSTR(m,A$,B$)
```

Wenn wir also beispielsweise erst ab der 8. Stelle der A$ zugewiesenen Zeichenkette nach dem Auftreten der B$ zugewiesenen Zeichenkette suchen wollen, so können wir codieren:

```
20  P = INSTR(8,A$,B$)
```

Der Variablen P wird auch bei diesem Format der Wert 0 zugewiesen, wenn die B$ zugewiesene Kettenkonstante nicht ab der 8. Stelle von A$ gefunden wird.

7.2.2 Regeln für die Beziehungen zwischen Ketten

Der Computer behandelt Beziehungen zwischen Ketten in ziemlich der gleichen Art und Weise, wie er die Beziehungen zwischen Zahlen handhabt. So sagen wir z.B. daß die Ketten A$ und B$ gleich sind, geschrieben zu A$=B$, vorausgesetzt, daß sie aus den gleichen Zeichen in der gleichen Reihenfolge bestehen. Anderenfalls sind die Ketten ungleich, wofür wir

```
A$<>B$        oder        A$><B$
```

schreiben können. Der Ausdruck A$<B$ besagt, daß die Kette von A$ der Kette von B$ vorausgeht, gesehen in alphabetischer Hinsicht. Scharf gilt diese Aussage natürlich nur für Ketten, die allein aus Buchstaben bestehen. Enthalten Ketten neben Buchstaben auch Ziffern und Sonderzeichen, so bestimmen die ASCII-Codes der Zeichen die Reihenfolge derselben. In ähnlicher Weise bedeutet A$>B$, daß A$ in alphabetischer Hinsicht bzw. gemäß der Reihenfolge der ASCII-Codes auf B$ folgt. Bleiben wir bei ein paar Beispielen, die Beziehungen sind bei jedem dieser Beispiele korrekt wiedergegeben.

```
"baer" < "ziege"
"maedchen" > "junge"
```

Der Ausdruck A$>=B$ bedeutet, daß A$ entweder größer als oder gleich B$ ist. Dafür können wir auch sagen, daß die Kette von A$ der Kette von B$ in alphabetischer Hinsicht oder gemäß der ASCII-Codes nachfolgt oder daß beide Ketten identisch sind. Die Beziehung A$<=B$ hat eine ähnliche Bedeutung (kleiner als oder gleich).

Beispiel 2:

Es ist ein Programm zu schreiben, daß die Wörter

Ei, Sellerie, Ball, Beutel, Handschuh, Mantel, Spinat, Unterhose, Klee, Anzug, Kleid, Gras, Kuh, Henne

in alphabetische Reihenfolge bringt.

Bevor wir das Programm niederschreiben, sind einige Bemerkungen zu dem von uns eingeschlagenen Lösungsweg angebracht. Wir führen einen Kettenbereich B$(J) mit $J=1,2,\cdots,13,14$ ein, der diese 14 Wörter enthält. Beginnend beim Bereichselement B$(1) vergleichen wir jedes Wort mit dem nächsten Wort im Bereich. Wenn diese beiden Wörter nicht in Reihenfolge angeordnet sind, vertauschen wir sie. Diese Prüfung setzen wir für jedes Paar aufeinanderfolgender Wörter bis zum Bereichsende fort. Diese Operationen reichen aus, um das erste Wort des Bereiches auf einen geeigneten Platz innerhalb des Bereiches zu verschieben, alle anderen Wörter erforderlichenfalls gerade um einen Platz nach vorn. Diesem ersten Vergleichszyklus lassen wir weitere folgen, jeweils von vorn beginnend. Der zweite Vergleichszyklus plaziert das zweite Wort auf einen geeigneten Platz innerhalb des Bereiches usw. Ingesamt werden 13 Vergleichszyklen durchgeführt. Dadurch werden schließlich alle Wörter auf die ihnen gebührenden Plätze innerhalb des Bereiches gebracht. Das unter diesen Aspekten entstandene Sortierprogramm ist in der Abb. 7.6 dargestellt.

Natürlich können wir das Programm noch so modifizieren, daß es zum Sortieren von beliebigen Listen irgendwelcher Zeichenketten taugt. Einzelheiten hierzu wollen wir den Übungsaufgaben überlassen.

```
100  DIM B$(14)
110  DATA Ei, Sellerie, Ball, Beutel, Handschuh, Mantel, Spinat
120  DATA Unterhose, Klee, Anzug, Kleid, Gras, Kuh, Henne
130  'Aufbau des Bereiches B$
140  FOR  J=1 TO 14
150    READ B$(J)
160  NEXT J
170  'Vergleichsschleifen
180  FOR J=1 TO 13
190    FOR K=1 TO 13
200      IF  B$(K+1) < B$(K)  THEN 210  ELSE 250
210      'Vertauschen von B$(K) mit B$(K+1)
220      C$ = B$(K)
230      B$(K) = B$(K+1)
240      B$(K+1) = C$
250    NEXT K
260  NEXT J
270  'Anzeige der Resultate
280  FOR J=1 TO 14
290    PRINT B$(J)
300  NEXT J
1000 END
```

Abb. 7.6 Sortierprogramm für Zeichenketten

Aufgabengruppe 23

1. Das Programm vom Beispiel 2 (siehe Abb. 7.6) soll dazu benutzt werden, die Wörter der nachfolgend aufgeführten Liste in ihre alphabetische Reihenfolge zu bringen.
   Gerechtigkeit, Mitte, Probe, Charakter, Kapital, Suchen, Ersetzen, Folgen, Passwort, Erweitern

2. Es ist ein Programm zu schreiben, durch das die Additionsaufgabe $15 + 48 + 97 = 160$ in der Form

$$\begin{array}{r} 15 \\ 48 \\ \underline{97} \\ 160 \end{array}$$

   ausgegeben wird.

3. Es ist ein Programm zu schreiben, das die Eingabe der Zeichenketten „$6718.49" und „$4801.96" anfordert und die Summe dieser beiden Dollarbeträge berechnet und ausgibt.

Antworten auf die Testübungen

7.2.1 "47"

7.2.2 `A$ = "Er sagte: " + CHR$(34) + "Nein" + CHR$(34) + "." + CHR$(13)`
`+ CHR$(10)`

7.2.3 Nach Eintippen des Befehles RUN ist die Eingabetaste zu betätigen. Beim Erscheinen der Systemanfrage ist die vorgegebene Zeichenkette einzugeben und wiederum die Eingabetaste zu drücken.

7.2.4 "omp"

7.2.5 Das nachstehende Programm löst dieses Problem:

```
10  A$ = "5 Prozent":  B$ = "758.45 DM"
20  A = VAL(A$):       B = VAL(B$)
30  PRINT  A$, "VON", B$, "IST GLEICH",
40  PRINT  A*B*0.01, "DM"
50  END
```

7.3 Steuerzeichen

In der Abb. 7.7 ist eine Aufstellung der Steuerzeichen enthalten, die mit den ASCII-Codes 0 bis 31 verschlüsselt sind. Es ist sicher sehr nützlich, wenn wir zu den verschiedenen Codes noch einige Erklärungen abgeben:

ASCII-Code	Zeichen	Steuerzeichen
0	Nichtigzeichen (Nullzeichen)	NUL
1	Graphisches Zeichen ☺	SOH
2	Graphisches Zeichen ●	STX
3	Graphisches Zeichen ♥	ETX
4	Graphisches Zeichen ♦	EOT
5	Graphisches Zeichen ♣	ENQ
6	Graphisches Zeichen ♠	ACK
7	(Piepzeichen oder Piepton)	BEL
8	Graphisches Zeichen ◘	BS
9	(Tabulatorzeichen)	HT
10	(Zeilenvorschubzeichen)	LF
11	(Haus- oder Heimzeichen)	VT
12	(Formularvorschubzeichen)	FF
13	(Wagenrücklaufzeichen)	CR
14	Graphisches Zeichen ♫	SO
15	Graphisches Zeichen ☼	SI
16	Graphisches Zeichen ►	DLE
17	Graphisches Zeichen ◄	DC1
18	Graphisches Zeichen ↕	DC2
19	Graphisches Zeichen ‼	DC3
20	Graphisches Zeichen ¶	DC4
21	Graphisches Zeichen §	NAK
22	Graphisches Zeichen ▬	SYN
23	Graphisches Zeichen ↨	ETB
24	Graphisches Zeichen ↑	CAN
25	Graphisches Zeichen ↓	EM
26	Graphisches Zeichen →	SUB
27	Graphisches Zeichen ←	ESC
28	(Cursor-Bewegung nach rechts)	FS
29	(Cursor-Bewegung nach links)	GS
30	(Cursor-Bewegung nach oben)	RS
31	(Cursor-Bewegung nach unten)	US

Anmerkung: Die in der letzten Spalte aufgeführten Abkürzungen sind genormte Abkürzungen.

Abb. 7.7 ASCII-Codes von Steuerzeichen

- *Code 0 (Nichtigzeichen oder Nullzeichen)*
  Der Name suggeriert bereits die Bedeutung dieses Zeichens; es bedeutet nämlich nichts,

d.h. das Zeichen veranlaßt keine Operation. Es wird oft bei der Nachrichtenübermittlung benutzt, um eine Nachricht mit einer Kette von Nichtigzeichen beginnen zu lassen.

- *Codes 1 bis 6*
  Diese Codes repräsentieren graphische Zeichen. Sie werden bei Anzeigen in Spielen verwendet.
- *Code 7 (Piepzeichen)*
  Dieses Zeichen veranlaßt den Computer, einen Piepton von sich zu geben.
- *Code 8 (Rücksetzzeichen)*
  Durch dieses Zeichen wird der Positionsanzeiger (Cursor) um eine Stelle zurückgesetzt.
- *Code 9 (Tabulatorzeichen)*
  Dieses Zeichen veranlaßt, daß der Positionsanzeiger (Cursor) bis zur nächsten Tabulatorstelle geführt wird. BASIC plaziert automatisch auf jede 5. Stelle der Zeilen einen Tabulator.
- *Code 10 (Zeilenvorschubzeichen)*
  Dieses Zeichen veranlaßt, daß der Positionsanzeiger (Cursor) um eine Zeile nach unten verschoben wird.
- *Code 11 (Haus- oder Heimzeichen)*
  Dieses Zeichen veranlaßt, daß der Positionsanzeiger (Cursor) in die linke obere Ecke des Bildschirmes gestellt wird.
- *Code 12 (Formularvorschubzeichen)*
  Dieses Zeichen veranlaßt, daß bei Druckern das zu bedruckende Papier bis zum Beginn der nächsten Seite vorgeschoben wird.
- *Code 13 (Schreibkopf- oder Cursorrücklaufzeichen)*
  Dieses Zeichen veranlaßt, daß die Positionsanzeiger (Cursor) auf die am weitesten links gelegene Stelle der Zeile rückt, auf der er sich gerade befindet (Einstellung auf Zeilenanfang).
- *Codes 14 bis 27*
  Diese Zeichen stellen weitere graphische Zeichen dar, die in Bildschirmanzeigen Verwendung finden.
- *Code 28 (Zeichen für Cursor-Bewegung nach rechts)*
  Dieses Zeichen veranlaßt, daß der Positionsanzeiger (Cursor) um eine Stelle nach rechts verschoben wird.
- *Code 29 (Zeichen für Cursor-Bewegung nach links)*
  Dieses Zeichen veranlaßt, daß der Positionsanzeiger (Cursor) um eine Stelle nach links verschoben wird.
- *Code 30 (Zeichen für Cursor-Bewegung nach oben)*
  Dieses Zeichen veranlaßt, daß der Positionsanzeiger (Cursor) um eine Stelle nach oben, d.h. in die vorhergehende Zeile, verschoben wird.
- *Code 31 (Zeichen für Cursor-Bewegung nach unten)*
  Dieses Zeichen veranlaßt, daß der Positionsanzeiger (Cursor) um eine Stelle nach unten, d.h. in die nachfolgende Zeile, verschoben wird.

Die ASCII-Steuerzeichen sind in den Ausgabestatements (PRINT bzw. LPRINT) genau so zu benutzen wie die druckbaren Zeichen. Wenn wir also z.B. den Cursor um eine Zeile nach oben rücken wollen, gebrauchen wir einfach das Statement

```
10  PRINT CHR$(30);
```

Es wäre hierzu noch zu bemerken, daß wir diese Anweisung durch ein Semikolon abgeschlossen haben. Dadurch können wir verhindern, daß BASIC ein Cursorrücklauf- und ein Zeilenvorschubzeichen im Anschluß an das PRINT-Statement erzeugt. Könnte nämlich BASIC die Generierung dieser beiden Zeichen vornehmen, dann würde die Cursorpositionierung, die durch den ASCII-Code 30 veranlaßt wird, wieder zunichte gemacht.

Testübung 7.3.1

Es ist ein Programm zu schreiben, durch das der Cursor zwei Stellen nach rechts und zwei Zeilen nach unten bewegt werden kann

7.3.1 Ergänzende Betrachtungen über den Positionsanzeiger (Cursor)

Die ASCII-Verschlüsselungen, die die Bewegungen des Positionsanzeigers steuern, gestatten die Einstellung des Cursors relativ zu seiner momentan eingenommenen Position. Darüberhinaus kann der Cursor außerdem in eine beliebige Position auf dem Bildschirm gebracht werden, indem man das LOCATE-Statement benutzt. Dieses Statement besitzt das folgende allgemeine Format (wir haben dieses Statement bereits im Abschnitt 3.4 gestreift, wiederholen es hier also):

```
LOCATE  z,s
```

Hierbei bedeuten:

z	Zeilennummer
s	Spaltennummer

Die Ausführung dieses Statements hat also zur Folge, daß der Positionsanzeiger auf die Spalte *s* der Zeile *z* eingestellt wird. Um beispielsweise den Cursor auf die 5. Spalte der 20. Zeile zu führen, ist demnach die Anweisung

```
100  LOCATE  20,5
```

erforderlich.

Die Spalte, auf der sich der Cursor augenblicklich befindet, kann durch den Aufruf der BASIC-Funktion POS(0) bestimmt werden. Wenn sich der Positionsanzeiger, um bei einem Beispiel zu bleiben, momentan auf der Spalte 37 befindet, dann liefert POS(0) den Funktionswert 37. Ebenso ergibt der Aufruf der Funktion CSRLIN stets die Nummer derjenigen Zeile, auf der sich der Cursor gegenwärtig befindet. Steht also z.B. der Positionsanzeiger auf der Zeile 5, dann ist der Funktionswert von CSRLIN gleich 5. Man kann die beiden Funktionen POS(0) und CSRLIN in BASIC-Programmen genauso benutzen wie die Variablen.

Aufgabengruppe 24

1. Durch ein Programm ist zunächst auf dem Bildschirm die Zeichenfolge HALLO auszugeben, und anschließend ist der Cursor auf den Buchstaben H einzustellen.
2. Es ist ein Programm zu schreiben, durch das der Cursor infolge bestimmter Rückantworten des Benutzers bewegt wird. Folgende Rückantworten sind zulässig:

S → Bewegung des Cursors um eine Stelle nach links
D → Bewegung des Cursors um eine Stelle nach rechts
E → Bewegung des Cursors um eine Zeile nach oben
X → Bewegung des Cursors um eine Zeile nach unten
Jede andere Rückantwort soll die Stellung des Cursors nicht beeinflussen.

3. Das Programm von Aufgabe 2. ist so zu modifizieren, daß es eine Folge von Cursorbewegungs-Zeichen akzeptiert, z. B. die nachstehende Folge:
   SSDDXEEEESSSDDD
4. Es ist die Bewegung des Cursors zu verschiedenen Stellen auf dem Bildschirm zu praktizieren.
5. Es ist ein Programm zu schreiben, durch das der Cursor auf diejenige Spalte der letzten Zeile bewegt werden kann, auf der er sich augenblicklich befindet.
6. Es ist ein Programm zu schreiben, durch das der Cursor auf den Anfang derjenigen Zeile bewegt werden kann, auf der er sich augenblicklich befindet.

Antwort auf die Testübung 7.3.1

```
10  PRINT   CHR$(28); CHR$(28); CHR$(31); CHR$(31)
20  END
```

7.4 Steuerzeichen für Drucker und für Formulare

Wir wollen uns noch einmal den Druckern zuwenden. Im Kapitel 2 befaßten wir uns zwar mit dem Statement LPRINT und dem Befehl LLIST, doch diskutierten wir bisher nicht die speziellen Gesichtspunkte, die den Gebrauch von Druckern betreffen, wie die Steuerung der Seitenformate usw. Für die Anwendungen der Textverarbeitung ist es für die endgültige Gestaltung der Dokumente absolut wichtig, daß wir auf die Randbegrenzungen, auf die Anzahl der Zeilen pro Seite usw. Einfluß nehmen können. In diesem Abschnitt wollen wir über die Steuerzeichen für die Drucker und im Abschnitt 7.5 über ihren Gebrauch bei der Erzeugung von Briefen unter Benutzung von Adreßliste sprechen.

7.4.1 Betrieb von Druckern

Von der erstmaligen Benutzung muß der Drucker in geeigneter Weise an den IBM Personalcomputer angeschlossen werden. Am einfachsten können die IBM Drucker angeschlossen werden (der Matrixdrucker oder der graphische Drucker). Der Matrixdrucker arbeitet mit einer Ausgabegeschwindigkeit von 80 Zeichen/Sekunde (diese Maßeinheit wird zukünftig von uns mit Z/s abgekürzt). Die Einfachheit des Anschlusses ist damit zu begründen, daß die für die Kommunikation zwischen Computer und Drucker erforderlichen elektronischen Schaltkreise bereits aufeinander abgestimmt sind, d. h. sie sind bereits vom Entwurf her auf Zusammenarbeit ausgerichtet. Wenn man sich jedoch dafür entscheidet, einen Drucker anzuschließen, der nicht von der IBM stammt, so vergewissere man sich unbedingt, ob der Anschluß möglich ist, besser noch, man lasse sich Unterstützung bei der Verbindung der beiden Geräte zusichern. (Der Anschluß von Fremdgeräten kann, aber muß nicht eine heikle Arbeit sein. Nebenbei ge-

sagt, sind die Anschlußbedingungen von der IBM offengelegt worden, so daß bei Einhaltung derselben durch Fremdhersteller keine Probleme aufzutreten brauchen, sofern dieselben ihre Drucker unter IBM-Spezifikationen gefertigt haben). Wir wollen annehmen, daß ein Drucker ordnungsgemäß angeschlossen und betriebsbereit ist.

Der Drucker akzeptiert eine Folge von ASCII-Zeichencodes. Die meisten dieser Codes entsprechen Buchstaben, Ziffern und Sonderzeichen, die gedruckt werden können. Einige jedoch entsprechen Steuerzeichen; durch sie werden Gerätefunktionen (Druckerfunktionen) angestoßen bzw. ausgelöst, wie Schreibkopfrücklauf, Zeilentransport (Zeilenvorschub), Vorschub zum Beginn einer neuen Druckseite (Formularvorschub) usw. Für den Drucker setzt sich eine Zeile aus einer Folge von druckbaren Zeichen zusammen. Diese endet mit einem <E>[1]. Durch <E> werden dem Drucker zwei Funktionen signalisiert:

1. Es wird ein Schreibkopfrücklauf veranlaßt.
2. Es wird dafür gesorgt, daß das Druckpapier um eine Zeile weitertransportiert wird; daher spricht man vom Zeilentransport.

Zu bemerken wäre noch, daß der graphische Drucker der IBM die Reproduktion der graphischen Zeichen ermöglicht, die den ASCII-Codes 128 bis 255 entsprechen. Im nächsten Kapitel werden wir über diese Zeichen sprechen. Gewisse Druckertypen gestatten ebenfalls das Drucken graphischer Zeichen, aber diese Zeichen müssen nicht unbedingt mit denen der IBM-Drucker übereinstimmen. Weitere Informationen über dieses Problem sind in den Handbüchern der Hersteller der betreffenden Drucker zu finden.

7.4.2 Zeilenlänge

Die Drucker weisen verschiedene Zeilenlängen auf. Außerdem kommt verschieden breites Druckpapier in den Handel. Aus diesem Grunde ist es notwendig, daß die Zeilenlänge auf die entsprechende Anwendung eingestellt wird. Hierfür steht das Statement WIDTH zur Verfügung. In diesem Statement wird die Zeilenlänge durch die Anzahl der auf einer Zeile vorgesehenen Stellen (Spalten) festgelegt. Wenn z. B. eine Zeile 80 Druckstellen umfassen soll, so hätte man eine Anweisung der Form

```
10  WIDTH "LPT1:",80
```

einzufügen. Unter LPT1: ist, wie schon besprochen, der Gerätename zu verstehen. Diese Anweisung kann irgendwo im Programm erscheinen. Durch Weglassung der Zeilennummer wird dieses Statement zu einem Sofortbefehl. Die Zeilenlänge bleibt, sofern sie nicht geändert wird, bis zum Ausschalten des Computers erhalten. Für die beiden IBM Druckertypen gilt 80 als Standardzeilenlänge (voreingestellte Zeilenlänge); diese Zeilenlänge braucht daher nicht gesondert festgelegt zu werden.

Betonen möchten wir ausdrücklich, daß die Zeilenlänge als Stellenzahl pro Zeile festzulegen ist und nicht in Zoll oder cm. Beim IBM Matrixdrucker ist außerdem zu berücksichtigen, daß die Lettern in drei verschiedenen Typenbreiten zur Verfügung stehen. Bei der Planung der Zeilenlänge sind deshalb notwendigerweise auch die Papierbreite und die Typenbreite in Rechnung zu ziehen.

---

[1] Unter <E> wollen wir wie üblich die Zeichenkombination verstehen, die dem Drücken der Eingabetaste gleichkommt.

7.4.3 Setzen der Zeichendichte

Die für eine Zeile festgelegte Druckstellenzahl hat, wie wir soeben betont haben, nicht allein Einfluß auf die Papierbreite. Vielmehr muß auch berücksichtigt werden, wieviel Zeichen auf einem Zoll (2,54 cm) untergebracht werden können, d. h. welche Zeichendichte vorliegt. Der IBM Matrixdrucker kann in einer von drei verschiedenen Zeichendichten drucken:

- 10 Zeichen/Zoll (einfache Zeichendichte)
- 5 Zeichen/Zoll (doppelte Zeichendichte)
- 132 Zeichen auf 8 Zoll, d. h. 16,5 Zeichen/Zoll (komprimierte Zeichendichte)

Normalerweise wird man die einfache Zeichendichte für die Druckausgaben wählen. Diese Zeichendichte steht auch unmittelbar nach dem Einschalten des Druckers zur Verfügung. Um die Druckeinrichtung auf die doppelte Zeichendichte einzustellen, ist das Statement

```
LPRINT CHR$(14);
```

zu verwenden, entweder als Anweisung in einem Programm (dann mit Zeilennummer) oder als Sofortbefehl (ohne Zeilennummer). Zur Auswahl der komprimierten Zeichendichte dient das Statement

```
LPRINT CHR$(15);
```

7.4.4 Setzen des vertikalen Zeilenabstandes (Zeilendichte)

Der Abstand der Druckzeilen voneinander errechnet sich aus der Anzahl der Druckzeilen pro Zoll (2,54 cm). Es sind die folgenden Einstellungen für den IBM Matrixdrucker möglich:

- 6 Zeilen/Zoll
- 8 Zeilen/Zoll
- 72/7 Zeilen/Zoll

Die ersten beiden Einstellungen eignen sich vor allem für die normalen Zeichendichten, die letztere wird vor allem für die komprimierte Zeichendichte benutzt. Beim Einschalten des Druckers wird die Druckeinrichtung auf 6 Zeilen/Zoll gesetzt. Um auf 8 Zeilen/Zoll überzugehen, ist die Verwendung des Statements

```
LPRINT CHR$(27);CHR$(69);
```

Die Rückkehr zur einfachen Zeichendichte, d. h. zu 10 Zeichen/Zoll, bewirkt das Statement

```
LPRINT  CHR$(27); "AO";
```

erforderlich. Der Übergang auf 72/7 Zeilen pro Zoll erfordert die Benutzung des Statements

```
LPRINT  CHR$(27); "A1";
```

Um den Zeilenabstand nach Wahl eines anderen Zeilenabstandes wieder auf den normalen (6 Zeilen/Zoll) einzustellen, ist die Verwendung des Statements

```
LPRINT  CHR$(27); "A2";
```

erforderlich.

7.4.5 Festlegung der Seitenlänge

Die Seitenlänge bestimmt sich aus der Anzahl der Zeilen pro Seite. Beim Einschalten des Computers ist eine standardisierte Seitenlänge von 66 Zeilen/Seite wirksam; diese Festlegung ist auf die Zeilendichte von 6 Zeilen/Zoll und auf eine Papierseitenlänge von 11 Zoll (27,94 cm, also etwas weniger als die Länge eines DIN A4 Bogens) abgestimmt. Durch Benutzung von Statements der Form

```
LPRINT  CHR$(27); "C"; CHR$(33);
```

kann die Seitenlänge geändert werden, in unserem Beispiel beträgt nunmehr die Seitenlänge nur noch 33 Zeilen/Seite.

7.4.6 Festlegung des Seitenanfangs

Der Computer verfolgt durch Zählen der Zeilen pro Seite, wieviel Zeilen bereits zum Drucken übermittelt wurden. Nach der letzten Zeile, die gemäß der festgelegten Seitenlänge auf eine Seite gedruckt werden kann, sendet der Computer das Formularvorschubsteuerzeichen (Steuerzeichen für Seitenvorschub oder neue Seite) zum Drucker. Das hat zur Folge, daß beim Drucker das Druckpapier weitertransportiert wird, und zwar bis zum Beginn der nächsten Seite (gemäß der definierten Seitenlänge). Zu Beginn einer Programmiersitzung wird der Zeilenzähler auf 1 gesetzt, entspricht damit der Einstellung des Druckpapiers auf den Anfang einer Seite.

Wenn man die Ausgabe eines Programmes gezielt auf den Beginn einer Seite drucken lassen will, muß man in das Programm eine BASIC-Anweisung der Form

```
10  LPRINT CHR$(12);
```

aufnehmen, wodurch beim Drucker eine Einstellung auf den Beginn einer neuen Seite veranlaßt wird; der ASCII-Code 12 entspricht ja bekanntlich dem Gerätesteuerzeichen für den Formularvorschub. Die Ausführung dieser Anweisung bewirkt also einen Papiertransport über die restlichen, noch unbedruckten Zeilen der laufenden Seite hinweg.

7.4.7 Randbegrenzungen

Die Parameter für Zeilenlänge und Seitenlänge berücksichtigen keine Randbegrenzungen, weder die Randbegrenzungen oben und unten auf einer Seite, noch die links und rechts auf einer Seite. Die Randbegrenzungen oben und unten auf einer Seite kann man durch Drucken von Leerzeilen abdecken. Die linke Randbegrenzung ist durch die Lage des Papiers im Drukker bestimmt, die rechte Randbegrenzung durch die Definition der Zeilenlänge.

Testübung 7.4.1

Angenommen, wir wollen mit einer Zeilendichte von 10 Zeichen/Zoll und einer Zeilendichte von 8 Zeilen/Zoll drucken. Es ist zu überlegen, durch welche Sofortbefehle erreicht werden kann, daß Randbegrenzungen von je einem Zoll auf beiden Seiten des (in den USA) genormten 8½ × 11 Zoll beim Drucken verbleiben.

Geisenbrunn, den 24. September 1984

.....
.....
.....
.....

Sehr geehrter Herr .....,

alle Mitarbeiter der "Althaus Renovierungsgesellschaft mbH" haben Sie in der Vergangenheit als Kunden schätzen gelernt. Wir teilen Ihnen nun heute mit, daß wir am 15. Okt. 1984 unsere Geschäftsräume in die Leipziger Str. 9 verlegen. Unsere neuen Räume werden größer als die alten sein; Sie werden sie auch übersichtlicher gestaltet finden. Zusätzlich werden wir unsere Angebotspalette durch ein umfassendes Sortiment von energiesparenden Türen und Fenstern erweitern. Wir erwarten Sie auch in unseren neuen Räumen als Kunden und werden Sie wie bisher zu Ihrer Zufriedenheit bedienen. Bitte lassen Sie uns wissen, wie wir Sie bei Ihren Umbauplänen unterstützen können.

Mit den besten Empfehlungen

Ihr Mathias Melker
Geschäftsführer der
"Althaus Renovierungsgesellschaft mbH"

Anmerkungen:
---

1) Die punktierten Zeilen werden durch die vier Zeichenketten ersetzt, die in jeder Eintragung der Datei KUNDEN enthalten sind (Adreßzeilen).
2) Die kursiv geschriebenen Zeilen werden durch das Programm hinzugestellt.
3) Der punktierte Teil der Anredezeile, d. h. der Zeile, die den Text
   Sehr geehrter Herr .....,
   enthält, wird durch den Nachnamen des Kunden ersetzt; dieser wird im Programm aus der ersten Zeichenkette jeder Kundeneintragung extrahiert.
4) Die erste Zeichenkette jeder Eintragung in der Datei KUNDEN enthält den gesamten Namen des Kunden in der Form
   Vorname Nachname
   Zwischen Vornamen und Nachnamen befindet sich also ein Leerzeichen (Zwischenraumzeichen).
5) Der Briefrahmen kann bei Verwendung des standardisierten, 11 Zoll langen Druckpapiers (66 Zeilen Umfang bei einer Zeilendichte von 6 Zeilen/Zoll) aus maximal 43 Zeilen bestehen. Die übrigen 23 Zeilen dienen folgenden Zwecken:
   --- 4 Adreßzeilen (aus Datei KUNDEN)
   --- 1 Anredezeile
   --- 4 Unterschriftszeilen
   --- je 3 Leerzeilen oben und unten auf jeder Briefseite
   --- 8 sonstige Leerzeilen

Abb. 7.8 Schematische Darstellung eines Rahmens für einen Geschäftsbrief

Antwort auf die Testübung 7.4.1

```
WIDTH "LPT1:",65
LPRINT CHR$(27);"AO";
```

Das Papier ist im Drucker so einzuführen, daß ein Zoll links vor der 1. Druckstelle verbleibt.

7.5 Gestaltung von Briefen

Man kann die Fähigkeiten des Computers zur Handhabung von Zeichenketten dazu benutzen, um Briefrahmen zu präparieren, die zum Schreiben von Briefen, die wie eine persönliche Korrespondenz aussehen, dienen können. Wir wollen diese Methode illustrieren, indem wir den in der Abb. 7.8 dargestellten Briefrahmen aufbauen und unter Benutzung desselben einen Stapel Briefe drucken; jeder dieser Briefe ist individuell adressiert.

Nehmen wir nun einmal an, daß der in der Abb. 7.8 dargestellte Rahmen als Geschäftsbrief an 100 Kunden, die in einer Adreßliste erfaßt sind, verteilt werden soll. Wir setzen weiterhin voraus, daß diese Adreßliste in einer sequentiellen Datei, die auf einer Diskette liegt, enthalten ist. Jede einzelne Adresse besteht aus vier Zeichenketten, die die folgenden Daten aufweisen:

- Namen des Empfängers (Kunden)
- Firma
- Geschäftsadresse (Straße)
- Postleitzahl und Ort

Diese Adreßdatei bestitzt den Namen KUNDEN . Es ist nun ein Programm zu schreiben, das den verlangten Stapel von Briefen produziert.

Unser Programm wird aus zwei Hauptteilen bestehen. Der erste Hauptteil ermöglicht uns die Eingabe des Briefrahmens (Beispiel siehe Abb. 7.8). Die einzelnen Zeilen des Briefrahmens sind genau so einzutippen, wie wir es bei einer Schreibmaschine tun würden. Der Computer speichert die Zeilen des Briefrahmens im Zeichenkettenbereich A$; in A$(1) wird die erste Zeile, in A$(2) die zweite Zeile usw. abgelegt. Maximal wollen wir für den Briefrahmen 50 Zeilen zulassen. Weiterhin wollen wir das Ende des Briefrahmens, damit es im Programm entsprechend interpretiert werden kann, durch Eingabe eines Prozentzeichens, gefolgt vom Anschlagen der Eingabetaste, anzeigen.

Sobald das Prozentzeichen erkannt worden ist, geht das Programm zum zweiten Hauptteil über. Dieser befaßt sich mit der eigentlichen Erzeugung des Briefstapels aus dem zuvor eingegebenen Adreßrahmen und der in der Datei KUNDEN enthaltenen Adreßliste. Zunächst wird die Datei KUNDEN als Eingabedatei eröffnet. Nacheinander werden die in dieser Datei stehenden Adreßeintragungen gelesen. Nach dem Lesen jeder Eintragung wird das Datum im Briefkopf gedruckt. Darauf folgt das Drucken der Adreßzeilen. Als nächstes ermittelt das Programm den Nachnamen des Kunden aus der ersten Adreßzeile und fügt die Wortfolge "Sehr geehrter Herr . . .," ein, generiert damit die Anrede. Danach erfolgt der Druck des eingegebenen Briefrahmens. Das auf diese Weise aufgebaute Programm ist in der Abb. 7.9 aufgelistet.

Das in der Abb. 7.9 vorgelegte Programm arbeitet nur dann einwandfrei, wenn einige Punkte, auf die wir nachstehend gesondert hinweisen wollen, beachtet werden. Die erste Zeile, die wir

```
10  DIM A$(43),B$(4)
20  PRINT "NACH JEDEM ? BITTE EINE BRIEFZEILE EINGEBEN;"
30  PRINT "DANACH IST DIE EINGABETASTE ZU DRUECKEN"
40  PRINT "ZUR BEENDIGUNG EINES BRIEFES IST % EINZUGEBEN,"
50  PRINT "GEFOLGT VOM DRUECKEN DER EINGABETASTE"
60  J = 1 :                    'Die Variable J enthaelt die Anzahl der Zeilen
70  INPUT A$(J)
80  IF  A$(J) = "%"  THEN 100
90  J = J + 1 :                'Uebergang zu naechsten Zeile
100 GOTO 70
110 OPEN "KUNDEN" FOR INPUT AS #1 :    'Eroeffnen der Kundendatei
120 FOR  N = 1  TO  100
130   INPUT #1, B$(1),B$(2),B$(3),B$(4)
140   LPRINT A$(1) :           'Drucken des Datums
150   LPRINT :                 'Erzeugung einer Leerzeile
160   LPRINT :                 'Erzeugung einer Leerzeile
170   'Zeilen 180 bis 210 ----> Drucken der Adressen
180   LPRINT B$(1) :           'Drucken der 1. Adresszeile
190   LPRINT B$(2) :           'Drucken der 2. Adresszeile
200   LPRINT B$(3) :           'Drucken der 3. Adresszeile
210   LPRINT B$(4) :           'Drucken der 4. Adresszeile
220   LPRINT :                 'Erzeugung einer Leerzeile
230   LPRINT :                 'Erzeugung einer Leerzeile
240   'Erkennen des Nachnamens des Adressaten
250   L = LEN(B$(1))
260   FOR  K = 0  TO  L-1
270     LET  C$ = MID$(B$(1),L-K,1)
280     IF  C$ = " "  THEN 300 :
                'Pruefen auf Zwischenraumzeichen vor dem Nachnamen
290   NEXT K
300   D$ = RIGHT$(B$(1),K) :   'D$ enthaelt den Nachnamen
310   LPRINT "Sehr geehrter Herr "; D$; ","
320   LPRINT :                 'Erzeugung einer Leerzeile
330   FOR  M = 2  TO  J :      'Drucken der Zeilen des Briefrahmens
340     LPRINT A$(M)
350   NEXT M
360   LPRINT :                 'Erzeugung einer Leerzeile
370   LPRINT :                 'Erzeugung einer Leerzeile
380   LPRINT TAB(23) "Mit den besten Empfehlungen"
390   LPRINT :                 'Erzeugung einer Leerzeile
400   LPRINT TAB(23) "Ihr  Mathias Melker"
410   LPRINT TAB(23) "Geschäftsführer der"
420   LPRINT TAB(23) CHR$(34);"Althaus Renovierungsgesellschaft mbH";CHR$(34)
430   LPRINT CHR$(12); :       'Papiervorschub auf neue Seite
440 NEXT N :                   'Uebergang zum naechsten Kunden
450 CLOSE :                    'Abschliessen der Kundendatei
500 END
```

Abb. 7.9 Programm zum Drucken eines Stapels von 100 Geschäftsbriefen

für die Zusammenstellung des Briefrahmens eingeben müssen, ist die Zeile mit dem Datum. Die Anredezeile, d.h. die Zeile, die

```
Sehr geehrter Herr.........,
```

lautet, darf nicht eingegeben werden, denn sie wird vom Programm zum Briefrahmen hinzugestellt. Man beachte außerdem, daß das vorliegende Programm keine Alternative zu dieser Anrede zuläßt. Dadurch wird eine Reihe von Kunden nicht in korrekter Weise angesprochen. Um Abhilfe zu schaffen, könnte man die Datei mit der Adreßliste entsprechend ergänzen, indem man beispielsweise der ersten Zeichenkette jeder Eintragung Herr oder Frau oder Frl. hinzufügt. Nach eingehendem Studium sollte eigentlich jeder Leser in der Lage sein, das vorliegende Programm so abzuändern, daß es diesen Gedankengängen Rechnung trägt.

Das in der Abb. 7.9 dargestellte Programm wurde im Zusammenhang mit einem bestimmten Brief (siehe Abb. 7.8) benutzt. Deshalb sollten wir noch einmal daran erinnern, daß unser Programm zur Erzeugung beliebiger Briefrahmen dienen kann. In den Aufgaben der Aufgabengruppe 24 wollen wir deshalb zu einigen Modifikationen dieses Programmes anregen. Unsere Vorschläge können sicher Veranlassungen geben, das Schreiben von Rechnungen oder irgendeine andere Korrespondenz mit unterschiedlichen Texten durch Programm abzudecken.

Aufgabengruppe 25

1. Die Briefrahmen, wie er im Abschnitt 7.5 vorgesehen war, ist so zu erweitern, daß auch der Text für eine zweite Briefseite erfaßt werden kann. Das Drucken von Briefen unter Zugrundelegung des auf diese Weise ergänzten Briefrahmens soll so erfolgen, daß auf der zweiten Breifseite zusätzlich die Seitennummer, noch einmal das Datum und ein Titel erscheint. Als Titel ist
   Geschäftsumzug
   vorzusehen. Der Text auf der zweiten Seite soll den folgenden Wortlaut aufweisen:

   ```
   Dieser Brief wird nur an unsere  treuesten Kunden versen-
   det! Bei Vorlage dieses Kupons gewähren wir einen Nachlaß
   von 10 % auf jeden Kaufabschluß,  der von Ihnen von heute
   ab bis zum  Ende des Monats September 1984 getätigt wird.
   ```

2. Der Briefrahmen von Abb. 7.8 ist so abzuändern, daß Einschlüsse in den Brieftext mit aufgenommen werden, um die Schreiben an die Kunden freundlicher zu gestalten. Folgende Erweiterungen sind vorzusehen:

Bisheriger Text	*Neuer Text*
`... sein; Sie werden ...`	`... sein; Sie, Herr ___, werden ...`
`... erwarten Sie ...`	`... erwarten Sie, Herr ___, ...`

   Anmerkung: Ergänzend zur Anrede „Herr" sollten auch die Anreden „Frau" und „Frl." vorgesehen werden.

3. Es ist ein Programm für die Rechnungsschreibung zu entwickeln. Dabei sind einige Vorgaben zu beachten. Zunächst ist davon auszugehen, daß alle Rechnungen komplett in einer Datei gespeichert sind, die RECHNUNG heißt. Jede einzelne Rechnung enthält für jeden ausgelieferten bzw. versendeten Artikel die folgenden Angaben: Menge, Einzelpreis, Artikelbezeichnung (begrenzt auf höchstens 15 Zeichen) und Gesamtpreis. Alle Rechnungseintragungen beginnen außerdem mit vier Zeichenketten, die der Reihe nach die nachstehenden Daten enthalten: Kundenname, Wohnadresse (Straße und Hausnummer), Wohnort und Rechnungsdatum. Jede Rechnungseintragung endet mit dem Prozentzeichen.

Die erste Eintragung in der Datei weist jedoch einen Aufbau auf, der sich vom Aufbau der übrigen Eintragungen unterscheidet; sie enthält nämlich nur die Anzahl der in die Datei aufgenommenen Rechnungen. Das zu schreibende Programm soll für alle Eintragungen, die in der Datei RECHNUNG enthalten sind, Rechnungen ausdrucken.

4. Angenommen, die Datei RECHNUNG (siehe Aufgabe 3.) enthält nur eine Kundennummer anstelle des Kundennamens und der Kundenadresse. Neben der Datei RECHNUNG existiere eine zweite Datei, in der, nach Kundennummern geordnet, Kundennamen und Kundenadressen verzeichnet sind. Zu dieser Kundendatei kann direkt zugegriffen werden. Das Programm von Aufgabe 3. ist nun so zu modifizieren, daß es sich aus der Kundendatei automatisch Kundenname und Kundenadresse in Abhängigkeit von der in der Rechnungsdatei enthaltenen Kundennummer herausholt und diese Angaben in die ausgedruckten Rechnungen übernimmt.

5. Bei einem Besuch in den USA passierte folgendes: Durch eine Änderung lokaler Verordnungen brauchte die örtliche Steuer nicht mehr den Kunden in Rechnung gestellt zu werden, die außerhalb des betreffenden Ortes (ZIP CODE: 91273) wohnen.
   Das Programm zum Schreiben von Briefen (siehe Abb. 7.9) ist deshalb so abzuändern, daß es den oben geschilderten Tatsachen gerecht wird. Es wird also einmal angenommen, daß das fragliche Programm für eine Firma des betreffenden USA-Ortes zu erstellen ist. In das Programm von Abb. 7.9 ist deshalb eine Prüfung auf den ZIP CODE aufzunehmen. Für diejenigen Kunden, deren Wohnort nicht den ZIP CODE 91723 aufweist, ist der Brief um den folgenden Passus zu erweitern:

```
Eine gute Nachricht für Sie! Ihnen brauchen wir zukünftig
nicht mehr die Ortssteuer in Rechnung zu stellen (örtli-
che Verordnung vom 20. 9. 1984). Unsere bekannt niedrigen
Preise werden dadurch für Sie noch geringer; unsere Ein-
sparungen geben wir voll an unsere Kunden weiter. Ein gu-
ter Grund, uns aufzusuchen!
```

Anmerkungen: Der ZIP CODE tritt an die Stelle der Postleitzahl. Als Briefunterschrift ist die einer fiktiven USA-Firma zu wählen.

8 Einführung in die Computergraphik

Bei vielen Anwendungen erweist es sich als vorteilhaft, wenn die Daten graphisch dargestellt werden können. Die graphische Darstellung ermöglicht oft erst die Gewinnung von Einsichten und Schlußfolgerungen auf Zusammenhänge, die aus den ursprünglichen Datensammlungen nur sehr schwer erkannt werden können.

In seiner Grundausrüstung kann mit dem IBM Personalcomputer Elementargraphik betrieben werden; diese wird Zeilengraphik genannt. Ihr charakteristisches Merkmal besteht darin, daß sich die graphischen Darstellungen aus den graphischen Zeichen zusammensetzen, die im Zeichenvorrat des Personalcomputers enthalten sind. Die beiden Abschnitte 1 und 2 dieses Kapitels befassen sich mit dieser Zeilengraphik.

Die graphischen Darstellungsmöglichkeiten des Personalcomputers werden durch seine Ausrüstung mit der Schnittstelle für Farben bzw. für Graphik erheblich gesteigert. Bei dieser Schnittstelle handelt es sich um eine der von der IBM zur Verfügung gestellten zusätzlich anschließbaren Erweiterungen. Mit dieser Erweiterung hat man die Wahl zwischen zwei graphischen Modi:

- *graphischer Modus mit mittlerem Auflösungsvermögen*
- *graphischer Modus mit hohem Auflösungsvermögen*

In beiden graphischen Modi verfügt BASIC über auf hoher Stufe stehende graphische Statements zum Zeichnen von Linien und Rechtecken, sowohl in Schwarzweiß, als auch in Farbe. Darüber hinaus gibt es zusätzlich noch eine umfassendere BASIC-Version, die *„erweitertes BASIC"* heißt. Diese Version enthält Statements zum Zeichnen von Kreisen, von Kreissektoren und zur Farbgebung von Flächen auf dem Bildschirm. In diese Statements des erweiterten BASIC wollen wir die Leser in den Abschnitten 8.5, 8.6 und 8.7 einführen.

Das erweiterte BASIC besitzt ferner Statements zum Spielen von Musik und zum Hervorrufen von akustischen Geräuschen. Diese wollen wir im Abschnitt 8.8 kennenlernen.

8.1 Zeilengraphik

Der Inhalt dieses Abschnittes gilt für alle IBM Personalcomputer.

Wie wir früher bereits festgestellt haben, reicht der Videoschirm (Bildschirm) des IBM Personalcomputers aus, um 25 Zeilen mit entweder 40 oder 80 Stellen anzuzeigen. Das ergibt maximal 25 mal 80 gleich 2000 Anzeigestellen auf dem Bildschirm. Durch diese einzelnen Zeichenstellen ist der Bildschirm in kleinste Rechtecke unterteilt. Die Abb. 8.1 zeigt diese Unterteilung des Bildschirmes unter Zugrundelegung von 80-stelligen Zeilen.

Die Rechtecke, in die wir den Schirm unterteilt haben, sind nach Zeilen und Spalten ausgerichtet. Die Zeilen werden von 1 bis einschließlich 25 numeriert, die Zeile 1 ist dabei die oberste, die Zeile 25 die unterste Zeile auf dem Bildschirm. Die Spalten werden von 1 bis 80 gezählt, die Spalte 1 befindet sich am weitesten links, die Spalte 80 am weitesten rechts auf dem Bildschirm. Jedes Rechteck auf dem Schirm kann damit durch ein Zahlenpaar, d. h. durch Koordinaten, gekennzeichnet werden. Das Zahlenpaar identifiziert Zeile und Spalte des Rechtecks. In der Abb. 8.2 haben wir beispielsweise das Rechteck (12,16) hervorgehoben, d. h. das Rechteck in der 12. Zeile auf der 16. Spalte.

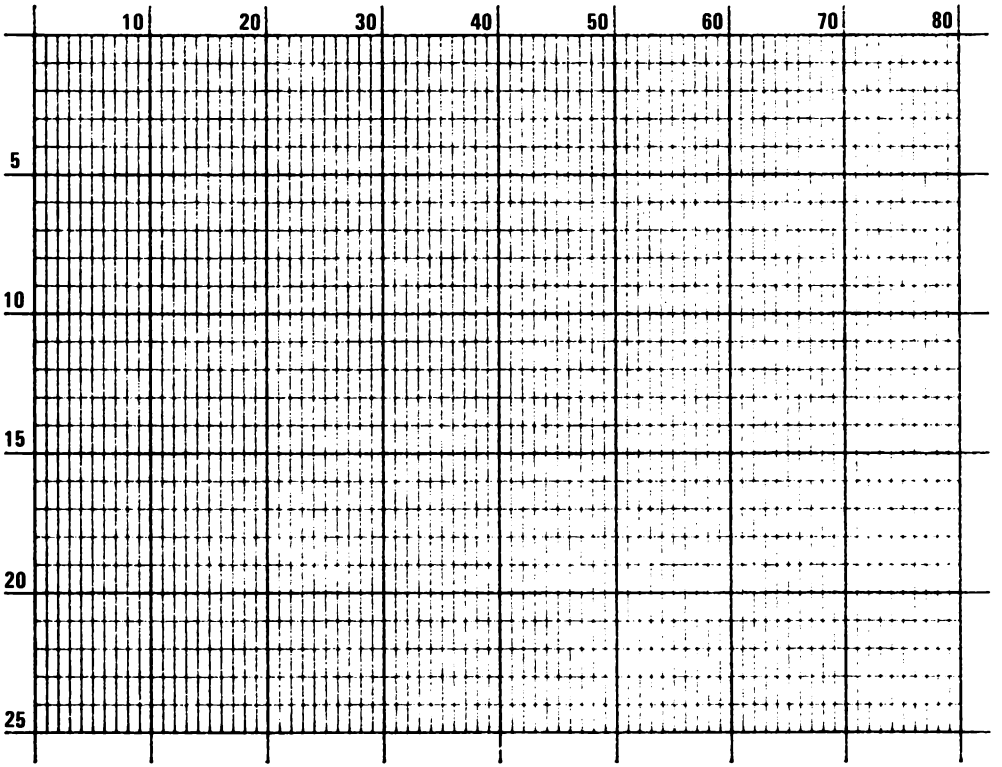


Abb. 8.1 Bildschirmauslegung im Textmodus bei einer Zeilenbreite von 80 Stellen (Spalten)

Wir können druckbare Zeichen auf den Bildschirm ausgeben, indem wir das Statement PRINT (mit oder ohne USING-Angabe) benutzen. Dieses Statement haben wir ausführlich im Abschnitt 4.3 behandelt. Für graphische Zwecke ist es wichtig, in der Lage zu sein, die Zeichen präzise auf dem Bildschirm zu positionieren. Durch das bereits bekannte LOCATE-Statement (siehe Abschnitte 3.4 und 7.3) können wir das auch erreichen. Erinnern wir uns, daß jede Ausgabe stets bei der augenblicklichen Cursorstellung beginnt. Um den Cursor auf die Spalte *y* der Zeile *x* zu führen, benutzen wir bekanntlich die Anweisung

```
100  LOCATE x,y
```

Beispiel 1:

Man schreibe eine Anweisungsfolge nieder, durch die der Text

```
IBM Personalcomputer
```

auf dem Bildschirm ausgegeben wird, beginnend auf der 11. Spalte der 20. Zeile.

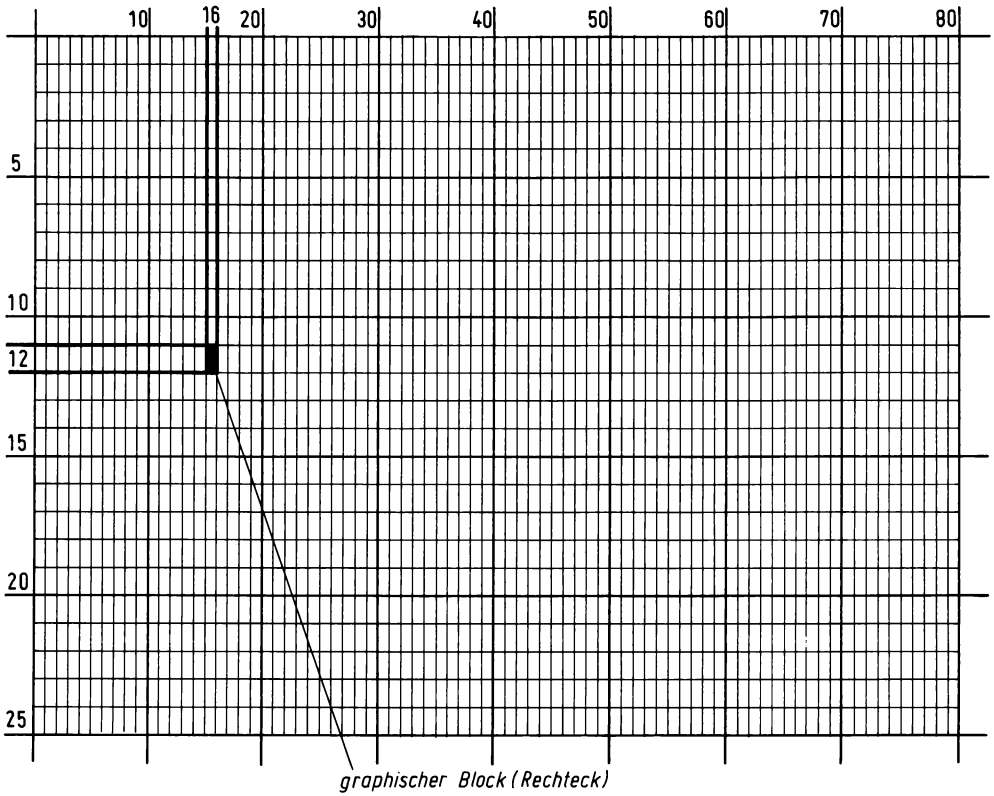


Abb. 8.2 Koordinatengitter auf dem Bildschirm

Die Lösung (Abb. 8.3) ist in ähnlicher Form von uns schon mehrere Male praktiziert worden, so daß sie keiner Kommentierung mehr bedarf.

```
:
10  LOCATE 20,11
20  PRINT "IBM Personalcomputer"
:
```

Abb. 8.3 Ausgabe eines Textes auf dem Bildschirm, beginnend an einer vorgegebenen Stelle

Bisher haben wir nur solche Zeichen ausgeben lassen, die auf der Tastatur des Computers zu finden sind, d.h. Buchstaben, Ziffern und gewisse Sonderzeichen (Satzzeichen, Rechenoperatoren usw.). In der Tat weist der IBM Personalcomputer jedoch einen erheblich umfangreicheren Zeichenvorrat auf; in diesen ist eine Anzahl graphischer Zeichen eingeschlossen (Abb. 8.4). Auch jedem graphischen Zeichen ist eine Dezimalzahl zugeordnet, der ASCII-

Code[1] des betreffenden Zeichens. Der Vertikalstrich besitzt z.B. den ASCII-Code 179. Um dieses graphische Zeichen auf die gegenwärtig vom Cursor eingenommene Position zu stellen, gebrauchen wir die Anweisung

```
30  PRINT CHR$(179)
```

Bei der Funktion CHR$ handelt es sich um die gleiche Funktion, die wir bereits im Abschnitt 7.1 kennengelernt haben.

ASCII-Wert	Zeichen	ASCII-Wert	Zeichen	ASCII-Wert	Zeichen	ASCII-Wert	Zeichen
128	Ç	166	ª	204	╠	242	≥
129	ü	167	º	205	═	243	≤
130	é	168	¿	206	╬	244	⌠
131	â	169	⌐	207	╧	245	⌡
132	ä	170	¬	208	╨	246	÷
133	à	171	½	209	╤	247	≈
134	å	172	¼	210	╥	248	°
135	ç	173	¡	211	╙	249	∙
136	ê	174	«	212	╘	250	·
137	ë	175	»	213	╒	251	√
138	è	176		214	╓	252	ⁿ
139	ï	177	░	215	╫	253	²
140	î	178	▒	216	╪	254	■
141	ì	179	│	217	┘	255	[2]
142	Ä	180	┤	218	┌		
143	Å	181	╡	219	[1]		
144	É	182	╢	220	▄		
145	Œ	183	╖	221	▌		
146	Æ	184	╕	222	▐		
147	ô	185	╣	223	▀		
148	ö	186	║	224	α		
149	ò	187	╗	225	β		
150	û	188	╝	226	Γ		
151	ù	189	╜	227	π		
152	ÿ	190	╛	228	Σ		
153	Ö	191	┐	229	σ		
154	Ü	192	└	230	µ		
155	¢	193	┴	231	τ		
156	£	194	┬	232	Φ		
157	¥	195	├	233	Θ		
158	Pts	196	─	234	Ω		
159	ƒ	197	┼	235	δ		
160	á	198	╞	236	∞		
161	í	199	╟	237	Ø		
162	ó	200	╚	238	∈		
163	ú	201	╔	239	∩		
164	ñ	202	╩	240	≡		
165	Ñ	203	╦	241	±		

Anmerkungen: [1] Dieses Zeichen soll „graphischer Block" genannt werden
[2] Leerzeichen (sedezimale Verschlüsselung 'FF')

Abb. 8.4 Der Zeichenvorrat des IBM Personalcomputers: Graphische Zeichen und Sonderzeichen

[1] ASCII ist die Abkürzung von „American standard code for information interchange" (dtsch.: Amerikanischer Standard-Code für Informationsaustausch)

Testübungen

8.1.1 Es sind die zwei Anweisungen niederzuschreiben, durch die das graphische Zeichen mit dem ASCII-Code 179 (Vertikalstrich) auf dem Bildschirm in der Zeile 18 auf der 22. Stelle ausgegeben wird.

8.1.2 Es ist ein Programm zu schreiben, durch das alle graphischen Zeichen, d.h. die Zeichen mit den ASCII-Codes 176 bis 223 auf dem Bildschirm angezeigt werden. Zwischen zwei aufeinanderfolgenden graphischen Zeichen soll dabei jeweils eine Stelle frei bleiben.

Wir können die graphischen Zeichen dazu benutzen, um auf dem Bildschirm die verschiedensten Bilder bzw. Figuren aufzubauen. Das nächste Beispiel befaßt sich damit.

Beispiel 2:

Es ist ein Programm zu schreiben, durch das auf dem Bildschirm eine waagerechte Linie (Horizontale) in Zeile 10 gezeichnet wird. Vorausgesetzt ist ein Bildschirm mit 80stelligen Zeilen.

Die Lösung kann relativ leicht entwickelt werden. Für den Fall, daß auf dem Bildschirm einige Zeichen angezeigt werden, die nicht zu dieser Problematik in Beziehung stehen, beginnen wir unser Programm mit der Anweisung CLS; diese Anweisung löscht bekanntlich den Bildschirm. Anschließend geben wir das Zeichen mit dem ASCII-Code 196 aus (Horizontalstrich), und zwar auf jeder Stelle der Zeile 10. Den Cursor stellen wir zuvor auf den Anfang dieser Zeile ein. Das unter diesen Überlegungen entstandene Programm ist in der Abb. 8.5 aufgelistet.

```
10  CLS
20  LOCATE 10,1
30  FOR  J=1 TO 80
40    PRINT CHR$(196);
50  NEXT J
60  END
```

Abb. 8.5 Programm zur Darstellung einer waagerechten Linie

Man beachte das Semikolon im PRINT-Statement. Dadurch erreichen wir, daß die Horizontalstriche ohne Lücken aufeinanderfolgend ausgegeben werden.

```
10  CLS:      KEY OFF
20  FOR K=1 TO 50:  'Die Laufvariable K steuert das Blinken
30    FOR J=5 TO 15
40      LOCATE J,25
50      PRINT CHR$(179);
60    NEXT J
70    CLS
80  NEXT K
90  END
```

Abb. 8.6 Programm zur Erzeugung einer blinkenden vertikalen geraden Linie

Beispiel 3:

Es ist ein Programm zu entwickeln, das eine Vertikallinie in der Spalte 25, beginnend auf der Zeile 5 und endend auf der Zeile 15, zeichnet. Die Gerade soll dabei 50mal blinken.

Das Blinken kann durch wiederholtes Löschen des Bildschirmes erreicht werden. Im übrigen liegen dem Programm (Abb. 8.6) die gleichen Aspekte zugrunde, die wir uns beim Programm vom Beispiel 2 erarbeitet haben.

Testübung 8.1.3

Es ist ein Programm zu schreiben, durch das auf dem Bildschirm eine vertikale Linie in Spalte 8 angezeigt wird, die bei Zeile 2 beginnt und bis zur Zeile 20 einschließlich reicht.

Beispiel 4:

Durch ein Programm ist auf dem Bildschirm das rechtwinklige kartesische Koordinatensystem zu zeichnen, das in der Abb. 8.7 dargestellt ist.

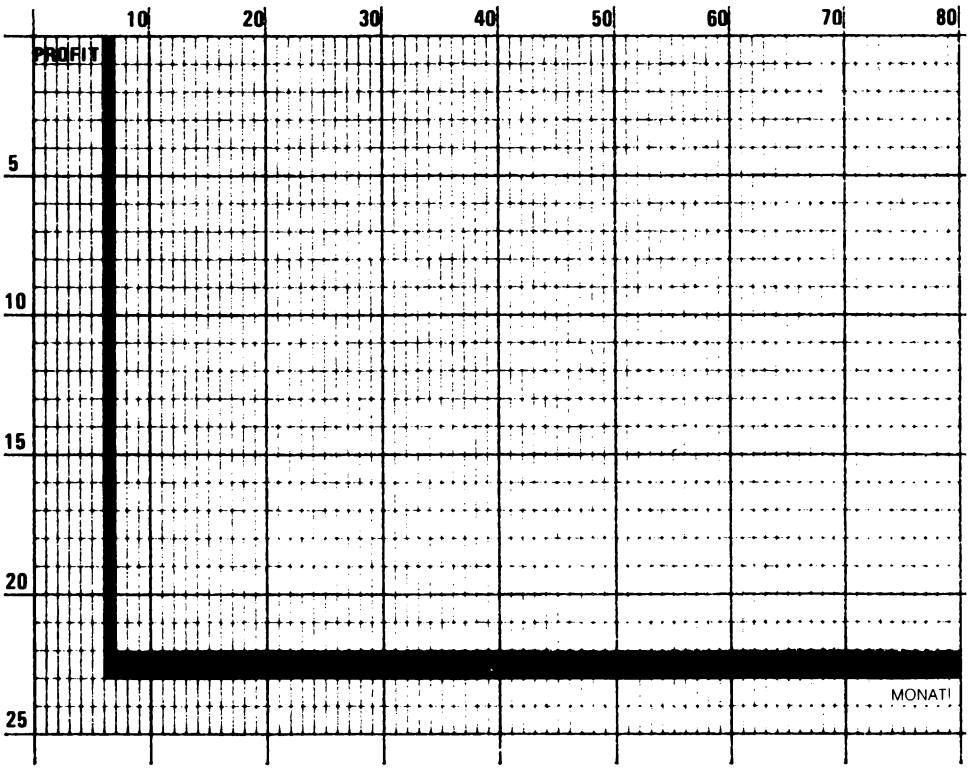


Abb. 8.7 Zu zeichnendes Koordinatensystem (Entwurf)

Die vertikale Achse (y-Achse) ist mit „Profit", die horizontale Achse mit „Monat" zu bezeichnen. Auch diese Aufgabe soll auf einen Bildschirm mit 80-stelligen Zeilen ausgelegt werden.

Für die Lösung stellen wir einige Vorüberlegungen an. Wir haben hier ein Programm zu entwickeln, durch das zwei Geraden gezeichnet und außerdem zwei Wörter ausgegeben werden: Probleme also, die wir im Einzelfall schon mehrere Male gelöst haben. Die einzige neue Schwierigkeit besteht nur darin, daß wir eine ordnungsmäßige Positionierung der Wörter vornehmen müssen. Das Wort „Profit" besitzt sechs Buchstaben. Deshalb wollen wir die Vertikallinie erst in der 7. Spalte zeichnen lassen; sie soll bei der 1. Zeile beginnen und bis zur drittletzten Zeile einschließlich reichen. Auf der vorletzten Zeile wollen wir das Wort „Monate" anzeigen lassen. Auf eine Ausgabe in der letzten Zeile wollen wir verzichten, da dadurch einiges von dem, was bisher ausgegeben wurde, durch eine Bildbewegung vom Bildschirm verschwinden würde. Das Programm zur Zeichnung des gewünschten Koordinatensystems ist in Abb. 8.8 aufgelistet.

```
10  CLS
20  LOCATE 1,1
30  PRINT "Profit"
40  LOCATE 23,76
50  PRINT "Monat"
60  FOR J=1 TO 22
70    LOCATE J,7
80    PRINT CHR$(179);
90  NEXT J
100 LOCATE 22,7
110 PRINT CHR$(192);
120 FOR J=8 TO 80
130   LOCATE 22,J
140   PRINT CHR$(196);
150 NEXT J
160 GOTO 160
170 END
```

Abb. 8.8 Programm zur Zeichnung eines Koordinatensystems (siehe Abb. 8.7)

Wir weisen insbesondere auf die unbegrenzt ablaufende Schleife im Programm hin, die durch das Statement auf der Zeile mit der Zeilennummer 160 verursacht wird. Diese Schleife hat zur Folge, daß die Bildschirmanzeige auf unbegrenzte Zeit, während der der Computer im Kreise läuft, erhalten bleibt. Um den Programmablauf zu stoppen, müssen die beiden Tasten *Ctrl* und *Break* gleichzeitig niedergedrückt werden. Um den Grund für die Aufnahme der endlosen Schleife ins Programm zu erkennen, versuche man das Programm nach Entfernen der mit 160 numerierten Zeile ablaufen zu lassen. Man wird dann feststellen, daß die BASIC-Rückmeldung *Ok* mit der graphischen Darstellung vermischt ist. Die endlose Schleife bewahrt uns also davor, daß die BASIC-Rückmeldung auf dem Bildschirm erscheint.

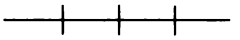
Aufgabengruppe 26

Durch Programme sind die folgenden Geraden auf dem Bildschirm anzuzeigen:

1. Waagerechte Linie über die ganze Zeile 18 hinweg
2. Senkrechte Linie über die gesamte Spalte 17 hinweg
3. Ein Paar von Geraden, das den Bildschirm in vier gleichgroße Quadrate unterteilt
4. Horizontale und vertikale Linien, die den Bildschirm in einen „Tic-Tac-Toe"-Spieltisch umwandeln
5. Vertikale Linie doppelter Stärke in Spalte 30, beginnend auf Zeile 1 und endend auf Zeile 24

6. Diagonale, die durch die Bildschirmpositionen (1,1), (2,2), ···, (24,24) verläuft

7. Eine horizontale Linie mit Teilstrichen, die das folgende Aussehen besitzt:



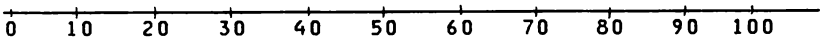
Hinweis: Der ASCII-Code für das graphische Zeichen „Teilstrich" ist in der Tabelle von Abb. 8.4 selbst nachzuschlagen.

8. Eine durch Teilstriche unterteilte vertikale Linie, die wie folgt aussieht:

9. Durch ein Programm ist auf dem Bildschirm in einem aus Sternzeichen geformten Kästchen der eigene Name anzuzeigen, beispielsweise also:

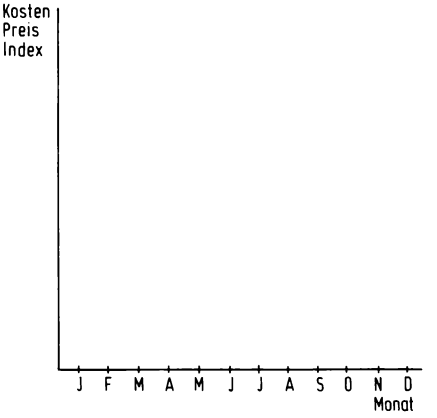
```
*****************
* Felix Schmitt *
*****************
```

10. Durch ein Programm ist die nachfolgende Zahlengerade anzuzeigen:



11. Es ist ein Programm zu schreiben, durch das ein graphisches Zeichen auf dem Bildschirm angezeigt wird! Das graphische Zeichen ist durch ein INPUT-Statement einzulesen!

12. Es ist ein Schaubild der folgenden Form auf dem Bildschirm zu erzeugen:



Antworten auf die Testübungen

8.1.1 Folgende beiden Statements lösen das Problem

```
10 LOCATE 18,22
20 PRINT CHR$(179)
```

8.1.2 Das Programm besitzt folgende Gestalt:

```
10 FOR J=176 TO 223
20   PRINT CHR$(J);" ";
30 NEXT J
40 END
```

8.1.3 Das Programm besitzt folgende Gestalt:

```
10 CLS
20 FOR J=2 TO 20
30   LOCATE J,8
40   PRINT CHR$(179);
50 NEXT J
60 END
```

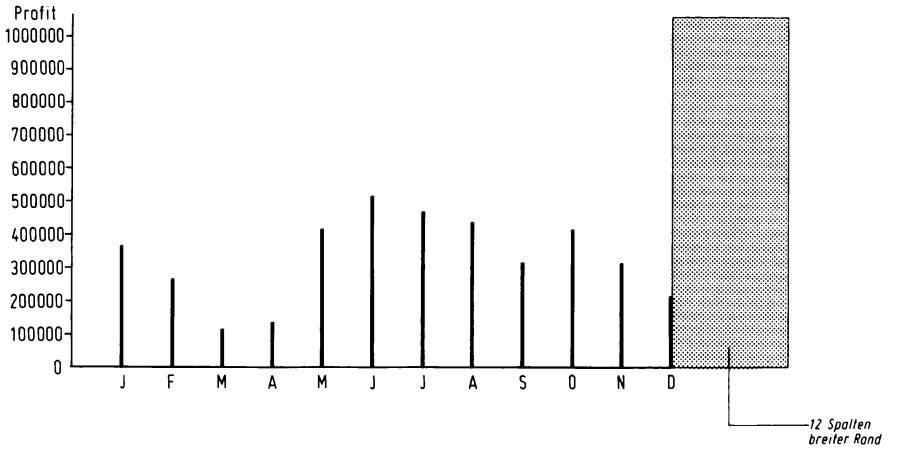
8.2 Darstellung von Balkendiagrammen

Dieser Abschnitt gilt für alle IBM Personalcomputer.

Man betrachte das in der Abb. 8.9 dargestellte Balkendiagramm, das die monatlichen Profite eines Geschäftes versinnbildlicht. Die Profite werden hierbei durch vertikale Balken dargestellt, deren Höhe durch den Betrag des Profits im betreffenden Monat bestimmt ist. Ein solches Diagramm wird Balkendiagramm genannt. Es ist allgemein üblich, Balkendiagramme bei Geschäftsberichten einzusetzen; damit können Trends in den verschiedensten Statistiken anschaulicher und übersichtlicher illustriert werden. In diesem Abschnitt wollen wir zeigen, wie man den IBM Personalcomputer zum Aufbau von Balkendiagrammen aus gegebenen Daten einsetzen kann.

Im vorhergehenden Abschnitt 8.1 erläuterten wir, wie wir horizontale und vertikale Achsen konstruieren können; diese Kenntnisse können wir für die Bildung von Balkendiagrammen gut gebrauchen. Wir wollen jetzt an diese wichtige Aufgabe herangehen. Um bei einem spezifischen Beispiel zu bleiben, wollen wir einmal das Diagramm entwerfen und zeichnen, das in Abb. 8.9 vorliegt. Bei der nachfolgenden Betrachtung wollen wir vorerst die meisten Berechnungen noch manuell erledigen. In den Aufgaben jedoch werden wir danach trachten, daß der Computer diese weitgehend selbst durchführt.

Das in der Abb. 8.9 dargestellte Balkendiagramm weist entsprechend der 12 Monate im Jahr 12 Balken oder Säulen auf. Für jeden Balken wollen wir vorerst gerade eine Spalte vorsehen. Da auf dem Bildschirm 80 Spalten zur Verfügung stehen, kommt es nun darauf an, diese den Erfordernissen des Problems gemäß zweckentsprechend zuzuordnen. Die ersten 8 Spalten wollen wir für das Wort „Profit", für die Vertikalachse und für die Teilstriche verwenden. Die rechten 12 Spalten wollen wir als Rand freilassen. Die verbleibenden 60 Spalten stehen für die



8.9 Balkendiagramm für die monatlichen Profite

Balken zur Verfügung. Diese Einteilung ist sicher als gut anzusehen, denn 60 ist ein Vielfaches von 12. Jeden Balken setzen wir daher sinnvollerweise an den rechten Rand eines 5 Spalten breiten Feldes. Da das erste, für den Monat Januar vorgesehene Feld die Spalten 9 bis 13 umfaßt, ordnen wir dem Balken für den Monat Januar die Spalte 13 zu. Der nächste Balken kommt in die Spalte 18, der übernächste in Spalte 23 usw.

Wenden wir uns nun der vertikalen Aufteilung des Bildschirmes zu. Die horizontale Achse plazieren wir am besten in die Zeile 22, damit bleibt Platz für die Monatsvermerke. Die Balken können unmittelbar darüber, also in Zeile 21, beginnen. Das Diagramm in der Abb. 8.9 verweist auf Profite zwischen 100000,– DM und 1 000000,– DM; jedenfalls ist die vertikale Achse mit entsprechenden Teilstrichen versehen. Es gibt 10 Teilstriche auf der vertikalen Achse. Somit können wir bei 20 Zeilen einen Teilstrich nach jeweils zwei Zeilen setzen, d.h. auf folgenden Zeilen

20	entspricht	100000,– DM
18	entspricht	200000,– DM
16	entspricht	300000,– DM
⋮		
2	entspricht	1000000,– DM

Damit haben wir den Bildschirmentwurf abgeschlossen.

Die Ergebnisse unserer Überlegungen haben wir auf einem Blatt, das zum Entwerfen von Bildschirmanzeigen bestimmt ist, festgehalten (siehe Abb. 8.10).

Testübung 8.2.1

Angenommen, ein Balkendiagramm besteht aus nur 8 Balken, während alle anderen Vorgaben (Achsenlage, Rand) die gleichen wie bei Abb. 8.9 sind. In welcher Spalte würden wir dann den ersten Balken unterbringen?

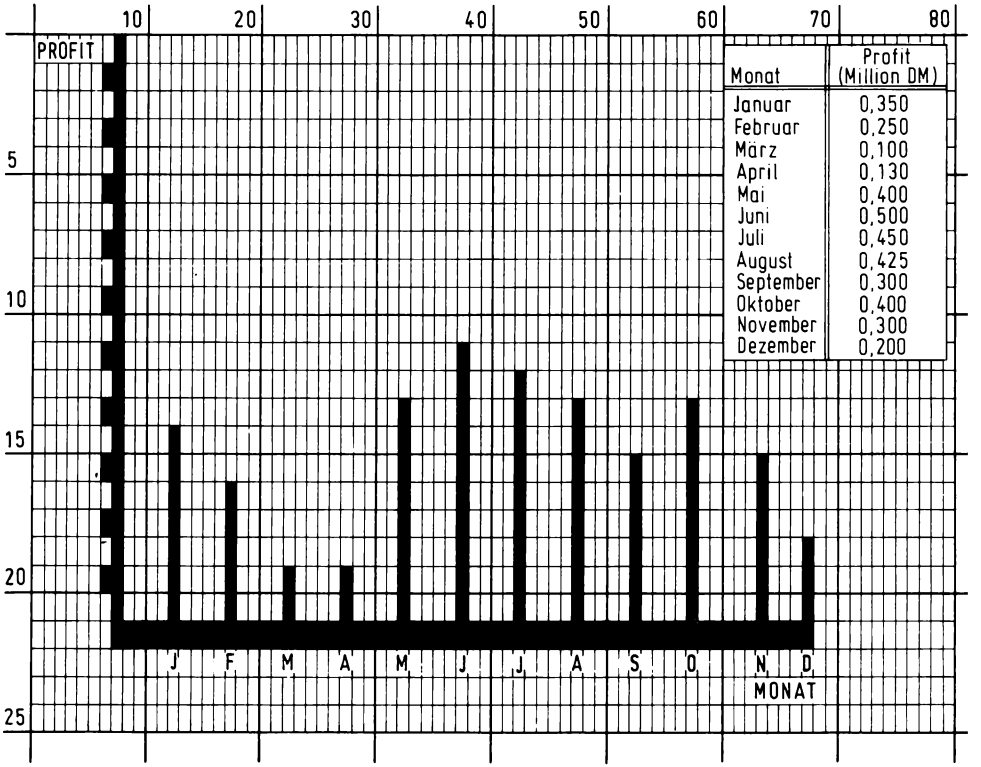


Abb. 8.10 Entwurfsblatt zum Entwerfen von Bildschirmanzeigen

Wir wollen uns nun der Planung und der Niederschrift des Programmes zuwenden, das uns dieses Balkendiagramm erzeugt. Für unser Programm werden wir zweckmäßigerweise drei Teile vorsehen

- Zeichnen der Achsen
- Anzeige des Textes
- Zeichnen der Balken

Zuerst wollen wir unser Augenmerk auf das Zeichnen der Balken lenken.

Um den Profit von J Hunderttausend DM anzuzeigen, muß ein Balken gezeichnet werden, der von der Zeile 22, d. h. von der Horizontalachse bis zur Zeile 22–2*J reicht. Zur Erläuterung wollen wir ein Beispiel heranziehen: Ein Profit von 200000,– DM entspricht J = 2 und somit einem Balken von Zeile 22 bis zur Zeile 22–2*2 = 18. In Abb. 8.11 haben wir ein Programm aufgelistet, das für den Monat Januar, in dem ein Profit von 350000,– DM erzielt wurde, den entsprechenden Balken zeichnet.

Die monatlichen Profite werden dem Programm durch DATA-Statements zugänglich gemacht. Der erste Programmteil liest die monatlichen Profite und speichert sie in den Elementen des Bereiches B(K) mit K = 1,2,3,· · ·,12. Danach zeichnet das Programm einen Bal-

```
100   P = 350000
110   J = 350000 / 100000
120   FOR  K = 22  TO  22-2*J  STEP -1
130     LOCATE K,13
140     PRINT CHR$(219);
150   NEXT K
9999  END
```

Abb. 8.11 Programm zur Zeichnung eines Balkens (Januar)

ken für jeden Monat; der entsprechende Programmteil ähnelt dem Programm der Abb. 8.11, von dem wir ausgegangen waren. Als einzig neuer Gesichtspunkt ist zu beachten, daß wir dieses Programm verallgemeinern müssen. Wir lassen also die Balken für die Monate M mit $M=1,2,3,\cdots,12$ in den Spalten $13+(M-1)*5$ zeichnen. Diese Lagebestimmung ist korrekt, denn für $M=1$ ergibt sich die Spalte 13, für $M=2$ die Spalte 18 usw. Ein Anwachsen von M um 1 läßt die Spaltenzahl um 5 wachsen, wie es auch beabsichtigt ist. Der dritte Programmteil ergibt somit die in der Abb. 8.12 dargestellte Statementfolge.

```
10    DIM  A(12)
20    DATA  350000, 250000, 100000, 130000, 400000, 500000
30    DATA  450000, 425000, 300000, 400000, 300000, 200000
40    FOR M=1 TO 12
50      READ A(M)
60    NEXT M
1000  FOR M=1 TO 12
1010    J=A(M)/100000
1020    FOR K=22 TO 22-2*J STEP -1
1030      LOCATE K,13+(M-1)*5
1040      PRINT CHR$(219); :  'Balkenzeichnung (Zeilen 1030 und 1040)
1050    NEXT K
1060  NEXT M
```

Abb. 8.12 Programmteil zum Zeichnen der Balken für die Monate Januar bis Dezember

Für den Einbau in das Gesamtprogramm werden wir das Einlesen der Profite vom 3. Programmteil abtrennen; dadurch wird die Programmpflege (Änderungsdienst) erheblich erleichtert. Diesem Aspekt haben wir bereits bei der Zeilennumerierung Rechnung getragen.

Zur Realisierung der beiden noch fehlenden Programmteile erinnern wir uns zunächst daran, daß wir die ersten acht Spalten für folgende Zwecke reserviert haben:

- Wort „Profit" (Spalten 1 bis 6)
- Teilstriche für die Unterteilung der Vertikalachse (Spalte 7)
- Vertikalachse (Spalte 8)

Diesen Entwurf wollen wir auch bei der Programmierung berücksichtigen. Das Wort „Profit" lassen wir also in der 1. Spalte der 1. Zeile beginnen, die Vertikalachse wird in der 8. Spalte gezeichnet und die Teilstriche plazieren wir in der 7. Spalte in den Zeilen $2,4,6,\cdots,20$. Die Horizontalachse wird in der Zeile 22 dargestellt, beginnend bei der Spalte 8 und endend bei der Spalte 68. Zum Schluß kommen die Buchstaben unter die Balken. Bezugnehmend auf die Abb. 8.10 stellen wir also die Monatsabkürzungen in die Spalten $13,18,23,\cdots,68$ der Zeile 23. Aus diesen Überlegungen heraus kristallisiert sich der in der Abb. 8.13 aufgelistete Programmteil.

```
100  DIM B$(12)
110  DATA  J, F, M, A, M, J, J, A, S, O, N, D
120  DATA  N, D
130  FOR J=1 TO 12
140    READ B$(J)
150  NEXT J
160  FOR J=1 TO 12
170    LOCATE  23,13+(J-1)*5
180    PRINT  B$(J); :        'Ausgabe der Monatsabkürzungen
190  NEXT J
200  LOCATE  1,1
210  PRINT  "Profit"
220  FOR J=1 TO 22 STEP 2
230    LOCATE  J,8
240    PRINT  CHR$(179); :    'Vertikalachse (Zeilen 230 und 240)
250  NEXT J
260  FOR J=2 TO 20 STEP 2
270    LOCATE  J,7
280    PRINT  CHR$(196); :    'Teilstriche (Zeilen 270 und 280)
290  NEXT J
300  FOR J=8 TO 68
310    LOCATE  22,J
320    PRINT  CHR$(196); :    'Horizontalachse (Zeilen 310 und 320)
330  NEXT J
340  LOCATE  22,8
350  PRINT  CHR$(192); :      'Koordinatenursprung (Zeilen 340 und 350)
```

Abb. 8.13 Programmteil zum Zeichnen der Achsen und zur Textanzeige

Nach der Entwicklung und Niederschrift der Programmteile können wir das Gesamtprogramm zusammenstellen. Zu den Teilen, die in der Abb. 8.12 und 8.13 aufgelistet sind, kommen noch die folgenden Statements hinzu:

- Löschen des Bildschirmes (zu Programmanfang)
- Zeichnen von Teilstrichen auf der Horizontalachse (zu Programmende)
- Generierung einer endlosen Schleife (zu Programmende) zum Festhalten des Bildes auf dem Bildschirm

Das Gesamtprogramm ist in der Abb. 8.14 aufgelistet

Natürlich ist es möglich, das in Abb. 8.14 aufgelistete Programm zu verfeinern. Insbesondere kann man dem Computer einen größeren Arbeitsanteil überlassen, indem wir z. B. alle Rechengänge von ihm ausführen lassen. In den Aufgaben der Aufgabengruppe 26 wollen wir zu einigen Verfeinerungen und Verbesserungen anregen.

Aufgabengruppe 27

Die Aufgaben 1 bis 6 beziehen sich auf das Programm zur Darstellung von Balkendiagrammen, das wir im Abschnitt 8.2 hergeleitet haben (siehe Abb. 8.14).

1. Das Programm ist einzugeben. Anschließend soll es durch Eingabe des Befehles RUN ausgeführt werden.

```
1     KEY OFF
5     CLS
10    DIM  A(12)
20    DATA  350000, 250000, 100000, 130000, 400000, 500000
30    DATA  450000, 425000, 300000, 400000, 300000, 200000
40    FOR M=1 TO 12
50      READ A(M)
60    NEXT M
100   DIM B$(12)
110   DATA  J, F, M, A, M, J, J, A, S, O, N, D
120   DATA  N, D
130   FOR J=1 TO 12
140     READ B$(J)
150   NEXT J
160   FOR J=1 TO 12
170     LOCATE  23,13+(J-1)*5
180     PRINT  B$(J); :        'Ausgabe der Monatsabkürzungen
190   NEXT J
200   LOCATE  1,1
210   PRINT  "Profit"
220   FOR J=1 TO 22 STEP 2
230     LOCATE  J,8
240     PRINT  CHR$(179); :    'Vertikalachse (Zeilen 230 und 240)
250   NEXT J
260   FOR J=2 TO 20 STEP 2
270     LOCATE  J,7
280     PRINT  CHR$(196); :    'Teilstriche (Zeilen 270 und 280)
290   NEXT J
300   FOR J=8 TO 68
310     LOCATE  22,J
320     PRINT  CHR$(196); :    'Horizontalachse (Zeilen 310 und 320)
330   NEXT J
340   LOCATE  22,8
350   PRINT  CHR$(192); :      'Koordinatenursprung (Zeilen 340 und 350)
1000  FOR M=1 TO 12
1010    J=A(M)/100000
1020    FOR K=22 TO 22-2*J STEP -1
1030      LOCATE K,13+(M-1)*5
1040      PRINT CHR$(219); :   'Balkenzeichnung (Zeilen 1030 und 1040)
1050    NEXT K
1052    LOCATE  22,13+(M-1)*5
1054    PRINT  CHR$(223); :    'Teilstriche (Zeilen 1052 und 1054)
1060  NEXT M
1100  GOTO 1100 :              'Endlose (unbegrenzte) Schleife
1200  END
```

Abb. 8.14 Programm zur Zeichnung eines Balkendiagrammes

2. Das Programm ist abzuändern. Anstelle der DATA-Statements zur Festlegung der Daten soll das Programm durch Rückfragen den Benutzer auffordern, die monatlichen Profite einzugeben. Die DATA- und READ-Statements sind also durch INPUT-Statements zu ersetzen.

3. Das zur Lösung der Aufgabe 2. entwickelte Programm ist zur Darstellung eines Balkendiagramms zu verwenden. Dabei sind die folgenden Daten zu berücksichtigen

Januar	175 238 DM	Juli	312 964 DM
Februar	35 275 DM	August	345 782 DM
März	240 367 DM	September	126 763 DM
April	675 980 DM	Oktober	324 509 DM
Mai	390 612 DM	November	561 420 DM
Juni	609 876 DM	Dezember	798 154 DM

4. Das zur Lösung der Aufgabe 2. entwickelte Programm ist wie folgt zu ergänzen:
   a) Links neben der Vertikalachse ist unterhalb des Wortes „Profit" auf der nächsten Zeile der Text „Mio DM" auszugeben.
   b) Links neben die Teilstriche auf der Vertikalachse sind die Maßangaben auszugeben, beginnend mit 0.1 und endend bei 0.9, jeweils um 0.1 anwachsend.

5. Das zur Lösung der Aufgabe 2. entwickelte Programm ist wie folgt abzuändern:
   Durch Rückfragen beim Benutzer soll das Programm ermitteln, welche Maßangaben links neben die Vertikalachse gesetzt werden sollen. Diese Maßnahme kann als erster Schritt zur Entwicklung eines allgemeinen Programmes zur Darstellung von Balkendiagrammen für beliebige Daten angesehen werden. Man sollte vorerst einmal noch von einer Limitierung ausgehen und deshalb nur Maßangaben für jede 2. Zeile vorsehen. Die Maßangaben sollten außerdem nicht mehr als sechs Zeichen umfassen, so daß die Lage der vertikalen Achse beibehalten werden kann. Bei Berücksichtigung dieser Voraussetzungen können die in das Programm aufzunehmenden Rückfragen zur Eingabe der folgenden Parameter auffordern:

   – Maßangabe, die an den ersten Teilstrich zu setzen ist,
   – Schrittweite zwischen den Maßangaben zweier aufeinanderfolgender Teilstriche.

   Durch ein Beispiel wollen wir das ganze näher erläutern. Angenommen die Teilstriche sollen mit

   0.3 , 0.7 , 1.1 , , 3.1

   bezeichnet werden. In diesem Fall wäre 0.3 als erste Teilstrichmarkierung und 0.4 als Schrittweite einzugeben; ins Programm müßten also entsprechende INPUT-Statements eingebaut werden. Die auf die erste Maßangabe (Beispiel: 0.3) folgenden Maßangaben müssen natürlich im Programm errechnet werden.

   Darüber hinaus könnte das Programm so ergänzt werden, daß es einen sogenannten „Skalenfaktor" berücksichtigt. Darunter versteht man, auf das ursprüngliche Programm bezogen, die Länge, der auf der Vertikalachse der Profitbetrag 100000,- DM entspricht. Genauer gesagt, stellt der Skalenfaktor die Länge dar, die auf der Vertikalachse dem Intervall (der Schrittweite) zwischen zwei aufeinanderfolgenden Maßangaben entspricht. Daraus ergibt sich: Multipliziert man ein Datenelement mit dem Skalenfaktor, so erhält man die Länge, d.h. die Höhe, gemessen in Intervallen, die dem Balken entspricht, der das Datenelement veranschaulicht.

6. Man erweitere das Programm zur Lösung der Aufgabe 5. derart, daß es auch nach der Bezeichnung der Teilstriche, d.h. der Balken, auf der Horizontalachse fragt, d.h. hier nach den Monaten. Das Programm soll in der Lage sein, nach einer variablen Anzahl von Balken zu fragen; maximal sollen jedoch nur 12 Balken erlaubt sein. Bei weniger als 12 Datenelementen sollen die Balken und Teilstrichbezeichnungen am rechten Ende des Diagramms weggelassen werden.

7. Man benutze das zur Lösung der Aufgabe 6. entwickelte Programm zur Darstellung eines Balkendiagramms für die folgenden Daten.

Einkommen	Prozentsatz der Bevölkerung
unter 25000 DM	15.8
25000 bis 50000 DM	25.7
50000 bis 75000 DM	27.4
75000 bis 100000 DM	11.1
über 100000 DM	20.0

Antwort auf die Testübung 8.2.1

Spalte 15

8.3 Computerkunst

Dieser Abschnitt gilt für sämtliche IBM Personalcomputer.

Mit dem Computer können recht interessante graphische Kunstwerke geschaffen werden. Mehrfach wurde schon eine Auswahl derselben auf Ausstellungen gezeigt.

Wir wollen über zwei Formen von Computerkunst sprechen. Der ersten Form begegnen wir, wenn wir dem Computer gewissermaßen künstlerische Freiheit zur Darstellung zufälliger Muster auf dem Bildschirm gewähren. Zur Demonstration dieser Form der Computerkunst wollen wir ein Programm verfassen, das über alle Stellen der einzelnen Zeilen hinwegläuft und an einzelnen, noch zu ermittelnden Stellen graphische Blöcke aufleuchten läßt; an welchen Stellen dabei die Blöcke aufleuchten, soll auf Zufallsbasis bestimmt werden. Das Verhältnis der aufleuchtenden Blöcke zu den nicht aufleuchtenden geben wir durch einen sogenannten Dichtefaktor vor; dieser soll zu Programmanfang eingegeben werden. Der Dichtefaktor ist als reelle Zahl zwischen 0 und 1 einzugeben; im Programm wird er der numerischen Variablen D zugewiesen. Unser Programm spricht jede einzelne Stelle an, die einen graphischen Block aufnehmen kann. Die Entscheidung darüber, ob ein graphischer Block aufleuchten soll oder nicht, wird vom ausgegebenen Wert des Zufallszahlengenerators RND abhängig gemacht. Wenn der von RND erzeugte Wert kleiner als der D zugewiesene Dichtefaktor ist, soll an der betreffenden Stelle der graphische Block aufleuchten, anderenfalls nicht. Das unter diesen Gedankengängen entwickelte Programm ist in der Abb. 8.15 aufgelistet.

Man sollte dieses wahrlich bescheidene Programm für mehrere verschiedene Werte von D ablaufen lassen, beispielsweise für

a) D = 0.1
b) D = 0.5
c) D = 0.7

Weil wir den Wert, den der Zufallszahlengenerator abliefert, nicht voraussagen können, wird der gleiche Wert, den wir für D in aufeinanderfolgenden Programmabläufen eingeben, im

```
100  INPUT "DICHTEFAKTOR"; D
110  RANDOMIZE
120  CLS
130  FOR  Z = 1  TO  23 :            'Zeilennummer
140    FOR  S = 1  TO  79 :          'Spaltennummer
150      IF  RND < D  THEN 160  ELSE 170
160      LOCATE  Z,S :      PRINT CHR$(219);
170    NEXT S
180  NEXT Z
200  GOTO 200                        'Endlose (unbegrenzte) Schleife
300  END
```

Abb. 8.15 Programm zur Schaffung graphischer Kunstwerke

allgemeinen zu gänzlich verschiedenen Bildern auf dem Bildschirm führen. Für einen mit D = 0.25 durchgeführten Programmablauf haben wir ein erzeugtes Bild in der Abb. 8.16 festgehalten.

Eine zweite Art und Weise, wie man Computerkunst erzeugen kann, besteht darin, daß man eine graphische Schablone benutzt und mit ihrer Hilfe eine Bildvorlage absteckt, gewissermaßen durchpaust. Um diese Methode näher zu erläutern, wollen wir einmal annehmen, daß das auf dem Bildschirm zu erzeugende Bild die Kopie einer Fotografie sein soll. Über die Fotografie lege man einfach ein Entwurfsblatt für Videoanzeigen (siehe Abb. 8.1) und fülle auf diesem alle Kästchen aus, die die Fotografie betreffen. Hierdurch kann man einen Abzug (Abdruck) des fotografierten Gegenstandes gewinnen, der als Basis für die Anzeige auf dem Bildschirm dienen kann.

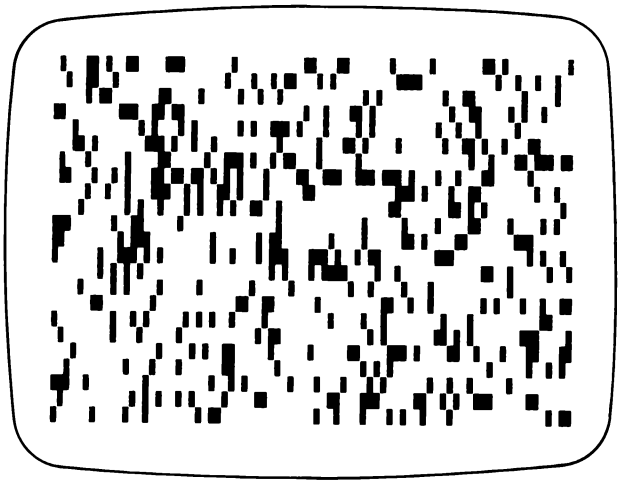


Abb. 8.16 Ein durch das Programm von Abb. 8.15 erzeugtes Bild auf der Basis von D = 0.25

Bevor wir unsere sehr knappe Besprechung abschließen, sollten wir noch erwähnen, daß in jüngster Zeit einige diesbezügliche Geräte auf den Markt gekommen sind, die auch preislich den Vorstellungen entsprechen, die diejenigen hegen, die den Computer als ihr Steckenpferd betrachten. In unseren vorhergehenden Ausführungen bezüglich der Gewinnung einer reproduktionsfähigen Vorlage verwiesen wir auf ein äußerst mühseliges Verfahren. Die eben er-

wähnten speziellen Geräte zur Digitalisierung von Bildvorlagen ermöglichen es dem Benutzer, eine Gestalt mit einem „elektronischen" Stift zu umranden, wodurch sie auf den Bildschirm übertragen wird. Zusätzlich gibt es dazu die sogenannten Lichtstifte, mit denen man Punkte auf dem Bildschirm berühren kann; die Berührung veranlaßt den Computer, die Lage dieses Punktes einzulesen. Diese Gerätetypen können bei der Schaffung von Computerkunstwerken recht nützlich sein. Man kann sie aber auch ebensogut für Computerspiele einsetzen.

Aufgabengruppe 28

1. Man lasse das Programm von Abb. 8.15 drei Mal für D = 0.4 ablaufen.
2. Man lasse das Programm von Abb. 8.15 nacheinander für die folgenden Werte von D ablaufen:
   0.1, 0.2, 0.3, ···, 0.9, 1.0
   Kann die Anzeige für D = 1.0 ohne Programmablauf vorausgesagt werden?
3. Man erzeuge eine Computerabbildung eines Familienmitgliedes; als Vorlage benutze man eine großformatige Fotografie.
   Anmerkung: Die Fotografie sollte mindestens das Format 13 × 18 cm aufweisen.

8.4 Betriebsarten für Farbe und Graphik

Dieser Abschnitt gilt nur für IBM Personalcomputer, die mit einer Schnittstelle (Interface) für Farbe/Graphik ausgerüstet sind.

Es gibt drei Anzeige-Betriebsarten (Modi) bei den IBM Personalcomputern:

a) *Textmodus*
b) *Graphischer Modus mittleren Auflösungsvermögens*
c) *Graphischer Modus hohen Auflösungsvermögens*

Den Textmodus (Text-Betriebsart) haben wir bis jetzt benutzt, um Zeichen, einschließlich der graphischen Zeichen, auf dem Bildschirm anzuzeigen. Er steht bei jeder Ausrüstung der IBM Personalcomputer zur Verfügung. Die beiden graphischen Modi erfordern jedoch die Ausrüstung des IBM Personalcomputers mit einer speziellen Schaltkreisplatine, die als Schnittstelle (Interface) für Farbe/Graphik bezeichnet wird. Der graphische Modus mittleren Auflösungsvermögens läßt zeichnerische Darstellungen in Farbe oder in Schwarz/Weiß zu. Der graphische Modus hohen Auflösungsvermögens gestattet sehr detaillierte Darstellungen, jedoch nur in Schwarz/Weiß. In diesem Abschnitt wollen wir uns hauptsächlich mit diesen zwei graphischen Modi beschäftigen; deshalb wollen wir ausdrücklich betonen, daß wir für diesen Abschnitt grundsätzlich voraussetzen, daß der benutzte Computer mit der Schnittstelle für Farbe/Graphik ausgerüstet ist.

Die Wahl zwischen den verschiedenen Anzeige-Betriebsarten kann mit Hilfe des Statements SCREEN erfolgen.

SCREEN 0	→	Textmodus
SCREEN 1	→	Graphischer Modus mittleren Auflösungsvermögens
SCREEN 2	→	Graphischer Modus hohen Auflösungsvermögens

Beim Starten von BASIC liegt automatisch der Textmodus vor. Die soeben aufgeführten Statements können zum Umschalten von einer Anzeige-Betriebsart zu einer anderen benutzt wer-

den, entweder innerhalb eines Programms als Anweisung oder als Befehl über die Tastatur. Man beachte jedoch, daß bei Verwendung von SCREEN automatisch der Bildschirm gelöscht wird.

8.4.1 Bildelemente und Farben beim graphischen Modus

Im graphischen Modus mittlerer Auflösung ist der Bildschirm in 320 Positionen von links nach rechts und in 200 Positionen von oben nach unten unterteilt. Um eine Vorstellung darüber zu verschaffen, welch feines Raster diese Unterteilung ergibt, betrachte man die Abb. 8.17; in dieser ist ein Raster mit 160 × 100 Linien dargestellt. (Das tatsächliche Bildschirmgitter ist viel zu fein, als daß man es drucken könnte. Würde man einen Druck versuchen, so würde nur eine beinahe vollständig schwarze Fläche herauskommen.)

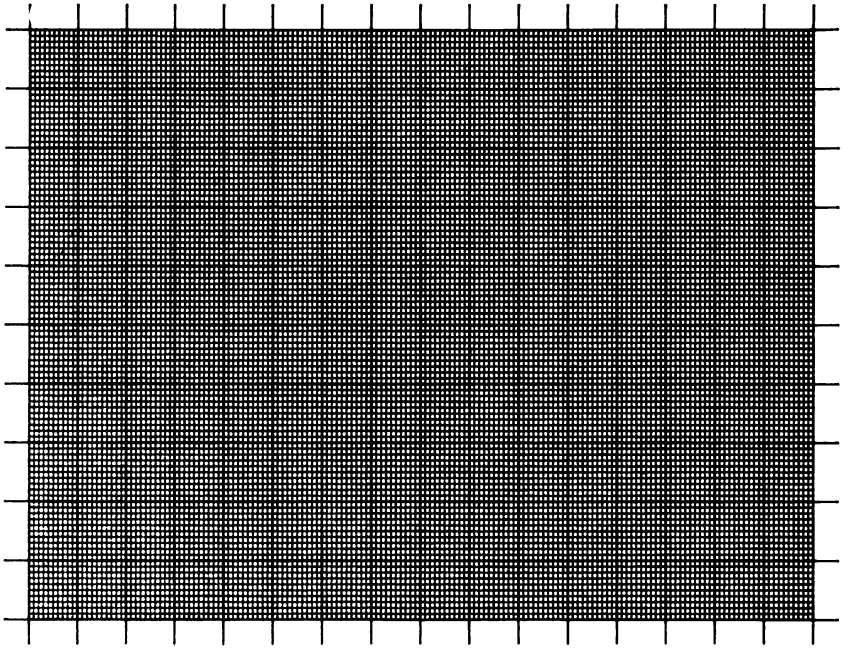


Abb. 8.17 Entwurfsblatt für Videoanzeigen für den graphischen Modus mittleren Auflösungsvermögens

Jedes dieser kleinen Rechtecke des Gitters wird Bildelement oder „Pixel"[1] genannt. Man kann jedes einzelne Pixel kontrollieren und steuern. Im graphischen Modus mittlerer Auflösung kann man ein Pixel in jeder der vier verfügbaren verschiedenen Grundfarben einfärben lassen.

[1] Pixel ist ein aus dem Englischen übernommenes Kunstwort, das durch Zusammenziehung der beiden Wörter „picture" und „element" entstanden ist, auf deutsch also „Bildelement".

Im graphischen Modus hoher Auflösung ist der Bildschirm von links nach rechts in 640 Positionen und von oben nach unten in 200 Positionen unterteilt. Auch hierbei kann jedes einzelne Pixel kontrolliert und gesteuert werden. Allerdings stehen nur zwei Farben zur Verfügung: ein Pixel ist entweder schwarz (gleichbedeutend mit OFF) oder weiß (gleichbedeutend mit ON).

a) *Graphische Koordinaten*
Die Lage jedes Pixels ist durch ein Koordinatenpaar (x, y) bestimmt. Hierbei bedeutet x die Spaltennummer und y die Zeilennummer. Auf die folgenden wichtigen Tatsachen sei verwiesen:
1. Die Zeilen- und Spaltennummern beginnen mit 0, nicht wie beim Textmodus mit 1. Beim graphischen Modus mittlerer Auflösung sind folglich die Zeilen von 0 bis 199 numeriert, die Spalten von 0 bis 319. Beim graphischen Modus hoher Auflösung lauten infolgedessen die entsprechenden Zahlen 0 bis 199 bzw. 0 bis 639.
2. Bei den graphischen Modi werden die Koordinaten in umgekehrter Reihenfolge wie beim Textmodus angegeben, d.h. die Spaltennummer kommt zuerst (x-Koordinate). Erinnern wir uns, daß beispielsweise beim LOCATE-Statement die Zeilennummer zuerst angegeben werden muß.

b) *Relative Koordinaten bei den graphischen Modi*
Bei den graphischen Anzeige-Betriebsarten ist der Positionsanzeiger (Cursor) nicht sichtbar. Stattdessen hält der Computer den letzten Punkt fest, auf den Bezug genommen worden ist. Darunter ist derjenige Punkt zu verstehen, dessen Koordinaten zuletzt in ein graphisches Statement eingegangen sind. Man kann die Position eines neuen Punktes dadurch festlegen, daß man relative Koordinaten vorgibt, d.h. Koordianten relativ zum zuletzt angesprochenen Punkt. Relativen Koordinaten muß immer das Schlüsselwort STEP vorausgehen. – Nehmen wir beispielsweise einmal an, daß der zuletzt angesprochene Punkt die Koordinaten (100,75) aufgewiesen hatte. Wenn man sich nun mit Hilfe von relativen Koordinaten durch

```
STEP (20,30)
```

auf einen neuen Punkt bezieht, so ist damit der Punkt mit den Koordinaten (120,105) gemeint, d.h. der Punkt, der gegenüber dem zuvor angesprochenen Punkt um 20 Einheiten nach rechts und um 30 Einheiten nach unten verschoben liegt. Man betrachte in gleicher Weise nunmehr den Punkt, der die relativen Koordinaten

```
STEP (-10,-40)
```

aufweist. Damit kommt man zu dem Punkt, der 10 Einheiten nach links und 40 Einheiten nach oben gegenüber dem zuletzt angesprochenen Punkt (120,105) versetzt ist, der also die Koordinaten (110,65) besitzt.

Testübung 8.4.1

Angenommen, der Punkt (50,80) wurde zuletzt angesprochen. Man bestimme die Koordinaten der Punkte, auf die man sich durch die folgenden relativen Koordinaten bezieht:

a) `STEP (50,50)` b) `STEP (-20,10)`
c) `STEP (10,-40)` d) `STEP (-20,-50)`

c) *Farbgebung im graphischen Modus mittleren Auflösungsvermögens*
Der graphische Modus mittleren Auflösungsvermögens erlaubt uns den Gebrauch von Farben auf dem Bildschirm. Der graphische Modus hohen Auflösungsvermögens gestattet jedoch, wie wir bereits wissen, nur die Verwendung von schwarz und weiß. Damit Anzeigen farblich gestaltet werden können, muß zunächst die Farbgebung durch das SCREEN-Statement aktiviert werden; das entsprechende Statement lautet:

```
SCREEN 1,0
```

Bedeutung: mittleres Auflösungsvermögen, Farbe eingeschaltet (ON)

Zur Ausschaltung von Farben bei Anzeigen dient das Statement:

```
SCREEN 1,1
```

Bedeutung: mittleres Auflösungsvermögen, Farbe ausgeschaltet (OFF)

Nach Aktivierung der Farbgebung kann man sowohl Hintergrund- als auch Vordergrundfarben festlegen, d.h. auswählen. Ein Pixel wird in einem bestimmten Augenblick als Teil des Hintergrundes betrachtet, es sei denn, daß seine Farbgebung explizit durch ein graphisches Statement neu festgesetzt wird. Wenn das Statement CLS ausgeführt wird, wird allen Pixels die Hintergrundfarbe zugewiesen. Man sagt, daß ein Pixel, das nicht die Hintergrundfarbe aufweist, zum Vordergrund gehört. Den einzelnen Farben ist jeweils eine Kennzahl zugewiesen.

Für die Hintergrundfarben gelten die folgenden 15 Kennzahlen:

0 → schwarz	*8 → grau*
1 → blau	*9 → hellblau*
2 → grün	*10 → hellgrün*
3 → kobaltblau	*11 → hellkobaltblau*
4 → rot	*12 → hellrot*
5 → violett	*13 → hellviolett*
6 → braun	*14 → gelb*
7 → weiß	*15 → leuchtend weiß*

Die Vordergrundfarben können aus einer von zwei Paletten ausgewählt werden.

Palette 0	*Palette 1*
1 → grün	*1 → kobaltblau*
2 → rot	*2 → violett*
3 → braun	*3 → weiß*

Hintergrund- und Vordergrundfarben werden durch das COLOR-Statement festgesetzt. Man kann die Hintergrundfarbe und die Palette mittels Statements auswählen, die wie das folgende Beispielstatement aufgebaut sind:

```
100 COLOR 12,0
```

Durch dieses Statement wird als Hintergrundfarbe hellrot (Farbkennzahl 12) und als Palette die Palette 0 festgesetzt. Diese Festlegungen gelten solange, bis sie mittels eines neuen COLOR-Statements geändert werden.

Testübung 8.4.2

Es sind die BASIC-Statements niederzuschreiben, durch die der graphische Modus mittleren Auflösungsvermögens festgelegt und als Hintergrundfarbe leuchtend weiß sowie die Palette 1 ausgewählt wird.

d) *Aufleuchten von Pixels*
Zum Aufleuchten von Pixels ist das PSET-Statement geschaffen worden. Die Ausführung der Anweisung

```
100 PSET (100,150),1
```

läßt beispielsweise das Pixel mit den Koordinaten (100,150) in der Farbe 1 der gegenwärtig festgelegten Palette aufleuchten. Um das Aufleuchten dieses Pixels wieder auszuschalten, muß man die Ausführung einer Anweisung der Form

```
200 PRSET (100,150)
```

veranlassen. Korrekt gesagt, veranlaßt die Ausführung der letzten Anweisung das Aufleuchten des Pixels (100,150) in der festgelegten Hintergrundfarbe, was dem Ausschalten der Vordergrundfarbe entspricht. Bei Benutzung der PSET- und PRSET-Statements kann man auch die Koordinaten der anzusprechenden Punkte relativ angeben. Das Statement

```
300 PSET STEP (100,150), 2
```

sorgt bei seiner Ausführung beispielsweise dafür, daß das Pixel, das 100 Einheiten rechts und 150 Einheiten nach unten vom zuletzt angesprochenen Pixel liegt, in der Farbe 2 der gegenwärtig gültigen Palette aufleuchtet.

8.4.2 Hinweise über Monitorgeräte

Wir sollten uns nunmehr Klarheit darüber verschaffen, über welche Fähigkeiten die unterschiedlichen Konfigurationen verfügen, die wir durch Kombination der verschiedenen anschließbaren Schnittstellen (Interfaces) und Monitore erhalten. Wir wollen versuchen, in einer kurzen gestrafften Übersicht einen Anhaltspunkt zu geben.

1. *Monochrome Anzeigeschnittstelle und monochromes Anzeigegerät der IBM*
   Mit dieser Konfiguration ist eine graphische Anzeige-Betriebsart nicht möglich, weder mit mittlerem noch mit hohem Auflösungsvermögen. Außerdem kommt die Beschränkung der Typengröße auf die „schmalen" Zeichen, d.h. große Zeichendichte, hinzu (80stellige Zeilen). Wenn man eine Zeilenlänge von 40 Stellen festlegt, ist die Anzeige auf die linke Hälfte des Bildschirmes beschränkt.

2. *Schnittstelle für Farbe und Schwarz/Weiß-Monitor*
   Die graphische Anzeige-Betriebsart ist sowohl mit mittlerem als auch mit hohem Auflösungsvermögen, jedoch stets nur in Schwarz/Weiß, durchführbar. Zusätzlich sind auch beide Typengrößen erlaubt.

Wenn man allerdings ein Programm ablaufen läßt, in das COLOR-Anweisungen aufgenommen sind, können die Grautöne, die den verschiedenen Farben entsprechen, zu unleserlichen Anzeigen führen. Bei dieser Konfiguration sollte man sich deshalb tunlichst an einem schwarzen Hintergrund und weißen Zeichen festhalten.

3. *Schnittstelle für Farbe und Fernsehgerät*
   Die Anzeige-Betriebsarten für Text und für Graphik mit mittlerem Auflösungsvermögen arbeiten vortrefflich. Der graphische Modus hohen Auflösungsvermögens ist jedoch nicht durchführbar.

4. *Schnittstelle für Farbe und Farbmonitor hohen Auflösungsvermögens*
   Bei dieser Ausrüstung sind alle Anzeige-Betriebsarten, also
   - Text
   - Graphik mit mittlerem Auflösungsvermögen
   - Graphik mit hohem Auflösungsvermögen

   durchführbar, die beiden zuerst genannten in Farbe. Es sollte jedoch unbedingt darauf hingewiesen werden, daß einige Programme für Computerspiele, die ausgeklügelt die Möglichkeiten ausnutzen, die Farben bieten, besser mit Fernsehgeräten arbeiten. Ihr Entwurf nutzt nämlich die technischen Gegebenheiten der Fernsehgeräte vorteilhaft aus.

Viele Besitzer von IBM Personalcomputern rüsten ihre Geräte sowohl mit der monochromen Schnittstelle als auch mit der Schnittstelle für Farbe/Graphik aus. Für diese Besitzer sei ausdrücklich darauf verwiesen, daß in einem bestimmten Augenblick nur eine dieser beiden Platinen angesprochen werden kann. Darüber hinaus geht BASIC am Anfang stets zur monochromen Anzeige über. Um zur Schnittstelle für Farbe/Graphik übergehen zu können, muß die monochrome Schnittstelle ausgeschaltet und die Schnittstelle für Farbe/Graphik eingeschaltet werden. Man kann das durch einen Programmabschnitt bewerkstelligen, den man zu Beginn des eigenen Programmes einfügt. Ein solcher Programmabschnitt ist in der Abb.8.18 aufgelistet.

```
10  WIDTH 80
20  DEF  SEG=0
30  A = PEEK(&H410)
40  B = (A AND &HCF) OR &H20
50  POKE &H410,B
60  WIDTH 40
70  SCREEN 0
80  LOCATE ,,1,6,7
```

Abb.8.18 Programmabschnitt für den Übergang von der monochromen Schnittstelle zur Schnittstelle für Farbe/Graphik

```
940  WIDTH 40
950  DEF  SEG=0
960  A = PEEK(&H410)
970  POKE &H410, A OR &H30
980  WIDTH 80
990  LOCATE ,,1,12,13
```

Abb.8.19 Programmabschnitt für den Übergang von der Schnittstelle für Farbe/Graphik zur monochromen Schnittstelle

Wenn man die Schnittstellen wieder wechseln will, um erfolgreich zum Betriebssystem DOS zurückkehren zu können, muß man zuerst die Schnittstelle für Farbe/Graphik ausschalten und dann die Schnittstelle für die monochrome Anzeige wieder einschalten. Für diesen Zweck ist es erforderlich, das eigene Programm durch einen Programmabschnitt abzuschließen, der die Gestalt aufweist, wie sie die Abb. 8.19 zeigt. Natürlich sind auch hier die Zeilennummern dem restlichen Programm sinnvoll anzupassen.

Aufgabengruppe 29

Es sind BASIC-Anweisungen niederzuschreiben, die die folgenden Operationen bewerkstelligen:

1. Als Hintergrundfarbe ist violett auszuwählen; die Vordergrundfarben sollen von der Palette 1 genommen werden.
2. Als Hintergrundfarbe ist hellrot auszuwählen; die Vordergrundfarben sollen von der Palette 0 genommen werden.
3. Das Pixel mit den Koordinaten (200,80) soll mit der Farbe 1 der gegenwärtig gültigen Palette aufleuchten.
4. Das Pixel mit den Koordinaten (100,100) soll rot auf der Hintergrundfarbe kobaltblau aufleuchten.
5. Das Pixel, das 200 Einheiten links und 100 Einheiten über dem zuletzt angesprochenen Pixel liegt, soll mit der Farbe 3 der augenblicklich geltenden Palette aufleuchten.
6. Das Pixel, das 100 Einheiten rechts vom zuletzt angesprochenen Pixel liegt, soll aufleuchten.

Antworten auf die Testübung 8.4.1

a) (100,130) b) (30,90) c) (60,40) d) (30,30)

Antwort auf die Testübung 8.4.2

```
10  SCREEN 1,0
20  COLOR 15,1
```

8.5 Gerade, Rechtecke und Kreise

Dieser Abschnitt besitzt nur für die IBM Personalcomputer Gültigkeit, die mit der Schnittstelle für Farbe/Graphik ausgerüstet sind.

In diesem Abschnitt wollen wir uns der Darstellung der einfachen geometrischen Grundfiguren zuwenden, nämlich den geraden Linien, den Rechtecken und den Kreisen.

8.5.1 Gerade Linien

Man kann zum Entwerfen graphischer Anzeigen ohne weiteres die beiden bereits bekannten Statements PSET und PRSET anwenden. BASIC weist jedoch ein reiches Repertoire von Statements auf, die Aufgaben dieser Art wesentlich vereinfachen. Einleitend zu dieser Problematik wollen wir die Aufgabe betrachten, gerade Linien auf dem Bildschirm darzustellen.

Wenn, wie schon in der Einleitung zu diesem Abschnitt gesagt, die Schnittstelle Farbe/Graphik vorhanden ist, kann man sich für diesen Zweck des Statements LINE bedienen. Sollen z.B. die Punkte (Pixels) mit den Koordinaten (20,50) und (80,199) durch eine Gerade verbunden werden, so kann man einfach das Statement

```
10  LINE (20,50)-(80,199)
```

codieren.

Um eine gerade Linie zeichnen zu lassen, die den zuletzt angesprochenen Punkt mit dem Punkt (100,90) verbindet, benutzt man die Anweisung

```
20  LINE -(100,90)
```

Ein weiteres Beispiel: Durch das Statement

```
30  LINE -STEP(80,-100)
```

wird auf dem Bildschirm eine gerade Linie gezeichnet, die den zuletzt angesprochenen Punkt mit dem Punkt verbindet, der 80 Einheiten rechts und 100 Einheiten darüber liegt.

Man kann darüber hinaus die Farbe für die darzustellende Gerade festlegen. Wenn z.B. die Gerade, die durch das Statement 10 gezeichnet wird, in der Farbe 1 der gegenwärtig gültigen Palette aufleuchten soll, so benutzt man einfach eine Erweiterung des o.a. Statements, nämlich

```
40  LINE (20,50)-(80,199),1
```

Die durch die Ausführung dieses Statements entstehende gerade Linie ist in der Abb.8.20 dargestellt.

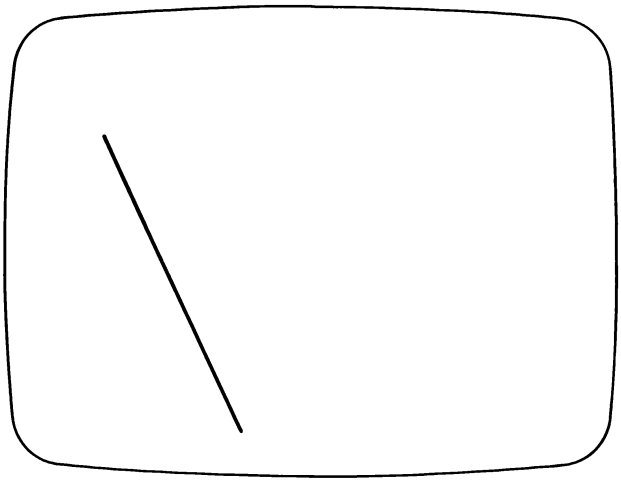


Abb.8.20 Gerade Linie zwischen den Punkten (20,50) und (80,199)

Wenn in der LINE-Anweisung keine Farbe spezifiziert ist, so erscheint die Gerade in der Farbe 3 der gegenwärtig geltenden Palette.

Es sei ausdrücklich darauf hingewiesen, daß der Computer keine perfekte gerade Linie darstellen kann. Schräg verlaufende gerade Linien bestehen aus einer Folge von sichtbaren „Stufen". Diese liegen so eng wie möglich an der exakten geraden Linie, die die beiden gegebenen Punkte verbindet. Die stufenförmige Annäherungslinie ist durch das begrenzte Auflösungsvermögen des ausgewählten graphischen Modus bedingt, d.h. beim hohen Auflösungsvermögen ist die Annäherung entsprechend feiner. Je höher also das Auflösungsvermögen ist, d.h. je größer die Anzahl der Bildelemente (Pixels) auf dem Bildschirm ist, um so besser kommt die Geradlinigkeit zur Geltung.

Testübung 8.5.1

a) Es ist die Zeichnung einer geraden Linie zu veranlassen, die die beiden Punkte (0,100) und (50,75) in der Farbe 2 der gegenwärtig geltenden Palette verbindet.

b) Es ist die Zeichnung eines Dreiecks mit den Eckpunkten (0,0), (50,50) und (100,30) zu veranlassen.

Die Version 2.00 des Betriebssystemes DOS gestattet es, darüber hinaus die „Linienart" zu spezifizieren. Man kann z.B. festlegen, daß bei einer Linie ständig fünf Leerzeichen auf fünf Punkte folgen sollen. Eine solche Gerade besitzt also folgendes Aussehen:

Auf diese Weise entsteht eine punktierte Linie mit Lücken zwischen den Punkten. Die Benutzung der Einrichtung zur Darstellung bestimmter Linienarten setzt gewisse Vorkenntnisse in der Dualarithmetik voraus; sie wird im Buch von *Larry Joel Goldstein*

ADVANCED BASIC AND BEYOND:
Techniques for the IBM PC

besprochen. Dieses Buch ist *1984* im *Verlag Robert J. Brady Co.* erschienen und kann als Fortsetzung des vorliegenden Buches angesehen werden.

8.5.2 Rechtecke

Das LINE-Statement besitzt mehrere sehr durchdachte Variationen. Um ein Rechteck zu zeichnen, braucht man nur die Koordinaten von zwei gegenüberliegenden Eckpunkten, d.h. von zwei Diagonalpunkten, im LINE-Statement anzugeben und den Code B am Ende des Statements hinzuzufügen; der Buchstabe B steht für „box" (Kasten). Wenn wir z.B. ein Rechteck zeichnen lassen wollen, von dem die Koordinaten zweier gegenüberliegender Eckpunkte zu (50,100) und (90,175) bekannt sind, gebrauchen wir die Anweisung

```
50  LINE (50,100)-(90,175),1,B
```

Die Ausführung dieses Statements bewirkt das Zeichnen des gewünschten Rechtecks in der Farbe 1 der augenblicklich aktivierten Farbpalette (siehe Abb. 8.21).

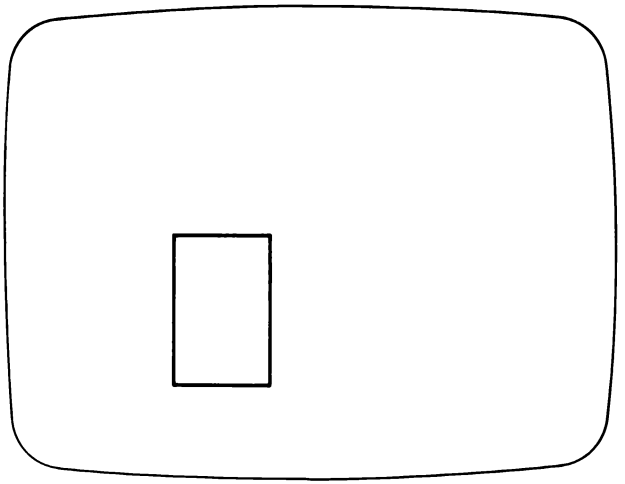


Abb. 8.21 Zeichnung eines Rechtecks mit der Option B

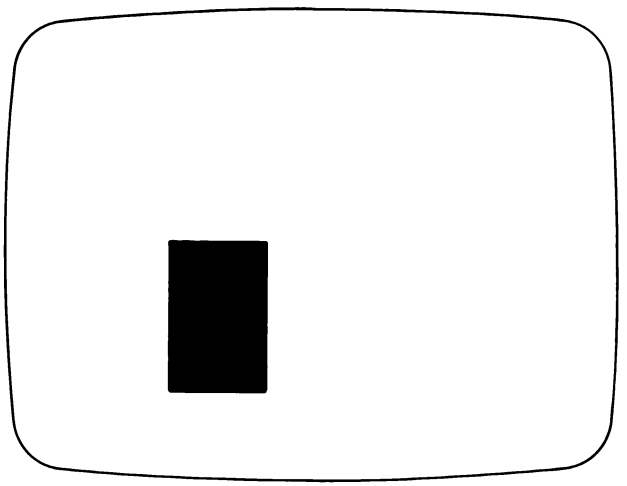


Abb. 8.22 Zeichnung eines „gefärbten" Rechtecks mit der Option BF

Das Innere des Rechtecks liegt dabei in der Hintergrundfarbe vor. Man kann aber auch veranlassen, daß das Innere in derselben Farbe wie die Seiten aufleuchtet. Man braucht hierzu nur den Code B durch den Code BF auszutauschen, also das o. a. Statement in der Form

```
60  LINE (50,100)-(90,175),1,BF
```

zu schreiben; der Buchstabe B steht hierbei nach wie vor für „box", der Buchstabe F für „filled", also etwa „gefüllter Kasten". Die durch das Statement auf der Zeile mit der Zeilennum-

mer 60 generierte Zeichnung finden wir in der Abb. 8.22; man muß sich natürlich das ganze in der entsprechenden Farbe vorstellen. Die eben besprochenen Formen der Anweisung LINE vereinfachen die Darstellung komplexer, geradlinig begrenzter Figuren beträchtlich.

Testübung 8.5.2

a) Es ist die Anweisung niederzuschreiben, durch deren Ausführung ein Rechteck gezeichnet wird, dessen Eckpunkte die folgenden Koordinaten besitzen:

```
(10,10), (10,100), (50,100), (50,10)
```

b) Es ist die Anweisung niederzuschreiben, durch deren Ausführung ein Rechteck gezeichnet wird, dessen Eckpunkte die folgenden Koordinaten besitzen:

```
(10,10), (10,100), (50,100), (50,10)
```

Die Seiten und das Innere des Rechtecks sollen dabei in der Farbe 2 der gegenwärtig gültigen Palette erscheinen.

8.5.3 Gemischte Anzeigen von Text und Graphiken

In die Zeichnungen können ohne weiteres auch Texte eingeschlossen werden. Man benutze hierzu wie üblich die PRINT-Anweisung ohne oder mit USING-Option. Man verhalte sich also genau so, als ob sich der Computer im Textmodus befindet. Man beachte dabei jedoch unbedingt die folgenden Hinweise:

1. Im graphischen Modus mit mittlerem Auflösungsvermögen kann man nur 40-stellige Zeilen verwenden. Das bedeutet, daß nur die „breiteren" Zeichen auf dem Bildschirm angezeigt werden. Im graphischen Modus hohen Auflösungsvermögens kann man hingegen nur 80-stellige Zeilen benutzen, d. h. alle Zeichen besitzen das „schmalere" Format.
2. Der Text erscheint in der Farbe 3 der gegenwärtig geltenden Palette.

Beim Planen der Ausgabe von Graphiken, die mit Texten unterlegt sind, sollte man daran denken, daß alle Zeichen unabhängig von der Zeilenlänge stets acht Pixels breit und acht Pixels hoch sind. Somit belegt z. B. das Zeichen in der linken oberen Ecke des Bildschirmes die Pixels (x,y), wobei sowohl x als auch y den Wertebereich 0 bis 7 durchlaufen.

Beispiel 1:

Man nenne den Befehl, durch den die Textzeile 1 auf dem Bildschirm bei vorliegendem graphischen Modus mittleren Auflösungsvermögens gelöscht wird.

Unser Vorgehen beim Löschen einer solchen Zeile besteht darin, daß wir ein die ganze Zeile belegendes Rechteck zeichnen lassen, wobei wir als Farbe die Hintergrundfarbe (Farbe 0) auswählen. Die erste Textzeile nimmt bekanntlich die Pixels (x,y) in Anspruch, deren x-Koordinate den Wertebereich 0 bis 319 und deren y-Koordinate den Wertebereich 0 bis 7 durchläuft. Erinnern wir uns, daß x der Spaltennummer und y der Zeilennummer entspricht. Das unter diesen Überlegungen zustande gekommene Statement besitzt infolgedessen das folgende Aussehen:

```
LINE (0,0)-(319,7),0,BF
```

8.5.4 Erweitertes BASIC

Alle bisher beschriebenen Statements stehen im Disketten-BASIC zur Verfügung. Zur Erinnerung: Hierbei handelt es sich um die BASIC-Version, die man erhält, wenn man auf die DOS-Systemanfrage mit „BASIC" antwortet. Es gibt jedoch eine umfangreichere Version von BASIC, die „Erweitertes BASIC" genannt wird. Diese enthält neben den vom Disketten-BASIC her bekannten Statements noch eine Reihe weiterer. Eine Kopie des erweiterten BASIC befindet sich auf der Diskette mit dem Betriebssystem DOS, die man in der Regel dann käuflich erwirbt, wenn man seinen Computer mit wenigstens einem Diskettenlaufwerk ausrüstet. Wenn man das erweiterte BASIC benutzen will, muß man zunächst eine DOS-Systemanfrage

```
A>
```

abwarten bzw. erhalten haben. In Reaktion auf diese muß nunmehr

```
BASICA
```

eingetippt und die Eingabetaste gedrückt werden. Daraufhin bekommt man vom DOS die obligatorische Rückmeldung

```
Ok
```

Von diesem Augenblick an kann man selbstverständlich weiterhin die BASIC-Anweisungen bzw. -Befehle benutzen, die man bisher kennengelernt hat. Zusätzlich können nunmehr außerdem noch die Anweisungen und Befehle verwendet werden, die nur für das erweiterte BASIC gelten. Dazu zählt u.a. das Statement CIRCLE, das wir jetzt behandeln wollen.

Testübung 8.5.3

Das erweiterte BASIC ist für die weiteren Tätigkeiten am Computer bereitzustellen, d.h. zu laden.

8.5.5 Kreise

Das erweiterte BASIC besitzt die Fähigkeit, Kreise und Kreisbögen zeichnen zu können. Wenn man einen Kreis zeichnen lassen will, muß man seinen Mittelpunkt und seinen Radius angeben und gegebenenfalls noch die Farbe, in der er erscheinen soll. Fangen wir mit einem Beispiel an und nennen einfach einmal den Befehl, durch den bei seiner Ausführung ein Kreis mit dem Radius von 50 Einheiten um den Mittelpunkt (100,100) gezeichnet wird.

```
CIRCLE (100,100),50
```

Da keine Farbe im Befehl spezifiziert wurde, wird der Kreis in der Farbe 3 der gegenwärtig geltenden Palette gezeichnet (siehe Abb. 8.23). Um den gleichen Kreis mit der Farbe 1 zu zeichnen, müssen wir den Befehl

```
CIRCLE (100,100),50, 1
```

eingeben.

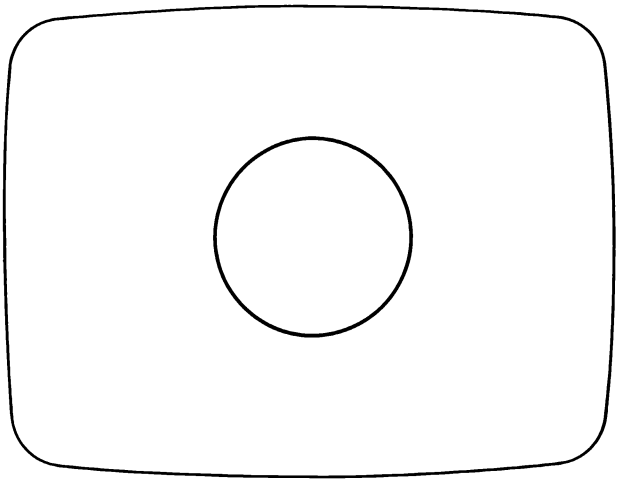


Abb. 8.23 Kreis um M(100,100) mit dem Radius 50

Die Kreise selbst erscheinen auf dem Bildschirm nicht glatt, sondern zackig und rauh, was auf das begrenzte Auflösungsvermögen zurückzuführen ist. Benutzt man hingegen die Anzeige-Betriebsart hohen Auflösungsvermögens, so wird man eine erheblich verbesserte Erscheinungsform der Kreise feststellen können.

Die Vorgaben für das Zeichnen von Kreisbögen sind komplizierter, denn sie basieren auf der Winkelmessung im Bogenmaß. Verweilen wir einen kurzen Augenblick bei der Beschreibung des Bogenmaßes. Frischen wir einmal unsere Schulkenntnisse auf und rufen uns zunächst die Bedeutung der Zahl π ins Gedächtnis. Unter π verstehen wir die reelle Zahl 3.1415926 . . .; der Näherungswert 3.14 reicht für viele Rechenzwecke aus. Gewöhnlich wird ein Winkel im Gradmaß gemessen, dem vollen Winkel kommen hierbei 360° zu. Das diesem Gradmaß entsprechende Bogenmaß beträgt 2π rad. Unter rad ist die Einheit des Bogenmaßes zu verstehen (rad kommt von Radiant). Wir haben also die folgenden Beziehungen zu verzeichnen:

$$2\pi \text{ rad} = 360°$$
$$1 \text{ rad} = 360/(2\pi)°$$

Setzt man in diese Beziehung für π einen Näherungswert ein und rechnet den Bruch auf der rechten Seite aus, so erhält man

$$1 \text{ rad} = 360/(2 \cdot 3.14159)°$$
$$1 \text{ rad} \approx 57°$$

Einer Einheit des Bogenmaßes entsprechen also näherungsweise etwa 57°. Winkel dem Computer vorzugeben, heißt sie prinzipiell im Bogenmaß anzugeben.

Damit auf dem Bildschirm ein Kreisbogen gezeichnet werden kann, müssen wir das CIRCLE-Statement in der folgenden Form benutzen:

```
CIRCLE (x,y),r,f,a,e
```

Hierbei bedeuten:

(x,y)	Koordinaten des Kreismittelpunktes
r	Radius des Kreises
f	Farbe der gegenwärtig geltenden Palette, in der der Kreisbogen dargestellt werden soll
a	Anfangswinkel des Kreisbogens, gemessen in rad (Winkel zwischen der Horizontalen und der Geraden vom Mittelpunkt zum Anfangspunkt des Kreisbogens)
e	Endwinkel des Kreisbogens, gemessen in rad (Winkel zwischen der Horizontalen und der Geraden vom Mittelpunkt zum Endpunkt des Kreisbogens)

Um einen Kreisbogen des zuvor behandelten Kreises mit einem Anfangswinkel von 0.1 rad und einem Endwinkel von 1.5 rad zu zeichnen, muß der entsprechende Befehl wie folgt lauten:

```
CIRCLE (100,100),50, 1, 0.1,1.5
```

Die bei der Ausführung dieses Befehles entstehende Figur ist in der Abb. 8.24 dargestellt.

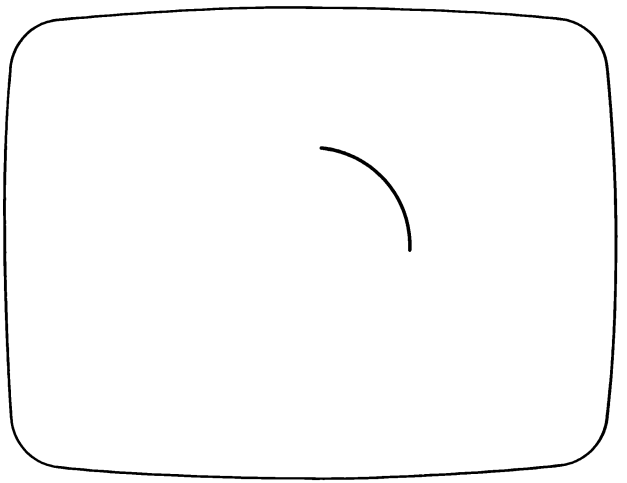


Abb. 8.24 Darstellung eines Kreisbogens

Die Abb. 8.24 stellt nur den Kreisbogen selbst dar. Um diesen zu einem Sektor zu ergänzen, d.h. es müssen noch die beiden Radien gezeichnet werden, ist jede Winkelangabe durch ein vorangestelltes Minuszeichen zu erweitern (soll nur ein Radius gezeichnet werden, so ist nur der zu diesem Radius gehörende Winkel zu kennzeichnen). Durch die Eingabe des Befehles

```
CIRCLE (100,100),50, 1, -0.1,-1.5
```

erhalten wir also die Darstellung des entsprechenden Kreissektors (siehe Abb. 8.25).

Wenn der Winkel 0 anzugeben ist und der zu diesem Winkel gehörende Radius gezeichnet werden soll, kann n i c h t −0 geschrieben werden. Um diesem Dilemma zu entgehen, ma-

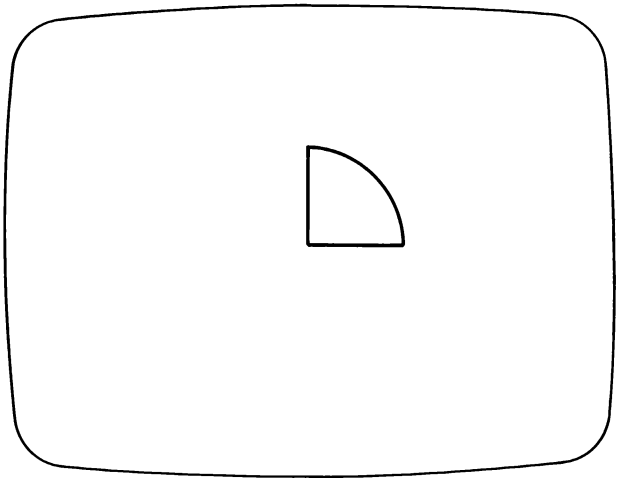


Abb. 8.25 Darstellung eines Kreissektors

chen wir von der Tatsache Gebrauch, daß der Winkel 0 rad dem Winkel 2π rad gleichkommt. Wir ersetzen also 0 durch 2π, d.h. durch 6.28318 ..., und können daher im Befehl auch -6.28318 niederschreiben.

Testübung 8.5.4

a) Es ist um den Mittelpunkt M(125,75) ein Kreisbogen mit dem Radius 60 zu zeichnen. Der Kreisbogen soll dabei einen Anfangswinkel von 0.25 rad und einen Endwinkel von 0.75 rad aufweisen.

b) Es ist der gleiche Kreisbogen wie bei a) zu zeichnen. Der Kreisbogen ist um die Radien zu seinem Anfangspunkt und zu seinem Endpunkt zu erweitern, so daß ein Kreissektor entsteht.

Aufgabengruppe 30

Es sind BASIC-Statements niederzuschreiben, durch die die Zeichnung der nachstehend beschriebenen Figuren bewirkt werden kann.

1. Verbindungsgerade zwischen den Punkten A(20,50) und B(40,100)
2. Gerade zwischen dem Punkt, auf dem augenblicklich der Cursor steht, und dem Punkt (250,150) in der Farbe 2 der gegenwärtig gültigen Palette
3. Gerade zwischen dem Punkt (125,50) und dem Punkt, der 100 Einheiten rechts davon und 75 Einheiten tiefer liegt, in der Farbe 1 der gegenwärtig gültigen Palette
4. Rechteck mit den Eckpunkten A(10,20), B(200,20), C(200,150) und D(10,150)
5. Rechteck wie bei Aufgabe 4., jedoch Seiten und Inneres in der Farbe 3 der gegenwärtig gültigen Palette
6. Kreis um den Mittelpunkt M(30,50) mit dem Radius $r = 20$

7. Kreisbogen des Kreises von Aufgabe 6. mit dem Anfangswinkel 1.5 rad und dem Endwinkel 3.1 rad

8. Kreissektor des Kreises von Aufgabe 6. mit dem Anfangswinkel 1.5 rad und dem Endwinkel 3.1 rad

Antworten auf die Testübungen

8.5.1 a)
```
10 LINE (0,100)-(50,75),2
```
b)
```
10 LINE (0,0)-(50,50)
20 LINE (50,50)-(100,30)
30 LINE (100,30)-(0,0)
```

8.5.2 a)
```
10 LINE (10,10)-(50,100),,B
```
b)
```
10 LINE (10,10)-(50,100),2,B
```

8.5.3 Ausgehend von der DOS-Systemanfrage ist BASICA einzugeben und die Eingabetaste zu drücken.

8.5.4 a)
```
10 CIRCLE (125,75),60,, 0.25,0.75
```
b)
```
10 CIRCLE (125,75),60,, -0.25,-0.75
```

8.6 Darstellung von Kreisdiagrammen

Voraussetzung für die Anwendung des in diesem Abschnitt behandelten Stoffes ist die Ausrüstung des IBM Personalcomputers mit der Schnittstelle für Farbe/Graphik.

Wir wollen uns die Aufgabe stellen, das Kreisdiagramm zu zeichnen, das in der Abb. 8.26 dargestellt ist. Wir finden, daß dieses Problem uns mit einer recht nützlichen Anwendung des Statements CIRCLE bekannt macht. – Das zu zeichnende Kreisdiagramm soll uns einen

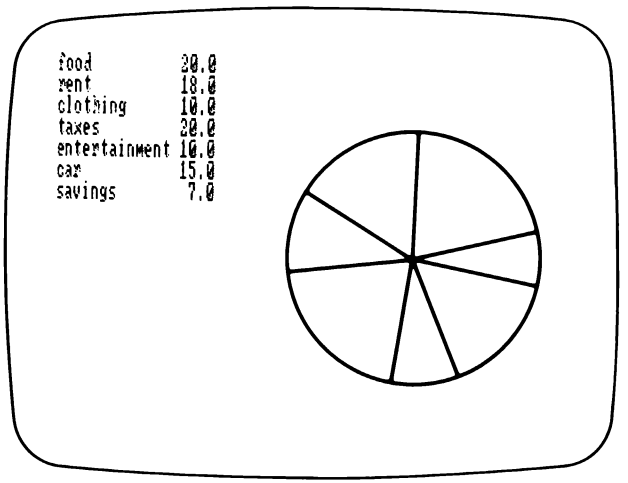


Abb. 8.26 Kreisdiagramm über die geplante prozentuale Verteilung der Ausgaben eines amerikanischen Haushalts

Überblick über die geplante prozentuale Verteilung der Ausgaben einer amerikanischen Familie vermitteln. Ihre vorgesehenen Ausgaben hat sie auf die folgenden Kategorien aufgeschlüsselt:

- *food (Lebensmittel):* *20.0%*
- *rent (Miete):* *18.0%*
- *clothing (Kleidung):* *10.0%*
- *taxes (Steuern und Abgaben):* *20.0%*
- *entertainment (Unterhaltung):* *10.0%*
- *car (Auto):* *15.0%*
- *savings (Sparen):* *7.0%*

Wir wollen das Programm, das wir zu entwickeln haben, in mehrere logisch gegenseitig abgegrenzte Abschnitte unterteilen. Der erste Programmabschnitt, den wir als Programmeinleitung bezeichnen wollen, soll sich mit der Bereitstellung der für den Programmablauf erforderlichen Hilfsmittel und Daten befassen. Dazu gehören:

- Anlegen von Bereichen für die Daten, die links auf dem Bildschirm erscheinen sollen
  - Bereich B$ für die Bezeichnungen der Kategorien
  - Bereich A für die Prozentsätze, die zu den Ausgabenkategorien gehören
- DATA-Statements für die Definition der Daten (die numerischen Werte dabei als Bruchteile des Ganzen)
- Statements zum Lesen der Daten aus den DATA-Statements und Zuweisung derselben zu den Bereichselementen
- Initialisierung der Anzeige-Betriebsart 2 (Graphischer Modus hohen Auflösungsvermögens) mit Hilfe des Statements SCREEN
- Ausschalten der Anzeige der Funktionstasten

Nach der Zusammenstellung der vom einleitenden Programmabschnitt zu lösenden Teilaufgaben können wir nun an seine Niederschrift herangehen (Abb. 8.27).

```
100  'Einleitender Programmabschnitt
110  DIM  A(7), B$(7),  WINKEL(7)
120  DATA  food,0.20, rent,0.18, clothing,0.10, taxes,0.20
130  DATA  entertainment,0.10, car,0.15, savings,0.07
140  FOR  J = 1  TO  7
150    READ B$(J),A(J)
160  NEXT J
170  SCREEN 2:        'Graphischer Modus hohen Aufloesungsvermoegens
180  KEY OFF:         'Ausschaltung der Anzeige der Funktionstasten
```

Abb. 8.27 Einleitender Programmabschnitt

Der nächste Programmabschnitt soll der Anzeige der Texte links oben auf dem Bildschirm gewidmet werden. Das erfordert von uns eine gewisse Überlegung und Sorgfalt. Wir wollen die Ausgabe in der 5. Zeile beginnen, d. h. die ersten vier Zeilen leer lassen. Da wir die Prozentsätze in den DATA-Statements als Anteile eines Ganzen angegeben haben, was uns später die Berechnung der Winkel erleichtern wird, müssen wir sie bei der Ausgabe mit 100 multiplizieren. Auf die Mitführung der Prozentzeichen bei der Ausgabe wollen wir verzichten, da kein Irr-

tum hinsichtlich der Bedeutung der ausgegebenen Werte möglich ist. Zwecks Erzielung einer formatierten Ausgabe verwenden wir die Option USING im PRINT-Statement. Um klare Spalten für die auszugebende Tabelle zu bekommen, wollen wir die Bezeichnungen der Ausgabekategorien in die Ausgabezone 1 stellen, die Prozentsätze in die Zone 2. Der unter Zugrundelegung dieser Überlegungen geschriebene Programmabschnitt ist in der Abb. 8.28 dargestellt; wir wollen unsere Aufmerksamkeit insbesondere auf die Ausgabestatements in den Zeilen mit den Zeilennummern 240 und 250 richten.

```
200  'Programmabschnitt fuer die Ausgabe der Textzeilen
210  CLS
220  PRINT:      PRINT:      PRINT:      PRINT
230  FOR  J = 1  TO  7
240    PRINT B$(J),;:     'Ausgabe und Uebergang zur 2. Ausgabezone
250    PRINT USING "##.#"; 100*A(J)
260  NEXT J
```

Abb. 8.28 Programmabschnitt für die Ausgabe der Textzeilen

Schließlich kommen wir zu dem Programmabschnitt, der die Zeichnung des Kreisdiagramms bewirken soll. Im Element J des Bereiches A ist das Verhältnis gespeichert, das der zugehörige Bogen zum gesamten Kreis einnimmt. Im Bogenmaß gemessen bedeutet das einen Mittelpunktswinkel von A(J)*(2*PI), den wir für den J-ten Kreisbogen vorsehen müssen. Bekanntlich ist ja das Bogenmaß für den Vollwinkel 2*PI (die Variable PI soll einen Näherungswert von π enthalten). Das erste „Tortenstück" beginnt somit mit dem Anfangswinkel 0. Diesen Wert weisen wir infolgedessen auch dem Element 0 des Bereiches WINKEL zu; dieser Bereich wurde im einleitenden Programmabschnitt ebenfalls definiert. Das erste „Tortenstück" endet beim Endwinkel WINKEL(1) = A(1)*(2*PI) . Der zweite Sektor beginnt dort, wo der erste Sektor endet, nämlich bei dem Winkel zur Horizontalen, dessen Bogenmaß unter WINKEL(1) gespeichert ist. Das zweite „Tortenstück" endet bei dem folgenden Winkel zur Horizontalen:

```
WINKEL(2) = WINKEL(1) + A(2)*(2*PI)
```

Diese Winkelberechnungen müssen solange fortgesetzt werden, bis der letzte Sektor (bekanntlich zu „savings" gehörig) erfaßt ist. Nach diesen Überlegungen sind wir in der Lage, auch den dritten und letzten Programmabschnitt (Abb. 8.29) niederzuschreiben. Man beachte, daß durch diesen Programmabschnitt jeder die Sektoren begrenzende Radius zweimal gezeichnet wird. Diese Angelegenheit braucht uns aber nicht zu kümmern.

```
300  'Programmabschnitt zum Zeichnen des Kreisdiagramms
310  WINKEL(0) = 0
320  PI = 3.14159:          'Zuweisung eines Naeherungswertes
330  FOR  J = 1  TO  7
340    T = A(J)*(2*PI):     'Zum laufenden Sektor gehoerender
                             Mittelpunktswinkel
350    WINKEL(J) = WINKEL(J-1) + T:    'Berechnung des Endwinkels
                                        des laufenden Sektors
360    CIRCLE (450,100),100,, -WINKEL(J),-WINKEL(J-1)
370  NEXT J
```

Abb. 8.29 Programmabschnitt zum Zeichnen des Kreisdiagramms

Im Statement CIRCLE (auf der Zeile mit der Zeilennummer 360) haben wir keine Farbe angegeben. Dennoch haben wir für den Farbparameter Platz gelassen, indem wir ein zusätzliches Komma eingefügt haben. (Der Platz für die Farbangabe ist nur imaginär frei gelassen worden, denn die beiden Kommas folgen unmittelbar aufeinander). Wenn BASIC nach einem Parameter verlangt, der an einem bestimmten Platz des Statements erwartet wird, kann man diesen gewöhnlich weglassen, Hauptsache, man läßt (auch imaginären) Platz für ihn. Geschieht nichts dergleichen, kann BASIC das betreffende Statement nicht interpretieren.

Vielleicht wundert es den Leser, daß wir den Mittelpunkt des Kreises auf das Pixel mit den Koordinaten (450,100) gelegt und den Radius 100 Einheiten groß gewählt haben. Gut, wir geben zu, daß wir vor dieser Entscheidung mehrere Male mit verschiedenen Mittelpunkten und Radien probiert haben und meistens mit dem Ergebnis unbefriedigt waren. Es dauerte lange, bis wir zu der vorgelegten Lösung gelangten, die uns auch ästhetisch einigermaßen zufriedenstellte. Bei graphischen Arbeiten sollte man grundsätzlich nicht davor zurückschrecken, sich vom eigenen Auge leiten zu lassen.

Wir wollen die Betrachtungen über diese Aufgabenstellung durch einen Blick auf das Gesamtprogramm abschließen (Abb. 8.30). Es ist natürlich durch die Zusammenziehung der einzelnen Programmabschnitte unter Hinzufügung des END-Statements entstanden. Das Gesamtprogramm kann natürlich nach eigenem Ermessen erweitert, geändert oder ergänzt werden; auf einige solche Aspekte verweisen die Aufgaben der nachfolgenden Aufgabengruppe 30.

```
100  'Einleitender Programmabschnitt
110  DIM  A(7), B$(7),  WINKEL(7)
120  DATA  food,0.20, rent,0.18, clothing,0.10, taxes,0.20
130  DATA  entertainment,0.10, car,0.15, savings,0.07
140  FOR  J = 1  TO  7
150    READ B$(J),A(J)
160  NEXT J
170  SCREEN 2:       'Graphischer Modus hohen Aufloesungsvermoegens
180  KEY OFF:        'Ausschaltung der Anzeige der Funktionstasten
200  'Programmabschnitt fuer die Ausgabe der Textzeilen
210  CLS
220  PRINT:      PRINT:      PRINT:      PRINT
230  FOR  J = 1  TO  7
240    PRINT B$(J),;:     'Ausgabe und Uebergang zur 2. Ausgabezone
250    PRINT USING "##.#"; 100*A(J)
260  NEXT J
300  'Programmabschnitt zum Zeichnen des Kreisdiagramms
310  WINKEL(0) = 0
320  PI = 3.14159:        'Zuweisung eines Naeherungswertes
330  FOR  J = 1  TO  7
340    T = A(J)*(2*PI):   'Zum laufenden Sektor gehoerender
                           Mittelpunktswinkel
350    WINKEL(J) = WINKEL(J-1) + T:    'Berechnung des Endwinkels
                                        des laufenden Sektors
360    CIRCLE (450,100),100,, -WINKEL(J),-WINKEL(J-1)
370  NEXT J
400  END
```

Abb. 8.30 Programm zum Zeichnen von Kreisdiagrammen mit zusätzlicher Textausgabe

Aufgabengruppe 31

1. Man ändere das in Abb. 8.30 vorgelegte Programm zur Darstellung von Kreisdiagrammen so ab, daß die Daten über die Tastatur eingegeben werden können (Verzicht auf die starren DATA-Statements). Die Aufforderungen zur Dateneingabe sollen enden, wenn der Benutzer als letztes Datenelement die Zeichenkette "@" eingegeben hat. - Im Programm ist Platz für maximal 20 Datenelemente vorzusehen.

2. Man ändere das zur Lösung der Aufgabe 1. geschaffene Programm so ab, daß das Kreisdiagramm in der Farbe 2 der gegenwärtig geltenden Palette gezeichnet wird.
   Anmerkung: Die Modifikation zieht eine Neuaufteilung der Bildschirmausgabe nach sich. Man denke u.a. daran, daß man jetzt den graphischen Modus mittleren Auflösungsvermögens vor sich hat, in dem bekanntlich die Zeilenlänge für die Textausgabe nur 40 Stellen beträgt.

8.7 Malen und Zeichnen

In diesem Abschnitt wollen wir über zwei weitergehende BASIC-Statements sprechen, die freilich nur im „erweiterten BASIC" zur Verfügung stehen. Das „erweiterte BASIC" kann bekanntlich im Betriebssystem DOS durch Eingabe von BASICA geladen werden. Die beiden fraglichen Statements betreffen das Malen (PAINT) und das Zeichnen (DRAW).

8.7.1 Statement PAINT

Mit den graphischen Statements von BASIC, die für den Personalcomputer geschaffen wurden, kann eine beachtliche Vielfalt von Figuren gezeichnet werden. Die Abb. 8.31 zeigt z.B. ein Dreieck, das mit Hilfe mehrerer LINE-Statements gezeichnet wurde. In Abb. 8.32 wiederum ist ein Kreis dargestellt, dessen Zeichnung durch das CIRCLE-Statement erfolgte. Unterhalb jeder Figur sind die Statements aufgeführt, durch die sie hervorgerufen wurde. Die Farbe der Randlinien der Figuren sind in üblicher Weise in den Statements genannt: Das Dreieck ist in der Farbe 2 gezeichnet, der Kreis in der Farbe 3.

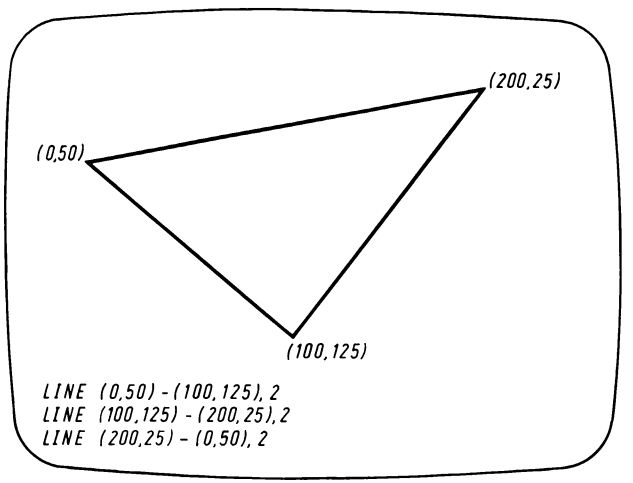


Abb. 8.31 Dreieck mit den Eckpunkten (0,50), (100,125) und (200,25) in der Farbe 2

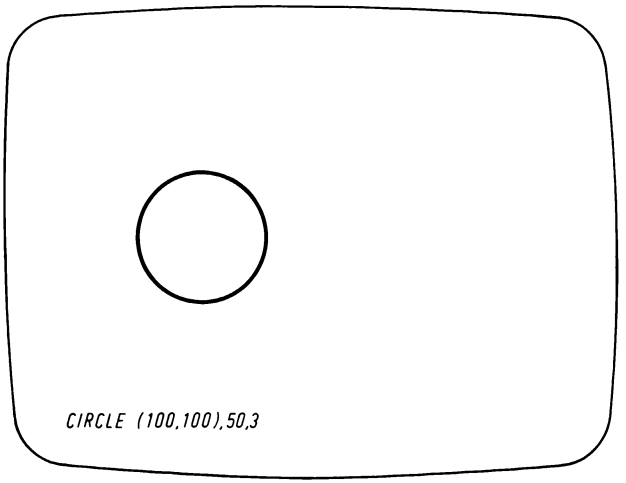


Abb. 8.32 Kreis mit einem Radius von 50 Einheiten um den Mittelpunkt M(100,100) in der Farbe 3

Mit dem PAINT-Statement kann man nun das Innere einer Figur kolorieren, gerade so, als ob sich die Figur in einem Malbuch befindet und man sie mit einem Buntstift ausmalt. Wir wollen nun zeigen, wie die PAINT-Statements aussehen, mit denen das Dreieck (Abb. 8.31) bzw. der Kreis (Abb. 8.32) ausgemalt werden können.

Das PAINT-Statement besitzt die folgende allgemeine Gestalt:

```
PAINT (x,y),farbe,randfarbe
```

Hierbei bedeuten:

(x,y)	Koordinaten eines Pixels (Punktes) im Innern der Figur
farbe	Farbnummer der Farbe, mit der das Innere der Figur ausgemalt werden soll
randfarbe	Farbnummer der Farbe, in der der Rand der Figur vorliegt

Das Ausmalen des Figureninnern beginnt beim Punkt (x,y) und erfolgt von dort aus nach allen Richtungen. Wenn in einer Richtung auf die Randfarbe gestoßen wird, wird das Ausmalen in dieser Richtung unterbunden. Betrachten wir z. B. das Dreieck in der Abb. 8.31, in dessen Innern der Punkt P(75,75) liegt. Wenn es mit der Farbe 3 ausgemalt werden soll, lautet das Statement wie folgt:

```
PAINT (75,75),3,2
```

Das Ergebnis dieses Statements finden wir in der Abb. 8.33; man vergleiche hiermit das in Abb. 8.31 dargestellte Ausgangsdreieck, das ja bekanntlich in der Farbe 2 vorliegt.

Für den Kreis (Abb. 8.32) heißt das Statement, wenn derselbe in der Farbe 3 ausgemalt werden soll:

```
PAINT (100,100),3,3
```

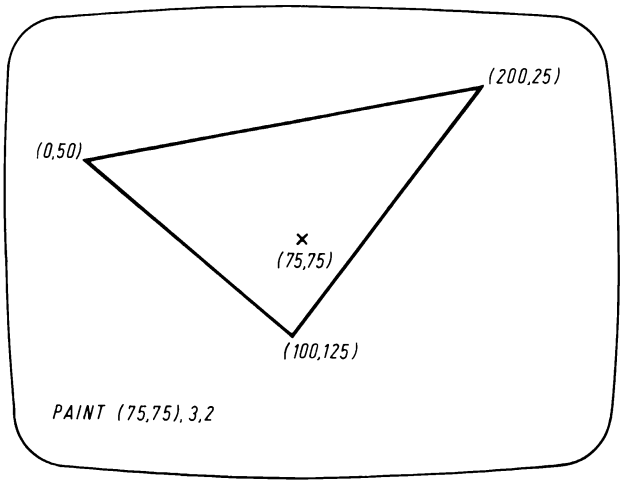


Abb. 8.33 In der Farbe 2 ausgemaltes Dreieck mit den Eckpunkten (0,50), (100,125) und (200,25) unter Benutzung des Startpunktes (75,75)

Testübung 8.7.1

Wie heißt das Statement, mit dem der Kreis von Abb. 8.32 in der Farbe 1 ausgemalt werden kann?

8.7.2 Statement DRAW

Unter Benutzung der verschiedenen graphischen Statements von BASIC kann man durchaus recht verwickelte Figuren auf dem Bildschirm darstellen. Die Programme freilich, die das leisten sollen, werden zusehends komplizierter, je komplexer die darzustellenden Figuren sind. Viele Figuren bestehen zudem aus geraden Linien in verschiedenen Richtungen und Lagen. Solche Zeichenaufgaben können bei Benutzung des DRAW-Statements knapp, doch präzis beschrieben werden.

Um das DRAW-Statement zu verstehen, bilde man sich einmal ein, daß man einen fiktiven Bleistift in der Hand hat, mit dem man auf dem Bildschirm zeichnen kann. Die Bewegung des Bleistifts wird durch Befehle gesteuert, die in das DRAW-Statement aufgenommen sind. Dieses besitzt das folgende allgemeine Format:

```
DRAW <zk>
```

Hierbei versteht man unter <zk> eine Zeichenkette, die eine Folge von Befehlen zur Steuerung der Zeichenbewegungen enthält. Anders ausgedrückt: Die anzufertigende Zeichnung wird mit einer Graphik-Definitionssprache beschrieben, deren einzelne Befehle die Zeichenbewegungen steuern. Die Folge der Sprachbefehle zur Beschreibung einer Zeichnung bildet den Inhalt der Zeichenkette <zk> .

Bei der Aufsetzung der Befehle muß man die Punkte auf dem Bildschirm ansprechen. Die von den meisten Befehlen veranlaßten Aktionen hängen nämlich von dem Punkt ab, auf den zuletzt eingegangen wurde. Darunter versteht man den Punkt, auf den zuletzt in einem Befehl des DRAW-Statements bezug genommen wurde. Es ist jedoch zu beachten, daß die beiden Statements CLS und RUN bei ihrer Ausführung zuletzt auf den Punkt bezug nehmen, der im Zentrum des Bildschirmes liegt, also auf den Punkt

- (160,100) im graphischen Modus mittleren Auflösungsvermögens
- (320,100) im graphischen Modus hohen Auflösungsvermögens

Die graphischen Befehle, die mit dem DRAW-Statement verknüpft sind, werden durch einzelne Buchstaben kenntlich gemacht. Fundamental ist der Befehl M, der nachstehend aufgeführt ist:

```
DRAW "M x,y"
```

Dieser Befehl veranlaßt das Zeichnen einer geraden Linie vom zuletzt angesprochenen Punkt zum Punkt mit den Koordinaten (x,y). Nach der Ausführung des Befehles gilt als zuletzt angesprochener Punkt natürlich der Punkt (x,y).

Wir wollen noch zwei Variationen des Befehles M erwähnen:

1. Wenn der Buchstabe N dem Buchstaben M vorangestellt wird, dann gilt als zuletzt angesprochener Punkt nach wie vor derjenige, der vor Ausführung des Befehles als zuletzt angesprochener Punkt galt. Diese Möglichkeit kann man z. B. wunderbar zur Zeichnung eines Winkels, ausgehend vom Scheitelpunkt S, benutzen (siehe Abb. 8.34). Der Scheitelpunkt S des Winkels möge bei S(100,90) liegen; außerdem liege der graphische Modus hohen Auflösungsvermögens vor. Die Zeichnung des fraglichen Winkels erreicht man durch das Statement

```
DRAW "NM 300,90 M 500,150"
```

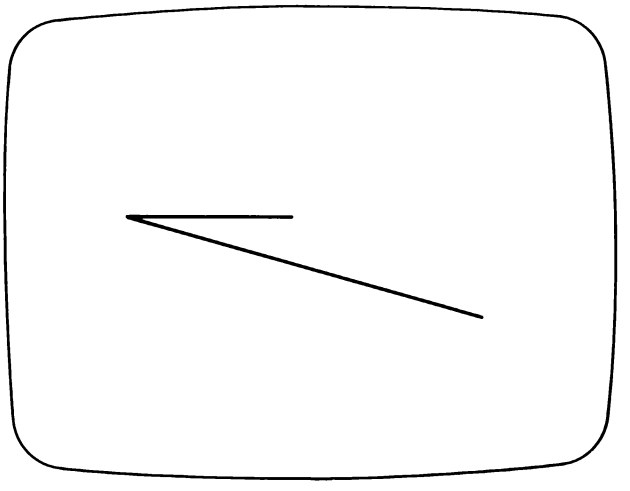


Abb. 8.34 Zeichnung eines Winkels durch das Statement DRAW

2. Wenn der Buchstabe B dem Buchstaben M vorangestellt wird, dann erfolgt wohl eine Änderung des zuletzt angesprochenen Punktes, die Zeichnung der Verbindungsgeraden unterbleibt jedoch. Der Befehl BM kann also zur Neueinstellung des „Zeichenstiftes" benutzt werden. Man betrachte hierzu die Winkelzeichnung in Abb. 8.35, wobei davon ausgegangen ist, daß der zuletzt angesprochene Punkt nicht der Scheitelpunkt ist. Der Scheitelpunkt S habe die Koordinaten (120,90). Das DRAW-Statement lautet, wenn der Winkel von S ausgehend gezeichnet wird:

```
DRAW "BM 120,90 NM 500,150 M 350,80"
```

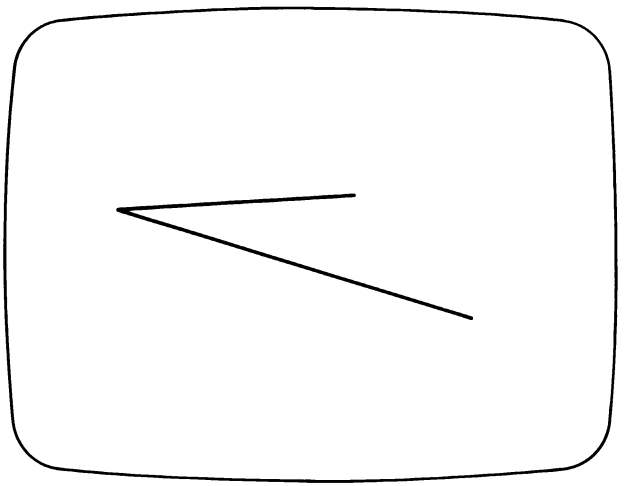


Abb. 8.35 Winkelzeichnung mit DRAW

Testübung 8.7.2

Wie heißt das DRAW-Statement, mit dem das Dreieck von Abb. 8.35 gezeichnet werden kann?

a) *Benutzung relativer Koordinaten*
Unsere bisherigen Ausführungen stützten sich auf absolute Koordinaten, d. h. von uns wurden stets die tatsächlichen Koordinaten genannt. Man kann jedoch auch die folgende Form des Befehles M benutzen:

```
M +w,+s
```

Durch diesen Befehl erfolgt die Zeichnung einer Verbindungsgeraden vom zuletzt angesprochenen Punkt zu einem Punkt, der sich w Einheiten nach rechts und s Einheiten nach unten entfernt vom zuletzt angesprochenen Punkt befindet, d. h. auf die Koordinaten

des zuletzt angesprochenen Punktes ist w bzw. s zu addieren. Diesem Befehl entsprechen die Befehle:

```
M -w,+s
M -w,-s
M +w,-s
```

Das Minuszeichen bedeutet eine Subtraktion bei den betreffenden Koordinaten, also eine Lokalisierung des Endpunktes links bzw. oberhalb des zuletzt angesprochenen Punktes.

b) *Angabe von Koordinaten unter Benutzung von Variablen*
Die Koordinaten in den M-Befehlen können auch durch Variablen <var1> und <var2> ausgedrückt werden. In diesem Fall besitzt der M-Befehl die folgende Form:

```
M =<var 1>;,=<var 2>;
```

Man beachte bei dieser Form die Verwendung und die Stellung der drei Sonderzeichen: Gleichheitszeichen (=), Semikolon (;) und Komma (,). Diese dürfen nicht weggelassen werden. Wenn man z.B. die Verbindungsgerade zwischen dem zuletzt angesprochenen Punkt und dem Punkt, dessen Koordinaten als Werte den Variablen A bzw. B zugewiesen sind, zeichnen lassen will, so verwende man den Befehl

```
M =A;,=B;
```

Wenn den Gleichheitszeichen Pluszeichen oder Minuszeichen vorangestellt werden, so werden die den Variablen zugewiesenen Werte als relative Koordinaten angesehen. Wenn man z.B. die Verbindungsgerade zwischen dem zuletzt angesprochenen Punkt und dem

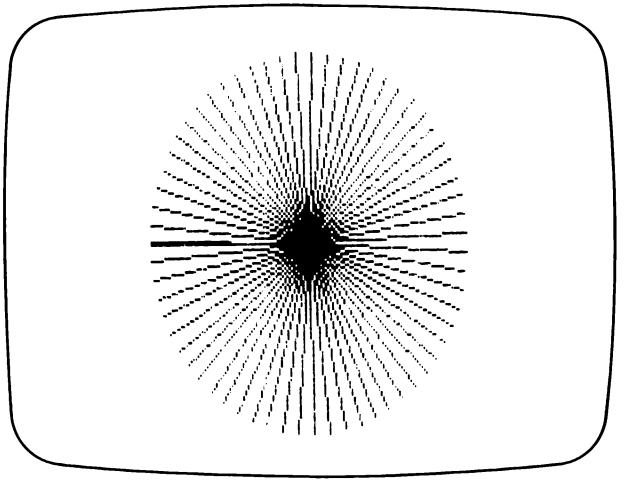


Abb. 8.36 Schmuckfigur auf dem Bildschirm

Punkt, dessen Koordinaten A Einheiten rechts und B Einheiten unterhalb des zuletzt angesprochenen Punktes liegen, so benutze man den Befehl

```
M +=A;,+=B;
```

Man beachte, daß nur das Vorzeichen, das vor einer Variablen steht, die Bewegungsrichtung bestimmt. Wenn z. B. das vor der Variablen A stehende Vorzeichen negativ ist, so erfolgt eine Bewegung um ABS(A) Einheiten nach links; unter ABS(A) ist hier der Absolutwert von A zu verstehen.

Ein ausgeklügeltes Bild, das mit Hilfe des DRAW-Statements entstanden ist, ist in der Abb. 8.36 zu finden. Das Programm, in das das DRAW-Statement eingebettet ist, ist in der Abb. 8.37 aufgelistet.

```
10 SCREEN 1:      CLS:      KEY OFF
20 FOR  R=0 TO 6.3 STEP 0.1
30   A = 160 + 70*COS(R)
40   B = 100 + 70*SIN(R)
50   DRAW "NM =A;,=B;"
60 NEXT R
70 END
```

Anmerkung: Unter SIN bzw. COS sind die trigonometrischen Funktionen Sinus bzw. Kosinus zu verstehen.

Abb. 8.37 Programm zur Erzeugung der in Abb. 8.36 dargestellten Figur

Testübung 8.7.3

Unter Benutzung des Zufallszahlengenerators sind 50 Paare von Zufallspunkten mit ihren Koordinaten zu erzeugen. Mit Hilfe von DRAW soll eine Verbindungsgerade zwischen den beiden Punkten jedes Paares gezogen werden.

c) *Ergänzungen zu den Relativbewegungen*
Bei den meisten Zeichenaufgaben werden die Koordinaten in relativer Form und nicht in absoluter Form angegeben. Um die Länge der Zeichenketten zu verkürzen, die solche Be-

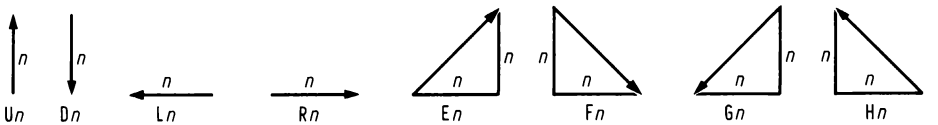


Abb. 8.38 Befehle U bis H für die relativen Bewegungen

wegungen beschreiben, können im DRAW-Statement auch die folgenden Befehle aufgeführt werden:

U n	Bewegung um n Einheiten nach oben (up)
D n	Bewegung um n Einheiten nach unten (down)
L n	Bewegung um n Einheiten nach links (left)
R n	Bewegung um n Einheiten nach rechts (right)
E n	Bewegung um n Einheiten nach „Nordosten", d.h. n Einheiten nach rechts und n Einheiten nach oben
F n	Bewegung um n Einheiten nach „Südosten", d.h. n Einheiten nach rechts und n Einheiten nach unten
G n	Bewegung um n Einheiten nach „Südwesten", d.h. n Einheiten nach links und n Einheiten nach unten
H n	Bewegung um n Einheiten nach „Nordwesten", d.h. n Einheiten nach links und n Einheiten nach oben

Die Wirkung ist in der Abb. 8.38 gezeigt.

Man kann die beiden optionalen Buchstaben N bzw. B auch den Befehlen U bis H voranstellen. So bewirkt z. B. das Statement

```
DRAW "NU 50",
```

das Zeichnen einer Vertikalen vom zuletzt angesprochenen Punkt um 50 Einheiten nach oben. Der Punkt, der vor Ausführung des Befehles NU als zuletzt angesprochener Punkt angesehen wurde, gilt dabei auch nach der Befehlsausführung als zuletzt angesprochener Punkt. Betrachten wir als zweites Beispiel nunmehr das Statement:

```
DRAW "BU 50"
```

Durch den Befehl BU wird als neuer zuletzt angesprochener Punkt derjenige Punkt installiert, der 50 Einheiten oberhalb des Punktes liegt, der vor der Befehlsausführung als zuletzt angesprochener Punkt galt. Die Zeichnung einer Geraden unterbleibt jedoch aufgrund des Befehlsteiles B.

Natürlich kann man auch Variablen im Zusammenhang mit den Befehlen U bis H benutzen. Man betrachte hierzu das Statement:

```
DRAW "U =A;"
```

Durch den in der Zeichenkette enthaltenen Befehl U wird eine Gerade gezeichnet, die den zuletzt angesprochenen Punkt P mit einem Punkt verbindet, der unter der Voraussetzung, daß n>0 ist, n Einheiten oberhalb von P liegt. Unter n sei hierbei der der Variablen A zugewiesene Wert verstanden. Ist n jedoch negativ, erfolgt die Zeichenbewegung um n Einheiten nach unten.

d) *Farbe*

In die Graphik-Definitionssprache, die für das Statement DRAW geschaffen wurde, ist auch ein Befehl aufgenommen worden, durch den die Farbe festgelegt werden kann, die für eine Bewegung zum Zeichnen oder eine Folge von Bewegungen zum Zeichnen gelten soll. Der Farbbefehl lautet allgemein:

```
C f
```

Unter f ist hierbei die Farbnummer der Farbe zu verstehen, die aus der gegenwärtig geltenden Farbpalette ausgewählt werden soll, d.h. f kann nur die Werte 0, 1, 2 oder 3 annehmen. Mit dem in der Abb. 8.39 aufgelisteten Programm kann das in der Abb. 8.40 dargestellte Segelboot gezeichnet werden.

```
10 SCREEN 1,0:      CLS:      KEY OFF
20 COLOR 7,0
30 DRAW  "C1 L60 E60 D80 C2 L60 F20 R40 E20 L20"
40 END
```

Abb. 8.39 Programm zum Zeichnen des in der Abb. 8.40 dargestellten Segelbootes

Durch das in der Abb. 8.39 aufgelistete Programm entsteht ein Bild auf weißem Hintergrund mit einem Segelboot, dessen Rumpf rot und dessen Segel grün ist.

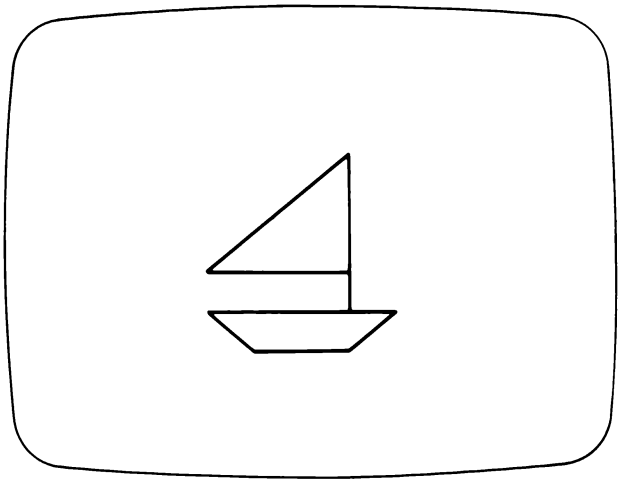


Abb. 8.40 Darstellung eines Segelbootes

e) *Drehwinkel*
Man kann dafür sorgen, daß ein Bild um einen rechten Winkel oder um das Vielfache eines rechten Winkels gedreht wird. Man muß hierzu nur der Befehlsfolge, die die zu zeichnende Figur in ungedrehter Lage beschreibt, einen Drehbefehl vorausgehen lassen. Dieser lautet allgemein:

```
An
```

Der Parameter n kann hierbei folgende Werte annehmen:

0	keine Drehung
1	Drehung um 90 Grad im Uhrzeigersinne
2	Drehung um 180 Grad im Uhrzeigersinne
3	Drehung um 270 Grad im Uhrzeigersinne

In Abb. 8.41 ist ein Programm aufgelistet, durch das das Segelboot von Abb. 8.40 in eine der möglichen Lagen gedreht wird. Die Lage hängt von der eingegebenen Kennung für den Drehwinkel ab.

```
10 SCREEN 1:     CLS:     KEY OFF:     PSET (160,100)
20 INPUT "BITTE KENNUNG (0 BIS 3) FUER DREHWINKEL EINGEBEN: "; N
30 DRAW  "A=N; BU40 L30 E30 D40 L30 F10 R20 E10 L10"
40 END
```

Abb. 8.41 Programm zum Zeichnen gedrehter Segelboote (siehe Abb. 8.40)

In der Abb. 8.42 ist das Resultat des Programmes von Abb. 8.41 zu sehen; dabei sind alle möglichen Lagen des Segelbootes berücksichtigt.

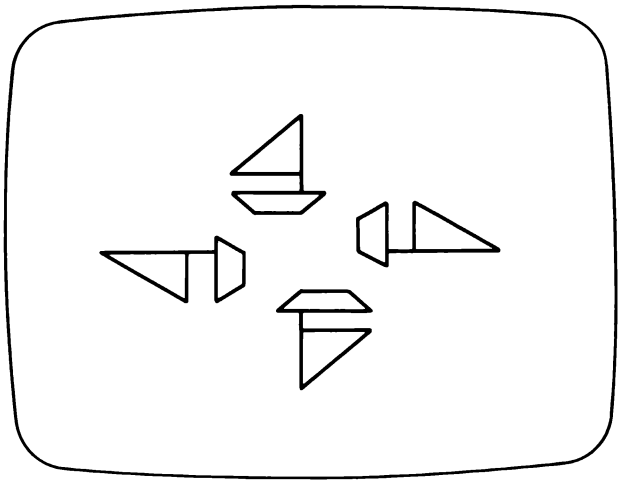


Abb. 8.42 Darstellung gedrehter Segelboote

f) *Maßstab*
Man kann dafür sorgen, daß ein Bild in einem bestimmten Maßstab abgebildet wird. Der hierfür geltende Befehl lautet allgemein:

S n

Dieser Befehl veranlaßt, daß alle Strecken, d.h. die Länge aller Linien, mit dem Faktor n/4 multipliziert werden, bevor sie auf dem Bildschirm erscheinen. Unter n ist eine natürliche Zahl aufzuführen, die im Wertebereich von 1 bis einschließlich 255 liegt.

Testübung 8.7.4

Man schreibe das DRAW-Statement nieder, durch das das Segelboot von Abb. 8.44 um die Hälfte verkleinert wird, d.h. im Maßstab 1:2 auf dem Bildschirm erscheint.

g) *Teilketten*

Man kann die Zeichenkette mit den Zeichnungsbefehlen auch außerhalb des DRAW-Statements definieren. Nehmen wir einmal an, daß eine solche Zeichenkette der Variablen A$ zugewiesen ist. Dann kann man sich im DRAW-Statement wie folgt auf diese Zeichenkette beziehen:

```
DRAW A$
```

Oft kommt es vor, daß man sich bei der Anfertigung einer Zeichnung mehrere Male auf eine Zeichenkette beziehen will oder muß, da man Teile der Abbildung mehrere Male wiederholen möchte. Man kann das erreichen, indem man eine Zeichenkette in eine andere, größere Zeichenkette einbettet. Um die Einbettung vorzunehmen, verlangt die Befehlssprache für Graphik, daß der eingebetteten Teilkette der Buchstabe X vorangestellt wird und daß ihr das Sonderzeichen ; folgt. Das nachfolgende DRAW-Statement veranlaßt z.B., daß zunächst die Figur gezeichnet wird, die in der der Variablen A$ zugewiesenen Zeichenkette beschrieben ist. Anschließend erfolgt eine Bewegung um 50 Einheiten nach oben. Nach dieser Bewegung wird die fragliche Figur in der neuen Lage noch einmal gezeichnet.

```
DRAW "XA$; BU50 XA$;"
```

Aufgabengruppe 32

1. Man schreibe ein Programm, mit dem die in Abb. 8.43 dargestellte Figur gezeichnet werden kann.

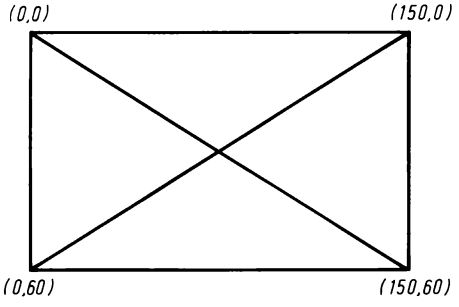


Abb. 8.43 Rechteck mit Diagonalen

2. Man schreibe ein Programm zur Darstellung der um 270 Grad im Uhrzeigersinne gedrehten Figur von Abb. 8.43!
3. Man schreibe ein Programm zur Darstellung der auf das Zweifache vergrößerten Figur von Abb. 8.43!

Antworten auf die Testübungen

8.7.1 `PAINT (100,100),1,3`

8.7.2 `DRAW "BM 0,50 M 100,125 M 400,25 M 0,50"`

8.7.3
```
10  DIM  X1(50),Y1(50), X2(50),Y2(50)
15  CLS
20  SCREEN 2
25  KEY OFF
30  FOR  J=1 TO 50
35    X1(J) = INT(RND*620):   X2(J) = INT(RND*620)
40    Y1(J) = INT(RND*200):   Y2(J) = INT(RND*200)
45    DRAW "BM =X1(J);,=Y1(J); M =X2(J);,=Y2(J);"
50  NEXT J
```

8.7.4
```
DRAW  "S2  C1 L60 E60 D80  C2 L60 F20 R40 E20 L20"
```

8.8 Tonerzeugung und Musik mit Personalcomputern

Beim Personalcomputer befindet sich links an der Vorderfront der Systemeinheit ein Lautsprecher. Man kann den Lautsprecher in Programmen ansprechen und damit die Erzeugung von Tönen, ja sogar das Spielen von Musikstücken programmieren. Für diese Zwecke gibt es drei verschiedene Statements in BASIC:

- BEEP
- SOUND
- PLAY

Wir wollen uns jetzt damit beschäftigen, einen Überblick über die Funktionen dieser Statements zu geben.

8.8.1 Statement BEEP

Als einfachstes der drei Statements zur Erzeugung von Tönen ist wohl das Statement BEEP anzusehen. Seine Ausführung läßt den Lautsprecher für ¼ Sekunde ertönen; Tonhöhe und Tondauer können dabei nicht gesteuert oder kontrolliert werden. Als Beispiel für die Verwendung von BEEP wollen wir ein Unterprogramm betrachten, das dem Zweck dienen soll, auf Eingabefehler des Benutzers zu reagieren (Abb. 8.44).

```
960  PRINT "ES LIEGT EIN FEHLER BEI DER EINGABE VOR"
970  PRINT "---> BITTE EINGABE WIEDERHOLEN"
980  BEEP
990  RETURN
```

Abb. 8.44 Unterprogramm mit Anweisung BEEP

Man kann das BEEP-Statement natürlich auch mit anderen Statements verbinden, beispielsweise mit einem IF-Statement:

```
10  IF  X=100  THEN BEEP
```

In die in der Praxis eingesetzten Programme sind gewöhnlich ausgeklügelte Eingaberoutinen aufgenommen, die die vom Benutzer eingegebenen Daten einer Vielzahl verschiedener Prüfungen unterziehen. Durch diese soll ermittelt werden, ob die eingegebenen Daten den vorgegebenen, an sie gestellten Bedingungen genügen (Plausibilitätsprüfungen). Diese Prüfungen befassen sich beispielsweise mit solchen Fragestellungen:

- Ist die Länge korrekt?
- Enthalten die Datenelemente ungültige Zeichen?

usw.

Ein einfaches, solchen Zwecken dienendes Unterprogramm wollen wir in der Abb. 8.45 vorstellen. Dieses Unterprogramm setzt voraus, daß im Hauptprogramm oder in einem anderen, dieses Unterprogramm aufrufenden Programm der Variablen LAENGE ein Wert zugewiesen ist, und zwar die Maximallänge der einzugebenden Zeichenkette. Das Unterprogramm selbst beleuchtet zunächst einen Kasten, der auf der Stelle (1,1), d.h. in der oberen linken Ecke auf dem Bildschirm, beginnt und der soviel Stellen umfaßt, wie als Maximallänge vorgegeben wurden. Damit wird bereits vor der Eingabe die maximal mögliche Länge der einzugebenden Daten angezeigt. Anschließend läßt das Unterprogramm die Eingabe eines einzelnen Datenzeichens zu und zeigt es sofort auf dem Bildschirm in der zugehörigen Position des beleuchteten Feldes an. Dabei verschwindet natürlich die Beleuchtung der betreffenden Stelle. Bei

```
5000 'Eingaberoutine (Unterprogramm)
5001 'Parameter LAENGE:  Maximalzahl der Zeichen in der
                         einzugebenden Zeichenkette
5002 'Unter STELLE ist die augenblickliche Position des
      Cursors im Eingabefeld zu verstehen
5010 STELLE = 1
5020 CLS
5030 LOCATE 1,1
5040 PRINT ""
5050 LOCATE 1,1
5060 FOR  I=1 TO LAENGE
5070   LOCATE 1,I:      PRINT CHR$(219);
5080 NEXT I
5090 LOCATE 1,1
5100 A$ = INKEY$:      'Einlesen eines Zeichens
5110 IF  A$ = ""  THEN GOTO 5100
5120 IF  A$ = CHR$(8)  THEN GOTO 5200
5130 IF  A$ = CHR$(13)  THEN GOTO 5240
5140 IF  STELLE = LAENGE + 1  THEN GOTO 5180
5150 LOCATE 1,STELLE:      PRINT A$;
5160 STELLE = STELLE + 1
5170 GOTO 5100
5180 BEEP
5190 GOTO 5090
5200 STELLE = STELLE - 1
5210 IF  STELLE = 0  THEN BEEP:      STELLE = STELLE + 1
5220 LOCATE 1,STELLE:      PRINT CHR$(219);
5230 GOTO 5090
5240 RETURN:           'Rueckkehr zur aufrufenden Routine
```

Abb. 8.45 Unterprogramm für die Eingabe längenbegrenzter Zeichenketten

Betätigung der Rücksetztaste (Tastenaufschrift: ←) wird darüber hinaus je ein Beleuchtungszeichen in das Eingabefeld (in den Kasten) zurückgespeichert. Wenn man versucht, Zeichen in Stellen[1] einzugeben, die jenseits des beleuchteten Eingabefeldes liegen, piepst der Lautsprecher. Die im Unterprogramm in der Zeile mit der Zeilennummer 5100 verwendete Variable INKEY$ ist eine in BASIC eingefügte Variable, die wir ausführlich im Abschnitt 10.1 besprechen werden. Die Bezugnahme auf sie, soviel sei jetzt schon gesagt, bewirkt das Einlesen eines Zeichens aus dem Puffer für die Tastatur.

8.8.2 Statement SOUND

Das zweite Statement, mit dem der Lautsprecher angeregt werden kann, ist das Statement SOUND. Dieses handliche kleine Statement ermöglicht die Generierung von Tönen zwischen 37 und 32767 Hertz (Schwingungen/Sekunde). Die Dauer des Tones wird in speziellen Zeiteinheiten, sogenannten Zeitschlägen, gemessen. In einer Sekunde laufen 18.2 Zeitschläge ab. Im Statement SOUND kann die Dauer durch einen numerischen Ausdruck, dessen Wert im Wertebereich von 0 bis einschließlich 65535 liegt, angegeben werden. Der maximale Wert (65535) entspricht einer Zeit, die gering über einer Stunde liegt. Um einen Ton von 500 Hz auf die Dauer von 40 Zeitschlägen zu erzeugen, müßte man das folgende Statement formulieren:

```
SOUND 500,40
```

```
5    'Loeschen der Anzeigenzeile fuer die Funktionstasten
10   KEY OFF
15   'Uebergang vom Textmodus in den graphischen Modus 1
20   SCREEN 1
30   FOR  I = 1  TO  100
35     'Zeichnung eines Kreises mit zufaelligen
        Mittelpunktskoordianten und dem Radius 30
40     CIRCLE (RND*250,RND*200),30
45     'Erzeugung eines Tones mit einer auf Zufallsbasis
        errechneten Frequenz zwischen 37 und 1037 Hz und
        einer Dauer von 2 Zeitschlaegen
50     SOUND RND*1000+37,2
60     CLS
65     'Zeichnung eines Dreiecks
70     DRAW  "E15; F15; L30"
75     'Erzeugung eines Tones mit einer auf Zufallsbasis
        errechneten Frequenz zwischen 37 und 1037 Hz und
        einer Dauer von 2 Zeitschlaegen
80     SOUND RND*1000+37,2
90     CLS
100  NEXT I
110  END
```

Abb. 8.46 Programm mit begleitenden Tönen

[1] Eine solche Routine führt eine passive Plausibilitätsprüfung durch, indem sie von vornherein die Eingabe längerer Zeichenketten unterbindet.

Zur Verdeutlichung dieses Statements wollen wir ein elementares graphisches Programm vorlegen (Abb. 8.46), in das eine SOUND-Anweisung aufgenommen wurde. Das vorgelegte Programm zeichnet gleichgroße Dreiecke sowie Kreise mit dem Halbmesser 30 und einem Mittelpunkt, dessen Koordinaten von der Zufallszahl abhängen, die der Zufallszahlengenerator RND liefert. Dadurch, daß die erzeugten Figuren blinken und von zwei Zeitschlägen andauernden Tönen zwischen 37 und 1037 Hz begleitet werden, wird fast die Illusion eines Rockkonzertes erweckt. Mit SOUND kann man also eine Art musikalische Begleitung für ein Programm hervorrufen. Das auf der Zeile mit der Zeilennummer 70 verwendete Statement DRAW ist bereits im Abschnitt 8.7 besprochen worden. Hier genügt der nochmalige Hinweis, daß mit dieser Anweisung beliebige Figuren konstruiert werden können.

8.8.3 Musik mit Personalcomputern (PC)

Der Beschreibung von Musikstücken kommt am meisten das PLAY-Statement entgegen. Durch seinen Gebrauch kann man praktisch den PC in ein Klavier verwandeln und auf diesem musikalische Kompositionen abspielen lassen, einfache und auch schwierigere, mit anderen Worten, welche man will. Nebenbei gesagt, gibt es mittlerweile sogar ein Arrangement für die Mondscheinsonate (op. 27, Nr. 2 cis-Moll) von Beethoven.

Dem mit den Grundbegriffen der Musik nicht vertrauten Leser wollen wir zunächst einige Fakten vermitteln:

1. Ebenso wie ein Klavier weist der PC einen Tonraum von 7 Oktaven auf, numeriert von 0 bis 6. Jede Oktave beginnt mit dem Ton C und endet mit dem Ton B.
2. Die Oktave 3 beginnt mit dem kleinen c (Frequenz 523,25 Hz).
3. Unter dem Tempo eines Musikstückes versteht man die Geschwindigkeit, mit der es abgespielt wird. Beim PC wird das Tempo durch die Anzahl der in einer Minute ertönenden Viertelnoten (Vierteltaktschlägen) angegeben. Das Tempo kann im Wertebereich von 32 bis 255 liegen.
4. Der PC gestattet die Angabe einer Stilart (Vortragsbezeichnung) für das Abspielen der Musikstücke. Zwischen drei Stilarten kann ausgewählt werden:
   a) *normal*
   b) *legato*
   c) *staccato*

   Die normale Stilart bedeutet, daß die Töne für die Dauer von ⅞ ihrer definierten Länge anhalten. Die Stilart legato bedeutet, daß die Töne während der vollen Zeitperiode gespielt werden, die für sie festgelegt wurde. Bei der Stilart staccato erklingen hingegen die Töne nur während ¾ der für sie festgelegten Zeit. Musik in legato erscheint „gebunden", solche in staccato „gestoßen", d. h. nicht gebunden.

8.8.4 Statement PLAY

Durch Benutzung des Statements PLAY kann man seinen schöpferischen musikalischen Genius beweisen, selbst wenn man kein Instrument spielen kann. Diesem Statement liegt eine Ausdrucksweise zugrunde, die es ermöglicht, die Melodie eines Musikstückes in Form von Zeichenketten, die quasi anstelle der Notenfolge treten, auszudrücken. Nach der Umsetzung in Noten ertönt die Musik über den Lautsprecher.

Die Benutzung des PLAY-Statements erfordert zweierlei:

1. Codierung der Noten des zu spielenden Musikstückes als Zeichenkette
2. Verwendung der codierten Zeichenkette im Statement PLAY in der Form:

```
PLAY <zeichenkette>
```

Um das Ganze zu untermauern, wollen wir als Beispiel das Programm von Abb. 8.47 betrachten. Warum sollten wir es nicht eintippen und dem Ergebnis seines Ablaufes lauschen?

```
10  A$ = "O3L4EDCDEE"
20  PLAY "XA$; E2DDD2EGG2; T255XA$; EEDDEDP2C1"
30  END
```

Abb. 8.47 Spielen eines Musikstückes über den Lautsprecher

Sieht man sich dieses Programm an, so erscheint uns das Musikstück ziemlich mysteriös. Es ist jedoch außerordentlich einfach, die Noten so auszudrücken, daß sie der Computer versteht. Der nachfolgende Überblick zeigt uns die Grundsätze.

a) *Noten*
Die Noten werden durch die Buchstaben A bis G ausgedrückt; daran kann eines der Sonderzeichen #, + oder – angehängt werden. Hierbei bedeuten:

# oder +	Erhöhung um einen halben Ton
–	Erniedrigung um einen halben Ton

b) *Dauer*
Durch L wird die Spieldauer (Länge) einer Note festgelegt. Es bedeuten:

L1	ganze Note
L2	halbe Note
L4	Viertelnote
...	.
...	.
...	.
L64	1/64 Note

Die durch L festgelegte Länge (Spieldauer) einer Note gilt ebenfalls für die folgenden Noten, solange keine Neufestsetzung der Länge erfolgt. Wenn z. B. die Tonleiter in C-Dur in Viertelnoten gespielt werden soll, verwenden wir die Zeichenkette

```
L4 CDEFGABC
```

Wenn man nur die Dauer einer einzelnen Note festlegen will, läßt man den Kennbuchstaben L weg und schreibt die die Spieldauer charakterisierende Zahl unmittelbar hinter die Note; C4 bedeutet infolgedessen eine Viertelnote der Tonhöhe C. Für die nachfolgenden Noten gilt in diesem Fall die Länge, die in der letzten Erwähnung von L festgelegt wurde. Wie in der normalen Notenschrift auch bedeutet ein Punkt, daß der betreffende Ton die 1½fache Zeit der für ihn geltenden Dauer anhalten soll.

c) *Pause*
Pausen werden durch den Buchstaben P angezeigt, die nachfolgende Zahl gibt die Länge der Pause an. Somit haben wir folgende Bedeutungen zu verzeichnen:

P1	Pause für die Dauer einer ganzen Note
P2	Pause für die Dauer einer halben Note
P4	Pause für die Dauer einer Viertelnote

usw.

Testübung 8.8.1

Es ist die Zeichenkette niederzuschreiben, durch die die Tonleiter in C-Dur in Achtelnoten aufsteigend gespielt werden kann. Nach einer Pause von einer halben Note Dauer soll anschließend die gleiche Tonleiter absteigend gespielt werden.

d) *Oktave*
Zu Beginn werden alle Töne aus der Oktave 4 (Standardoktave) genommen, d.h. aus der Oktave, die über der mit dem kleinen c beginnenden Oktave liegt. Die Oktave kann mittels des Kennbuchstabens O geändert werden. Wenn z.B. in die Oktave 2 gewechselt werden soll, gibt man einfach O2 an. Nach der Festlegung der Oktave werden alle angegebenen Noten als zur betreffenden Oktave gehörig interpretiert. Diese Festlegung gilt solange, bis eine neue Oktavenfestlegung erfolgt bzw. die Oktavenfestlegung temporär überschrieben wird.

Testübung 8.8.2

Es ist die Zeichenkette niederzuschreiben, durch die die gleiche Tonleiter wie bei der Testübung 8.7.1 gespielt wird, jedoch in der Oktave 5.

e) *Tempo*
Unter dem Tempo versteht man die Geschwindigkeit, mit der ein Musikstück abgespielt wird. Das Tempo wird in Vierteltaktschlägen pro Minute gemessen. Wenn kein anderes Tempo angegeben ist, gilt das Tempo 120. Eine Änderung des Tempos bedingt die Verwendung des Kennbuchstabens T, gefolgt von einer Zahl, die das von da ab geltende neue Tempo in Vierteltaktschlägen pro Minute angibt. Soll beispielsweise das Tempo 80 eingeführt werden, muß man die Kennung T80 benutzen. Das neu eingeführte Tempo gilt solange, bis eine neue Kennung auftaucht.

Testübung 8.8.3

Es ist die Zeichenkette niederzuschreiben, durch die die gleiche Tonleiter wie bei der Testübung 8.7.1 gespielt wird, jedoch dieses Mal im Tempo 80 und anschließend im Tempo 150.

f) *Stilarten (Vortragsbezeichnungen)*
Man kann die Stilart der Musik selbst bestimmen. Zur Auswahl stehen die folgenden Stilarten:

normal → MN
legato → ML
staccato → MS

Die Kennungen für die Auswahl der Stilarten sind also MN, ML und MS. Ohne Angabe einer Kennung wird die normale Stilart herangezogen. Eine einmal ausgewählte Stilart bleibt solange in Effekt, bis sie durch die Festlegung einer anderen Stilart aufgehoben wird.

Testübung 8.8.4

Es ist die Zeichenkette niederzuschreiben, durch die die gleiche Tonleiter wie bei der Testübung 8.7.1 gespielt wird, jedoch dieses Mal in der Stilart legato und anschließend in der Stilart staccato.

Gewöhnlich bewirkt das Statement PLAY, daß BASIC für die Dauer des Abspielens der angegebenen Melodie gestoppt wird. Das PLAY-Statement kann jedoch auch im Hintergrund (Background) betrieben werden. Bei dieser Betriebsart führt BASIC während des Abspielens der Melodie über den Lautsprecher das Programm weiter aus, beginnend mit dem unmittelbar auf das PLAY-Statement folgenden Statement. Der Hintergrundbetrieb kann mittels des Statements MB gestartet werden. Die Rückkehr zur normalen Betriebsart, auch Vordergrundbetrieb (Foreground) genannt, erfolgt durch die Niederschrift des Statements MN.

Bei der Abfassung von Musikstücken wünscht man oft, daß eine gewisse Notenfolge mehrere Male gespielt wird (Refrain). In diesen Fällen braucht man die zugehörige Zeichenkette nicht mehrere Male niederzuschreiben bzw. einzugeben. Durch den Kennbuchstaben X ist es möglich, Wiederholungen anzuzeigen. Man muß dazu zunächst die zu wiederholende Zeichenkette einer Zeichenkettenvariablen, beispielsweise A$ zuweisen. Immer, wenn die Zeichenkette wiederholt werden soll, braucht man nunmehr nur noch

```
XA$;
```

codieren (siehe auch Abb. 8.47, Zeilen 10 und 20).

Aufgabengruppe 33

Man wähle ein Klavierstück aus und setze seine Notenschrift in die für das Statement PLAY benötigte Codierung um.

Antworten auf die Testübungen

8.8.1 `A$ = "L8CDEFGABCP2CBAGFEDC"`

8.8.2 `A$ = "L805CDEFGABCP2CBAGFEDC"`

8.8.3 `A$ = "L8T80CDEFGABCP2CBAGFEDC"`
`A$ = "L8T150CDEFGABCP2CBAGFEDC"`

8.8.4 `A$ = "L8MLCDEFGABCP2CBAGFEDC"`
`A$ = "L8MSCDEFGABCP2CBAGFEDC"`

9 Textverarbeitung

9.1 Einführung

Mikrocomputer sind heutzutage auf dem besten Wege, eine Bürorevolution zu verursachen. Da sie in den letzten Jahren enorm billiger und auch erheblich leichter zu gebrauchen sind, bahnen sie sich unaufhaltsam ihren Weg in alle Geschäftsangelegenheiten. Nirgends jedoch verspricht der revolutionäre Anstoß, für den die Mikrocomputer verantwortlich zeichnen, größere Erfolge als bei der Textverarbeitung. Kurz gesagt, man versteht unter einem Textprozessor eine Einrichtung, bei der die Eigenschaften der herkömmlichen Schreibmaschine mit den Fähigkeiten des Computers zum Speichern, Edieren, Wiederauffinden, Anzeigen und Drucken von Informationen gepaart sind. Es ist keine Übertreibung, wenn man prophezeit, daß die altehrwürdige Schreibmaschine bald genau so veraltet ist wie der zur Zeit unserer Väter gebaute Opel P4. Es gehört nicht viel dazu, um vorausschauend zu sagen, daß im nächsten Jahrzehnt etwa die einfachen Schreibmaschinen vollständig von ausgeklügelten, mit ständig wachsender Intelligenz versehenen Textverarbeitungseinrichtungen abgelöst werden.

Das prinzipielle Konzept eines Textprozessors (einer Textverarbeitungseinrichtung) besteht darin, daß ein Mikrocomputer als Schreibmaschine eingesetzt wird. Es wird dabei jedoch kein Papier benutzt, um die Wörter aufzuzeichnen, vielmehr werden diese im Speicher des Computers aufbewahrt. Zuerst kommt es zur Speicherung derselben im RAM, einer temporären Speicherung also. Wenn man hingegen die Texte auf Dauer aufzeichnen will, muß man sie auf einem Datenträger (Kassette, Diskette, Platte usw.) in eine Datendatei hineinstellen. Während der Eingabe kann der Text gleichzeitig auf dem Bildschirm angezeigt werden. Dieser Aspekt der Textverarbeitung erscheint uns nicht so umwälzend zu sein wie die anderen Gesichtspunkte. Die wahre Stärke eines Textprozessors zeigt sich erst dann, wenn es darum geht, die Daten eines Schriftstückes zu edieren. Unter Ausnutzung der Fähigkeiten eines Computers kann man die folgenden Aufgaben schnell und ohne große Mühe bewerkstelligen:

- Freizügiger Übergang zu jeder Stelle im Schriftstück
- Beliebiges Hinzufügen von Buchstaben, Silben, Wörtern, Ausdrücken, Sätzen und sogar von ganzen Abschnitten
- Beliebiges Entfernen (Löschen) von Textteilen (Buchstaben, Silben, Wörter usw.)
- Übertragen von Textblöcken von einer Stelle des Schriftstückes zu einer anderen Stelle
- Einfügen von aktuellen Informationen (standardisierte Textabschnitte wie Beschreibungen, Vertragsbestimmungen, Geschäftsbedingungen, Adressen usw.) aus anderen Datendateien, z. B. Adressen aus einer Kundenadreßdatei
- Selektive Änderung bestimmter Textteile, z. B. des Wortes „Hans" in das Wort „Fritz" überall dort, wo es im Text auftritt
- Drucken des Inhaltes einer Datei, abgestimmt auf das verlangte Format

In diesem Kapitel wollen wir über die Eigenschaften der verschiedenen Textprozessorprogramme[1] sprechen, die man für den IBM Personalcomputer käuflich erwerben kann. Um dem

---

[1] In den USA spricht man von „Wortverarbeitung" (Word Processing) anstelle von Textverarbeitung. Ein Textprozessor wird deshalb oft auch als „Wortprozessor" (Word Processor) bezeichnet.

Leser ein passendes Gefühl für die Textverarbeitung zu vermitteln, wollen wir zusätzlich einen rudimentären Textprozessor zusammenbauen, der zur Vorbereitung von Briefen, Schul- und Studienarbeiten, Denkschriften und anderen Schriftstücken verwendet werden kann.

9.2 Die Benutzung des Computers zur Textverarbeitung

Unter einem Textverarbeitungssystem versteht man ein Computerprogramm zur Schaffung, Speicherung und zur Überarbeitung sowie zum Drucken von Texten.

Auf der allertiefsten Stufe kann man ein Textverarbeitungssystem genau so wie eine Schreibmaschine benutzen. Nehmen wir einmal an, daß wir ein Schriftstück vorbereiten wollen. Wir schalten den Computer ein und lassen das Textverarbeitungsprogramm laufen. Am Anfang erkundigt sich das Programm, welche Art von Arbeit der Benutzer durchzuführen beabsichtigt. Folgende Möglichkeiten werden angeboten:

- Eingabe des Textes für ein neues Schriftstück
- Edieren eines bestehenden Schriftstückes
- Sicherstellung eines Schriftstückes auf einer Diskette
- Drucken eines Schriftstückes

Für unsere geplante Arbeit müßten wir die erste Möglichkeit auswählen. Als nächstes hätten wir die verschiedenen Formatparameter zu nennen, mit denen der Textprozessor arbeiten soll. Dazu zählen:

- Zeilenlänge
- Zeichendichte (Anzahl der Zeichen pro Zoll)
- Anzahl der Zeilen pro Seite
- Umfang des Papiertransportes nach jeder Textzeile (Anzahl der Leerzeilen zwischen den Textzeilen)
  usw.

Danach würden wir den Text des Schriftstückes genau so eingeben, wie wenn wir ihn auf der Schreibmaschine schreiben würden. Es sind jedoch einige wesentliche Ausnahmen zu beachten. Zunächst sei vermerkt, daß wir uns nicht mit den Wagenrückläufen herumplagen müssen. Der Textprozessor kümmert sich selbst um die Anordnung der Zeilen. Er nimmt den Text, den wir eingegeben haben, auf, entscheidet selbst, wieviel davon auf eine Zeile geht und zeigt die Zeile an. Die über eine Zeile hinausgehenden Textteile werden zurückgehalten und automatisch für die Folgezeile aufbewahrt. Die einzige Funktion, die für den Schreibkopfrücklauf noch verbleibt, ist, daß wir durch seine Betätigung anzeigen, daß wir definitiv den Übergang auf eine neue Zeile wünschen, wie es am Ende eines Paragraphen, eines Abschnittes usw. der Fall sein wird.

Ein zweiter Vorteil des Textprozessors zeigt sich bei der Fehlerkorrektur. Wir brauchen hierzu nur den Positionsanzeiger (Cursor) auf die fehlerhafte Stelle zu führen und die entsprechenden Buchstaben oder Wörter zu löschen (sei es durch Betätigung einer Taste oder sei es durch Eingabe eines Befehles) oder mit den richtigen Buchstaben bzw. Wörtern zu überschreiben. Natürlich wird ein solches Vorgehen oft die bisherige Zeilengliederung zerstören, einige Zeilen werden zu lang, andere zu kurz werden. Aber darum braucht man sich wahrlich keine Sorgen zu machen, denn Textprozessoren enthalten in der Regel recht einfache Befeh-

le, durch die eine Neuorientierung der Zeilen erreicht werden kann. Die Abstimmung wird dabei stets entsprechend dem vorgesehenen Format erfolgen.

Kennzeichnend für Textprozessoren ist ebenfalls, daß sie Befehle aufweisen, die es ermöglichen, auf einfache Art und Weise den Text eines Schriftstückes nach einem bestimmten Paragraphen zu durchsuchen. Einige Textprozessoren gestatten sogar die Markierung gewisser Textstellen. Derartige Markierungen wiederum können schnell „aufgeschlagen" werden, ohne daß erst ein mühseliges visuelles Suchen nach ihnen erfolgen muß.

Wenn ein so geschaffenes und bearbeitetes Schriftstück seine endgültige Fassung bekommen hat, zu unserer Zufriedenheit ausfällt, kann dem Textprozessor ein Befehl gegeben werden, dessen Ausführung die Sicherstellung des Schriftstückes in einer Datei auf einer Diskette zur Folge hat. Später kann man dann das Schriftstück durch Bezugnahme auf den Dateinamen zurückrufen und neuen Erfordernissen entsprechend bearbeiten und damit ändern; nicht nur ganze Paragraphen können entfernt oder hinzugefügt werden, sondern auch Textstellen innerhalb von Paragraphen. In der Regel sind Textprozessoren auch mit Befehlen ausgerüstet, durch die Operationen mit ganzen Textblöcken ausgelöst werden können. Nach der Markierung eines Textblockes kann man denselben entfernen, kopieren oder als ganzes an eine andere Stelle des Schriftstückes übertragen. Man kann auch in ein gerade bearbeitetes Schriftstück andere Schriftstücke oder Teile anderer Schriftstücke aufnehmen, d.h. einfügen. Das erweist sich als sehr nützlich, wenn man ein Schriftstück um bereits existierende Zusammenfassungen, Überblicke, Adressen usw. ergänzen muß. Durch die schon erwähnten Blockoperationen können darüber hinaus die Einfügungen geändert und somit den speziellen Bedürfnissen des in Bearbeitung befindlichen Schriftstückes angepaßt werden.

Man braucht ein Schriftstück nicht in einer einzigen Sitzung fertigzustellen; man kann die Arbeit an einem Dokument auf soviel Sitzungen verteilen, wie man will. Wenn auf einer Diskette schließlich das Schriftstück in seiner endgültigen Fassung gespeichert ist, kann man es dann drucken lassen. Über den Drucker wird damit eine fehlerfreie Schlußkopie des Dokumentes erzeugt.

Die soeben geschilderten Möglichkeiten stellen bereits einen erheblichen Fortschritt gegenüber der Schreibarbeit mit den konventionellen Schreibmaschinen dar, doch die charakteristischen Textprozessoren können noch weitaus mehr. Natürlich sind solche Erweiterungen abhängig von den jeweiligen Textprozessoren; jeder muß sich also genau und sorgfältig überlegen, welchen Textprozessor er erwerben will, er muß Vor- und Nachteile gegenseitig abwägen. Einige der soeben angedeuteten nützlichen Erweiterungen wollen wir jetzt vorstellen.

1. *Globales (umfassendes) Durchsuchen und Ersetzen*
   Nehmen wir einmal an, daß das zu erarbeitende Schriftstück einen Vorschlag beinhaltet, der einem bestimmten Kunden offeriert werden soll. Nehmen wir ferner an, daß dieser Vorschlag zu einem späteren Zeitpunkt auch anderen Firmen unterbreitet werden soll. Im Originalvorschlag sind sicherlich eine Reihe von Referenzen und Bemerkungen enthalten, die nur für den ursprünglichen Empfänger bedeutungsvoll sind; die neuen Empfänger sollen und dürfen davon aber nichts wissen. Wenn nun eine Einrichtung zum globalen Durchsuchen und Ersetzen in den Textprozessor eingebaut ist, kann man jedes Auftauchen der betreffenden Phrasen schnell und sicher feststellen und sie durch andere Phrasen ersetzen. So könnte man z.B. jede Textstelle, die „Energie durch Erdgas" lautet, durch „Sonnenenergie" ersetzen. Das globale Durchsuchen und Ersetzen kann sogar noch sinnvoller gestaltet sein. In einigen Systemen kann der Textprozessor so durchdacht sein, daß man ihm vorher vorgeben kann, zunächst den Benutzer zu fragen, ob eine Änderung vorgenommen werden soll. Der Benutzer kann somit bestimmen, ob alle Textstellen oder nur einige, von ihm ausgewählte, geändert werden sollen. Eine ande-

re Variation findet man ebenfalls. Man kann dabei den betreffenden Textprozessor instruieren, daß jede Großschreibung ersetzt werden soll.

2. *Zentrierung*
   Nach Eingabe eines Textstückes kann dieses durch einen simplen Befehl auf einer Zeile zentriert werden. Nehmen wir dazu einmal an, daß das eingegebene Textstück 30 Zeichen umfaßt. Die Zeilenlänge betrage 80 Stellen. Dann hat das Zentrieren dieser Zeile zur Folge, daß das Textstück um die Zeilenmitte herum angeordnet wird, d.h. die ersten und letzten 25 Stellen dieser Zeile bleiben frei.

3. *Fettschrift*
   Man kann bestimmte Wörter durch Fettschrift hervorheben lassen; diese erscheinen dann dunkler.

4. *Unterstreichung*
   Man kann Textstellen unterstreichen lassen.

5. *Tief- und Hochschreibung*
   Durch entsprechende Befehle kann man veranlassen, daß Zeichen tiefer, wie beispielsweise bei a_1, bzw. höher, wie bei a^2, gestellt werden. Diese Möglichkeit ist von entscheidender Bedeutung bei wissenschaftlichen, beginnend bei mathematisch-physikalischen, Schriftstücken (Dokumenten).

6. *Ausrichtung*
   Man kann die betreffenden Textprozessoren instruieren, daß die einzelnen Zeilen nach rechts ausgerichtet werden, so daß rechts die Zeilen genau so an einer scharfen Begrenzung (Zeilenende) enden wie links. Oft wird freilich das nur dadurch ermöglicht, daß ein Drucker angeschlossen ist, der mit kleineren Schreibschritten drucken kann als es die normale Zeichenbreite erfordert.

7. *Berichtigung von Rechtschreibfehlern*
   Augenblicklich existieren bereits eine Reihe von Programmen für die Korrektur von Rechtschreibfehlern; diese vergleichen die Wörter im Schriftstück mit einem (20000 bis 70000 umfassenden) Wörterbuch. Wenn nun ein solches Korrekturprogramm keine Übereinstimmung mit einem Wort des Wörterbuches findet, fragt es beim Benutzer zurück, ob das betreffende Wort richtig buchstabiert ist und gibt ihm außerdem die Chance, das Wort in das Wörterbuch neu aufzunehmen. Auf diese Weise kann die Ausgabe eines Textprozessors vom Computer bereits vorgeprüft gelesen werden.

Als dieser Abschnitt von uns geschrieben wurde, gab es bereits mehr als ein Dutzend Textverarbeitungsprogramme (Textprozessoren) für den IBM Personalcomputer; bei praktisch fast allen sind die gefragtesten Forderungen verwirklicht. Man sollte in die Softwarebibliothek des eigenen Personalcomputers möglichst rasch einen geeigneten Textprozessor aufnehmen. Tut man es nicht, so vergibt man sich eine der nützlichsten Anwendungen des Personalcomputers.

9.3 Zusammenstellung eines eigenen primitiven Textprozessors

Es scheint uns wirklich unzweckmäßig zu sein, einen eigenen Textprozessor zu schreiben bzw. überhaupt zu planen. Man muß sich ständig darüber im klaren sein, daß ein solches Programm zu umfangreich und zu verwickelt sein wird. Wenn man es darüber hinaus in BASIC verfassen würde, würde dieses Programm dazu tendieren, zu langsam abzulaufen. Ein wirklich effizienter Textprozessor wird beinahe immer in einer maschinenorientierten Sprache des jeweiligen Computers geschrieben. – Nichtsdestoweniger wollen wir das ignorieren, was wir soeben gesagt haben. Es kommt uns darauf an, den Leser mit einigen der Vorzüge bekannt zu machen, die ein Textprozessor nun einmal bietet. Wir wollen uns also selbst einen einfachen primitiven Textprozessor schreiben.

Unser Textprozessor soll zeilenorientiert arbeiten: Man soll eine Textzeile genau so eintippen können wie bei Benutzung einer Schreibmaschine. Nach Eingabe einer Zeile soll ein Schreibkopfrücklauf durch Drücken der Eingabetaste veranlaßt werden. Die j-te Zeile soll im Element A$(J) eines Zeichenkettenbereiches gespeichert werden. Wir wollen annehmen, daß 32k[1] Stellen des Speichers für die Speicherung des Schriftstückes zur Verfügung stehen. Dieser Speicherraum erlaubt uns, ein Dokument zu speichern und zu edieren, das aus nahezu fünf Seiten (bei doppeltem Abstand der Zeilen) bestehen kann. Unser Textprozessor soll mit fünf verschiedenen Arbeitsarten (Modi) ausgestattet werden. Im ersten Modus soll die Texteingabe möglich sein.

Diese Funktion soll genau so funktionieren wie bei einer Schreibmaschine. Zu Beginn jeder Zeile zeigt der Prozessor ein Fragezeichen an, um damit zu signalisieren, daß nunmehr eine Zeile eingegeben werden kann. Im Anschluß an das Fragezeichen kann jetzt Text eingetippt werden. Die Eingabe des zur fraglichen Zeile gehörenden Textes muß durch das Anschlagen der Eingabetaste abgeschlossen werden. Um dem Textprozessor mitzuteilen, daß man keine weitere Zeile eingeben will, hat man das Prozentzeichen einzutippen und sofort anschließend die Eingabetaste zu betätigen. Der Umfang des Schriftstückes ist im Programm auf 150 Zeilen beschränkt (siehe DIM-Statement auf der Zeile mit der Zeilennummer 120).

Ein zweiter Modus gestattet uns die Sicherstellung eines Schriftstückes in einer Datendatei, deren Name vom Programm angefordert wird. Als erstes Datenelement wird in diese Datei die Anzahl der Zeilen geschrieben, die das Schriftstück besitzt. Im Programm ist diese Anzahl der Variablen L zugewiesen. Danach kommen die Textzeilen, die als Zeichenkettenkonstanten den Bereichselementen A$(1),A$(2),A$(3),···,A$(L) zugewiesen sind.

Ein dritter Modus läßt das Drucken des Schriftstückes in einer Entwurfsversion zu. Hierbei geht den einzelnen Zeilen des Dokumentes die Zeilennummer voraus. Dadurch ist es möglich, fehlerhafte Zeilen leichter zu identifizieren, was für das Edieren wichtig ist. Es sei ausdrücklich darauf hingewiesen, daß das Drucken eines Schriftstückes voraussetzt, daß es zuvor in eine Datendatei weggespeichert wurde.

Ein vierter Modus ermöglicht das Edieren eines Schriftstückes. Die Fehlerkorrektur ist so organisiert, daß man fehlerhafte Zeilen über ihre Zeilennummer heranholen muß. Sie werden dann auf dem Bildschirm angezeigt, und man hat sie daraufhin neu einzugeben. Um das Edieren beenden zu können, ist es notwendig, anstelle einer Textzeile das Prozentzeichen einzugeben. Nach dessen Eintippen ist natürlich die Eingabetaste zu drücken. Die Erkennung des Prozentzeichens führt zum Programmanfang zurück und verschafft damit dem Benutzer die Möglichkeit, andere Arbeiten (Sicherstellung, Drucken) mit dem gleichen Schriftstück vorzunehmen. Nach Beendigung des Edierens sollte man als nächste Aktion die Sicherstellung des edierten Dokuments in einer Datendatei (zweiter beschriebener Modus) vornehmen. Der fünfte und letzte Bearbeitungsmodus erlaubt schließlich das Drucken des fertiggestellten Schriftstückes.

Schließlich kann aus dem Hauptmenü noch eine letzte Funktion angesteuert werden. Nachdem alle Bearbeitungsaufgaben erledigt sind, muß der Textprozessor verlassen werden können. Dafür ist die Kennung BE geschaffen worden.

Nach Beginn des Programmablaufes unseres bescheidenen Textprozessors erscheint ein Haupt- oder Primärmenü auf dem Bildschirm. Aus diesem ist ersichtlich, welche Buchstaben-

[1] Unter 1k Speicherstellen versteht man 1 024 Speicherstellen.

kombinationen einzugeben sind, um die gewünschte Arbeitsart auswählen zu können (siehe Abb. 9.1).

```
TEXT-VERARBEITUNGSPROGRAMM
ES IST EINE DER FOLGENDEN ARBEITSARTEN AUSZUWAEHLEN:

   TEXTEINGABE (TE)
   AUSGABE DES ENTWURFS EINES SCHRIFTSTUECKES (AE)
   AUSGABE EINES SCHRIFTSTUECKES (AS)
   SICHERSTELLUNG IN EINER DATEI (SI)
   EDIEREN (ED)
   BEENDIGUNG DER TEXTVERARBEITUNG (BE)
```

Abb. 9.1 Haupt- oder Primärmenü

Als Erwiderung auf diese Anzeige ist eine der folgenden Kennungen einzugeben:

TE, AE, AS, SI, ED, BE

Wenn die Texteingabe (TE) ausgewählt wird, wird zunächst der Bildschirm gelöscht; danach kann man mit der Eingabe der Textzeilen beginnen. Bei den anderen Arbeitsarten kommt es durch das Programm zu Rückfragen an den Benutzer, die diesem mitteilen, was er zu tun hat. Das gesamte Programm ist in der Abb. 9.2 aufgelistet.

Man sollte dieses Programm ruhig einmal zum Schreiben einiger Briefe einsetzen. Schon dabei wird man erkennen, welcher beachtliche Fortschritt gegenüber dem Briefeschreiben mit einer Schreibmaschine eingetreten ist. Darüber hinaus wird sicher das unbezähmbare Verlangen nach leistungsfähigeren Textprozessoren entstehen, die mit den im vorigen Abschnitt beschriebenen Funktionen ausgerüstet sind.

Aufgabengruppe 34

1. Der Textprozessor von Abb. 9.2 ist so zu modifizieren, daß er auch zuerst zu Beginn die Eingabe der Zeilenlänge ermöglicht. Erinnern wir uns in diesem Zusammenhang, daß nicht mehr als 80 Zeichen auf einer Bildschirmzeile angezeigt werden können. Die Zeichenkettenvariablen können jedoch bis zu 255 Zeichen enthalten.

2. Der Textprozessor (Abb. 9.2) ist so abzuändern, daß eine Zeile verlängert werden kann. Diese Modifikation soll es ermöglichen, daß eine korrigierte Zeile erforderlichenfalls in die nächste Textzeile überlaufen kann. Das Programm sollte daran anschließend alle Nachfolgezeilen neu anordnen, um die Hinzufügung auszugleichen.

3. Der Textprozessor (Abb. 9.2) ist so zu erweitern, daß er auch das Löschen von Textstellen in einer Zeile ermöglicht. In Reflektion auf eine derartige Zeilenverkürzung sollten die Folgezeilen entsprechend neu angeordnet werden. Begonnen werden soll damit, daß die Zeile, aus der ein Textstück entfernt wurde, wieder aufgefüllt wird.

Abb. 9.2 Textprozessor (1. Teil) ▷

```
100  'Ausgabe des Haupt- oder Primaermenues
110  CLS
120  DIM A$(150)
130  PRINT "TEXT-VERARBEITUNGSPROGRAMM"
140  PRINT "ES IST EINE DER FOLGENDEN ARBEITSARTEN AUSZUWAEHLEN:"
150  PRINT
160  PRINT "   TEXTEINGABE (TE)"
170  PRINT "   AUSGABE DES ENTWURFS EINES SCHRIFTSTUECKES (AE)"
180  PRINT "   AUSGABE EINES SCHRIFTSTUECKES (AS)"
190  PRINT "   SICHERSTELLUNG IN EINER DATEI (SI)"
200  PRINT "   EDIEREN (ED)"
210  PRINT "   BEENDIGUNG DER TEXTVERARBEITUNG (BE)"
220  INPUT X$
230  IF  X$ = "TE"  THEN 310
240  IF  X$ = "AE"  THEN 400
250  IF  X$ = "AS"  THEN 500
260  IF  X$ = "SI"  THEN 600
270  IF  X$ = "ED"  THEN 690
280  IF  X$ = "BE"  THEN 840
290  GOTO 140:              'Fehlerhafte Eingabe ---> Wiederholung
300  'Texteingabe (TE)
310  L = 1
320  PRINT "NACH EINGABE JEDER TEXTZEILE EINGABETASTE DRUECKEN"
330  LINE INPUT "? "; A$(L)
340  IF  A$(L) = "%"  THEN   L = L - 1:     GOTO 140
350  L = L + 1
360  IF  L <= 150  THEN 330
370  PRINT "SCHRIFTSTUECK ZU UMFANGREICH"
380  GOTO 140               'Neubeginn (Rueckkehr zum Primaermenue)
390  'Ausgabe eines in Bearbeitung befindlichen Schriftstueckes
      (Entwurfsausgabe)
400  INPUT "BEZEICHNUNG DES SCHRIFTSTUECKES (DATEINAME): "; Y$
410  OPEN Y$  FOR INPUT AS #1
420  INPUT #1, L
430  FOR  K = 1  TO  L
440    INPUT #1, A$(K)
450    LPRINT K; " --->"; TAB(10) A$(K)
460  NEXT K
470  CLOSE #1
480  GOTO 140               'Rueckkehr zum Primaermenue
490 'Ausgabe der Endfassung eines Schriftstueckes
500  INPUT "BEZEICHNUNG DES SCHRIFTSTUECKES (DATEINAME): "; Y$
510  OPEN Y$  FOR INPUT AS #1
520  INPUT #1, L
530  FOR  K = 1  TO  L
540    INPUT #1, A$(K)
550    LPRINT A$(K)
560  NEXT K
570  CLOSE #1
580  GOTO 140               'Rueckkehr zum Primaermenue
590  'Sicherstellung des Schriftstueckes in einer Datei
600  INPUT "BEZEICHNUNG DES SCHRIFTSTUECKES (DATEINAME): "; Y$
610  OPEN Y$  FOR OUTPUT AS #1
620  WRITE #1, L
630  FOR  K = 1  TO  L
640    WRITE #1, A$(K)
650  NEXT K
660  CLOSE #1
670  GOTO 140               'Rueckkehr zum Primaermenue
```

```
680  'Edieren eines Schriftstueckes
690  INPUT "BEZEICHNUNG DES SCHRIFTSTUECKES (DATEINAME): "; Y$
700  OPEN Y$  FOR INPUT AS #1
710  INPUT #1, L
720  FOR  K = 1  TO  L
730    INPUT #1, A$(K)
740  NEXT K
750  INPUT "NUMMER DER AUFZUBEREITENDEN ZEILE: "; NR
760  CLS
770  PRINT A$(NR)
780  PRINT "BITTE KORRIGIERTE ZEILE EINGEBEN: "
790  LINE INPUT  A$(NR)
800  IF  A$(NR) <> "%"  THEN 750
810  CLOSE #1
820  GOTO 140              'Rueckkehr zum Primaermenue
830  'Beendigung der Textverarbeitung (Ausgang aus dem Programm)
840  CLS
850  PRINT "TEXTVERARBEITUNG BEENDET"
860  END
```

Abb. 9.2 Textprozessor (2. Teil)

10 Zusätzliche Programmierhilfen

In diesem Kapitel wollen wir uns mit vier zusätzlichen Programmierhilfen beschäftigen. Ihre Verwendung beim Programmieren läßt einerseits einfachere und wirkungsvollere Programme entstehen, andererseits können wir aber auch Probleme angehen, die ohne diese Hilfe kaum oder nur schwer lösbar sind.

10.1 Funktion INKEY$

10.1.1 Der Puffer für die Tastatur

Viele Programme hängen von der Eingabe ab, die der Bediener bzw. Benutzer vornimmt. Wir haben gelernt, für solche Eingaben die Statements INPUT und LINE INPUT vorzusehen. Wenn der Programmablauf auf diese beiden Statements stößt, so pausiert er und wartet die Eingabe ab. Die Ausführung des Programmes wird erst dann fortgeführt, wenn eine gültige Eingabe abgeschlossen ist (Drücken der Eingabetaste). Die eingebaute Funktion INKEY$ liefert uns eine alternative Methode, über die Tastatur eingegebene Zeichen zu lesen.

Beim Anschlagen einer Taste auf der Tastatur unterbricht nämlich BASIC seine momentane Arbeit, um den ASCII-Code des eingegebenen Zeichens auf einen entsprechenden Platz in einer reservierten Speichersektion zu übertragen. Diese besondere Speichersektion wird Tastaturpuffer genannt. Der Tastaturpuffer hat Platz genug, um eine Reihe von ASCII-Codes aufnehmen zu können. Der Prozeß der Aufzeichnung der eingegebenen Zeichen in diesen Puffer schreitet zügig voran, so daß der Benutzer sich kaum vergegenwärtigt, daß ein solcher Puffer überhaupt vorhanden ist. Wenn man z. B. Programmzeilen eingibt, liest BASIC fortwährend aus dem Tastaturpuffer und zeigt die zu den betreffenden ASCII-Codes gehörenden Zeichen auf dem Bildschirm an. In gleicher Weise liest das INPUT-Statement die Zeichen aus dem Puffer und zeigt die entsprechenden Zeichen auf dem Bildschirm an. Ein durch das Drücken der Eingabetaste verursachter Cursorrücklauf (Schreibkopfrücklauf) sagt dem INPUT-Statement, daß es mit dem Einlesen aufhören soll.

Sowie die Zeichen aus dem Puffer gelesen sind, wird der von ihnen besetzte Platz wieder für eine erneute Benutzung freigegeben. Ist allerdings durch fortlaufende Eingabe der Puffer gefüllt, und versucht man, weiterhin Zeichen einzugeben, so ertönt ein Piepton aus dem Lautsprecher. Dies dient zur Information des Benutzers; die dennoch weiterhin eingegebenen Zeichen gehen solange verloren, bis der Puffer gelesen ist.

Es sei ausdrücklich darauf hingewiesen, daß man über die Tastatur während eines Programmablaufes eingeben kann. Auch wenn BASIC mit der Ausführung eines Programmes beschäftigt ist, hält es an, um eingegebene Zeichen in den Tastaturpuffer zu stellen, und nimmt danach die Programmausführung wieder auf. Wenn der Puffer anschließend gelesen wird, werden selbstverständlich die Zeichen in der Reihenfolge gelesen, in der sie vom Benutzer eingetippt wurden. Auf diese Art kann man im voraus eingeben, was vom Programm später als Eingabe benötigt wird.

10.1.2 Gebrauch von INKEY$

Die eingefügte Funktion INKEY$ ermöglicht das Lesen eines Zeichens aus dem Tastaturpuffer. Wenn das Programm auf diese Funktion stößt, wird das Zeichen aus dem Puffer geholt, das am längsten in ihm gespeichert ist; durch INKEY$ wird es als Zeichenkette zur Verfügung

gestellt. Nach dem Lesen wird das Zeichen im Puffer gelöscht. Wenn der Puffer leer ist, wird als Wert von INKEY$ die leere Zeichenkette zurückgegeben.

Die Funktion INKEY$ kann man recht vielfältig einsetzen. Man nehme beispielsweise einmal an, daß man wünscht, daß der Programmablauf solange pausiert, bis eine Taste angeschlagen wird. Durch das folgende Statement kann das bewerkstelligt werden:

```
100  IF  INKEY$ = ""  THEN 100
```

In diesem Fall würde das Programm ständig den Inhalt des Tastaturpuffers prüfen. Wenn kein Zeichen eingelesen werden kann, weil der Puffer leer ist, wird die Prüfung wiederholt. Das geschieht solange, bis eine Taste angeschlagen wird.

Dabei gilt es eines zu beachten. Wir haben die Wirkungsweise der eingefügten Funktion INKEY$ anhand des Tastaturpuffers so genau erklärt, daß man sicherlich den nachstehenden Gedankengang verfolgen kann. Wenn der Tastaturpuffer nicht leer ist, sorgt eine Bezugnahme auf INKEY$ dafür, daß ein Zeichen aus dem Puffer entfernt wird. Wenn man nun ein zweites Mal die Funktion INKEY$ aufruft, wird der Tastaturpuffer erneut angesprochen und der bei der ersten Bezugnahme gelesene Wert geht verloren. Man muß deshalb den durch INKEY$ gelesenen Wert einer Zeichenkettenvariablen zuweisen, wenn man ihn noch verwenden muß und andererseits INKEY$ erneut aufrufen muß. Das könnte beispielsweise durch eine Ergibtanweisung der Form

```
10  A$ = INKEY$
```

geschehen.

Aufgabengruppe 35

Man nehme einmal an, daß der Tastaturpuffer leer ist. Nunmehr tippe man der Reihe nach die folgenden Buchstaben ein:

```
A F c
```

1. Welchen Wert wird INKEY$ beim ersten Aufruf annehmen?
2. Angenommen, man habe die Aufgabe 1. erledigt. Anschließend wird das folgende Statement ausgeführt:

   ```
   120  IF  INKEY$ <> ""  THEN PRINT INKEY$
   ```

   Welcher Buchstabe wird durch dieses Statement auf dem Bildschirm angezeigt?
3. Es ist ein Programm zu schreiben, das den Inhalt des Tastaturpuffers prüft und die Zeichen auf dem Bildschirm anzeigt, deren zugehörige Tasten angeschlagen wurden. Die Anzeige sollte in einer einzigen Zeile erfolgen; zwischen aufeinanderfolgenden Zeichen sollen dabei keine Leerzeichen verbleiben.

10.2 Die Funktionstasten und das Auffangen von Ereignissen

Die zehn Funktionstasten befinden sich auf der linken Seite der Tastatur des Personalcomputers. Sie sind mit

```
F1, F2, F3, ···, F10
```

beschriftet.

10.2.1 Die Benutzung der Funktionstasten durch den Benutzer

Die Funktionen, die den Funktionstasten zugeordnet sind, können durch den Benutzer festgelegt werden. Man bezeichnet diesen Vorgang oft als „Definition der Funktionstasten durch den Benutzer".

Jeder Funktionstaste kann eine Zeichenkettenkonstante zugewiesen werden, die aus bis zu 15 Zeichen bestehen darf. Wenn dann eine so definierte Funktionstaste niedergedrückt wird, dient die betreffende Zeichenkette als Eingabe für BASIC. Auf diese Weise kann man immer wiederkehrende (Standard)-Eingaben auf das Anschlagen einzelner Tasten reduzieren und damit Eingabefehler eliminieren. Nehmen wir z.B. einmal an, daß ein INPUT-Statement eine der folgenden Eingaben erwartet:

```
HOCH<R>
TIEF<R>
MITTEL<R>
```

Unter <R> sei hier das Cursorrücklauf-(Schreibkopfrücklauf-)Zeichen verstanden. Man könnte dann ohne weiteres den Funktionstasten F1 bis F3 die folgenden Zeichenkettenkonstanten zuweisen:

```
F1 ← HOCH<R>
F2 ← TIEF<R>
F3 ← MITTEL<R>
```

Ein Anschlag der Taste F1 z.B. wirkt nunmehr genau so, als ob die Zeichenfolge HOCH, gefolgt vom Drücken der Eingabetaste, eingegeben wird.

Das Setzen der Funktionstasten, d.h. die Zuweisung von Zeichenkettenkonstanten kann entweder im Befehlsmodus oder im Ausführungsmodus erfolgen. Um <kette> zur Funktionstaste n zuzuweisen, ist das Statement

```
KEY n,<kette>
```

zu benutzen. Nehmen wir z.B. einmal an, daß wir die Funktionstaste F1 mit der Kette LIST<R> belegen wollen, so können wir das mit Hilfe des Statements

```
10 KEY 1, "LIST"+CHR$(13)
```

bewerkstelligen. Danach wird immer, wenn wir die Taste F1 anschlagen, die genannte Kette von BASIC eingelesen. Geschieht das insbesondere dann, wenn man sich im Sofortmodus

befindet, verursacht die Eingabe dieser Kette die Auflistung des gegenwärtigen Programmes. Man hat damit in der Tat die Funktionstaste F1 für eine bestimmte Anwendung gebrauchsfähig gemacht. In gleicher Weise kann man natürlich den anderen Funktionstasten ebenfalls Befehle oder Folgen von Tastenanschlägen unterlegen, die bei den eigenen Arbeiten am PC häufig vorkommen.

Um die augenblicklichen Belegungen der Funktionstasten kennenzulernen, braucht man nur den Befehl

```
KEY LIST
```

einzugeben; die Anzeige erfolgt hier wie üblich im normalen Anzeigebereich (Zeilen 1 bis 24) auf dem Bildschirm.

Beim Schreiben oder beim Ausführen von Programmen ist es oft zweckmäßig, wenn zur besseren Erinnerung die Belegung der Funktionstasten jederzeit auf dem Bildschirm angezeigt wird. Das kann durch Eingabe des Befehles

```
KEY ON
```

bewerkstelligt werden. Die ersten sechs Zeichen der Zeichenkettenkonstanten, mit denen die Funktionstasten belegt sind, werden dann in der Zeile 25 auf dem Bildschirm angezeigt. Bei einer vorhandenen Zeilenlänge von nur 40 Stellen ist die Anzeige beschränkt auf die Belegung der ersten fünf Funktionstasten. Die Anzeige über die Belegung der Funktionstasten kann durch den Befehl

```
KEY OFF
```

ausgeschaltet werden.

Testübung 10.2.1

a) Es sind die Befehle zu nennen, durch die die Funktionstasten F1, F2 und F3 mit den folgenden Zeichenkettenkonstanten belegt werden:

F1 ← "ADDITION"
F2 ← "SUBTRAKTION"
F3 ← "MULTIPLIKATION"

Alle übrigen Funktionstasten sind zu deaktivieren.

b) In der Zeile 25 ist auf dem Bildschirm die Belegung der Funktionstasten sichtbar zu machen. Wie lautet der entsprechende Befehl?

10.2.2 Das Auffangen von Ereignissen

Das Auffangen von Ereignissen kann nur im „Erweiterten BASIC" erfolgen, das bekanntlich durch Eingabe von BASICA nach der DOS-Systemmeldung A> zur Verfügung steht.

Wir haben in früheren Kapiteln und im jetzigen Abschnitt erörtert, wie die Dateneingabe mittels der Statements

```
INPUT,  LINE INPUT  und  INKEY$
```

erfolgt. Diese Eingabemöglichkeiten haben eins gemeinsam: Die Logik des Programmes ist maßgebend dafür, wann nach einer Eingabe angefragt wird. Eine davon sehr verschiedene Form der Eingabe kann nun bei Benutzung der Funktionstasten realisiert werden.

Nehmen wir einmal an, daß das Programm auf die Funktionstaste F1 achtgeben soll. Unmittelbar nach dem Anschlagen dieser Taste soll das Programm zur Unterroutine verzweigen, die auf der Zeile mit der Zeilennummer 1000 beginnt. Um diese Absicht auszuführen, muß zuerst die Einrichtung zum Auffangen von Ereignissen für die Taste F1 eingeschaltet werden. Hierzu dient das folgende Statement:

```
10  KEY(1) ON
```

Durch diese Anweisung wird dem Programm gesagt, daß die Taste F1 nach Ausführung jeder Anweisung auf ihren Zustand hin überprüft werden soll. Nach der KEY-Anweisung müssen wir in unser Programm eine zweite Anweisung aufnehmen, die dafür sorgt, daß nach dem Drükken von F1 der Übergang zu der Unterroutine erfolgen soll, die auf der Zeile mit der Zeilennummer 1000 beginnt. Durch die folgende Anweisung teilen wir das dem Programm mit:

```
20  ON KEY(1)  GOSUB 1000
```

Das ablaufende Programm inspiziert nun nach der Ausführung jedes Statements den Tastaturpuffer. Wenn es dabei feststellt, daß F1 gedrückt wurde, wird

```
GOSUB 1000
```

ausgeführt. Man kann das Auffangen von Ereignissen in ein Menü münden lassen, wie es das folgende Beispiel für mehrere Funktionstasten zeigt.

Beispiel 1:

Man schreibe ein Programm, das die Rechenfähigkeiten anhand der Addition, Subtraktion und Multiplikation zweistelliger ganzer Zahlen testet. Dabei soll der Benutzer die Rechenoperationen mittels der Tasten F1, F2 und F3 auswählen können. Über die Taste F4 soll das Programmende erreicht werden können.

Bei der Erarbeitung der Lösung gehen wir davon aus, daß wir vier Unterprogramme benötigen, je eins für die drei Rechenoperationen (Addition, Subtraktion und Multiplikation) und eins für das Programmende. Diese vier Unterprogramme wollen wir aus Gründen der Übersichtlichkeit auf den Zeilen mit den Zeilennummern 1000, 2000, 3000 bzw. 4000 beginnen lassen. In dem in der Abb. 10.1 vorgelegten Lösungsprogramm sind für uns am interessantesten die Statements auf den Zeilen mit den Zeilennummern 10 bis 230. Wir wollen sie infolgedessen kurz besprechen, wobei wir das wichtigste ausdrücklich betonen wollen. Zunächst wird der Bildschirm gelöscht (Zeile 20). Anschließend werden die Zeichenkettenkonstanten definiert (Zeilen 30 bis 60), die mit den Funktionstasten F1, F2, F3 und F4 verknüpft werden sollen, nämlich

- "ADDITION"
- "SUBTRAKTION"
- "MULTIPLIKATION"
- "ENDE"

Den übrigen Funktionstasten (F5 bis F10) wird eine leere Zeichenkette zugewiesen, d. h. sie werden deaktiviert (Zeilen 70 bis 120). Das Auffangen von Ereignissen für die ersten vier Funktionstasten wird durch die

```
10   'Initialisieren der Funktionstasten
20   CLS
30   KEY 1, "ADDITION"
40   KEY 2, "SUBTRAKTION"
50   KEY 3, "MULTIPLIKATION"
60   KEY 4, "ENDE"
70   KEY 5, ""
80   KEY 6, ""
90   KEY 7, ""
100  KEY 8, ""
110  KEY 9, ""
120  KEY 10,""
130  KEY ON
140  ON KEY(1)  GOSUB 1000
150  ON KEY(2)  GOSUB 2000
160  ON KEY(3)  GOSUB 3000
170  ON KEY(4)  GOSUB 4000
180  FOR  J=1 TO 4
190     KEY(J) ON
200  NEXT J
210  X = INT(100*RND)
220  Y = INT(100*RND)
230  GOTO 210
1000 'Addition
1010    CLS
1020    PRINT "ADDITION"
1030    PRINT "AUFGABE"
1040    PRINT X;" + ";Y; " IST GLEICH ?"
1050    INPUT ANTWORT
1060    IF  ANTWORT = X+Y  THEN 1070  ELSE 1090
1070    PRINT "RICHTIG"
1080    GOTO 1100
1090    PRINT "FALSCH --- DAS RICHTIGE ERGEBNIS LAUTET: ";X+Y
1100 RETURN
2000 'Subtraktion
2010    CLS
2020    PRINT "SUBTRAKTION
2030    PRINT "AUFGABE"
2040    PRINT X;" - ";Y; " IST GLEICH ?"
2050    INPUT ANTWORT
2060    IF  ANTWORT = X-Y  THEN 2070  ELSE 2090
2070    PRINT "RICHTIG"
2080    GOTO 2100
2090    PRINT "FALSCH --- DAS RICHTIGE ERGEBNIS LAUTET: ";X-Y
2100 RETURN
3000 'Multiplikation
3010    CLS
3020    PRINT "MULTIPLIKATION"
3030    PRINT "AUFGABE"
3040    PRINT X;" * ";Y; " IST GLEICH ?"
3050    INPUT ANTWORT
3060    IF  ANTWORT = X*Y  THEN 3070  ELSE 3090
3070    PRINT "RICHTIG"
3080    GOTO 3100
3090    PRINT "FALSCH --- DAS RICHTIGE ERGEBNIS LAUTET: ";X*Y
3100 RETURN
```

Abb. 10.1 Programm zum Auffangen von Ereignissen (Drücken von Funktionstasten), Teil 1

```
4000 'Beendigung des Programmes
4010    CLS
4020    PRINT "BEENDIGUNG DES PROGRAMMES"
4030    KEY OFF
4040 END
```

Abb. 10.1 Programm zum Auffangen von Ereignissen (Drücken von Funktionstasten), Teil 2

Anweisungen in den Zeilen 140 bis 170 aufgesetzt. Zuguterletzt wird schließlich das Auffangen der Ereignisse eingeschaltet (Zeilen 180 bis 200).

In den Zeilen 210 und 220 werden die beiden Zahlen ausgewählt, die in die zu stellende arithmetische Aufgabe eingehen sollen; das geschieht, wir erkennen es an der Niederschrift von RND, auf Zufallsbasis. Die Zeile mit der Zeilennummer 230 enthält eine Rückverzweigung zum Statement auf der Zeile mit der Zeilennummer 210. Es entsteht somit eine endlose (unbegrenzte) Schleife (Zeilen 210 bis 230). In dieser endlosen Schleife werden nun fortlaufend, d.h. bei jeder Wiederholung, Paare von Zufallszahlen gebildet. Auf diese Weise hängt die Aufgabenstellung, die ein Benutzer bekommt, von der Zeit ab, die er braucht, um eine der Funktionstasten F1 bis F4 niederzudrücken. Deshalb ist auch hier nicht nötig, das Statement RANDOMIZE einzusetzen, um zu garantieren, daß die gebildeten Zufallszahlen sich wiederholen. Die Programmausführung verbleibt solange in der Schleife, bis eine der Funktionstasten F1 bis F4 niedergedrückt wird. Danach verzweigt die Programmablaufsteuerung nach der entsprechenden Unterroutine. Es sei noch darauf hingewiesen, daß die Belegung der Funktionstasten in Zeile 25 auf dem Bildschirm angezeigt wird. Das haben wir durch das Statement veranlaßt, das wir in der Zeile mit der Zeilennummer 130 codiert haben.

In Programmen kann die Notwendigkeit entstehen, in einigen Teilen derselben das Auffangen von Ereignissen, die sich auf Funktionstasten beziehen, zu unterbinden. Nehmen wir einmal an, daß davon die Funktionstaste n betroffen ist. Wir haben in diesem Fall einfach eines der beiden Statements

```
KEY(n) STOP
KEY(n) OFF
```

vorzusehen. Die Wiederaufnahme des Auffangens von Ereignissen bei der Funktionstaste n erfordert dann bloß die Niederschrift des Statements

```
KEY(n) ON
```

Wenn die Funktionstaste n nach dem Wirksamwerden von STOP gedrückt wird, wird das Ereignis in der Erinnerung behalten. Nach Einschalten des Auffangens erfolgt dann sofort der Sprung zur entsprechenden Unterroutine. Benutzt man hingegen das Statement

```
KEY(n) OFF
```

zum Ausschalten des Auffangens, so unterbleibt das Merken evtl. auftretender Ereignisse, hier des Drückens der Funktionstaste n.

Außer des Auffangens von den mit Funktionstasten verbundenen Ereignissen kann man auch Ereignisse auffangen, die durch das Drücken der Tasten zur Bewegung des Positionsanzeigers (Cursors) entstehen. Die vier Cursorbewegungstasten befinden sich bekanntlich auf dem numerischen Block und sind durch Pfeile gekennzeichnet, die in die vier möglichen Rich-

tungen zeigen, in die der Cursor bewegt werden kann. Die Statements zum Auffangen dieser Ereignisse lauten genau so wie die entsprechenden Statements bei den Funktionstasten, also:

```
ON KEY(n)  GOSUB ···
KEY(n)  ON
KEY(n)  OFF
KEY(n)  STOP
```

Durch n sind hierbei die einzelnen Cursorbewegungstasten zu kennzeichnen. Es bedeuten:

n = 11	→	Cursorbewegung nach oben
n = 12	→	Cursorbewegung nach links
n = 13	→	Cursorbewegung nach rechts
n = 14	→	Cursorbewegung nach unten

Aufgabengruppe 36

1. Es ist das Statement niederzuschreiben, durch das die Funktionstaste F5 deaktiviert wird!
2. Es ist das Statement niederzuschreiben, durch das der Funktionstaste F1 die Zeichenkettenkonstante "LLIST<R>" zugewiesen wird. Unter <R> ist hier das Schreibkopfrücklaufzeichen (Cursorrücklaufzeichen) zu verstehen.
3. Es ist ein Programm zu schreiben, in dem das Anschlagen der Funktionstaste F1 bewirkt, daß der Bildschirm gelöscht wird und das laufende Programm aus dem RAM verschwindet (Vorbereiten eines neuen Programmes). Wie lautet das entsprechende Statement?
4. Das Programm, das wir im Beispiel 1 (Abb. 10.1) vorgestellt haben, ist so zu modifizieren, daß das Auffangen von Ereignissen während der Zeiten unterbunden wird, in der die Unterroutinen ablaufen, die auf den Zeilen mit den Zeilennummern 1000, 2000 bzw. 3000 beginnen.
5. Es ist das Statement niederzuschreiben, durch das das Drücken der Taste aufgefangen wird, die die Cursorbewegung nach oben verursacht.

Antworten auf die Testübung 10.2.1

a)
```
10  DATA  ADDITION, SUBTRAKTION, MULTIPLIKATION
20  FOR  J = 1  TO  3
30    READ A$(J)
40  NEXT J
50  FOR  J = 1  TO  10
60    KEY J,A$(J)
70  NEXT J
```

b)
```
KEY ON
```

10.3 Das Auffangen von Fehlern

Bezüglich der Behandlung von auftretenden Fehlern haben wir bis zu diesem Augenblick den folgenden Wissensstand erreicht: Wir kennen nur eine einzige Art und Weise, in der ein Programm auf Fehler reagiert, nämlich Stoppen der Programmausführung und Anzeige einer Fehlernachricht auf dem Bildschirm. Mitunter wird der Programmablauf aus einem guten Grund angehalten, da ein logischer Fehler BASIC daran hindert, einen vernünftigen Gebrauch vom Programm zu machen. Jedoch gibt es auch andere Fälle, in denen ein Fehler vorkommt, der ziemlich harmlos und daher einfach zu beheben ist. Beispiele hierfür sind:

- Man hat vergessen, den Drucker einzuschalten
- Man hat die falsche Diskette in ein Laufwerk eingelegt
- Man hat auf eine Systemmeldung mit einer falschen Antwort reagiert

In jedem dieser Fälle wäre es sicherlich wünschenswert für den weiteren Programmablauf, wenn man verlangen könnte, daß die Fehlerursache dem Benutzer mitgeteilt wird und anschließend auf Maßnahmen gewartet werden kann, die der Benutzer ergreift. Wir wollen nunmehr lernen, wie ein Programm zu gestalten ist, das Maßnahmen im Fehlerfall zuläßt.

Wie wir gesehen haben, reagiert ein Programm im Fehlerfall durch Anhalten. Jedoch existiert hierzu eine Alternative, wenn man das Statement

```
ON ERROR  GOTO <zn>
```

verwendet. Wenn ein Programm eine solche Anweisung enthält, wird BASIC zu der Zeile mit der Zeilennummer <zn> verzweigen, sobald ein Fehler während des Programmablaufes eintritt. Wenn wir z. B. in unser Programm die Anweisung

```
ON ERROR  GOTO 5000
```

aufgenommen haben, geht beim Auftreten eines Fehlers der Programmablauf zur Zeile mit der Zeilennummer 5000 über. Ab der Zeile 5000 hätten wir dann sinnvollerweise eine Fehlerbehandlungsroutine (eine Routine zum Auffangen von Fehlern) niederzuschreiben. Diese müßte unbedingt folgendes enthalten:

1. *Analyse des Fehlers*
2. *Benachrichtigung des Benutzers über den Fehler*
3. *Wiederaufnahme des Programmablaufes und bzw. oder Warten auf weitere Befehle seitens des Benutzers*

Das Statement

```
ON ERROR  GOTO···
```

wird als „Fehlerauffangstatement" bezeichnet. Es darf in einem Programm an einer beliebigen Stelle vorkommen. Nach dem Eintippen des Befehles RUN untersucht BASIC, ob irgendwo im Programm eine Fehlerauffanganweisung präsent ist. Wenn das der Fall ist, setzt es einen Code auf, der zu der angegebenen Programmstelle geschickt wird, falls ein Fehler auftreten sollte. Es ist natürlich anzuraten, ein Fehlerauffangstatement am Anfang des Programmes vorzusehen. Dadurch kann man die Zeit minimieren, die BASIC zum Durchsuchen eines Programmes nach einer solchen Anweisung braucht.

Um zu sehen, wie eine Fehlerbehandlungsroutine aufzubauen ist, wollen wir ein spezielles Beispiel betrachten. Dazu wollen wir annehmen, daß in unserem Programm das Lesen von einer Datendatei enthalten ist; diese muß sich auf der Diskette im Standardlaufwerk befinden. Der Benutzer dieses Programmes kann möglicherweise die falsche Diskette ins Laufwerk einlegen, er kann aber auch das Einlegen einer Diskette gänzlich vergessen. Wir wollen unsere zu schreibende Fehlerbehandlungsroutine auf diese beiden Fehlerarten beschränken, d.h. sie soll nur auf diese Fehlerarten reagieren.

Unsere Fehlerbehandlungsroutine wollen wir auf der Zeile mit der Zeilennummer 5000 beginnen lassen. Zu Beginn unseres Programmes, z.B. auf der Zeile mit der Zeilennummer 10, wollen wir das Statement zum Auffangen von Fehlern plazieren. Es lautet gemäß der geschilderten Voraussetzungen:

```
10  ON ERROR  GOTO 5000
```

Tritt nun ein Fehler auf, so notiert BASIC die Nummer der den Fehler enthaltenden Zeile in der Variablen ERL (ERror Line, auf deutsch Fehlerzeile) und eine Fehlernummer in der Variablen ERR. Anschließend verzweigt das Programm zur Zeile mit der Zeilennummer 5000. Über die Werte der beiden Variablen ERL und ERR können wir genau so verfügen wie über die Werte der von uns definierten und benutzten Variablen. – Die Fehlernummern, die der Variablen ERR zugewiesen werden, sind gemäß der Fehlerart fest vergeben, so daß man umgekehrt aus der Fehlernummer Rückschlüsse auf die Art des Fehlers ziehen kann.

In unserem speziellen Beispiel interessieren uns nur zwei bestimmte Fehlerarten:

- File not Found (Datei nicht gefunden) mit der Fehlernummer 53
- Disk not Ready (Diskette nicht betriebsbereit) mit der Fehlernummer 71

Die erstgenannte Fehlerart wird dann signalisiert, wenn die angeforderte Datei auf der eingelegten Diskette nicht gefunden wird. Zu einem Fehler der zweitgenannten Art kommt es, wenn entweder die Verriegelung des Laufwerks offen oder keine Diskette eingelegt ist. Die Fehlernummern können entweder in der diesem Buch beigelegten Übersichtsfaltkarte oder in dem Handbuch der IBM über BASIC nachgeschlagen werden. Im Fehlerfall soll und wird unsere Fehlerbehandlungsroutine den Benutzer informieren und auf seinen Eingriff zur Bereinigung der Situation warten. Die Auflistung unserer Fehlerbehandlungsroutine ist in der Abb. 10.2 festgehalten.

```
5000  'Routine zum Auffangen von Fehlern (Fehlerbehandlungsroutine)
5010  IF  ERR = 53  THEN PRINT "File not Found"
5020  IF  ERR = 71  THEN PRINT "Disk not Ready"
5030  IF  ERR<>53 AND ERR<>71  THEN PRINT "Unrecoverable Error"
5040  IF  ERR<>53 AND ERR<>71  END
5050  PRINT "Diskette korrigieren"
5060  PRINT "Nach der Korrektur irgendeine Taste druecken"
5070  IF  INKEY$=""  THEN 5070
5080  RESUME
```

```
Die Fehlernachrichten haben folgende Bedeutungen:
  1)   File not Found        --->  Datei nicht gefunden
  2)   Disk not Ready        --->  Diskette nicht betriebsbereit
  3)   Unrecoverable Error   --->  nicht behebbarer (nicht kor-
                                   rigierbarer) Fehler
```

Abb. 10.2 Beispiel für eine einfache Fehlerbehandlungsroutine

Zu der in Abb. 10.2 aufgelisteten Fehlerbehandlungsroutine sind noch einige Kommentare angebracht. Wie von uns beabsichtigt, erlaubt sie nur im Fall von Fehlern der Fehlernummern 53 und 71 eine Wiederaufnahme des Programmablaufes. Beim Auftreten eines Fehlers irgendeiner anderen Fehlerart bewirken die Anweisungen auf der Zeile mit der Zeilennummer 5040, daß das Programm beendet wird. Das Statement auf der Zeile mit der Zeilennummer 5050 fordert den Benutzer auf, die Fehlersituation zu bereinigen. In der darauffolgenden Zeile wird das Warten veranlaßt und zwar solange, bis der Benutzer die Fehlerbehebung signalisiert. Die Anweisung RESUME in der Zeile mit der Zeilennummer 5070 löscht die Fehlerbedingung und veranlaßt, daß die Ausführung des Programmes bei der Anweisung wiederaufgenommen wird, die zum Auftreten des Fehlers führte. Man beachte, daß unsere Fehleranalyse auf die Variable ERR zurückgegriffen hat. Wir hätten uns auch ebensogut der Variablen ERL bedienen können, um das Verhalten des Programmes im Fehlerfall zu leiten.

Die Anweisung RESUME kann außerdem noch in mehreren anderen nützlichen Varianten gebraucht werden:

- RESUME NEXT
  Diese Variante bewirkt die Wiederaufnahme des Programmablaufes bei der Anweisung, die unmittelbar auf die den Fehler verursachende Anweisung folgt.
- RESUME <znr>
  Diese Variante bewirkt die Wiederaufnahme des Programmablaufes bei der Zeile mit der angegebenen Zeilennummer <znr>.

Beim Entwerfen und Testen von Fehlerbehandlungsroutinen ist es außerordentlich hilfreich, wenn man Fehler der aufzufangenden Fehlerarten künstlich generieren kann. Das kann mittels des ERROR-Statements geschehen. Um z. B. einen Fehler der Fehlernummer 50 (Field Overflow, zu deutsch Feldüberlauf) auf der Zeile 75 zu erzeugen, hat man

```
75  ERROR 50
```

zu codieren. Wenn die Programmausführung jetzt die Zeile mit der Zeilennummer 75 erreicht, wird der Fehler mit der Fehlernummer 50 simuliert. Der Programmablauf verzweigt nun zu der zu testenden Fehlerbehandlungsroutine.

Aufgabengruppe 37

1. Es ist eine Fehlerbehandlungsroutine zu schreiben, die alle beim Programmablauf auftretenden Fehler ignoriert.
2. Es ist eine Fehlerbehandlungsroutine zu schreiben, die die Entdeckung eines Fehlers vom Typ „Type mismatch (Keine Typübereinstimmung)" auf Zeile 500 ermöglicht. Die Antwort auf das Auftreten eines solchen Fehlers soll in der Anzeige einer Fehlerbeschreibung und einer nachfolgenden Verzweigung zur Zeile mit der Zeilennummer 600 bestehen.

   Anmerkung: Die Fehlerart „Keine Typübereinstimmung" ist durch die Fehlernummer 13 gekennzeichnet.

10.4 Anhängen von Programmen

Das Statement CHAIN ermöglicht den Aufruf eines in BASIC geschriebenen Programmes von dem gerade ablaufenden Programm. So hat z.B. die Anweisung

```
2000  CHAIN "QUADRATE"
```

in einem Programm zur Folge, daß das Programm "QUADRATE" in den RAM geladen und danach ausgeführt wird. Das Programm selbst, in dem diese Anweisung enthalten ist, geht dabei genau so verloren wie die Variablen dieses Programmes. Dem die Steuerung übertragenen Programm "QUADRATE" werden also keine Variablen übergeben. Die Ablaufsteuerung beginnt mit der ersten Zeile von "QUADRATE".

Wenn man will, daß die Ablaufsteuerung des angehängten Programmes erst mit den Statements ab der Zeile mit der Zeilennummer 300 beginnt, kann das durch einen zweiten Operanden im CHAIN-Statement dem angehängten Programm mitteilen. Erweitern wir in diesem Sinne das o.a. Beispiel:

```
2000  CHAIN "QUADRATE",300
```

Das CHAIN-Statement bietet aber dem Programmierer noch weitere Möglichkeiten an. Durch den dritten Operanden ALL kann er ausdrücken, daß alle Variablen des gegenwärtig ablaufenden Programmes[1] an das angehängte Programm überstellt werden. Bei Weglassung des zweiten Operanden wird gesagt, daß das angehängte Programm mit der ersten Anweisung seine Ausführung beginnt. Die Erweiterung unseres Beispielstatements sieht also wie folgt aus:

```
2000  CHAIN "QUADRATE",,ALL
```

Zur Übergabe aller Variablen des augenblicklich ablaufenden Programmes an das angehängte Programm, das seine Ausführung mit dem ersten Statement auf Zeile 300 beginnen soll, ist demnach, am Beispiel des angehängten Programmes "QUADRATE" gesehen, die folgende Anweisung zu codieren:

```
2000  CHAIN "QUADRATE",300,ALL
```

Die CHAIN-Anweisung erweist sich dann als besonders nützlich, wenn ein einzelnes Programm zu umfangreich ist und daher nicht mehr in den Hauptspeicher (in den RAM) paßt. Um das Problem der zu umfangreichen Programme zu umgehen, kann man ein Programm in mehrere einzelne Programme zerlegen. Durch das CHAIN-Statement lassen sich diese dann zu einem Programm verbinden. Im Interesse der Speicherplatzersparnis wird man in der Mehrzahl aller Fälle wünschen, daß an ein angehängtes Programm nur einzelne Variablen des gerade ablaufenden Programmes übergeben werden. Um das zu bewerkstelligen, bedient man sich des COMMON-Statements. Um z.B. aus dem gegenwärtigen Programm die Variablen A, B und C$ an das angehängte Programm zu überstellen, muß das gegenwärtige Programm das Statement

```
10  COMMON A,B,C$
```

[1] *kurz:* des gegenwärtigen Programmes

enthalten. Wenn ein Bereich auf diese Weise übergeben werden soll, so muß das ebenfalls im COMMON-Statement ausgedrückt werden. Beispielsweise wollen wir neben den bisher schon erwähnten Variablen noch den Bereich GEHALT an das angehängte Programm übergeben. Das ins gegenwärtige Programm aufzunehmende COMMON-Statement könnte dann wie folgt lauten:

```
10  COMMON A,B,C$, GEHALT()
```

Man kann beliebig viele COMMON-Statements in ein Programm einfügen. Eine bestimmte Variable darf jedoch nur ein einziges Mal in ihnen aufgeführt sein. Die Stellung der COMMON-Statements in einem Programm kann freizügig gewählt werden. Der Übersichtlichkeit wegen ist freilich der Programmanfang vorzuziehen.

Mit dem CHAIN-Statement sollte sorgfältig umgegangen werden. Man sollte sich stets an einige signifikante Effekte erinnern.

1. Es gibt keine Möglichkeit, vom Benutzer definierte Funktionen an das angehängte Programm zu übergeben.
2. Die mit Hilfe der Statements DEFINT, DEFSNG oder DEFDBL definierten Variablentypen bleiben nicht erhalten; die durch diese Statements definierten Variablentypen werden im Kap. 12 besprochen.
3. Eine durch das Fehlerauffangen erhaltene Zeilennummer wird nicht sichergestellt.
4. Alle Dateien werden abgeschlossen, sie müssen also im angehängten Programm wieder neu eröffnet werden.

Durch das CHAIN-Statement geht das augenblicklich ablaufende Programm vollständig verloren. Man kann allerdings einen Teil desselben (und damit bei passender Formulierung auch das ganze selbst) durch Benutzung der MERGE-Option im RAM festhalten. Die Anweisung

```
CHAIN MERGE "QUADRATE",300
```

bewirkt z.B., daß das Programm "QUADRATE" mit dem augenblicklich ablaufenden Programm vermischt wird; die Programmausführung wird nach dem Mischen bei der Zeile 300 wieder aufgenommen. Das Mischen erfolgt so, daß die Zeilen von "QUADRATE" mit den Zeilen des gegenwärtig ablaufenden Programmes durchschossen werden; maßgebend hierfür sind die Zeilennummern. Kommt allerdings in "QUADRATE" eine Zeilennummer vor, die bereits in dem Programm enthalten ist, das das CHAIN-Statement enthält, so wird die betreffende Zeile im gegenwärtigen Programm gelöscht, nur die in "QUADRATE" bleibt bestehen. Das angehängte Programm wird also bevorzugt.

Das hineinzumischende Programm muß im ASCII-Format vorliegen; dieses Format wird bei Verwendung des SAVE-Befehles mit dem Operanden A hergestellt. Ist das nicht geschehen, liegt also ein für das Mischen unbrauchbares Format vor. BASIC reagiert darauf mit der Fehlermeldung „Bad File Mode Error (Falscher Dateityp)", d.h. mit der Fehlernummer 54.

Bei manchen Anwendungen ist es sinnvoll, wenn vor dem Mischen ein Teil des gegenwärtigen Programmes zerstört wird. Auch das kann man durch eine geeignete Form des CHAIN-Statements erreichen. Sehen wir uns dazu das folgende Beispiel an:

```
CHAIN MERGE "QUADRATE",300, DELETE 310-1000
```

Hier werden vor dem Mischen des gegenwärtigen und des angehängten Programmes die Zeilen des gegenwärtigen Programmes mit den Zeilennummern 310 bis 1000 zerstört. Danach wird, wie schon bekannt, der Programmablauf bei der Zeile 300 des resultierenden Programmes wieder aufgenommen.

Das CHAIN-Statement mit der Option MERGE bewirkt, daß die zum Zeitpunkt der Ausführung dieses Statements eröffneten Dateien eröffnet bleiben. Ebenfalls bleiben die definierten Variablentypen und die vom Benutzer definierten Funktionen erhalten.

Aufgabengruppe 38

1. Es ist das Statement niederzuschreiben, durch das das gegenwärtige Programm mit dem Programm "L" gemischt wird. Der Programmablauf soll anschließend bei der ersten Zeile des resultierenden Programmes wieder aufgenommen werden.
2. Es ist ein Programm zu schreiben, das so gestaltet ist, daß die Programme "A", "B" und "C" einzeln nacheinander ablaufen.

11 Computerspiele

In den letzten paar Jahren begeisterten sich Millionen Menschen an Computerspielen, ja nahmen geradezu ihre Vorstellungswelt gefangen. Wir beabsichtigen deshalb, in diesem Kapital einige Spiele zu erörtern, die sowohl vom Zufallszahlengenerator als auch von den graphischen Möglichkeiten des IBM Personalcomputers Gebrauch machen. Bei vielen dieser Spiele benötigen wir eine Uhr, um die Zeitdauer von Bewegungen festlegen zu können. Deshalb beginnen wir damit, wie man Zeiten mit dem Computer abstimmt.

11.1 Abstimmung von Zeiten mit dem Computer

Über das Betriebssystem DOS kann man den Uhrzeitgeber des IBM Personalcomputers ansprechen. Er liefert sowohl das Datum als auch die Tageszeit. Man kann den Uhrzeitgeber für viele Zwecke einsetzen, beispielsweise für eine Zeitbegrenzung in einem Programmteil (siehe hierzu das Beispiel 1).

11.1.1 Auslesen des Uhrzeitgebers

Der Uhrzeitgeber des IBM Personalcomputers liefert sechs verschiedene Informationen. Sie liegen stets in der nachstehenden Reihenfolge vor:

- Monat (01 bis 12)
- Tag (01 bis 31)
- Jahr (80 bis 99)

} *Datum*

- Stunden (00 bis 23)
- Minuten (00 bis 59)
- Sekunden (00 bis 59)

} *Tageszeit*

Das Datum wird auf die folgende Art und Weise angezeigt:

2-15-84 → (15. Februar 1984)

Die Tageszeit wird auf die folgende Art und Weise angezeigt:

14:38:27 → (14.38:27 Uhr)

Die Jahreszahlangabe erfolgt nur mit den letzten beiden Ziffern. Durch sie werden die Jahre von 1980 bis 2099 abgedeckt. Der Uhrzeitgeber ist so programmiert, daß er die Tage in den Monaten zählt (28, 30 oder 31); Schaltjahre werden jedoch nicht erkannt.

In BASIC wird das Datum durch die eingefügte Funktion DATE$ erhalten. Um z. B. das augenblickliche Datum auf dem Bildschirm anzeigen zu lassen, muß man in ein Programm ein Statement der Form

```
10  PRINT DATE$
```

aufnehmen. Am 12. März 1984 kommt es damit zur folgenden Anzeige:

```
03-12-1984
```

In BASIC wird die Tageszeit durch die eingefügte Funktion TIME$ erhalten. Um z.B. die augenblickliche Tageszeit auf dem Bildschirm anzeigen zu lassen, muß man in ein Programm ein Statement der Form

```
20  PRINT TIME$
```

aufnehmen. Um 12.38:45 Uhr kommt es damit zur folgenden Anzeige:

```
12:38:45
```

Testübung 11.1.1

Es soll das augenblickliche Datum und die augenblickliche Tageszeit angezeigt werden. Welche Statements sind an der Tastatur einzugeben?

11.1.2 Setzen des Uhrzeitgebers

Man hat Gelegenheit, den Uhrzeitgeber beim Starten des Betriebssystemes DOS zu setzen. Man erinnere sich, daß das DOS anfangs eine Frage nach dem Datum anzeigt. Nach der akkuraten Beantwortung dieser Frage führt der Computer das Datum solange fort, wie er in Betrieb ist. Das Datum geht, auch dies sei noch einmal betont, freilich in dem Augenblick verloren, in dem der Computer ausgeschaltet wird. Dasselbe gilt sinngemäß für die Tageszeit.

Zur Festlegung von Datum und Tageszeit kann man aber auch die beiden in BASIC eingebauten Funktionen DATE$ und TIME$ benutzen. Angenommen, es ist am 28.4. 1984 um 11.00:00 Uhr. Dann könnte man die Befehlsfolge

```
DATE$ = "04-28-1984"
TIME$ = "11:00:00"
```

eingeben und damit Datum und Tageszeit für den Computer festlegen. Diese Befehle können immer dann eingetippt werden, wenn der Computer nicht mit der Ausführung eines Programmes beschäftigt ist. Da es sich um Befehle handelt, müssen hier natürlich die Zeilennummern fehlen. Man kann diese Befehle natürlich auch als Anweisungen in Programme aufnehmen, freilich dann mit Zeilennummern. Um z.B. die Tageszeit in einem Programm auf 0.00 Uhr zurückzusetzen, kann man in ein Programm die Anweisung

```
10  TIME$ = "00:00:00"
```

einfügen. Beim Setzen des Datums kann man von zwei Varianten Gebrauch machen. So ist es zuerst möglich, die Bindestriche durch Schrägstriche zu ersetzen. Somit gelten die folgenden vier Datumsformen als gleichwertig:

```
10/31/1984    10-31-1984
10/31-1984    10-31/1984
```

Außerdem ist es möglich, an Stelle der vierstelligen Jahreszahl nur die beiden letzten Ziffern derselben vorzusehen. So kann man also 84 eingeben, d.h. auf 1984 verzichten. Intern wird hier automatisch 19 hinzugefügt.

Testübungen

11.1.2 Es sind die Statements niederzuschreiben, durch die der Uhrzeitgeber auf 14.00 Uhr am 1. Januar 1984 gesetzt wird.

11.1.3 Es sind die Statements niederzuschreiben, durch die der Uhrzeitgeber auf die augenblickliche Tageszeit und das augenblickliche Datum gesetzt wird.

11.1.4 Es sind die Statements niederzuschreiben, durch die auf dem Bildschirm ständig die augenblickliche Tageszeit angezeigt wird.

11.1.3 Ermittlung von abgelaufenen (verstrichenen) Zeiten

Der Uhrzeitgeber kann zur Messung von verstrichenen Zeiten eingesetzt werden. Man kann vom Computer verlangen, daß er 10 Sekunden oder 3 Tage zählt. Bei solchen Messungen ist es sicher äußerst sinnvoll, die Bestandteile von Zeiten (Stunden, Minuten und Sekunden) bzw. von Datumsangaben (Tage, Monate und Jahre) in individuellen Variablen festzuhalten. Wir wollen uns deshalb zunächst darüber unterhalten, mit welcher Methode wir die entsprechenden Zahlen einfach bestimmen können.

Beginnen wollen wir mit der TIME$ zugewiesenen Zeichenkette. Nehmen wir dazu an, diese betrage z.Z.:

```
"10:07:32"
```

Um die Sekunden (hier: 32) zu isolieren, müssen wir die ersten sechs Zeichen der Zeichenkette als 10:07: abhacken. Wir können das durch Verwendung der eingefügten Funktion RIGHT$ bewerkstelligen. Durch RIGHT$(TIME$,2) erreichen wir, daß die rechten zwei Zeichen von TIME$ extrahiert werden; es entsteht damit also die Zeichenkettenkonstante "32". Bei den meisten Anwendungen brauchen wir jedoch die Sekunden als Zahl und nicht als Zeichenkettenkonstante. Für die Umwandlung einer aus Ziffern bestehenden Zeichenkettenkonstante in eine numerische Konstante gibt es die eingebaute Funktion VAL. Sehen wir uns dazu einige Beispiele an:

```
VAL("32")  →  32
VAL("-15") → -15
```

Wenn wir also die Sekunden aus der Tageszeit in Form einer numerischen Konstanten extrahieren wollen, können wir wie folgt codieren:

```
10  SEKUNDEN = VAL(RIGHT$(TIME$,2))
```

In ähnlicher Weise können wir auch die Stunden aus der Tageszeit herausholen. Wir ermitteln die linken zwei Zeichen und wandeln diese in eine ganze Zahl um. Eine diesbezügliche Anweisung würde somit wie folgt lauten:

```
20  STUNDEN = VAL(LEFT$(TIME$,2))
```

Wenn wir schließlich noch die Minuten aus der Tageszeit extrahieren wollen, müssen wir zunächst aus der Mitte von der TIME$ zugewiesenen Zeichenkettenkonstanten eine zwei Zei-

chen umfassende Kette herausholen, beginnend auf der Stelle 4, und diese in eine Zahl umwandeln. Hierzu können wir die eingebaute Funktion MID$ aufrufen. Ein entsprechendes Statement würde wie folgt aussehen:

```
30  MINUTEN = VAL(MID$(TIME$,4,2))
```

Um die Tage, Monate und Jahre als ganze Zahlen aus dem Datum zu ermitteln, kann man die Statementfolge codieren, die in der Abb. 11.1 aufgeführt ist.

```
40  TAG = VAL(MID$(DATE$,4,2))
50  MON = VAL(LEFT$(DATE$,2))
60  JHR = VAL(RIGHT$(DATE$,4))
```

Abb. 11.1 Statementfolge zur Ermittlung von Tag, Monat und Jahr

Beispiel 1:

Wir haben uns früher schon mit dem Problem beschäftigt, die Fertigkeit des Addierens zweistelliger Zahlen zu überprüfen. Wir wollen nunmehr die Rechenzeit beschränken. Für die Lösung jeder Additionsaufgabe sollen nunmehr nur noch 12 Sekunden zur Verfügung stehen.

Zur Beschränkung der Lösungszeit für die gestellten Aufgaben bedienen wir uns des Uhrzeitgebers. Nach Stellung der Aufgabe setzen wir deshalb die Tageszeit auf Null und konstruieren eine Schleife, in der kontinuierlich geprüft wird, ob die Sekunden in der Tageszeit schon auf 12 aufgelaufen sind. Wenn das der Fall ist, soll auf dem Bildschirm die Nachricht

```
ZEIT ABGELAUFEN --- WIE LAUTET DIE SUMME?
```

erscheinen. Das unter diesen Voraussetzungen geschriebene Programm ist in der Abb. 11.2 aufgelistet. Die Zeilen mit den Zeilennummern 30 bis 60 beinhalten die Schleife.

```
10  FOR  J = 1  TO  10:        'Schleife zur Stellung von 10 Aufgaben
20    INPUT "BITTE ZWEI ZWEISTELLIGE ZAHLEN EINGEBEN:"; A,B
30    PRINT "WIE LAUTET IHRE SUMME?"
40    TIME$ = "00.00.00":      'Setzen der Tageszeit auf den Augangswert
50    SEKUNDEN = VAL(RIGHT$(TIME$,2))
60    IF  SEKUNDEN < 12  THEN 50
100   INPUT "ZEIT ABGELAUFEN --- WIE LAUTET DIE SUMME?"; SUMME
110   IF  A+B=SUMME  THEN 200
120   PRINT "FALSCH --- DAS RICHTIGE ERGEBNIS LAUTET: "; A+B
130   GOTO 220:                'Uebergang zur naechsten Aufgabe
200   PRINT "GLUECKWUNSCH: DAS ERGEBNIS IST RICHTIG"
210   PKTZAHL = PKTZAHL + 1: 'Anzahl der richtig geloesten Aufgaben
                              um 1 erhoeht
220 NEXT J
300 PRINT "VON 10 GESTELLTEN AUFGABEN WURDEN "; PKTZAHL;
          " RICHTIG GELOEST"
400 PRINT "WENN EINE WIEDERHOLUNG GEWUENSCHT WIRD, ";
          "IST DER BEFEHL  RUN  EINZUGEBEN"
500 END
```

Abb. 11.2 Programm zur zeitabhängigen Überprüfung von Rechenfertigkeiten

Testübung 11.1.5

Das Programm von Abb. 11.2 ist so zu modifizieren, daß für die Lösung jeder gestellten Aufgabe eine beliebig lange Zeit zur Verfügung gestellt wird. Die zur Lösung einer Aufgabe benötigte Zeit soll jeweils nach Eingabe der Lösung angezeigt werden.

Aufgabengruppe 39

1. Dem Uhrzeitgeber sollen folgende Werte zugewiesen werden:
   a) Augenblickliches Datum
   b) Augenblickliche Tageszeit
2. Die augenblickliche Tageszeit soll auf dem Bildschirm angezeigt werden.
3. Es ist ein Programm zu schreiben, durch das im Abstand von einer Sekunde Datum und Uhrzeit auf dem Bildschirm angezeigt werden.
4. Es ist ein Programm zu schreiben, durch das im Abstand von einer Minute Datum und Uhrzeit auf dem Bildschirm angezeigt werden.

Antworten auf die Testübungen

11.1.1 Es sind folgende Eingaben vorzunehmen:

```
10  PRINT DATE$
20  PRINT TIME$
30  END
RUN
```

11.1.2 Die Befehlsfolge lautet wie folgt:

```
TIME$ = "14.00.00"
DATE$ = "1/1/84"
```

11.1.3 Es sind folgende Eingaben vorzunehmen:

```
TIME$ = "........"
DATE$ = "......"
PRINT DATE$,TIME$
```

Anmerkung: An die Stelle der Punkte tritt das Datum bzw. die augenblickliche Uhrzeit.

11.1.4 Das Programm besteht aus einer endlosen Schleife. Die Beendigung des Programmes kann nur durch das gleichzeitige Drücken der beiden Tasten *Ctrl* und *Break* erreicht werden.

```
10  PRINT TIME$
20  CLS
30    FOR  K = 1  TO  500
40    NEXT K:          'Verzoegerungsschleife
50  GOTO 10
60  END
```

11.1.5 Das Programm von Abb. 11.2 ist wie folgt zu ändern:
a) Die Zeilen mit den Zeilennummern 50 und 60 sind zu entfernen.
b) An Stelle der Zeile mit der Zeilennummer 100 sind die nachfolgenden Zeilen einzufügen:

```
100 INPUT "WIE LAUTE DIE SUMME? "; SUMME
101 MINUTEN = VAL(MID$(TIME$,4,2))
102 SEKUNDEN = VAL(RIGHT$(TIME$,2))
103 PRINT "VERBRAUCHTE ZEIT FUER DIE LOESUNG: ";
          60*MINUTEN+SEKUNDEN; " SEKUNDEN"
```

11.2 Blindekuhschießen

Dieses Spiel kann im Textmodus unter Einsatz des „Erweiterten BASIC" ausgeführt werden.

Gegenstand dieses Spieles ist, auf eine Zielscheibe zu schießen, die auf dem Bildschirm dargestellt ist. Das Schießen geschieht dadurch, daß man den Cursor (Positionsanzeiger) durch Anschlagen der entsprechenden Tasten solange bewegt, bis die Zielscheibe getroffen ist. Der Haken dabei ist, daß die Zielscheibe für den Schützen nur ganze zwei Sekunden zu sehen ist. Das dem „Blindekuhschießen" zugrundeliegende Programm fordert am Anfang den Schützen auf, mit dem Spiel zu beginnen, indem er irgendeine Taste anschlägt. Das Programm wählt dann auf Zufallsbasis einen Punkt aus, auf den es die Zielscheibe plaziert. Diese leuchtet dann ganze zwei Sekunden auf. Danach wird der Cursor in die linke obere Ecke des Bildschirmes, in die sogenannte Heimposition, gestellt. Anschließend muß nun der Schütze durch Drücken der entsprechenden Tasten versuchen, den Cursor so zu bewegen, daß er auf die nunmehr unsichtbare Zielscheibe, deren Lage nur durch den zuvor getätigten flüchtigen Blick in seinem Gedächtnis haftet, trifft. Für die Cursorbewegungen steht ihm nur eine Zeit von höchstens fünf

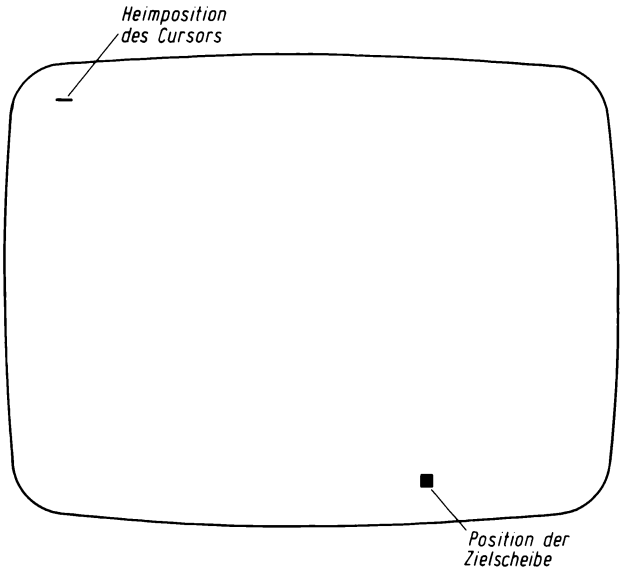


Abb. 11.3 Ausgangssituation beim „Schuß auf eine unsichtbare Zielscheibe" („Blindekuhschießen")

Sekunden zur Verfügung. Die Ausgangssituation dieses Spieles ist in der Abb. 11.3 dargestellt.

Der Trefferfolg des Blindekuhschießens wird mit einer Punktzahl gemessen. Diese wird aus der Anzahl der Einzelschritte ermittelt, die der Cursor noch in horizontaler und vertikaler Richtung zurücklegen muß, um von seiner letzten Position bis zur Position des Zieles zu gelangen. Die Punktzahl wird gemäß der nachfolgenden Tabelle bestimmt.

Einzelschritte bis zum Ziel	*Punktzahl*
0	100
1 oder 2	90
3 bis 5	70
6 bis 10	50
11 bis 15	30
16 bis 20	10
über 20	0

Wie schon gesagt, wird der Cursor durch das Anschlagen der Cursorbewegungstasten, die bekanntlich zum numerischen Block der Tastatur gehören, bewegt. Die Einrichtung zum Auffangen von Ereignissen (siehe Abschnitt 10.2) wird benutzt, um den Programmablauf zu unterbrechen, wenn eine dieser Tasten angeschlagen wird.

Zur weiteren Erläuterung dieses reizvollen Spieles wollen wir jetzt ein Beispiel verfolgen. Die Zeile, die eine Eingabe des Spielers beinhaltet, wollen wir dabei unterstreichen.

```
RUN
SCHUSS AUF EINE UNSICHTBARE ZIELSCHEIBE
BEGINN: ANSCHLAGEN IRGENDEINER TASTE
```

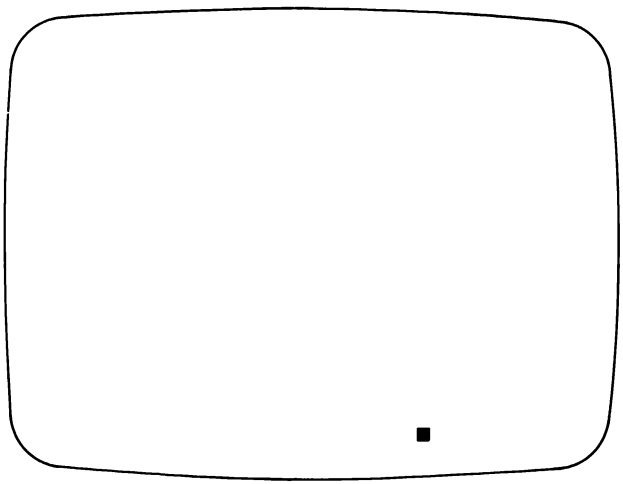
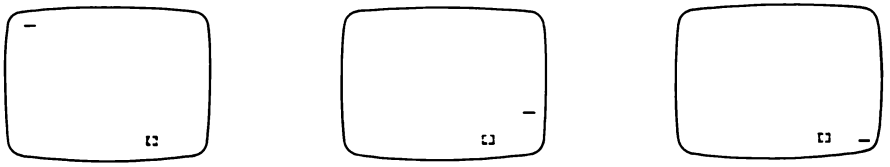


Abb. 11.4 Aufleuchten der Zielscheibe

Nach dem Anschlagen irgendeiner Taste auf der Tastatur wird der Bildschirm gelöscht und die Zielscheibe leuchtet auf (siehe Abb. 11.4).

Nach zwei Sekunden wird die Zielscheibe verdunkelt; sie verschwindet vom Bildschirm. Außerdem wird der Cursor in die Heimposition gebracht (Abb. 11.5,ⓐ).



ⓐ *Verdunklung der Zielscheibe und Stellung des Cursors in der Heimposition*

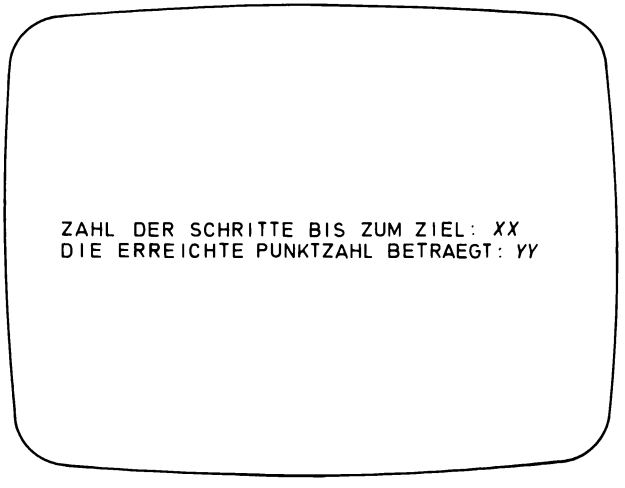
ⓑ *Zwischenstellung des Cursors nach fortlaufendem Anschlagen der Cursorbewegungstasten*

ⓒ *Endstellung des Cursors nach Ablauf der vorgegebenen Zeit (5s)*

Abb. 11.5 Einzelne Spielphasen

Nunmehr muß der Spieler durch Anschlagen der Cursorbewegungstasten den Cursor auf die Stellung hinbewegen, die er sich als Position der Zielscheibe gemerkt hat (Momentaufnahme in Abb. 11.5,ⓑ). Während sich der Spieler bemüht, die Zielscheibe anzusteuern, läuft die ihm zur Verfügung gestellte Zeit von fünf Sekunden ab; danach kann er den Cursor nicht mehr weiterbewegen (Abb. 11.5,ⓒ).

Nach Ablauf der „Schießzeit" erscheinen auf dem Bildschirm die zwei Zeilen, die das Spielergebnis anzeigen. Wird die Zielscheibe getroffen, d. h. steht der Cursor nach Ablauf der Zeit auf



Anmerkungen: 1) Unter *xx* ist der der Variablen SCHRITTZAHL zugewiesene Wert zu verstehen.
2) Unter *yy* ist der der Variablen PUNKTZAHL zugewiesene Wert zu verstehen.

Abb. 11.6 Ergebnisausgabe des „Blindekuhschießens"

der Position der Zielscheibe, wird zwischen diesen beiden Zeilen noch eine dritte, eine Glückwunschzeile, ausgegeben. Die Normalausgabe ist in der Abb. 11.6 dargestellt.

Nach der Besprechung des Spielverlaufes anhand eines Beispieles soll zum Schluß das Spielprogramm aufgelistet werden (Abb. 11.7).

```
100  'Anzeige der Titelzeilen auf dem Bildschirm
110     CLS
120     KEY OFF
130     WIDTH 40
140     RANDOMIZE VAL(RIGHT$(TIME$,2))
150     PRINT "SCHUSS AUF EINE UNSICHTBARE ZIELSCHEIBE"
160     PRINT "BEGINN: ANSCHLAGEN IRGENDEINER TASTE"
170     IF  INKEY = ""  THEN 170
180     CLS
190  'Initialisierung (Zuweisung von Anfangswerten)
200     TIME$ = "0:0:0":          'Anfangswert fuer die Tageszeit
210     LOCATE ,,0:               'Ausschalten des Cursors
                                   (Positionsanzeigers)
220  'Festlegen der Lage der Zielscheibe (Zeile,Spalte)
230     ZIELZEILE = INT(25*RND) + 1
240     ZIELSPALT = INT(40*RND) + 1
250     LOCATE ZIELZEILE,ZIELSPALT
260     PRINT CHR$(219):          'Aufleuchten der Zielscheibe
270  'Zeitbegrenztes Aufleuchten der Zielscheibe
280     SEKUNDEN = VAL(RIGHT$(TIME$,2))
290     IF  SEKUNDEN<2  THEN 280  ELSE 310
300     'Ablauf von zwei Sekunden
310     LOCATE ZIELZEILE,ZIELSPALT
320     PRINT " ";:               'Verdunkeln der Zielscheibe
330     LOCATE  ,,1
340     PRINT CHR$(11)
350     'Zuruecksetzen der Tageszeit (Setzen der Anfangszeit)
360     TIME$ = "0:0:0"
370  'Aktivieren des Auffangens von Ereignissen: Druecken der
      Tasten fuer die Bewegung des Cursors (Positionsanzeigers)
380     ON KEY(11)  GOSUB 480
390     ON KEY(12)  GOSUB 520
400     ON KEY(13)  GOSUB 560
410     ON KEY(14)  GOSUB 600
420     KEY(11)  ON
430     KEY(12)  ON
440     KEY(13)  ON
450     KEY(14)  ON
460     SEKUNDEN = VAL(RIGHT$(TIME$,2))
470     IF  SEKUNDEN<5  THEN 370  ELSE 700:   'Schusszeitvergleich
480  'Bewegung des Cursors (Positionsanzeigers) nach oben
490        GOSUB 640
500        PRINT CHR$(30)
510     RETURN
520  'Bewegung des Cursors (Positionsanzeigers) nach links
530        GOSUB 640
540        PRINT CHR$(29)
550     RETURN
```

Abb. 11.7 Programm des „Blindekuhschießens" (1. Teil)

```
560  'Bewegung des Cursors (Positionsanzeigers) nach rechts
570        GOSUB 640
580        PRINT CHR$(28)
590     RETURN
600  'Bewegung des Cursors (Positionsanzeigers) nach unten
610        GOSUB 640
620        PRINT CHR$(31)
630     RETURN
640  'Deaktivieren des Auffangens von Ereignissen: Druecken der
      Tasten fuer die Bewegung des Cursors (Positionsanzeigers)
650        KEY(11)  OFF
660        KEY(12)  OFF
670        KEY(13)  OFF
680        KEY(14)  OFF
690     RETURN
700  'Errechnen der erreichten Punktzahl
710     SCHRITTZAHL = ABS(POS(0)-ZIELSPALT) + ABS(CSRLIN-ZIELZEILE)
720     CLS
730     LOCATE 12,1
740     PRINT "ZAHL DER SCHRITTE BIS ZUM ZIEL: "; SCHRITTZAHL
750     PKTZAHL = 100
760     IF  SCHRITTZAHL=0   THEN PRINT "FABELHAFT --- ZIEL GETROFFEN"
770     IF  SCHRITTZAHL>0   THEN PKTZAHL=PKTZAHL-10
780     IF  SCHRITTZAHL>2   THEN PKTZAHL=PKTZAHL-20
790     IF  SCHRITTZAHL>5   THEN PKTZAHL=PKTZAHL-20
800     IF  SCHRITTZAHL>10  THEN PKTZAHL=PKTZAHL-20
810     IF  SCHRITTZAHL>15  THEN PKTZAHL=PKTZAHL-20
820     IF  SCHRITTZAHL>20  THEN PKTZAHL=PKTZAHL-10
830     PRINT "DIE ERREICHTE PUNKTZAHL BETRAEGT: "; PKTZAHL
840     INPUT "WIEDERHOLUNG GEWUENSCHT? (J/N): "; ANTWORT$
850     IF  ANTWORT$="J" OR ANTWORT$="j"  THEN 180
860     END
```

Abb. 11.7 Programm des „Blindekuhschießens" (2. Teil)

Aufgabengruppe 40

1. Mit dem Programm für das „Blindekuhschießen" ist zu experimentieren. Das Aufleuchten der Zielscheibe soll auf eine Sekunde begrenzt werden. Die Schußresultate sind mit der ursprünglichen Dauer des Aufleuchtens zu vergleichen.

2. Mit dem Programm für das „Blindekuhschießen" ist zu experimentieren. Das Aufleuchten der Zielscheibe soll auf fünf Sekunden ausgedehnt werden. Die Schußresultate sind mit der ursprünglichen Dauer des Aufleuchtens zu vergleichen.

3. Das Programm für das „Blindekuhschießen" ist so zu modifizieren, daß zehn aufeinanderfolgende Schüsse „abgefeuert" werden können. Am Schluß ist die Gesamtzahl der erreichten Punkte zu bestimmen.

4. Das Programm für das „Blindekuhschießen" ist so zu modifizieren, daß von zwei Spielern je zehn aufeinanderfolgende Schüsse „abgefeuert" werden können. Am Schluß ist für jeden Spieler die Gesamtzahl der erreichten Punkte zu bestimmen und der Sieger des Wettstreites zu ermitteln.

11.3 Sicherstellen und Zurückholen von Abbildungen

Das erweiterte BASIC weist Sprachelemente auf, durch die die auf dem Bildschirm angezeigten Inhalte beliebiger Rechtecke sichergestellt und später wieder zurückgeholt werden können. Bei vielen graphischen Anwendungen erweist sich diese Möglichkeit als außerordentlich vorteilhaft, insbesondere beim Wiederherstellen von graphischen Abbildungen.

Wir wollen unsere Besprechung mit einer Beschreibung der Abbildungen beginnen, die sichergestellt werden können. Jede Abbildung muß einem Bildschirmteil mit rechteckiger Gestalt entsprechen. Das fragliche Rechteck kann irgendwo auf dem Bildschirm liegen. Sein Inhalt kann aus Textzeichen, Teilen derselben oder aus graphischen Abbildungen bestehen.

Das sicherzustellende Rechteck muß durch Angabe der Koordinaten der Eckpunkte, die zu einer Diagonalen gehören, umrissen werden, d.h. entweder der Koordinaten des linken oberen und des rechten unteren oder der Koordinaten des linken unteren und des rechten oberen Eckpunktes. Erinnern wir uns, wir müssen somit die gleiche Beschreibung des Rechtekkes vornehmen, die wir bei der Behandlung des LINE-Statements (Zeichnen eines Rechteckes) kennengelernt haben. Einige Beispiele sollen das ganze noch einmal erhärten:

```
(0,0)-(100,100)
(3,8)-(30,80)
```

Bei der Angabe der Rechteckskoordinaten ist der graphische Modus zu berücksichtigen, der im Augenblick der Sicherstellung gültig ist, also der graphische Modus mittleren bzw. hohen Auflösungsvermögens. In beiden Fällen belegt ein Textzeichen ein Rechteck von 8 mal 8 Bildpunkten (Pixels). Das Zeichen in der linken oberen Ecke des Bildschirmes wird also vom Rechteck (0,0)-(7,7) umrandet. Wichtig ist der nicht oft genug zu betonende Hinweis, daß jede Textzeile stets 8 Pixels hoch ist.

Testübung 11.3.1

Es ist das Rechteck zu benennen, das die zweite Textzeile auf dem Bildschirm beinhaltet. Als Voraussetzung soll gelten, daß der graphische Modus mittleren Auflösungsvermögens vorliegt.

Das Statement GET gestattet die Speicherung des Inhaltes eines Rechtecks in einem numerischen Bereich, der groß genug für die Aufnahme der Abbildung ist, d.h. er muß aus einer ausreichenden Anzahl von Elementen bestehen. Nehmen wir einmal an, daß das Rechteck eine Länge von x Pixels und eine Höhe von y Pixels besitzt. Dann muß das DIM-Statement zur Definition des Bereiches mindestens mit der Dimension

```
4 + INT((2*x+7)*y/32)
```
[1]

[1] Als Funktionswert der in BASIC eingefügten Funktion INT ergibt sich die größte ganze Zahl, die kleiner als oder gleich dem Argument dieser Funktion ist. Die Behandlung dieser Funktion erfolgt im Abschnitt 12.3, zusammen mit anderen eingefügten Funktionen.

erklärt werden (beim mittleren Auflösungsvermögen) bzw. mindestens mit der Dimension

```
4 + INT((x+7)*y/32)
```

(beim hohen Auflösungsvermögen). Anhand eines Beispiels wollen wir uns die Definition eines solchen Bereiches ansehen. Nehmen wir zu diesem Zweck einmal an, daß ein umschließendes Rechteck 10 Pixels lang und 50 Pixels hoch sei. Außerdem soll der graphische Modus mittleren Auflösungsvermögens vorliegen. Der für die Speicherung des Inhaltes dieses Rechteckes erforderliche Bereich muß dann mindestens eine Dimension von

```
4 + INT((2*10+7)*50/32),
```

aufweisen. Einen solchen Bereich könnten wir durch das Statement

```
DIM A(46)
```

definieren, vorausgesetzt, daß wir ihn mit A() bezeichnen wollen. Nach der Definition eines ausreichend umfangreichen Bereiches kann der Inhalt des Rechteckes mittels des GET-Statements in diesen Bereich gespeichert werden. Allgemein weist dieses Statement die folgende Form auf:

```
GET (x1,y1)-(x2,y2),bereichsname
```

auf. Um den Inhalt des Rechtecks (0,0)-(9,49), also ein Rechteck mit einer Länge von 10 Pixels und einer Höhe von 50 Pixels, zu speichern, ist somit das Statement

```
GET (0,0)-(9,49),A
```

zu formulieren.

Fassen wir zusammen: Um den Inhalt eines Rechteckes in einem Bereich speichern zu können, müssen wir

1. ein DIM-Statement benutzen, um einen genügend großen Bereich zu definieren, und
2. ein GET-Statement ausführen lassen.

Man kann den Inhalt eines Rechteckes auf dem Bildschirm an irgendeiner Stelle wieder anzeigen lassen, indem man das PUT-Statement dafür einsetzt. Wenn wir z.B. den zuvor im Bereich A() weggespeicherten Rechteckinhalt auf dem Bildschirm wieder anzeigen lassen wollen, könnten wir das Statement

```
PUT (100,125),A
```

benutzen. Dieses spezielle Statement würde bei seiner Ausführung bewirken, daß der in A() gespeicherte Rechteckinhalt wieder auf dem Bildschirm erscheint, und zwar in dem Rechteck, dessen linker oberer Eckpunkt die Koordinaten (100,125) besitzt. Um das Zusammenwirken der Statements GET und PUT zu betrachten, wollen wir uns das in der Abb. 11.8 aufgelistete Programm ansehen.

Wir wollen zunächst einmal die von dem Programm in der Abb. 11.8 ausgeübten Funktionen beschreiben. Das Statement auf der Zeile mit der Zeilennummer 10 versetzt BASIC in den gra-

```
10  SCREEN 1
20  DIM LETTER(9)
30  LOCATE 1,1
40  PRINT "A"
50  GET (0,0)-(7,7),LETTER
60  CLS
70  PUT (100,100),LETTER
80  END
```

Abb. 11.8 Zusammenwirken der Statements GET und PUT

phischen Modus mittleren Auflösungsvermögens. Da wir den Inhalt eines Rechteckes von 8 x 8 Pixels speichern wollen, müssen wir zunächst die entsprechende Formel benutzen, um die Anzahl der Elemente bestimmen zu können, aus denen der Bereich bestehen muß. Die Werte von x und von y in die für den graphischen Modus mittleren Auflösungsvermögens geltende Formel eingesetzt, führt zu der im Statement auf der Zeile mit der Zeilennummer 20 vorgenommenen Bereichsdefinition. Durch die Anweisungen auf den Zeilen mit den Zeilennummern 30 und 40 wird das Textzeichen "A" in der linken oberen Ecke des Bildschirmes angezeigt. Diese Abbildung eines einzigen Textzeichens wird durch das nachfolgende Statement (Zeile mit der Zeilennummer 50) im Bereich LETTER gespeichert. Anschließend wird durch CLS der Bildschirm gelöscht. Die Anweisung auf der Zeile mit der Zeilennummer 70 holt die im Bereich LETTER gespeicherte Abbildung zurück und zeigt sie nunmehr auf dem Bildschirm in einem (gedachten) Rechteck an, dessen linker oberer Eckpunkt die Koordinaten (100,100) aufweist.

Wir wollen nach dem Ablauf des Programmes den Bildschirm noch nicht löschen. Vielmehr wollen wir sofort danach erst einmal den Befehl

```
PUT (100,100),LETTER
```

eingeben und die Eingabetaste drücken. Als Resultat der Ausführung dieses Befehles stellen wir fest, daß die Anzeige des Buchstabens "A" auf dem Bildschirm verschwindet. Tippen wir denselben Befehl anschließend noch einmal ein und drücken danach die Eingabetaste, so erscheint das Textzeichen "A" erneut auf dem Bildschirm. Dieses Phänomen können wir ausnutzen, um die Illusion einer Bewegung quer über den Bildschirm zu erzeugen. Angenommen, wir wollen dafür sorgen, daß sich scheinbar das Textzeichen "A" quer über den Schirm fort-

```
10  SCREEN 1
20  DIM LETTER(9)
30  LOCATE 1,1
40  PRINT "A"
50  GET (0,0)-(7,7),LETTER
60  CLS
70  FOR  XPOSITION = 0  TO  312
80    PUT (XPOSITION,0),LETTER
90    PUT (XPOSITION,0),LETTER
100 NEXT XPOSITION
110 END
```

Abb. 11.9 Bewegung eines Textzeichens über den Bildschirm

bewegt. Es ist hierzu bloß notwendig, daß wir auf fortlaufenden Positionen das Textzeichen anzeigen und sofort wieder löschen lassen. Die Anzeigen werden schneller erzeugt, als sie das Auge betrachten kann. Was sich infolgedessen ins Bewußtsein einprägt, ist eine fortlaufende Bewegung des Textzeichens. In der Abb. 11.9 ist ein Programm aufgeführt, das dem Betrachter die Illusion einer Bewegung vermittelt.

Zu dem Programm von Abb. 11.9 wollen wir noch einige Hinweise geben. Die Variable XPOSITION durchläuft den Wertebereich von 0 bis 312. Obgleich die Zeile eines Bildschirmes beim graphischen Modus mittleren Auflösungsvermögens aus insgesamt 320 Bildpunkten (numeriert von 0 bis 319) besteht, kann als Maximalwert nur 312 angegeben werden, da das Rechteck, das das Textzeichen "A" umschließt, 8 x 8 Pixels umfaßt und, wie bekannt, stets die Koordinaten des linken oberen Eckpunktes im PUT-Statement aufgeführt werden müssen. Somit ist in diesem Fall 312 der Maximalwert, den die Variable XPOSITION annehmen darf.

Rückgrat aller Computerspiele, die in der jüngsten Zeit so populär geworden sind, ist das Leben, das sich scheinbar auf dem Bildschirm ausbreitet. Im Abschnitt 11.4 wollen wir diese Vorstellung vom Leben auf dem Bildschirm ausnutzen, um uns mit einem Computerspiel zu befassen, das wir „Schießbude" nennen wollen.

Wir müssen abschließend noch einige Worte über ein kritisches Problem verlieren. Die Speicherung von größeren graphischen Abbildungen erfordert die Bereitstellung eines beachtlichen Teiles des Hauptspeichers, des RAM also. So erfordert die Unterbringung einer den gesamten Bildschirm umfassenden Abbildung nahezu 16000 Bytes (Speicherstellen). Man vergleiche diese Aussage mit dem Fakt, daß BASIC maximal 65536 Bytes benutzen kann. Weil die graphischen Abbildungen dazu tendieren, solche gewaltige Teile des Hauptspeichers zu beanspruchen, erweist es sich sicher oft als notwendig, Bildschirmabbildungen auf Disketten sicherzustellen und dort aufzubewahren. Statements, die für eine Sicherstellung des augenblicklichen Bildschirminhaltes in einer auf einer Diskette liegenden Datei sorgen, können z. B. wie folgt lauten:

```
10  DEF SEG=&HB800
20  BSAVE "SCREEN",0,&H4000
```

Als Dateiname der Datei, die den Bildschirminhalt aufnehmen soll, ist bei diesen Beispielstatements "SCREEN" vorgegeben. Soll der durch die o.a. Statements weggespeicherte Bildschirminhalt wieder auf den Schirm zurückgeholt, d.h. angezeigt werden, so müßte man die nachstehende Statementfolge benutzen:

```
10  DEF SEG=&HB800
20  BLOAD "SCREEN",0
```

Eine Erklärung darüber, wie die hier verwendeten Statements arbeiten, kann hier nicht gegeben werden; sie würde den Rahmen dieses Buches sprengen. Nichtsdestotrotz kann aus den Beispielen ohne weiteres geschlossen werden, wie diese Statements zu handhaben sind.

Aufgabengruppe 41

1. Es ist das Rechteck zu beschreiben, dessen linker oberer Eckpunkt die Koordinaten (10,10) besitzt und das 80 Pixels lang sowie 40 Pixels hoch ist.
2. Es ist das Rechteck zu beschreiben, das die ersten zwei Textzeilen des Bildschirmes umgibt.

3. Es ist das DIM-Statement zu formulieren, das einen Bereich definiert, der den Inhalt des Rechteckes aufnehmen kann, das in der Aufgabe 1. beschrieben ist.

4. Es ist das DIM-Statement zu formulieren, das einen Bereich definiert, der den Inhalt des Rechteckes aufnehmen kann, das in der Aufgabe 2. beschrieben ist.

5. Es sind die Statements zu formulieren, durch die der augenblickliche Bildschirminhalt auf eine Diskette gespeichert werden kann.

6. Es sind die Statements zu formulieren, durch die die folgenden Funktionen durchgeführt werden können:
   - Löschen des Bildschirmes
   - Wiederanzeigen der Abbildung, die durch die Statements von Aufgabe 5. weggespeichert wurde.

7. Es sind die Statements zu formulieren, durch die das graphische Zeichen mit dem ASCII-Code 2 („glückliches Gesicht") in einen Bereich gespeichert werden kann.

8. Es sind die Statements zu formulieren, durch die das durch die Statements von Aufgabe 7. weggespeicherte graphische Zeichen mit dem ASCII-Code 2 („glückliches Gesicht") an folgenden Stellen auf dem Bildschirm wieder angezeigt werden kann:
   a) (0,0)
   b) (50,50)
   c) (0,100)

9. Es sind die Statements zu formulieren, durch die das folgende lebende Bild auf dem Bildschirm erscheint:
   Das Zeichen mit dem ASCII-Code 2 („glückliches Gesicht") soll sich auf dem Bildschirm entlang der Zeile 10 bewegen.

10. Es sind die Statements zu formulieren, durch die das folgende lebende Bild auf dem Bildschirm erscheint:
    Das Zeichen mit dem ASCII-Code 2 („glückliches Gesicht") soll sich diagonal über den Bildschirm bewegen.

Antwort auf die Testübung 11.3.1

(0,8)-(319,15)

11.4 Das Schießbudenspiel

In diesem Abschnitt wollen wir uns mit einem Computerspiel beschäftigen, das wir „Schießbudenspiel" nennen wollen, weil es das Treiben in den Schießbuden auf den Rummel- und Volksfestplätzen nachahmt. Dem Spieler wird „ein Gewehr in die Hand gegeben", mit dem er auf eine sich bewegende Zielscheibe „feuern" kann. Solche Spiele gibt es in mannigfachen Variationen auf den Rummelplätzen. Die Grundkonstellation unseres Computerspieles ist in der Abb. 11.10 dargestellt. Im Verlaufe unseres Spieles erscheinen nacheinander 20 Zielscheiben auf dem Bildschirm.

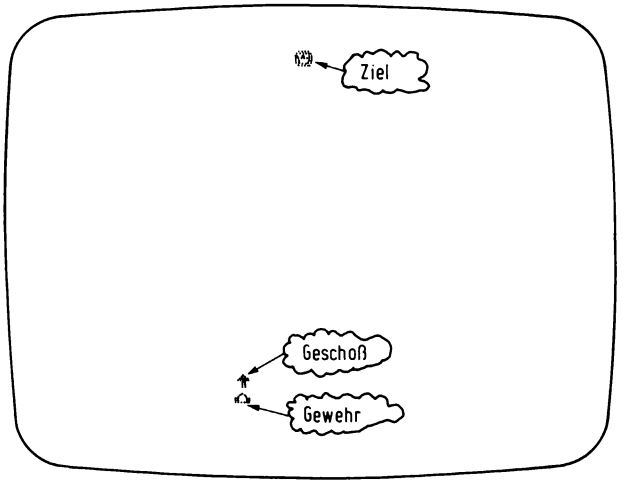


Abb. 11.10 Ausgangsstellung des Schießbudenspieles

Analysieren wir nun das Programm für das Schießbudenspiel; es ist in der Abb. 11.11 dargestellt.

Die Programmierung des Schießbudenspieles erfordert die Anwendung fast aller bisher erworbenen Kenntnisse. Wir beginnen sinnvollerweise damit, das Auffangen von Ereignissen für drei Cursorbewegungstasten (Bewegung nach oben, nach links und nach rechts) einzurichten. Das Drücken der Tasten für die Cursorbewegungen nach links und nach rechts entspricht den Bewegungen des Gewehrs nach links und rechts. Dem Anschlagen der Taste für die Cursorbewegung nach oben kommt dem Abschießen des Gewehres gleich. In den Zeilen mit den Zeilennummern 10 bis 80 nehmen wir diese Initialisierung, d.h. die Schaffung der soeben beschriebenen Ausgangsbedingungen, für das Spiel vor.

Die Anzeigen auf dem Bildschirm erfolgen im graphischen Modus mittleren Auflösungsvermögens. Das Gewehr wird zu Beginn des Spieles in der Mitte der letzten verfügbaren Textzeile plaziert. Im Augenblick des Abfeuerns befindet sich folglich das Geschoß in der Zeile 184. Die jeweilige horizontale Position des Gewehres halten wir in der Variablen GEWEHRPOS fest, die jeweilige Lage des Geschosses in den Variablen GESCHOSSPALTE und GESCHOSSZEILE (horizontale bzw. vertikale Koordinate). In der Zeile mit der Zeilennummer 90 sind die Anfangswerte für die Gewehrstellung und für das Geschoß festgelegt.

Die Darstellung der Spielfiguren muß natürlich durch irgendwelche von uns ausgewählte Symbole erfolgen. Für das Gewehr entscheiden wir uns für das Symbol „kleines Haus" (ASCII-Verschlüsselung 127, siehe Abb.7.1); wir markieren gewissermaßen dadurch den Schießstand. Das Geschoß wird durch den Vertikalpfeil (ASCII-Verschlüsselung 24, siehe Abb.7.7) und die Zielscheibe durch das Symbol „glückliches Gesicht" (ASCII-Verschlüsselung 2, siehe Abb.7.7) dargestellt. Alle Figuren müssen „lebend" sein; infolgedessen werden wir sie in Bereiche (A%, B% und C%) stellen. Dies geschieht durch die Statements auf den Zeilen mit den Zeilennummern 100 bis 210. Die Benutzung des Prozentzeichens bei den Variablennamen bedeutet, daß die Variablen nur ganzzahlige Werte annehmen können. Die Beschränkung auf diesen Zahlentyp für die Variablen hat zur Folge, daß das Programm schneller ausgeführt wird. Diese Vorkehrung ist sehr zu empfehlen, da bei Benutzung von BASIC die „lebenden Figuren" auf dem Bildschirm dazu neigen, sich sehr langsam zu bewegen.

Das Statement in der Zeile mit der Zeilennummer 220 plaziert das Gewehr in seine Ausgangsposition. Das Hauptprogramm umfaßt die Anweisungen in den Zeilen, deren Zeilennummern 230 bis 330 lauten. Es besteht aus einer äußeren Schleife, die für die aufeinanderfolgen-

```
10  'Initialisierung (Zuweisung von Anfangswerten)
20      KEY OFF
30      ON KEY(11)  GOSUB 590:      'Cursorbewegung nach oben
40      ON KEY(12)  GOSUB 530:      'Cursorbewegung nach links
50      ON KEY(13)  GOSUB 470:      'Cursorbewegung nach rechts
60      KEY(11) ON:     'Auffangen der Cursorbewegung nach oben
70      KEY(12) ON:     'Auffangen der Cursorbewegung nach links
80      KEY(13) ON:     'Auffangen der Cursorbewegung nach rechts
90      GEWEHRPOS = 160:      GESCHOSSZEILE = 185
100     DIM A%(100), B%(100), C%(100)
110     SCREEN 1,0
120     CLS
130     PRINT CHR$(2):      'Anzeige der Zielscheibe
140     GET (0,0)-(7,7),A%
150     CLS
160     PRINT CHR$(127):    'Anzeige des Gewehres
170     GET (0,0)-(7,7),B%
180     CLS
190     PRINT CHR$(24):     'Anzeige des Geschosses
200     GET (0,0)-(7,7),C%
210     CLS
220     PUT (GEWEHRPOS,185),B%
230  Schleife des Hauptprogrammes (20-malige Wiederholung)
240     FOR  ZIELSCHEIBE=1 TO 20
250       PUT (0,8),A%
260       FOR  SPALTE=2 TO 312  STEP 2
270         GOSUB 340:      'Bewegung der Zielscheibe
280         GOSUB 380:      'Bewegung des Geschosses
290       NEXT SPALTE
300       IF  SPALTE=316  THEN 320
310       PUT (312,8),A%
320     NEXT ZIELSCHEIBE
330 END
```

Abb. 11.11 Programm für das Schießbudenspiel (1. Teil) – Auflistung mit Matrixdrucker –

```
340 'Bewegung der Zielscheibe
350     PUT (SPALTE-2,8),A%
360     PUT (SPALTE,8),A%
370  RETURN
380 'Bewegung des Geschosses
390     IF  GMARKE=0  THEN 460
400     PUT (GESCHOSSPALTE,GESCHOSSZEILE),C%
410     GESCHOSSZEILE = GESCHOSSZEILE - 8
420     IF  GESCHOSSZEILE<10  THEN GOSUB 670  ELSE 450
430     GMARKE = 0
440     GOTO 460
450     PUT (GESCHOSSPALTE,GESCHOSSZEILE),C%
460  RETURN
470 'Bewegung des Gewehres um 8 Schritte nach rechts  -Cursorbewegung rechts-
480     PUT (GEWEHRPOS,185),B%
480     PUT (GEWEHRPOS,185),B%
490     GEWEHRPOS = GEWEHRPOS + 8
500     IF  GEWEHRPOS>311  THEN GEWEHRPOS = 311
510     PUT (GEWEHRPOS,185),B%
520  RETURN
530 'Bewegung des Gewehres um 8 Schritte nach links   -Cursorbewegung links-
540     PUT (GEWEHRPOS,185),B%
550     GEWEHRPOS = GEWEHRPOS - 8
560     IF  GEWEHRPOS<0  THEN GEWEHRPOS = 0
570     PUT (GEWEHRPOS,185),B%
580  RETURN
590 'Schiessen (Schussabgabe)  -Cursorbewegung nach oben-
600     IF  GMARKE=1  THEN 650
610     GMARKE = 1
620     GESCHOSSPALTE = GEWEHRPOS
630     GESCHOSSZEILE = 177
640     PUT (GESCHOSSPALTE,GESCHOSSZEILE),C%
650  RETURN
660 'Trefferbestimmung (Zielscheibe getroffen?)
670     IF  ABS(GESCHOSSPALTE-SPALTE)<7  THEN GOSUB 690
680  RETURN
690 'Loeschen der Zielscheibe und des Geschosses
700     PUT (SPALTE,8),A%
710     BEEP
720     TREFFERZAHL = TREFFERZAHL + 1
730     LOCATE 1,1
740     PRINT "Zahl der Treffer: "; TREFFERZAHL
750     SPALTE = 314
760  RETURN
```

Abb. 11.11 Programm für das Schießbudenspiel (2. Teil) - Auflistung mit Matrixdrucker -

de Erzeugung der 20 vorgesehenen Zielscheiben verantwortlich ist; in diese ist eine innere Schleife eingebettet. Jeder Durchlauf der inneren Schleife bewegt die Zielscheibe zwei Spalten quer über den Bildschirm und das Geschoß, sofern es abgefeuert wurde, acht Zeilen aufwärts. Wenn der Schuß mittels Anschlagen der Taste zur Bewegung des Cursors nach oben abgefeuert wird, wird die Programmausführung unterbrochen, und die „Schußabgaberoutine" (Beginn auf der Zeile mit der Zeilennummer 590) angesteuert. Dadurch erscheint das Geschoß in seiner Anfangslage, d. h. in der Lage nach Verlassen des Gewehrlaufes. Alle nachfolgenden Geschoßbewegungen werden durch das Hauptprogramm gesteuert. Das Geschoß verschwindet vom Bildschirm, wenn es die Zeile erreicht hat, in der sich die Zielscheibe befindet. Diese hingegen verlischt, wenn sie beim Bewegen auf das rechte Ende des Bildschirmes

gestoßen ist. Nehmen Geschoß und Zielscheibe zum gleichen Zeitpunkt eine gemeinsame Position ein, so werden beide Figuren gelöscht und der Schütze bekommt einen Punkt gutgeschrieben.

Die uns bereits bekannte Anweisung BEEP (siehe Kap. 8) läßt den Lautsprecher ertönen, wenn es zu einem Treffer gekommen ist. – Es sollte auch noch auf die Verwendung der Absolutfunktion ABS(X) in der Zeile mit der Zeilennummer 650 hingewiesen werden. Durch ABS(X) ist der vorzeichenfreie Wert der Variablen X bestimmt. Beispielsweise ergibt ABS(+5) den gleichen Wert wie ABS(−5), nämlich 5.

Aufgabengruppe 42

1. Das in Abb. 11.11 dargestellte Programm für das Schießbudenspiel sollte man mehrere Male ablaufen lassen, um ein Gefühl für seine Wirkungsweise zu bekommen.
2. Das in Abb. 11.11 dargestellte Programm für das Schießbudenspiel soll in der Hinsicht modifiziert werden, daß die Geschoßgeschwindigkeit um das zweifache vergrößert wird. Dadurch läßt es sich leichter spielen.
3. Das in Abb. 11.11 dargestellte Programm für das Schießbudenspiel soll in der Hinsicht modifiziert werden, daß die Geschoßgeschwindigkeit auf die Hälfte verringert wird.
4. Das in Abb. 11.11 dargestellte Programm für das Schießbudenspiel soll in der Hinsicht modifiziert werden, daß jede fünfte Zielscheibe durch das Sonnensymbol (ASCII-Code 15, siehe Abb. 7.7) dargestellt wird. Die Punktezählung soll dabei ebenfalls abgeändert werden: Das Treffen einer Sonnenzielscheibe soll mit 5 Punkten bewertet werden.

11.5 Programm für das Spiel „Tic Tac Toe" mit dem Computer als Partner

Für das in diesem Abschnitt vorgelegte Programm gelten die folgenden Voraussetzungen:

- Graphischer Modus
- Erweitertes BASIC

In diesem Abschnitt wollen wir ein Programm für das traditionelle Spiel „Tic Tac Toe" vorstellen. Wir wollen hierbei nicht versuchen, dem Computer irgendeine ausgeklügelte Strategie zu unterlegen. Vielmehr wollen wir ihn gänzlich stumpfsinnig handeln lassen; er soll seine Züge nur auf einer Zufallsbasis auswählen können. Außerdem werden wir den Zufallszahlengenerator dazu benutzen, um denjenigen zu bestimmen, der den ersten Zug machen kann. Durch das ganze Programm hindurch lassen wir die vom Computer besetzten Felder durch „X" markieren, die vom Gegenspieler des Computers besetzten durch „O". Das Gesamtprogramm ist in der Abb. 11.12 aufgelistet.

Testübung 11.5.1

Auf welche Art und Weise kann der Computer bestimmen, wer den ersten Zug ausführen darf?

```
1000  'Initialisierung
1010    CLEAR:              KEY OFF
1020    SCREEN 1
1030    RANDOMIZE VAL(RIGHT$(TIME$,2))
1040    DIM A$(9)
1050    DIM B$(3,3,3)
1060    CLS:                SPIELENDE = 0
1070    PRINT "TIC TAC TOE"
1080    PRINT "DER SPIELER IST GLEICH O, DER COMPUTER GLEICH X"
1090    PRINT "DIE FELDER AUF DEM SPIELBRETT SIND WIE FOLGT"
1100    PRINT "NUMERIERT"
1110    PRINT "1"; TAB(8) "2"; TAB(16) "3"
1120    PRINT "4"; TAB(8) "5"; TAB(16) "6"
1130    PRINT "7"; TAB(8) "8"; TAB(16) "9"
1140    PRINT "DER COMPUTER WIRFT, WER BEGINNT"
1150    FOR J=1 TO 2000:  NEXT J:     'Verzoegerungsschleife
1160    IF  RND(1) > 0.5  THEN 1170  ELSE 1210
1170    PRINT "DER SPIELER BEGINNT ..."
1180    FOR J=1 TO 2000:  NEXT J:     'Verzoegerungsschleife
1190    Z=0:              'Der Spieler beginnt
1200    GOTO 1240
1210    PRINT "ICH BEGINNE ..."
1220    FOR J=1 TO 2000:  NEXT J:     'Verzoegerungsschleife
1230    Z=1:              'Der Computer beginnt
1240    PRINT "ZUM SPIELBEGINN IST IRGENDEINE TASTE ZU DRUECKEN"
1250    IF  INKEY$ = ""  THEN 1250
1260    CLS

2000  'Hauptprogramm
2010    GOSUB 3000:       'Zeichnung des Spielbrettes
2020    FOR  M=1 TO 9:    'M enthaelt die Nummer des Zuges
2030      IF  Z=0  THEN GOSUB 5000
2040      IF  Z=1  THEN GOSUB 6000
2050      Z = 1 - Z
2060      IF  SPIELENDE = 1  THEN 2100
2070    NEXT M
2080    PRINT "DAS SPIEL ENDETE OHNE GEWINNER"
2090    FOR J=1 TO 2000:  NEXT J:     'Verzoegerungsschleife
2100    CLS
2110    LOCATE 1,1
2120    INPUT "EIN WEITERES SPIEL? (J/N): "; E$
2130    IF  E$="J" OR E$="j"  THEN 1060  ELSE END

3000  'Zeichnung des Spielbrettes
3010    CLS
3020    LINE (103,8)-(103,91)
3030    LINE (206,8)-(206,91)
3040    LINE (8,70)-(311,70)
3050    LINE (8,132)-(311,132)
3060  RETURN
```

Abb. 11.12 Programm für das Spiel „Tic Tac Toe" (Teil 1)

```
4000  'Anzeige des augenblicklichen Spielstandes
4010    LOCATE 5,7:        PRINT A$(1);
4020    LOCATE 5,20;       PRINT A$(2);
4030    LOCATE 5,33:       PRINT A$(3);
4040    LOCATE 14,7:       PRINT A$(4);
4050    LOCATE 14,20:      PRINT A$(5);
4060    LOCATE 14,33:      PRINT A$(6);
4070    LOCATE 21,7:       PRINT A$(7);
4080    LOCATE 21,20:      PRINT A$(8);
4090    LOCATE 21,33:      PRINT A$(9);
4100  RETURN

5000  'Zug des Spielers
5010    LOCATE 1,1
5020    INPUT "EINGABE DES NAECHSTEN ZUGES (1 BIS 9): "; S
5030    IF  A$(S)=""  THEN 5090
5040    LOCATE 1,1
5050    LINE (0,0)-(319,7),0,BF:   'Loeschen der ersten Zeile
5060    PRINT "UNGUELTIGER ZUG"
5070    FOR J=1 TO 2000:  NEXT J:     'Verzoegerungsschleife
5080    GOTO 5000
5090    A$(S) = "O"
5100    LINE (0,0)-(319,7),0,BF:   'Loeschen der ersten Zeile
5110    GOSUB 7000:        'Ist das Spiel vorbei?
5120    GOSUB 4000:        'Anzeige des Zuges
5130  RETURN

6000  'Zug des Computers
6010    LOCATE 1,1
6020    PRINT "HIER IST MEIN ZUG"
6030    GOTO 8000:         'Handelt es sich um einen Gewinnzug?
6040    'Liegt kein Gewinnzug vor, dann Ausfuehrung eines Zuges
         auf Zufallsbasis
6050    S = INT(9*RND+1)
6060    IF  A$(S)=""  THEN 6070  ELSE 6050
6070    A$(S) = "X"
6080    FOR J=1 TO 2000:  NEXT J:     'Verzoegerungsschleife
6090    GOSUB 7000:        'Ist das Spiel vorbei?
6100    GOSUB 4000:        'Anzeige des Zuges
6110  RETURN
```

Abb. 11.12 Programm für das Spiel „Tic Tac Toe" (Teil 2)

```
7000  'Entscheidung darueber, ob das Spielende erreicht ist
7010    IF  Z=0  THEN C$="O"  ELSE C$="X"
7020    IF  A$(1)=A$(2)  THEN 7030  ELSE 7050
7030    IF  A$(2)=A$(3)  THEN 7040  ELSE 7050
7040    IF  A$(3)=C$  THEN 7260
7050    IF  A$(1)=A$(4)  THEN 7060  ELSE 7080
7060    IF  A$(4)=A$(7)  THEN 7070  ELSE 7080
7070    IF  A$(7)=C$  THEN 7260
7080    IF  A$(1)=A$(5)  THEN 7090  ELSE 7110
7090    IF  A$(5)=A$(9)  THEN 7100  ELSE 7110
7100    IF  A$(9)=C$  THEN 7260
7110    IF  A$(2)=A$(5)  THEN 7120  ELSE 7140
7120    IF  A$(5)=A$(8)  THEN 7130  ELSE 7140
7130    IF  A$(8)=C$  THEN 7260
7140    IF  A$(3)=A$(6)  THEN 7150  ELSE 7170
7150    IF  A$(6)=A$(9)  THEN 7160  ELSE 7170
7160    IF  A$(9)=C$  THEN 7260
7170    IF  A$(4)=A$(5)  THEN 7180  ELSE 7200
7180    IF  A$(5)=A$(6)  THEN 7190  ELSE 7200
7190    IF  A$(6)=C$  THEN 7260
7200    IF  A$(7)=A$(8)  THEN 7210  ELSE 7230
7210    IF  A$(8)=A$(9)  THEN 7220  ELSE 7230
7220    IF  A$(9)=C$  THEN 7260
7230    IF  A$(3)=A$(5)  THEN 7240  ELSE 7320
7240    IF  A$(5)=A$(7)  THEN 7250  ELSE 7320
7250    IF  A$(7)=C$  THEN 7260  ELSE 7320
7260    GOSUB 4000
7270    LOCATE 1,1
7280    PRINT SPACE$(80)
7290    LOCATE 1,1
7300    PRINT C$, "GEWINNT DIESES SPIEL":     SPIELENDE = 1
7310    FOR J=1 TO 2000:  NEXT J:     'Verzoegerungsschleife
7320  RETURN
```

Abb. 11.12 Programm für das Spiel „Tic Tac Toe" (Teil 3)

```
8000  'Nachsehen , ob ein Gewinnzug vorliegt
8010    ZAEHLER = 0
8020    FOR  I=1 TO 9
8030      IF  A$(I)="X"  THEN B(I)=1
8040      IF  A$(I)=""   THEN B(I)=0
8050      IF  A$(I)="O"  THEN B(I)=-1
8060    NEXT I
8070    READ I,J,K
8080    ZAEHLER = ZAEHLER + 1
8090    IF  ZAEHLER=8  THEN 8180
8100    S = B(I) + B(J) + B(K)
8110    IF  S=2  THEN 8120  ELSE 8070
8120    IF  B(J)=0  THEN A$(J)="X"  ELSE 8140
8130    GOTO 8310
8140    IF  B(K)=0  THEN A$(K)="X"  ELSE 8160
8150    GOTO 8310
8160    IF  B(I)=0  THEN A$(I)="X"  ELSE 8070
8170    GOTO 8310
8180    RESTORE
8190    ZAEHLER = 0
8200    READ I,J,K
8210    ZAEHLER = ZAEHLER + 1
8220    S = B(I) + B(J) + B(K)
8230    IF  ZAEHLER=8  THEN 8320
8240    IF  S=-2  THEN 8250  ELSE 8200
8250    IF  B(J)=0  THEN A$(J)="X"  ELSE 8270
8260    GOTO 8310
8270    IF  B(K)=0  THEN A$(K)="X"  ELSE 8290
8280    GOTO 8310
8290    A$(I) = "X"
8300    GOTO 8310
8310    RESTORE:          GOTO 6080
8320    RESTORE:          GOTO 6040
8330    DATA 1,2,3,4,5,6,7,8,9,1,4,7,2,5,8,3,6,9,1,5,9,3,5,7
```

Anmerkung: Die in der Auflistung vorhandenen Leerzeilen sind nicht Bestandteil des Programmes; sie sind nur der Übersichtlichkeit und der leichteren Lesbarkeit wegen eingefügt.

Abb. 11.12 Programm für das Spiel „Tic Tac Toe" (Teil 4)

Nach den für das Spiel erforderlichen Initialisierungen (u.a. Bestimmung des Beginners) zeichnet der Computer auf dem Bildschirm ein Spielbrett (Abb. 11.13).

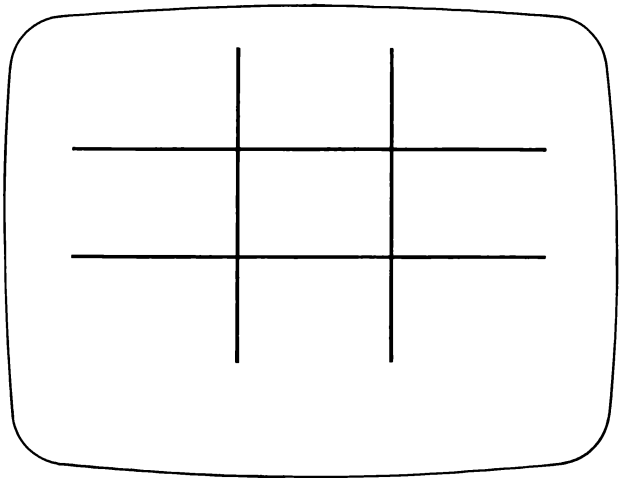
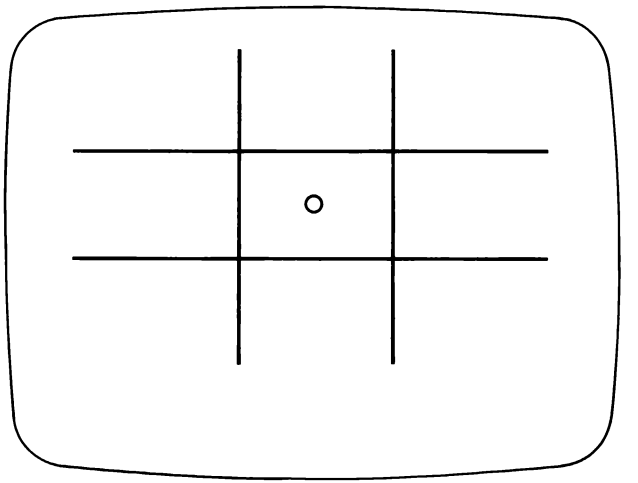


Abb. 11.13 Schematische Darstellung des Spielbrettes für das Spiel „Tic Tac Toe"

Danach zeigt der Computer den ersten Spielzug seines Mitspielers an und tätigt einen eigenen, sofern der Mitspieler beginnen darf (Abb. 11.14).



Anmerkung: Vom Spieler wurde 5 eingegeben

Abb. 11.14 Schematische Darstellung des Spielbrettes nach Beginn des Spiels (Beispiel)

Nun wechseln Mitspieler und Computer einander mit den weiteren Spielzügen solange ab, bis einer gewonnen hat oder sich ein unentschiedener Spielausgang ergibt.

Die im Programm (Abb. 11.12) verwendeten Variablen haben die folgenden Bedeutungen:

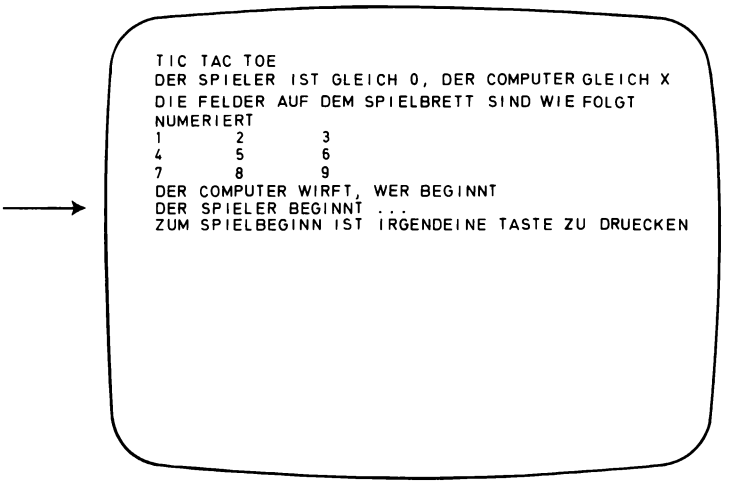
- Z — Schalter, der denjenigen kenntlich macht, der am Zuge ist:
  - Z=1 → Der Computer ist am Zug
  - Z=0 → Der Mitspieler ist am Zug
- A$(J) — Anzeiger für den Spielstand mit J=1,2,···,9
  - A(J)="X" → Das betreffende Feld ist vom Computer besetzt
  - A(J)="O" → Das betreffende Feld ist vom Mitspieler besetzt
  - A(J)="" → Das betreffende Feld ist noch unbesetzt
- S — Position des augenblicklichen Zuges
- M — Anzahl der gespielten Züge, einschließlich des augenblicklich getätigten

Zur Verfolgung des Spielprogrammes benutzen wir am zweckmäßigsten ein Arbeitsblatt für Bildschirmentwürfe und tragen in dieses die Koordinaten der Punkte bzw. der Spielfelder ein und markieren die besetzten Felder durch die Buchstaben X bzw. O.

Wenn das Programm unter dem Namen "TICTAC" auf der Diskette im Laufwerk A gespeichert ist, so hat man zwecks Ausführung des Programmes nacheinander die folgenden Befehle einzugeben:

```
LOAD "TICTAC"
RUN
```

Auf dem Bildschirm erscheint durch den Initialisierungsabschnitt zunächst das in der Abb. 11.15 dargestellte Bild.



Anmerkung: oder ICH BEGINNE

Abb. 11.15 Startbild des Spieles „Tic Tac Toe"

Aufgabengruppe 43

1. Das Programm für das Spiel „Tic Tac Toe" ist so abzuändern, daß der Computer mit seinem Spielpartner nacheinander zehn Spiele durchführen kann. Nach dieser Spielserie soll das Programm anzeigen, wer der Gesamtgewinner ist.
2. Das Programm für das Spiel „Tic Tac Toe" ist so abzuändern, daß „Tic Tac Toe" auf einem Spielbrett mit 4 × 4 Feldern gespielt werden kann.

Antwort auf die Testübung 11.5.1

Die Antwort findet sich in den Zeilen 1140 bis 1230 des in der Abb. 11.12 vorgelegten Programmes.

12 Zahlenarten in BASIC bei den IBM Personalcomputern

In diesem Kapitel wollen wir über die verschiedenen Zahlenarten sprechen, die in die Programmiersprache BASIC eingegangen sind. Darüberhinaus werden wir die Bibliothek der *eingefügten* (eingebauten) mathematischen Funktionen behandeln; diese in Programmen einzusetzen, wird vielfach zur unumgänglichen Notwendigkeit.

12.1 Einfache und doppelte Genauigkeit bei Zahlendarstellungen

Wir haben uns bis jetzt kaum Gedanken darüber gemacht, mit welcher *Genauigkeit* die arithmetischen Operationen von den Computern ausgeführt werden. Es ist jedoch bei wissenschaftlich-technischer Programmierung geradezu wichtig, daß man über die Zahl der Dezimalstellen eine Aussage treffen kann, die bei den Ergebnissen von Berechnungen als noch richtig betrachtet werden können. Die Erörterungen darüber wollen wir damit beginnen, daß wir uns mit den Formen beschäftigen, in der bei BASIC Zahlen gespeichert und nutzbar gemacht werden.

BASIC erkennt drei verschiedene Arten von numerischen Konstanten:

- *ganzzahlige* Konstanten
- Konstanten mit *einfacher Genauigkeit*
- Konstanten mit *doppelter* (oder erhöhter) *Genauigkeit*

a) *Ganzzahlige Konstanten*
Unter einer *ganzzahligen Konstanten* versteht man eine positive bzw. negative ganze Zahl im Wertebereich von −32768 bis +32767. Die Zahl 32768 ist bekanntlich gleich der 15. Potenz von 2; diese Zahl ist bezeichnend für die interne Arbeitsweise der IBM Personalcomputer. Einige Beispiele für ganzzahlige Konstanten sollen folgen:

7, 58, 3782, −16, 3407, −9724, 13528

Ganzzahlige Konstanten können im RAM äußerst wirkungsvoll gespeichert werden. Die mit ihnen ausgeführten arithmetischen Operationen beanspruchen überdies die geringste Zeit. Um diese augenscheinlichen Vorteile zu realisieren, kann deshalb der IBM Personalcomputer ganzzahlige Konstanten erkennen und mit ihnen in einer besonderen Weise umgehen.

b) *Konstanten mit einfacher Genauigkeit*
Unter einer *Konstanten mit einfacher Genauigkeit* versteht man eine Zahl mit sieben oder weniger gültigen Ziffern, die keine ganzzahlige Konstante darstellt. Beispiele für Konstanten mit einfacher Genauigkeit sind:

13.0528, −97.1924, 1234567, −1.467654E12

Wie an dem letzten Beispiel zu ersehen ist, können Konstanten mit einfacher Genauigkeit auch in der „wissenschaftlichen" Schreibweise dargestellt werden; diese wird auch als „Gleitkomma- oder Gleitpunktdarstellung" bezeichnet. Die Anzahl der Ziffern ist natürlich auch bei dieser Schreibweise ebenfalls auf sieben oder weniger beschränkt; der Exponent nach dem Kennbuchstaben E mit seinen ein oder zwei Ziffern ist hierbei nicht zu berück-

sichtigen. Bei der Programmiersprache BASIC der IBM Personalcomputer sind die Konstanten mit einfacher Genauigkeit auf die beiden folgenden Wertebereiche beschränkt:

$$-1 \cdot 10^{38} \quad \text{bis} \quad -1 \cdot 10^{-38}$$
$$1 \cdot 10^{-38} \quad \text{bis} \quad 1 \cdot 10^{38}$$

Hinzu kommt noch die Zahl 0. Diese Begrenzungen stellen für praktische Berechnungen höchst selten eine Einschränkung dar. Man halte sich als Beispiel vor Augen, daß

$$1 \cdot 10^{38} = 100000000000000000000000000000000000000$$

Hinter der 1 folgen also 38 Nullen; das sollte man nie vergessen. Eine solche Zahl dürfte für die meisten praktischen Berechnungen groß genug sein.

c) *Konstanten mit doppelter (oder erhöhter) Genauigkeit*
Unter einer Konstanten mit doppelter Genauigkeit versteht man eine Zahl mit mehr als sieben gültigen Ziffern. Beispiele hierfür sind:

9.0719241928, −34598636, −6142.7286456

Die wissenschaftliche Schreibweise darf ebenso benutzt werden, um Konstanten mit doppelter Genauigkeit darzustellen: Man beachte jedoch, daß dem Exponenten der Buchstabe D vorangehen muß. So kann z. B. die Konstante 0.00027575757575 auch zu

2.7575757575D-4

geschrieben werden. Ein zweites Beispiel möge die Betrachtung abschließen:

1314159265350000 ≙ 1.31415926535D15

Konstanten mit doppelter Genauigkeit können bis zu 17 Ziffern enthalten. Sie sind denselben Beschränkungen hinsichtlich der Wertebereiche unterworfen wie die Konstanten mit einfacher Genauigkeit.

Konstanten mit einfacher Genauigkeit belegen mehr Speicherplatz im RAM als ganzzahlige Konstanten. Außerdem laufen die arithmetischen Operationen mit ihnen langsamer ab als die mit ganzzahligen Konstanten. Konstanten mit doppelter Genauigkeit belegen, was eigentlich selbstverständlich ist, sogar noch mehr Speicherplatz im RAM als die Konstanten einfacher Genauigkeit. Die arithmetischen Operationen mit ihnen sind ferner langsamer als die gleichen Operationen mit Konstanten, die nur die einfache Genauigkeit aufweisen. Die IBM Personalcomputer erkennen diese drei Arten numerischer Konstanten und setzen folgerichtig stets nur die notwendigen arithmetischen Fähigkeiten ein.

Eine Zusammenfassung der für die Bestimmung des Typs numerischer Konstanten geltenden Regeln soll diese Betrachtung abschließen.

1. Jede ganze Zahl im Wertebereich von −32768 bis 32767 gilt als ganzzahlige Konstante
2. Jede Zahl mit sieben oder weniger Ziffern, die nicht eine ganzzahlige Konstante darstellt, ist eine Konstante mit einfacher Genauigkeit. Jede Zahl in der wissenschaftlichen Schreibweise, bei der dem Exponenten ein E vorausgeht, wird als Konstante mit einfacher Genauigkeit angesehen. Besitzt eine Konstante in der wissenschaftlichen Schreibweise jedoch mehr als sieben Ziffernstellen, so wird sie als Konstante mit doppelter Genauigkeit angesehen, auch wenn der Exponent durch ein E eingeleitet wird. Z. B. wird die Konstante

   1.23456789E15

   als Konstante mit doppelter Genauigkeit interpretiert, d. h. zu

   1.23456789D15

3. Eine Zahl mit mehr als sieben Ziffern wird als Konstante mit doppelter Genauigkeit angesehen. Wenn mehr als 17 Ziffern niedergeschrieben sind, dann werden die Ziffern abgeschnitten, die nach der 17. Ziffer erscheinen: außerdem erfolgt eine Umformung in die wissenschaftliche Schreibweise. Beispielsweise wird die Zahl

   123456789123456789

   als doppelt genaue Konstante

   1.2345678912345678D17

   interpretiert.

Der Typ einer numerischen Konstanten kann mittels eines „Typerklärungsanhanges" festgelegt werden. Wenn beispielsweise eine numerische Konstante mit dem Anhängsel % versehen wird, wird sie als ganzzahlige Konstante betrachtet. Das Prozentzeichen wird jedoch bei Zahlen, die einen Dezimalpunkt enthalten, ignoriert. So wird z. B. 1.85% als einfach genaue Konstante 1.85 angesehen.

Wenn die Konstante, die mit einem Prozentzeichen abgeschlossen ist, zu groß für eine ganzzahlige Konstante, d. h. für eine Konstante im Wertebereich −32768 bis +32767, ist, tritt ein Überlauffehler (Overflow) auf.

Eine numerische Konstante, die mit dem Ausrufungszeichen abgeschlossen ist, wird als einfach genaue Konstante interpretiert und entsprechend gerundet. Z. B. wird die Konstante

−1.23456789!

als Konstante

−1.234567

interpretiert. Die Konstante

123456789!

wird auf sieben gültige Ziffern gerundet und in die wissenschaftliche Form umgewandelt, d. h. in

1.234568E9

Wird als Anhängsel das Nummernzeichen # benutzt, so wird die damit versehene Konstante als doppelt genaue Konstante ausgelegt. Beispielsweise wird die Konstante

1.2#

als doppelt genaue (17-stellige) Konstante angesehen:

1.2000000000000000

Bei der wissenschaftlichen Schreibweise dient, zur Wiederholung sei es noch einmal gesagt, das den Exponenten einleitende D nach wie vor als Kennzeichen für diesen Typ.

Ein Typerklärungsanhang setzt jedoch in jedem Fall die weiter oben beschriebenen Regeln 1. bis 3. für die Typerkennung von numerischen Konstanten außer Kraft.

Testübung 12.1.1

Die folgenden numerischen Konstanten sind in der Dezimalform niederzuschreiben!

a) −7.5%
b) 4.589234501836 49E12
c) 270D-2
d) 12.55#
e) −1.62!

Wenden wir nun unsere Aufmerksamkeit den arithmetischen Operationen zu, in die verschiedene Konstantentypen eingehen, und fragen uns, wie BASIC diese handhabt. Der Typ der Variablen, die das Ergebnis einer arithmetischen Operation aufnimmt, ist durch den Typ der Variablen bestimmt, deren Werte in die Operation eingehen. Beispielsweise ergibt sich als Summe von zwei ganzzahligen Konstanten eine ganzzahlige Konstante, sofern das Ergebnis im zulässigen Wertebereich ganzzahliger Konstanten liegt. Ist das nicht der Fall, erscheint die Summe als einfach genaue Konstante. Arithmetische Operationen mit einfach genauen Konstanten liefern stets einfach genaue Ergebnisse. Gehen sie hingegen in arithmetische Operationen mit doppelt genauen Konstanten ein, so ergeben sich doppelt genaue Resultate. Durch einige Beispiele wollen wir die soeben durchgeführten Besprechungen erhärten:

1. 5% + 7%
   Die beiden ganzen Zahlen 5 und 7 werden addiert und führen zur ganzzahligen Konstanten 12 als Resultat.

2. 4.21! + 5.2!
   Die Addition der beiden einfach genauen Konstanten ergibt das einfach genaue Resultat 9.41.

3. 3/2
   Die Division der beiden ganzen Zahlen 3 durch 2 ergibt einen Quotienten, der nicht ganzzahlig ist. Folglich ist der Typ des Resultats einfach genau.

4. 1/3
   Die Division führt hier zum einfach genauen Quotienten 0.3333333.

5. 1#/3#
   In diesem Fall besitzt der Quotient die doppelte Genauigkeit, d.h. er wird zu 0.33333333333333333 gebildet. Dividend und Divisor sind hier nämlich als doppelt genaue Konstanten vorgegeben.

Testübung 12.1.2

Welches Resultat errechnet der Computer bei den folgenden Aufgaben?

a) 2/5 + 1/3
b) 0.4% + 0.33333333333333333333%
c) 0.4# + 0,3333333333333333333#
d) 0,4! + 0,3333333333333333333!

Wenn Zahlen keine exakte dezimale Darstellung besitzen, was z. B. beim Bruch $\frac{1}{3} = 0{,}\bar{3}\ldots$ der Fall ist, oder wenn sie mehr gültige Ziffern aufweisen, als der für sie vorgesehene Konstantentyp zuläßt, gehen in die vom Computer auszuführenden Rechenoperationen nur Näherungswerte ein und nicht die Zahlen selbst. Die durch die Benutzung von Näherungswerten verursachten, nicht zu vermeidenden Fehler werden „Rundungsfehler" genannt. Hierzu wollen wir ein kleines Problem betrachten, nämlich die Berechnung des Ausdrucks

1/3 + 1/3 + 1/3

Wir wissen bereits, daß der Wert von 1/3 gleich der einfach genauen Konstanten 0.3333333 ist. Der Computer bildet also die Summe zu

0.3333333 + 0.3333333 + 0.3333333 = 0.9999999

Der Rundungsfehler beträgt hier also 0.0000001.

Bei den IBM Personalcomputern werden bei einfach genauen numerischen Konstanten bis zu sieben Ziffern angezeigt. Bedingt durch die unvermeidlichen Rundungsfehler ist jedoch das Ergebnis jeder einfachen arithmetischen Operation nur bis auf sechs Ziffernstellen genau gewährleistet. Doppelt genaue numerische Konstanten werden mit einer Rundung auf die 16. Ziffernstelle angezeigt bzw. ausgegeben. Die vom Computer ausgeführten einfachen arithmetischen Operationen führen demnach zu Ergebnissen, die akkurat bis auf die 16. Ziffernstelle sind. Wenn nun nacheinander viele einfache Rechenoperationen auszuführen sind, dann kann es vorkommen, daß sich die Rundungsfehler der einzelnen Operationen so aufschaukeln, daß auch noch die 15. Ziffernstelle oder sogar noch eine frühere nicht mehr als korrekt angesehen werden können.

•

Aufgabengruppe 44

Zu jeder der nachstehenden Konstanten ist die Zahl zu nennen, die vom Computer gespeichert wird!

1.	3	2.	2,37
3.	5,78E5	4.	2#
5.	3!	6.	−4.1!
7.	−4.1%	8.	3500.6847586958658!
9.	2.176D2	10.	−5.94E12
11.	3.5869504003837265374	12.	−234542383746.21
13.	−2.367D20	14.	457000000000000000

Bei jeder der nachfolgenden Rechenaufgaben ist das Ergebnis zu nennen, das vom Computer gebildet wird!

15.	1 + 45	16.	2/4
17.	3#/5#	18.	3!/5! + 1
19.	2/3	20.	2#/3# + 0.53#
21.	2/3	22.	2/3 + 0.53
23.	0.5E4 − 0.37E2	24.	1.75D3 − 1.0D-5

25. Bei jeder der Aufgaben 15. bis 24. ist anzugeben, wie der Computer das Resultat anzeigt.

26. Es ist die Summe von

    1/3 + 1/3 + + 1/3

    zu bestimmen; dabei liegen 1000 Summanden 1/3 vor. Welches Ergebnis wird gebildet? Ist das Ergebnis genau bis auf die sechste Ziffernstelle? Wenn nicht, so ist der Grund dafür zu erklären!

27. Es liegt dieselbe Problemstellung wie bei Aufgabe 26. vor, jedoch bezogen auf doppelt genaue Konstanten mit 17 gültigen Ziffernstellen.

Antworten auf die Testübungen

12.1.1 **a)** −7.5
b) 4589235000000
c) 2.7000000000000000
d) 12.550000000000000
e) −1.620000

12.1.2 **a)** 0.7333333
b) 0
c) 0.73333333333333333
d) 0.7333333

12.2 Variablentypen

Im vorhergehenden Abschnitt führten wir die verschiedenen Arten oder Typen von numerischen Konstanten ein:

- ganzzahlige Konstanten
- einfach genaue Konstanten
- doppelt genaue Konstanten

Zu diesen Konstantentypen gibt es die entsprechenden Variablentypen, die wir im Kapitel 11 bei den Computerspielen schon einmal andeutungsweise gestreift hatten.

a) *Ganzzahlige Variablen*
Eine Variable des ganzzahligen Typs nimmt nur Werte an, die ganzzahlige Konstanten darstellen. Variable dieses Typs werden durch ein an den Variablennamen angehängtes Prozentzeichen gekennzeichnet. Beispiele hierfür sind:

A%, BB%, 51%

Bei Zuweisungen von Werten zu Variablen des ganzzahligen Typs rundet der Computer die gebrochenen Anteile der Werte, so daß ganze Zahlen entstehen. So wird z.B. durch die Anweisung

```
10 A% = 2.54
```

der Variablen A% die ganzzahlige Konstante 3 zugewiesen. Variablen vom ganzzahligen

Typ sind nützlich, wenn es gilt, ganzzahlige Mengen, wie z.B. Lagerbestandsmengen, Personenanzahlen usw., auf dem laufenden zu halten. Dazu zählen natürlich auch die Zeilennummern von Programmen.

b) *Einfach genaue Variablen*
Die Variablen des einfach genauen Typs können nur einfach genaue Konstanten als Werte annehmen. Variablen dieses Typs tragen als Kennzeichen ein Ausrufungszeichen hinter ihrem Namen. Einige Beispiele hierfür sind:

M!, A7!, ZS!

Bei der Zuweisung eines Wertes zu einer Variablen des einfach genauen Typs erfolgt eine Rundung auf die siebente gültige Ziffer. Die Variable A! wird beispielsweise auf den Wert 1.234568 gesetzt, wenn die Anweisung

```
20  A! = 1.23456789
```

ausgeführt wird.
Wird ein Variablenname ohne Kennzeichnung niedergeschrieben, so wird vom Computer unterstellt, daß es sich um eine einfach genaue Variable handelt. Infolgedessen gehörten alle von uns bisher in Programmen benutzten Variablen zu diesem Typ. Bei weitem sind die einfach genauen Variablen diejenigen Variablen, die im allgemeinen am meisten benutzt werden.

c) *Doppelt genaue Variablen*
Die Variablen des doppelt genauen Typs nehmen nur Werte an, die doppelt genaue Konstanten darstellen. Dieser Art von Variablen kommt eine bedeutende Rolle in Berechnungen zu, die zu einer größtmöglichen Exaktheit der Ergebnisse führen müssen.
Die Variablen dieses Typs werden eingeführt, indem man dem Variablennamen ein Nummernzeichen anfügt. Beispiele hierfür sind:

M1#, BK350#, IFA#

Beim Zuweisen von Werten zu Variablen dieses Typs erfolgt eine Rundung der Werte auf die 17. Ziffernstelle.
Es sei ausdrücklich darauf hingewiesen, daß beispielsweise

A%, A!, A# und A$

vier unterschiedliche Variablen bezeichnen. Man kann, sofern man will, alle vier in einem einzigen Programm benutzen, doch empfiehlt sich eine solche Handlungsweise kaum. Die Verwirrung würde wahrscheinlich sehr groß sein. Dadurch könnte es infolge der Verwechslungsmöglichkeiten sehr leicht zu Irrtümern bzw. Fehlern kommen.

Testübung 12.2.1

Welche Werte werden den Variablen durch die folgenden Statements zugewiesen?

a) A# = 1#
b) C% = 5.22%
c) BB! = 1387.5699

Die Benutzung der Anhängsel für die Typerklärungen %, ! und # zieht einen gewissen Ärger nach sich, da diese Anhängsel immer dann niedergeschrieben werden müssen, wenn auf die betreffenden Variablen bezug genommen wird. Diese Pingeligkeit läßt sich umgehen. Durch die Vereinbarungen

- DEFINT
- DEFSNG
- DEFDBL
- DEFSTR

können die Variablentypen für das gesamte Programm festgelegt werden; dadurch erspart man sich die Benutzung der Typanhängsel. Man betrachte hierzu beispielsweise die Vereinbarung

```
100  DEFINT A
```

Durch sie wird festgelegt, daß jede Variable, deren Name mit dem Buchstaben A beginnt, also z.B. A, A1 und AB, als Variable des ganzzahligen Typs betrachtet wird. Zwei Variationen dieses Statements sollen folgen:

```
200  DEFINT A,B,C
300  DEFINT A-G
```

Durch das Statement auf der Zeile mit der Zeilennummer 200 ist festgelegt, daß alle mit den Buchstaben A, B und C beginnenden Variablennamen zu Variablen des ganzzahligen Typs gehören. Beim Statement auf der Zeile mit der Zeilennummer 300 gelten alle Variablen als Variablen des ganzzahligen Typs, deren Namen mit einem der Buchstaben A bis G anfangen. DEFINT-Vereinbarungen werden in der Regel an den Programmanfang gestellt; die sich aus ihnen ergebenden Definitionen bleiben dann für das gesamte Programm wirksam.

Die Vereinbarung DEFSNG wirkt ähnlich der Vereinbarung DEFINT, nur werden durch sie solche Variablen festgelegt, die zum einfach genauen Typ gehören sollen. Die verbleibenden, oben erwähnten Vereinbarungen DEFDBL und DEFSTR dienen schließlich dazu, Variablen des doppelt genauen Typs (DEFDBL) bzw. Variablen des Zeichenkettentyps (DEFSTR) festzulegen.

Es sei jedoch abschließend ausdrücklich darauf hingewiesen, daß die Anhängsel für Typerklärungen die Typvereinbarungen DEF... überschreiben, d.h. außer Kraft setzen. Nehmen wir zur Erläuterung dieser Regel einmal an, daß die Variable A mittels eines DEFSNG-Statements zu Programmanfang als Variable des einfach genauen Typs festgelegt wurde. Man kann dann ohne weiteres im Programm die Variable A# als Variable des doppelt genauen Typs benutzen, da das Anhängsel für die Typerklärung # die durch DEFSNG erfolgte Typdefinition überschreibt.

Es bleibt uns nur noch übrig, vor leicht zu übersehenden Fehlerquellen zu warnen. Man betrachte hierzu das Programm in Abb. 12.1; es scheint ziemlich harmlos zu sein.

```
10  LET A# = 1.7
20  PRINT A#
30  END
```

Abb. 12.1 Problematik bei Typumwandlungen

Durch das erste Statement dieses Programmes wird der doppelt genauen Variablen A# der Wert 1.7 zugewiesen; die nächste Anweisung sorgt für die Ausgabe dieses Wertes. Man erwartet dann auf dem Bildschirm höchstwahrscheinlich die Anzeige von

```
1.7000000000000000
```

Läßt man jedoch dieses Programm ausführen, so wird man über die tatsächlich erscheinende Anzeige

```
1.700000047683716
```

sehr erstaunt sein. Was ist hierbei schiefgelaufen? Das inkorrekte Resultat ergibt sich aus der Art und Weise, in der die interne Computerlogik arbeitet, sowie aus der im Computer vorliegenden Binärdarstellung der Zahlen. Ohne ins Detail zu gehen, wollen wir bloß darauf hinweisen, daß der Computer 1.7 als numerische Konstante einfacher Genauigkeit ansieht. Wird nun diese in eine Konstante doppelter Genauigkeit umgewandelt (bei dieser Operation wird aber von der Binärdarstellung von 1.7 Gebrauch gemacht), so stimmt das Resultat in den ersten 16 Ziffern mit der gegebenen Zahl überein. Bedeutet das nun, daß wir uns wegen dieser Ungereimtheit beunruhigen müssen? Natürlich nicht, denn wir brauchen, um ein korrektes Resultat zu erzielen, nur das erste Statement in der Form

```
10  LET  A# = 1.7#
```

niederzuschreiben. Die jetzt vom Programm erzeugte Ausgabe entspricht der von uns erwarteten.

Aufgabengruppe 45

Unter Benutzung der Arithmetik einfacher Genauigkeit sind die Ergebnisse der folgenden Aufgaben zu ermitteln!

1. (5.87 + 3.85 − 12.07)/11.98
2. (15.1 + 11.9)∧4/12.88
3. (32485 + 9826)/(321.5 − 87.6∧2)
4. Die Aufgabe 1. ist unter Benutzung der Arithmetik doppelter Genauigkeit zu wiederholen!
5. Die Aufgabe 2. ist unter Benutzung der Arithmetik doppelter Genauigkeit zu wiederholen!
6. Die Aufgabe 3. ist unter Benutzung der Arithmetik doppelter Genauigkeit zu wiederholen!
7. Es ist ein Programm zu schreiben, das die größte ganze Zahl bestimmt, die kleiner als oder gleich X ist. Der Wert von X ist dabei zu Programmanfang über ein INPUT-Statement einzugeben.

Man bestimme den Wert, der den Variablen bei jeder der folgenden Aufgaben zugewiesen wird!

8. A% = −5
9. A% = 4.8
10. A% = −11.2
11. A! = 1.78
12. A# = 1.78#
13. A! = 32.653426278374645237
14. A! = 4.25234544321E21
15. A! = −1.23456789E−32
16. A# = 3.28364649302927364643 4
17. A# = −5.74#

Antworten auf die Testübung 12.2.1

a) 1.0000000000000000
b) 5
c) 1387.570

12.3 Mathematische Funktionen in BASIC

Bei technisch-wissenschaftlichen Berechnungen muß man öfters eine Reihe von mathematischen Funktionen benutzen, beispielsweise die Funktion zur Bestimmung natürlicher Logarithmen, die Exponentialfunktion und die trigonometrischen Funktionen. In den IBM Personalcomputern sind eine ganze Anzahl dieser Funktionen „eingebaut". Man bezeichnet sie generell als „eingefügte Funktionen". In diesem Abschnitt wollen wir diese Funktionen kennen und gebrauchen lernen.

Alle in BASIC eingefügten mathematischen Funktionen arbeiten in der gleichen Art und Weise. Sie werden durch eine bestimmte Folge von Buchstaben, d.h. durch eine bestimmte Zeichenkette angesprochen. Für diese Zeichenketten, die als Funktionsnamen gelten, sollen nun einige Beispiele folgen:

- SIN → *Sinusfunktion*
- COS→ *Kosinusfunktion*
- LOG→ (natürliche) *Logarithmusfunktion*

Um den Funktionswert für ein Argument X zu ermitteln, muß man das Argument hinter dem Funktionsnamen in Klammern notieren. So wird z.B. der natürliche Logarithmus von X wie folgt geschrieben:

```
LOG(X)
```

Bei der Ausführung des Programmes, in dem dieser Funktionsaufruf codiert ist, wird der augenblicklich der Variablen X zugewiesene Wert als Argument benutzt und der natürliche Logarithmus dieses Wertes als Funktionswert bestimmt. Wenn also X den Wert 2 besitzt, wird somit der natürliche Logarithmus von 2 ermittelt.

Anstelle von X, einer Variablen des einfach genauen Types, kann auch jeder andere Typ von numerischen Variablen verwendet werden, also ganzzahlige Variablen genau so wie Variablen des doppelt genauen Typs. Als Argumente können ebenso auch numerische Konstanten benutzt werden; der Konstantentyp ist hierbei genau so freizügig wählbar. So bedeutet z.B.

```
SIN(0.435678889658595)
```

die Ermittlung des Sinuswertes für die in Klammern stehende doppelt genaue Konstante. Von einigen Ausnahmen abgesehen, werden die Funktionswerte in BASIC stets als Werte einfacher Genauigkeit ermittelt. Die Werte sind dabei bis auf die 6. gültige Ziffer korrekt. Der weiter oben angeforderte Wert der Sinusfunktion ergibt sich somit zu

```
SIN(0.435678889658595)=0.422026
```

In BASIC können als Argumente einer Funktion auch arithmetische Ausdrücke benutzt werden. Man betrachte z. B. den Ausdruck

```
X^2 + Y^2 - 3*X
```

Von diesem Ausdruck läßt sich auf einfache Art der Sinuswert ermitteln, indem man einfach niederschreibt:

```
SIN(X^2 + Y^2 - 3*X)
```

In diesem Fall berechnet der Computer zunächst den Wert des als Argument aufgeführten Ausdrucks, indem er die augenblicklich den Variablen X und Y zugewiesenen Werte benutzt. Wenn also z. B. X gleich 1 und Y gleich 4 ist, dann ergibt sich, wenn für den Argumentausdruck

```
X^2 + Y^2 - 3*X
```

die vorgegebenen Werte eingesetzt werden,

$$1^2 + 4^2 - 3 \cdot 1 = 14$$

Somit wird aus

```
SIN(X^2 + Y^2 - 3*X)
```

zunächst

```
SIN(14),
```

was den Sinuswert 0.9906074 ergibt.

12.3.1 Trigonometrische Funktionen

Bei den IBM Personalcomputern stehen die folgenden trigonometrischen Funktionen zur Verfügung:

- SIN(X) → Sinus des Winkels X
- COS(X) → Kosinus des Winkels X
- TAN(X) → Tangens des Winkels X

Hierbei wird X stets im Bogenmaß aufgefaßt. Bekanntlich entsprechen 360 Grad des Gradmaßes einem Bogenmaß von 2π. Aus dieser Grundbeziehung ergeben sich die folgenden Umrechnungsformeln:

- $1° = 0.017453$ rad
- 1 rad $= 57.29578°$

Die Einheit des Bogenmaßes wird hier, wie allgemein üblich, mit *rad* abgekürzt. Diese Maßeinheitabkürzung kommt vom Begriff „Radiant". Wenn man also die Werte der trigonometrischen Funktionen für einen Winkel X ausrechnen lassen will, der im Gradmaß gemessen ist, muß man infolgedessen die folgende Codierung vornehmen:

- SIN(0.017453*X)
- COS(0.017453*X)
- TAN(0.017453*X)

Die drei restlichen trigonometrischen Funktionen

- *Kotangens*
- *Secans*
- *Kosecans*

können mit Hilfe der bekannten Beziehungen errechnet werden. Der Vollständigkeit halber wollen wir hier diese noch einmal aufführen:

```
cot x = cos x / sin x = 1 / tan x
sec x = 1 / cos x
cosec x = 1 / sin x
```

Auch hier gilt, daß bei einem im Gradmaß vorliegenden Winkel dieser ins Bogenmaß umzurechnen ist.

Die Programmiersprache BASIC für die IBM Personalcomputer weist eine einzige eingefügte inverse trigonometrische Funktion auf, nämlich arctan[1]. Um diese Funktion mit dem Argument X anzusprechen, ist die Codierung von

```
ATN(X)
```

notwendig. Der Funktionswert dieser Funktion ergibt sich als ein im Bogenmaß ausgedrückter Winkel, d.h. er liegt in *rad* vor. Um das Gradmaß dieses Winkels zu erhalten, ist also die Bildung des nachstehenden arithmetischen Ausdruckes erforderlich:

```
57.29578*ATN(X)
```

Testübung 12.3.1

Es ist ein Programm zu schreiben, das die folgenden Werte errechnet:

a) sin 45°
b) cos 45°
c) tan 45°

[1] Die Bezeichnung arctan kennzeichnet eine Arkusfunktion: Es ist der Winkel im Bogenmaß zu bestimmen, dessen Tangens gleich x ist.

12.3.2 Logarithmische Funktion und Exponentialfunktion

BASIC gestattet die Berechnung von e^x, indem die eingefügte Exponentialfunktion

```
EXP(X)
```

aufgerufen wird. Weiterhin kann für das Argument X auch der natürliche Logarithmus durch Aufruf der eingefügten Funktion

```
LOG(X)
```

bestimmt werden. Um aus diesen Logarithmen zur Basis b zu ermitteln, muß man sich der bekannten Beziehung

```
logb x = ln x / ln b
```

bedienen. In dieser Formel ist unter ln der natürliche Logarithmus zu verstehen, d.h. der Logarithmus zur Basis 2,7182818 . . . (diese Basis wird mit e bezeichnet).

Beispiel 1:

Für die natürlichen Logarithmen von

```
x = 1.0, 2.0, 3.0, ···, 98.0, 99.0, 100.0
```

ist eine Logarithmentafel zu erstellen. Die Logarithmentafel soll über den Drucker ausgegeben werden.

Das in der Abb. 12.2 aufgelistete Lösungsprogramm erstellt eine zweispaltige Tabelle; jede Spalte ist dabei mit einer Spaltenüberschrift versehen.

```
10  LPRINT  "X", "LOG(X)"
20  J = 1.0
30  LPRINT  J, LOG(J)
40  IF  J=100.00  THEN GOTO 70
50  J = J + 1.0
60  GOTO 30
70  END
```

Abb. 12.2 Programm zur Ausgabe einer Tafel von natürlichen Logarithmen

Testübungen 12.3.2

1. Man schreibe das Programm von Abb. 12.2 so um, daß zur Steuerung der Berechnungsschleife das FOR-Statement herangezogen wird!
2. Man schreibe ein Programm zur Berechnung der Funktion

   $f(x) = \sin x / (\ln x + e^x)$

   für $x = 0.45$ und $x = 0.7$!

Beispiel 2:

Zur Altersbestimmung von Gegenständen wird sehr oft die Kohlenstoffmethode[1] herangezogen. Hierbei wird der prozentuale Anteil des im Gegenstand verbliebenen radioaktiven Kohlenstoffs vom Atomgewicht 14 bestimmt. Der prozentuale Anteil wird dabei auf die Menge bezogen, die im Gegenstand vorhanden wäre, wenn dieser erst heute hergestellt worden wäre. Bezeichnet man mit r den anteiligen Gehalt an ^{14}C, so ergibt sich das ungefähre Alter A des Gegenstandes nach der Formel:

$$A = -\frac{1}{0.00012} \cdot \ln r$$

Nehmen wir einmal an, daß eine alte ägyptische Papyrusrolle 47% der Menge an ^{14}C enthält, die bei einer heutigen Herstellung vorhanden wäre. Durch ein Programm ist das Alter der Papyrusrolle zu errechnen.

Durch Einsetzen von $r = 0.47$ in die obige Formel ergibt sich das in der Abb. 12.3 aufgelistete Programm.

```
10  LET  R = 0.47
20  LET  A = -(1/0.00012)*LOG(R)
30  PRINT  "Das Alter des Papyrus betraegt";A;" Jahre"
40  END
```

Abb. 12.3 Programm zur Altersbestimmung von Objekten

12.3.3 Potenzen

In die Programmiersprache BASIC der IBM Personalcomputer ist eine Funktion zur Berechnung von Quadratwurzeln eingebaut, die mit

```
SQR(X)
```

bezeichnet wird; unter X sei dabei auch hier das Argument, in diesem Fall ist es der Radikand, verstanden. Wie bei allen bisher betrachteten Funktionen, kann für das (numerische) Argument ein beliebiger Typ gewählt werden; der Funktionswert gehört auf jeden Fall zum einfach genauen Typ. Beispielsweise wird sich also für Y nach Ausführung der Anweisung

```
10  LET Y = SQR(2.0000000000000000)
```

der Wert 1.414214 ergeben.

Die Berechnung von Potenzen unter Benutzung des Operators ^, von dem wir schon seit langem Gebrauch machen, ist genau so gut auch mit negativen und gebrochenen Exponenten möglich. Daher stellt sie auch für die Berechnung von Quadratwurzeln eine alternative Methode dar. Somit kann die Berechnung der Quadratwurzeln von X genausogut durch die Codierung des Ausdrucks

```
X^(1/2)        bzw.        X^0.5
```

erreicht werden. Der Aufruf der eingefügten Funktion SQR führt freilich zu einer schnelleren Berechnung und sollte deshalb auch bevorzugt werden. Andererseits ist die Codierung mit

[1] Auch als Radiokarbonmethode bezeichnet

dem Operator ^ flexibler. Mit seiner Hilfe läßt sich ohne große Mühe auch die Kubikwurzel aus X ziehen bzw., um bei einem weiteren Beispiel zu bleiben, X in die 5.389-te Potenz erheben.

Codierungsbeispiele: `X^(1/3)` bzw. `X^5.389`

12.3.4 Verschiedene andere Funktionen

Die in diesem Unterabschnitt erwähnten Funktionen können bei der Lösung verschiedener Probleme sehr von Nutzen sein. Besprechen sie wir der Reihe nach.

Die größte ganze Zahl, die kleiner als oder gleich einem Wert ist, der der Variablen X zugewiesen ist, wird durch die Bezugnahme auf die Funktion INT(X) bestimmt. Bleiben wir bei einem Beispiel. Die größte ganze Zahl, die kleiner als oder gleich 5.4578 ist, ist 5. Somit ergibt auch

```
INT(5.4578)
```

den Wert 5. Ein weiteres Beispiel betrifft negative Zahlen. Die größte ganze Zahl, die kleiner als oder gleich −3.4 ist, ist −4 (veranschaulichen wir uns das: −4 ist auf der Zahlengeraden die erste ganze Zahl links von −3.4!). Der Aufruf von

```
INT(-3.4)
```

ergibt demzufolge auch den Funktionswert −4. Wir erkennen also ohne weiteres, daß bei Argumenten, die positive Zahlen darstellen, die Funktion INT einfach den gebrochenen Anteil des Argumentes streicht, um den Funktionswert zu ermitteln. Bei negativen Argumenten arbeitet jedoch die Funktion INT ein wenig anders. Wollen wir, daß von einer Zahl, gleichgültig, ob sie positiv oder negativ ist, der gebrochene Anteil weggelassen wird, so können wir uns einer anderen Funktion, nämlich der Funktion FIX(X) bedienen. Zwei Beispiele mögen die Wirkungsweise dieser Funktion erläutern:

```
FIX(5.4578) → 5
FIX(-3.4)   → -3
```

Der absolute Wert einer Zahl ergibt sich als Funktionswert der eingefügten Funktion ABS(X). Man erinnere sich, daß der Absolutwert einer Zahl gleich der Zahl selbst ist, wenn diese positiv oder gleich Null ist; der Absolutwert einer negativen Zahl ist hingegen der entsprechenden positiven Zahl, d.h. der Zahl, die bei Weglassung des Minuszeichens erhalten wird. Auch hier sollen einige Beispiele folgen:

```
ABS(5.4578) → 5.4578
ABS(0)      → 0
ABS(-3.4)   → 3.4
```

Die Funktion ABS wirft also gewissermaßen das Vorzeichen einer Zahl weg. Im Gegensatz hierzu behält die Funktion SGN(X) das Vorzeichen bei und ersetzt die X zugewiesene Zahl durch 1. Zwei Beispiele mögen diese Aussage erläutern:

```
SGN(5.4578) → +1
SGN(-3.4)   → -1
```

12.3.5 Konvertierungsfunktionen

Die Programmiersprache BASIC enthält bei den IBM Personalcomputern auch Funktionen, die die Konvertierung einer Zahl von einem Typ in einen anderen Typ zur Folge haben.

Soll z. B. die der Variablen X zugewiesene Zahl in den ganzzahligen Typ, d. h. in eine ganze Zahl, umgewandelt werden, braucht man sich nur der Funktion CINT(X) zu bedienen. Man beachte hierbei, daß der gebrochene Anteil von X zur Rundung herangezogen wird. Die sich als Funktionswert ergebende ganzzahlige Konstante muß natürlich im Wertebereich von −32768 bis 32767 liegen; andererseits wird ein Fehler entstehen.

Um die X zugewiesene Zahl in eine Zahl einfacher Genauigkeit umzuwandeln, benutzt man einfach die Funktion CSNG(X). Verbirgt sich hierbei unter dem Argument X eine ganze Zahl, so wird zur Ermittlung des Funktionswertes hinter dem Dezimalpunkt eine geeignete Anzahl von Nullen angehängt; ist hingegen unter X ein doppelt genauer Wert zu finden, so wird zur Bestimmung des Funktionswertes der Wert des Arguments auf sieben gültige Ziffern gerundet.

Um eine Zahl in eine Zahl des doppelt genauen Typs umzuwandeln, steht die Funktion CDBL(X) zur Verfügung. Bei dieser Funktion entsteht der Funktionswert dadurch, daß eine passende Anzahl von Nullen an die Zahl angehängt wird, die sich hinter dem Argument X verbirgt.

Aufgabengruppe 46

Bei den Aufgaben 1. bis 10. sind die folgenden Werte zu bestimmen!

1. $e^{1.54}$
2. $e^{-2.376}$
3. ln 58
4. ln 0.0000975
5. sin 3.7
6. cos 45°
7. arctan 1
8. tan 0.682
9. ARCTAN 2 (das Ergebnis soll im Gradmaß vorliegen!)
10. $\log_{10} 18.9$
11. Es ist eine Wertetafel für die Exponentialfunktion e^x aufzustellen. Das Argument soll hierbei die folgenden Werte durchlaufen:

    $$x = -5, -4.9, -4.8, \cdots, 0, 0.1, \cdots, 4.8, 4.9, 5$$

12. Die Funktion

    $$y = \sqrt[4]{3x} \cdot \ln 5x + e^{1.8x} \cdot \tan x$$

    ist für

    $$x = 1.7, 3.1, 5.9, 7.8, 8.4, 10.1$$

    zu berechnen!

13. Es ist ein BASIC-Programm zur graphischen Darstellung der Funktion

$$y = \sin x$$

im Intervall von $x = 0$ bis $x = 6.28$ zu schreiben! Die x-Werte sollen dabei mit einer Schrittweite von 0.05 aufeinander folgen.

14. Es ist ein BASIC-Programm zur graphischen Darstellung der Funktion

$$y = |x|$$

im Intervall von $x = -5$ bis $x = 5$ zu schreiben.
Anm.: Unter $|x|$ ist der absolute Betrag von x zu verstehen.

15. Es ist ein Programm zu schreiben, das den gebrochenen Anteil von x bestimmt.
Anm.: Unter dem gebrochenen Anteil von x sind die auf den Dezimalpunkt folgenden Ziffern von x zu verstehen.

Antworten auf die Testübungen

12.3.1 Die Aufgabenstellung wird durch das nachfolgende Programm gelöst.

```
10  A = 0.017453
20  PRINT  SIN(45*A), COS(45*A), TAN(45*A)
30  END
```

12.3.2 Die Aufgabenstellungen werden durch die nachfolgenden Programme gelöst.

1. *Benutzung einer FOR-Schleife*

```
1Ø  LPRINT  "X", "LN(X)"
2Ø  FOR  J = 1.Ø  TO 1ØØ.Ø  STEP 1.Ø
3Ø    LPRINT  J, LOG(J)
4Ø  NEXT J
5Ø  END
```

2. *Formelberechnung*

```
10  DATA  0.45, 0.7
20  FOR  J = 1  TO  2
30    READ A
40    PRINT  SIN(A)/(LOG(A)+EXP(A))
50  NEXT J
60  END
```

12.4 Definition eigener Formelfunktionen des Benutzers

Viele in der Mathematik und in den technisch-wirtschaftlichen Disziplinen verwendeten Funktionen werden gewöhnlich durch eine oder mehrere Formeln definiert. Zur Erinnerung an diese Tatsache wollen wir hier zunächst einige Funktionen als Beispiele aufführen:

$$f(x) = \sqrt{x^2 - 1}$$

$$g(x) = 3x^2 - 5x - 15$$

$$h(x) = \frac{1}{x-1}$$

Jede dieser Beispielfunktionen ist von uns mit einem Buchstaben benannt worden; hier also mit f, g bzw. h. Bei den IBM Personalcomputern gestattet die Programmiersprache BASIC die Definition von Funktionen wie der soeben als Beispiele aufgeführten und ihre spätere Verwendung im Programm durch einfache Nennung der von uns für sie festgelegten Bezeichnung, des Funktionsnamens also. Zur Definition einer Funktion benutzen wir die Vereinbarung

```
DEF FN ...
```

Anstelle der Pünktchen ist eine Zeichenfolge einzusetzen, die zusammen mit FN den Funktionsnamen bildet. Funktionsvereinbarungen müssen in einem Programm vor dem ersten Aufruf der betreffenden Funktionen vorgenommen werden. Um die von uns mit f(x) oben aufgeführte Beispielfunktion für ein Programm zu definieren, können wir somit die Vereinbarung

```
10  DEF FNF(X) = (X^2-1)^(1/2)
```

niederschreiben. In gleicher Weise kann die o.a. Funktion g(x) für die Benutzung in einem Programm durch die Niederschrift des Statements

```
20  DEF FNG(X) = 3*X^2 - 5*X - 15
```

definiert werden.

In diesen beiden Fällen haben wir zur Unterscheidung unserer beiden Funktionen nur einen einzigen Buchstaben (F bzw. G) benutzt. Wenn wir nun in Programm den Funktionswert von f(x) für x = 12.5 ermitteln müssen, so genügt die Niederschrift von FNF(12.5), vorausgesetzt, die Funktionsvereinbarung ist zuvor bereits erfolgt. Derartige Berechnungen können durch das ganze Programm hindurch immer wieder aufs neue erfolgen, ohne daß wir gezwungen sind, in jedem Einzelfall die Formel erneut aufzuführen.

Jeder gültige Variablenname kann zusammen mit FN als gültiger Funktionsname gewählt werden. Man kann z.B. für die Berechnung von Zinsen die Funktion

```
30  DEF FNZINSEN = K*I*P / 36000
```

definieren, die bekannte Zinsformel also.

Bei der Definition von Formelfunktionen kann man darüberhinaus auch auf bereits definierte Funktionen (auch auf eingefügte Funktionen) bezug nehmen. Sind FNF(X) und FNG(X) gemäß

der obigen Vereinbarungen auf den Zeilen mit den Zeilennummern 10 und 20 bekannt, so kann man als weitere Funktion das Produkt der beiden Funktionen f(x) und g(x) einführen, indem man die Vereinbarung

```
40  DEF FNP(X) = FNF(X)*FNG(X)
```

Alle bisher als Beispiele definierten Formelfunktionen waren Funktionen, die nur von einer Variablen abhängen. BASIC gestattet jedoch auch die Festlegung von Formelfunktionen, die von mehreren Variablen abhängig sind. Veranschaulichen wir diese Feststellung durch ein Beispiel. Die Funktion

$$q(x,y,z) = x^2 + y^2 + z^2$$

muß in einem Programm mehrfach aufgerufen werden. Deshalb werden wir zweckmäßigerweise eine Vereinbarung über diese Funktion treffen:

```
50  DEF FNQ(X,Y,Z) = X^2 + Y^2 + Z^2
```

Als unabhängige Variablen können in einer Funktionsdefinition auch Zeichenkettenvariablen benutzt werden. So wird z.B. durch die Vereinbarung

```
60  DEF FNB(A$) = LEN(A$) + 6
```

die Funktion FNB definiert, die als Funktionswert die um 6 erhöhte Länge der Zeichenkettenvariablen A$ besitzt.

Schließlich können auch Funktionen festgelegt werden, deren Funktionswert gleich einer Zeichenkette ist. Der Funktionsname einer solchen Funktion muß in analog zu früher gesagten mit dem Währungszeichen (Dollarzeichen) enden. Als Beispiel hierfür sie die Funktion FND$ definiert:

```
70  DEF FND$(A$,J) = LEFT$(A$,J) + " PC"
```

Diese Funktion mit den beiden unabhängigen Variablen A$ und J besitzt als Funktionswert die Zeichenkette, die aus den J linksbündigen Zeichen von A$ besteht, an die die Zeichenkette " PC" angehängt ist. Wenn also A$ die Zeichenkette "COMPUTER" und J der Wert 4 zugewiesen ist, ergibt sich als Funktionswert für FND$ die Zeichenkette "COMP PC".

Aufgabengruppe 47

Bei den Aufgaben 1. bis 6. sind die Vereinbarungen zur Definition der aufgeführten Funktionen niederzuschreiben!

1. $y = x^2 - 5x$
2. $y = \frac{1}{x} - 3x$
3. $y = 5e^{-2x}$
4. $y = x \ln \frac{x}{2}$

5. $y = \frac{\tan x^2}{x}$

6. $y = \cos 2x + 1$

7. Es ist die Funktion zu definieren, deren Funktionswert gleich der Zeichenkette ist, die aus den beiden rechtsbündigen Zeichen der unabhängigen Variablen C$ besteht.

8. Es ist die Funktion zu definieren, deren Funktionswert gleich der Zeichenkette ist, die aus den an der Stelle j der Variablen A$ beginnenden vier aufeinanderfolgenden Zeichen besteht.

9. Es ist die Funktion zu definieren, deren Funktionswert die aus dem mittleren Zeichen der unabhängigen Variablen B$ bestehende Zeichenkette darstellt. Es ist dabei vorausgesetzt, daß die B$ zugewiesene Zeichenkette stets nur aus einer ungeraden Anzahl von Zeichen besteht.

10. Schreibe ein Programm, das eine Wertetafel für die in der Aufgabe 3. genannte Funktion ausgibt. Die unabhängige Variable x soll dabei den Wertebereich von [0,10] mit einer Schrittweite von 0.1 durchlaufen.

13 Simulationen unter Zuhilfenahme von Computern

Als äußerst wirkungsvolles Instrument für Analysen erweisen sich immer mehr die Simulationen. Wenn zu ihrer Durchführung darüberhinaus Computer eingesetzt werden, kann ein weites Spektrum von Problemen in Angriff genommen und auch gelöst werden, die auf andere Weise kaum oder nur sehr schwierig zu lösen wären.

13.1 Grundlagen der Simulationstechniken

Um den Leser klarzumachen, was man unter einer Simulation versteht, wollen wir ein konkretes Beispiel betrachten. Versetzen wir uns deshalb einmal in die Person des Inhabers eines Geschäftes, das mit Computern und deren Software handelt. Im Verkauf hat er momentan nur einen Angestellten beschäftigt, aber er denkt über die Einstellung eines zweiten Verkäufers nach. Er stellt sich infolgedessen gewissenhaft die Frage, ob das Geschäft diese Einstellung verkraften, d.h. ob er sie verantworten kann. Da er eine gründliche, alle Fakten abwägende, analytisch geschulte Person ist, hat er zur Entscheidungsfindung eine Reihe von Daten gesammelt. Der Kundenverkehr schwankt während der Geschäftsstunden erheblich. Um ihn analysieren zu können, hat er deshalb während des letzten Monats Tag für Tag in einem Geschäftstagebuch die Anzahl der Kunden vermerkt, die während der einzelnen Geschäftsstunden vorgesprochen haben. Es ist ihm somit möglich, die durchschnittliche Anzahl potentieller Kunden zu schätzen, die zu den einzelnen Geschäftsstunden vorsprechen werden. Seine Erkenntnisse sind von ihm in einer Tabelle festgehalten worden (Abb. 13.1).

Geschäftsstunden (Uhrzeit)	*Anzahl potentieller Besucher*
9 bis 10	10
10 bis 11	15
11 bis 12	15
12 bis 13	40
13 bis 14	30
14 bis 15	10
15 bis 16	10
16 bis 17	8
17 bis 18	25
18 bis 19	50
19 bis 20	45
20 bis 21	30

Abb. 13.1 Durchschnittliche Anzahl der zu erwartenden potentiellen Kunden

Der Inhaber hat weiterhin richtig beobachtet, daß er augenblicklich einen geschäftlichen Schaden tragen muß, wenn ihm ein zweiter Verkäufer fehlt. Ein potentieller Kunde verläßt die Geschäftsräume, ohne etwas zu kaufen, wenn er zu lange auf Bedienung warten muß. Seine Beobachtungen gipfelten in der Feststellung, daß der Prozentsatz potentieller Kunden, die un-

bedingt die Geschäftsräume verlassen, abhängig von der Länge der Schlange derjenigen ist, die vor ihm auf Bedienung warten; hier liegt also ein Warteschlangenproblem vor. Auch diese Beobachtungsergebnisse hat der Inhaber in einer Tabelle (Abb. 13.2) zusammengefaßt.

Länge der Warteschlange (Anzahl der auf Bedienung wartenden Kunden)	*Anteil der das Geschäft unbedient verlassenden Kunden*
0	0
1	0.20
2	0.20
3	0.30
4	0.30
5	0.40
6	0.40
7	0.40
8	0.50
9	0.60
10	0.65
11	0.70
12	0.75
13	0.75
14	0.75
15 und mehr	0.75

Abb. 13.2 Durchschnittlicher Anteil der das Geschäft verlassenden Kunden (abhängig von der Länge der Warteschlange)

Vom Inhaber wurde weiterhin ermittelt, daß eine Person im Durchschnitt in 2 Minuten abgefertigt wird und daß der durchschnittliche Rechnungsbetrag der Verkäufe 75.– DM beträgt. Andererseits wird ein neu einzustellender Verkäufer ein Gehalt von wöchentlich 750.– DM (einschließlich der Sozialabgaben) erhalten müssen. Es wird ferner vorausgesetzt, daß das Verkaufspersonal während der Geschäftsstunden ohne Pausen arbeitet. Das sind im groben die Grundlagen, auf die der Geschäftsinhaber seine Entscheidung gründen muß. Was ist also zu tun?

Das soeben umrissene Problem ist ziemlich typisch für die im Geschäftsleben auftretenden Probleme. Diese zeichnen sich dadurch aus, daß einerseits angehäufte Beobachtungsdaten vorliegen und andererseits Vorfälle zu beachten sind, die nicht vorausgesagt werden können. Solche Vorfälle drücken sich in Fragen aus, die für unseren Fall etwa wie folgt formuliert werden können:

a) Wann betritt ein Kunde die Geschäftsräume?
b) Muß sich der Kunde (die Kundin) in eine lange Schlange einreihen?
c) Gehört der Kunde (die Kundin) zu den ungeduldigen Menschen, d.h. hier zu denjenigen, die beim Vorfinden einer langen Warteschlange die Geschäftsräume sofort verlassen, ohne etwas zu kaufen?

usw.

Alle Fragen ändern nichts an der Tatsache, daß eine Entscheidung unaufschiebbar ist, daß der Geschäftsinhaber sie einfach fällen muß. Wie müßte man vorgehen, um sich zu einer vertretbaren Entscheidung durchzuringen?

Eine der für solche Entscheidungsfindungen angewandten Methoden besteht darin, daß man einen Computer dazu benutzt, den Geschäftsverkehr „nachzuahmen". Man läßt gewissermaßen den Computer ein Spiel „spielen", das darin besteht, daß zu willkürlichen, zufälligen Zeitpunkten „Kunden" erzeugt werden. Diese Kunden betreten dann die „Geschäftsräume" und entscheiden für sich selbst, ob sie auf Bedienung warten oder die Geschäftsräume unverrichteter Dinge wieder verlassen wollen, abhängig von der Warteschlange, die sie vorfinden. Der Computer verwaltet und verfolgt die Schlange der auf Bedienung wartenden Personen, die Anzahl der die Geschäftsräume verlassenden Personen, die von den Kunden getätigten Einkäufe und die verlorengehenden Einkünfte infolge der nicht bedienten Personen. Weiterhin führt er Buch über den simulierten Geschäftsverkehr während eines ganzen „Tages" und gibt dem Benutzer ein Fazit über den Ablauf eines „Geschäftstages". Man kann dagegen freilich einwenden, daß die vom Computer gelieferten Ergebnisse nicht zutreffend sind. Angenommen, daß der Computer einen „untypischen" Geschäftstag generiert; den für diesen gelieferten Ergebnissen wird man dann voreingenommen, zumindest kritisch gegenüberstehen. Das kann tatsächlich vorkommen. Um nicht in eine solche „Fallgrube" zu stürzen, muß man deshalb ein solches Programm für viele simulierte Tage laufen lassen und die Ergebnisse mitteln, d.h. ihre Durchschnittswerte bestimmen. Einen solchen Prozeß, wie wir ihn soeben beschrieben haben, bezeichnet man als Simulation.

In diesem Kapitel wollen wir nur einen flüchtigen Blick auf die Simulationstechniken werfen. Dieser soll uns helfen, ihre Möglichkeiten zu erkennen. Er soll uns ferner genügend Ideen verschaffen, um einfache Simulationen aufzubauen, die für uns von Nutzen sein können.

Nunmehr wollen wir uns an einige mathematische Grundgedanken erinnern, auf die wir im nächsten Abschnitt beziehen müssen. Die benötigten Gedankengänge konzentrieren sich auf die grundsätzliche Frage: Wie können wir es erreichen, daß der Computer ein unvorhergesehenes Ereignis nachahmen kann? Man betrachte beispielsweise den wütenden Kunden, der das Geschäft zu einem Zeitpunkt betritt, in dem er eine Schlange von vier wartenden Kunden vorfindet. Entsprechend der Tabelle von Abb. 13.2 wird erfahrungsgemäß der Kunde in 30% aller Fälle das Geschäft sofort wieder verlassen, in 70% der Fälle hingegen sich in die Warteschlange einreihen. Wie kann der Computer nun die Entscheidung in bezug auf den jeweiligen Kunden treffen?

So schwierig ist die Beantwortung dieser Frage gar nicht! Man benutze einfach den Zufallszahlengenerator. Man denke daran, daß RND eine Zufallszahl zwischen 0 (einschließlich) und 1 (ausschließlich) erzeugt. Wir wollen uns einmal fragen, wie oft RND einen Wert abliefert, der größer als 0.30 ist. Wenn in der Tat der Zufallszahlengenerator keine außergewöhnlichen Schwankungen zeigt, können wir erwarten, daß annähernd 70% der von ihm produzierten Zahlen im gegebenen Intervall liegen, denn das von uns angesprochene Intervall nimmt 70% des Intervalls von 0 bis 1 in Anspruch. Die zu untersuchende Kundenentscheidung stellt sich also für uns in der Programmierung wie folgt dar:
Wenn RND>0.30 ist, reiht sich der potentielle Kunde in die Schlange ein, anderenfalls verläßt der Kunde verärgert das Geschäft. Wir werden diese simple Idee mehrmals beim Entwerfen unserer Simulation aufgreifen.

13.2 Simulationsbeispiel: Computergeschäft

Wir wollen nunmehr eine Simulation zusammenstellen, die uns das im vorigen Abschnitt 13.1 geschilderte Problem lösen hilft. Wir müssen dabei Vorgehensweisen entwickeln, die jeden wichtigen Aspekt unseres Problems nachbilden.

Da das vorliegende Problem die Analyse von Aktionen erfordert, die während einer Zeitperiode (hier: während eines Tages) ablaufen, müssen wir auf irgendeine Weise die verstreichende, natürlich nur simulierte Zeit messen können. Die Erfassung der Zeit wollen wir mit Hilfe der Variablen STUNDEN und MINUTEN vornehmen; in diesen Variablen werden die Stunden und die Minuten der augenblicklichen Uhrzeit festgehalten. Als Einheit der (simulierten) Uhrzeit legen wir einen Zeitraum von vier Minuten fest, den wir als Zeittakt bezeichnen wollen. Ein Zeittakt dauert also solange, wie zur Abfertigung von zwei Kunden[1] benötigt wird. Unser Programm wird also im Abstand von vier Minuten die ablaufende Zeit verfolgen. Während eines jeden dieser vier Minuten andauernden Zeiträume finden gewisse geschäftliche Aktionen statt. Nach Durchführung derselben erfolgt der Übergang zum nächsten Zeitraum; das geschieht durch Aufdatierung der Variablen STUNDEN und MINUTEN.

Die aus den geschilderten Beobachtungen herrührenden statistischen Daten über das Eintreffen von Kunden in den Geschäftsräumen (Abb. 13.1) wollen wir in den Elementen des Bereiches VERKEHR(J) mit $J = 9,10,11,\cdots,19,20$ speichern. Im Element VERKEHR(9) ist die Anzahl der Kunden enthalten, die von 9.00 Uhr bis 10.00 Uhr die Geschäftsräume betreten, im Element VERKEHR(10) die zwischen 10.00 Uhr und 11.00 Uhr eintreffenden Kunden. Das letzte Element VERKEHR(20) nimmt schließlich die Anzahl der zwischen 20.00 Uhr und 21.00 Uhr eintreffenden Kunden auf. In unserem zu entwickelnden Programm werden wir also zuallererst den Elementen des Bereiches VERKEHR die ihnen zukommenden Werte zuweisen müssen.

Im nächsten Programmteil werden wir uns der zweiten, aus Beobachtungen herrührenden Tabelle von Daten (Abb. 13.2) annehmen, d. h. uns mit den „ungeduldigen" Kunden beschäftigen. Den Anteil der Kunden, die wegen zu großer Warteschlangen die Geschäftsräume wieder verlassen, ohne auf Bedienung zu warten, wollen wir in den Elementen des Bereiches OHNEBED(K) mit $K = 0,1,2,\cdots,13,14,15$ festhalten. Die entsprechenden Daten der genannten Tabelle werden also in diesem Programmteil in die entsprechenden Bereichselemente eingelesen.

Außerdem müssen wir in unserem Programm noch einige andere relevante Variablen benutzen. Diese benennen wir wie folgt:

EINNAHMEN	*Gesamter Verkaufserlös während eines Geschäftstages*
BEDIENTE KUNDEN	*Anzahl der an einem Geschäftstag bedienten Kunden*
EINNAHMEVERLUST	*Gesamter Einnahmeverlust während eines Geschäftstages infolge „ungeduldiger" Kunden*
MAXSCHLANGE	*Anzahl der Personen in der längsten Warteschlange, die sich während eines Geschäftstages gebildet hat*

Neben diesen Variablen werden wir im Programm selbstverständlich auch noch weitere Variablen benötigen; deren Bedeutung ist ohne weiteres aus dem Programm bzw. aus den verwendeten Bezeichnungen ersichtlich.

---

[1] Wenn wir von Kunden sprechen, meinen wir sowohl männliche als auch weibliche Personen

Zu Beginn einer jeden Stunde setzt das Programm die Zeitpunkte für das Eintreffen der Kunden fest (Unterprogramm auf den Zeilen mit den Zeilennummern 570 bis 650). Für die j-te Stunde ist somit das Eintreffen derjenigen Anzahl von Kunden zu planen, die im Bereichselement VERKEHR(J) vermerkt ist. Jedem dieser Kunden wird dabei ein Zeittakt zugeordnet, zu dem er die Geschäftsräume betreten wird (ein Zeittakt dauert bekanntlich vier Minuten). Bei der Festlegung des Eintreffzeitpunktes wird vom Zufallszahlengenerator Gebrauch gemacht. Da irgendwelche anderen Informationen fehlen, wollen wir eben annehmen, daß die die Geschäftsräume betretenden Kunden sich in einer rein zufälligen, doch gleichförmigen Weise über die fragliche Stunde verteilen. Die Methode, mit der wir diese Überlegungen innerhalb des Programmes handhaben wollen, wird nachfolgend beschrieben. Zu Beginn jeder (simulierten) Stunde versorgen wir einen Bereich mit 15 Elementen, den wir mit TAKTKUNDEN bezeichnen wollen, mit Werten. Jedes Element dieses Bereiches steht für einen der zu dieser Stunde gehörenden Zeittakte von vier Minuten Dauer und enthält die Anzahl der Personen, die während des Ablaufes dieses Zeittaktes die Geschäftsräume betreten. Wenn also z.B. TAKTKUNDEN(10) den Wert 4 enthält, so wird damit ausgesagt, daß vier Kunden in der Zeit von 36 bis 40 Minuten die Geschäftsräume betreten werden, also im 10-ten Vier-Minuten-Zeittakt. Auf die Bereichselemente von TAKTKUNDEN wird die in VERKEHR(J) vermerkte Anzahl der während der ab J beginnenden Stunde eintreffenden Personen gemäß der Ergebnisse des Zufallszahlengenerators verteilt (Zeilen mit den Zeilennummern 620 und 630).

Das Programm (Abb. 13.3) schreitet durch die simulierte Stunde in den von uns eingeführten Zeittakten fort. Für den z-ten vierminütigen Zeittakt veranlaßt es also, daß TAKTKUNDEN(z) Kunden die Geschäftsräume betreten. Das Programm läßt jede ankommende Person einen Blick auf die Warteschlange werfen und danach entscheiden, ob sie die Geschäftsräume, ohne auf Bedienung zu warten, wieder verläßt. Entscheidet sich die betreffende Person für das Verbleiben in den Geschäftsräumen, dann wird sie in die Warteschlange hineingestellt. Entscheidet sich der Kunde jedoch für das Verlassen der Geschäftsräume, so notiert der Computer einen Einnahmeverlust von 75.– DM; um diesen Betrag wird der bisherige Wert von EINNAHMEVERLUST erhöht. Nachdem die ankommenden Personen entweder in die Warteschlange eingereiht worden oder wieder hinausgegangen sind, erfolgt durch den Verkäufer die Bedienung von 1 oder 2 Personen aus der Warteschlange (2 Personen nur dann, wenn in der Warteschlange noch 2 oder mehr Personen vorhanden sind). Jede erfolgte Bedienung schlägt sich in einer Addition von 75.– DM auf die Summe der Einnahmen (EINNAHMEN) nieder. Nach Durchführung dieser Aktionen wird die Zeit aufdatiert, und es erfolgt ein Übergang zum nächsten Zeittakt. Bezüglich der Kunden, die bei Geschäftsschluß noch auf Bedienung warten, wollen wir uns recht hartherzig verhalten. Wir beabsichtigen nicht, diese noch zu bedienen; den damit nicht zustandegekommenen Einkauf buchen wir einfach als Einnahmeverlust. Dieses ziemlich sonderbare Geschäftsgebahren entspricht jedoch völlig unseren Überlegungen, die wir für die Simulation angestellt hatten. Wir wollten untersuchen, ob sich die Notwendigkeit erweist, zusätzliches Personal einzustellen. Jede Überzeit (Bedienung noch in der Warteschlange befindlicher Kunden) aber sollte deshalb in die Analyse so eingehen, daß sie sich zugunsten der Notwendigkeit, mehr Bedienungspersonal zur Verfügung zu haben, niederschlägt.

Das unter diesen Überlegungen geschriebene einfache Simulationsprogramm ist in der Abb. 13.3 aufgelistet.

```
10    'Initialisierung
20     DIM  VERKEHR(20), OHNEBED(15), TAKTKUNDEN(15)
30     RANDOMIZE  VAL(RIGHT$(TIME$,2))
40     'Einlesen der Daten hinsichtlich der ankommenden Kunden
        (pro Tag treffen im Schnitt 288 Kunden ein)
50      DATA  10,15,15,40,30,10,10,8,25,50,45,30
60      FOR  STUNDEN = 9 TO 20
70        READ VERKEHR(STUNDEN)
80      NEXT STUNDEN
90     'Einlesen der Daten hinsichtlich der Kunden, die sich nicht
        in die Warteschlange einreihen wollen
100     DATA  0, 0.2,0.2, 0.3,0.3, 0.4,0.4,0.4, 0.5, 0.6,0.65,
              0.7,0.75,0.75,0.75,0.75
110     FOR  SCHLANGE = 0 TO 15
120       READ OHNEBED(SCHLANGE)
130     NEXT SCHLANGE
140    'Zuweisung von Anfangswerten zu den Variablen
150     SCHLANGE = 0
170     EINNAHMEVERLUST = 0
180     EINNAHMEN = 0
160     MAXSCHLANGE = 0
190     BEDIENTEKUNDEN = 0

200   '***************  H A U P T P R O G R A M M  ***************
210    CLS:     PRINT "SIMULATION ---> BITTE WARTEN"
220    FOR  STUNDEN = 9 TO 20
230     FOR  MINUTEN = 0  TO 56  STEP 4
240       'Aufdatierung der Uhrzeit
250       IF  MINUTEN=0  THEN GOSUB 570:     'Planung der Stunden
260       ZEITTAKT = MINUTEN/4 + 1
270       'Simulation des Eintreffens von Kunden in den Geschaefts-
           raeumen fuer den augenblicklichen Zeittakt (4 Minuten)
280       FOR  J = 1  TO TAKTKUNDEN(ZEITTAKT)
290         GOSUB 390
300       NEXT J
310       'Simulation der Kundenbedienung
320       GOSUB 490
330     NEXT MINUTEN
340    NEXT STUNDEN
350    'Berechnung der (statistischen) Ergebnisse der Simulation
360    GOSUB 660
370    END

380   '***************  U N T E R P R O G R A M M E  *************
390       'Eintreffen eines Kunden in den Geschaeftsraeumen
400        IF  SCHLANGE>15  THEN L=15  ELSE L=SCHLANGE
410        IF  RND>OHNEBED(L)  THEN 420  ELSE 460
420        'Kunde reiht sich in die Warteschlange ein
430        SCHLANGE = SCHLANGE + 1
440        IF  SCHLANGE>MAXSCHLANGE  THEN MAXSCHLANGE=SCHLANGE
450        GOTO 480
460        'Kunde verlaesst die Geschaeftsraeume ohne Bedienung
470        EINNAHMEVERLUST = EINNAHMEVERLUST + 75
480    RETURN
```

Abb. 13.3 Programm zur Simulation eines Computergeschäfts (1.Teil)

```
490       'Bedienung von Kunden
500        FOR  J = 1  TO 2
510          IF  SCHLANGE=0  THEN 550
520          SCHLANGE = SCHLANGE - 1
530          EINNAHMEN = EINNAHMEN + 75
540          BEDIENTEKUNDEN = BEDIENTEKUNDEN + 1
550        NEXT J
560    RETURN

570       'Planung des Eintreffens der zu erwartenden Kunden fuer
           die naechste Stunde in den Geschaeftsraeumen
580        FOR  ZEITTAKT = 1  TO 15
590          TAKTKUNDEN(ZEITTAKT) = 0
600        NEXT ZEITTAKT
610        FOR  I = 1  TO VERKEHR(STUNDEN)
620          X = INT(15*RND) + 1
630          TAKTKUNDEN(X) = TAKTKUNDEN(X) + 1
640        NEXT I
650    RETURN

660       'Drucken der Tagesergebnisse
670        CLS
680        PRINT
690        PRINT  "EINNAHMEN:"; TAB(35) EINNAHMEN
700        PRINT  "BEDIENTE KUNDEN:"; TAB(35) BEDIENTEKUNDEN
710        PRINT  "NICHT BEDIENTE KUNDEN:";
                  TAB(35) 288-BEDIENTEKUNDEN
720        PRINT  "EINNAHMEVERLUST:"; TAB(35) EINNAHMEVERLUST
730        PRINT  "MAXIMALE WARTESCHLANGENLAENGE:";
                  TAB(35) MAXSCHLANGE
740    RETURN
```

Anmerkung: In das Programm wurden zusätzlich Leerzeilen eingefügt, um seine Lesbarkeit zu verbessern und damit seine Übersichtlichkeit zu erhöhen. Sie beeinflussen den Programmablauf natürlich nicht.

Abb. 13.3 Programm zur Simulation eines Computergeschäfts (2.Teil)

Die Ausgaben unseres Simulationsprogrammes haben wir für vier Tage festgehalten (Abb. 13.4), d.h. wir haben das Programm vier Mal ablaufen lassen.

```
1. Simulation
   ==========

   | EINNAHMEN:                          17250
   | BEDIENTE KUNDEN:                    230
   | NICHT BEDIENTE KUNDEN:              58
   | EINNAHMEVERLUST:                    4350
   | MAXIMALE WARTESCHLANGENLAENGE:      9

2. Simulation
   ==========

   | EINNAHMEN:                          16950
   | BEDIENTE KUNDEN:                    226
   | NICHT BEDIENTE KUNDEN:              62
   | EINNAHMEVERLUST:                    4650
   | MAXIMALE WARTESCHLANGENLAENGE:      9

3. Simulation
   ==========

   | EINNAHMEN:                          17550
   | BEDIENTE KUNDEN:                    234
   | NICHT BEDIENTE KUNDEN:              54
   | EINNAHMEVERLUST:                    4050
   | MAXIMALE WARTESCHLANGENLAENGE:      13

4. Simulation
   ==========

   | EINNAHMEN:                          18300
   | BEDIENTE KUNDEN:                    244
   | NICHT BEDIENTE KUNDEN:              44
   | EINNAHMEVERLUST:                    3300
   | MAXIMALE WARTESCHLANGENLAENGE:      10
```

Abb. 13.4 Simulationsergebnisse von vier Tagen

Bei der Betrachtung der Ergebnisse von Abb. 13.4 fallen uns einige interessante Fakten auf. Natürlich ergaben die vier Programmabläufe unterschiedliche Ergebnisse. Das ist darauf zurückzuführen, daß der Zufallszahlengenerator unterschiedliche Muster hinsichtlich des Eintreffens der Kunden in den Geschäftsräumen erzeugt. Man beachte zweitens, daß die Ergebnisse der einzelnen Tage prozentual nur geringfügig voneinander abweichen. Es scheint, daß sich ein statistisches Modell aufgetan hat, daß über die einzelnen Tage hinweg bestehen bleibt.

Kommen wir schließlich zum Fazit der Simulationsabläufe. Von äußerster Wichtigkeit für den Inhaber eines solchen Computergeschäftes dürfte die Erkenntnis sein, daß er in seinem Geschäft pro Tag mehrere Tausend DM an Einnahmen allein dadurch verliert, daß nicht alle potentiellen Kunden bedient werden können. Der Wochenverdienst von 750.– DM, den ein zusätzlicher Verkäufer erhalten würde, läßt ihn als gute Investition erscheinen. Allein die entgangenen Einnahmen eines Tages reichen aus, um sein Gehalt zu zahlen. Es scheint deshalb für den Inhaber richtig zu sein, einen weiteren Verkäufer einzustellen. Doch ein wenig mehr Vorsicht ist geboten! Wir haben der Simulation den Umsatz und nicht den Verdienst zugrundegelegt. Um eine endgültige Entscheidung fällen zu können, müssen wir noch den Profit ermitteln, der durch die zusätzlichen Verkäufe zu erzielen wäre. Wenn beispielsweise der Gewinn 40% des Umsatzes ausmachen würde, dann würde der zusätzliche Angestellte durch die von ihm getätigten Mehrverkäufe leicht mehr als 750.– DM Gewinn pro Woche hereinbringen. Seine Einstellung sollte also erfolgen.

Das soeben behandelte Beispiel ist ziemlich typisch für die Fälle, in denen Simulationen angewendet werden sollten, um sogar recht verwickelte Situationen bei kleineren Unternehmen klären zu helfen. In der nachfolgenden Aufgabengruppe wollen wir uns deshalb u.a. auch mit einigen Variationen hinsichtlich der soeben durchgesprochenen Aufgabenstellung beschäftigen.

Aufgabengruppe 48

1. Das Simulationsprogramm von Abb. 13.3 ist 10 Mal hintereinander auszuführen. Die Ergebnisse aller 10 Läufe sind aufzuzeichnen. Kommen die erzielten Ergebnisse den Daten nahe, die wir in der Abb. 13.4 aufgelistet haben?
   (Man erinnere sich, daß infolge des im Programm eingesetzten Zufallszahlengenerators man nicht darauf hoffen kann, daß die Ergebnisse sich einander gleichen. Die Fragestellung ist deshalb so zu verstehen, daß man bei der Wertung der Ergebnisse die im statistischen liegenden Abweichungen berücksichtigt.)

2. Es soll einmal angenommen werden, daß die potentiellen Kunden ungeduldiger bezüglich der Bedienung sind, als wir dem ursprünglichen Programm zugrundegelegt haben (siehe Tabelle in Abb. 13.2). Die Wahrscheinlichkeit, daß sie die Geschäftsräume verlassen, ohne auf Bedienung zu warten, soll deshalb in jedem Fall auf das Doppelte erhöht werden. Nach Vornahme der Programmänderung soll das Programm erneut mehrere Male ablaufen. Die dabei festgestellten Einnahmeverluste sind mit denjenigen zu vergleichen, die sich bei der ursprünglichen Programmfassung ergaben.

3. Es soll einmal angenommen werden, daß die potentiellen Kunden geduldiger hinsichtlich der Bedienung sind, als wir dem ursprünglichen Programm zugrundegelegt haben (siehe Tabelle in Abb. 13.2). Die Wahrscheinlichkeit, daß sie die Geschäftsräume, verlassen, ohne auf Bedienung zu warten, soll deshalb in jedem Fall auf die Hälfte verringert werden. Nach Vornahme der Programmänderung soll das Programm erneut mehrere Male ablaufen. Die dabei festgestellten Einnahmeverluste sind mit denjenigen zu vergleichen, die sich bei der ursprünglichen Programmfassung ergaben.

4. Dieser Aufgabenstellung lege man die Daten zugrunde, von denen wir ausgegangen sind (Abb. 13.1 und 13.2). Wir wollen nunmehr jedoch annehmen, daß ein zweiter Verkäufer bereits eingestellt ist. Nach Vornahme der dadurch erforderlichen Programmänderung soll das Programm erneut mehrere Male ablaufen. Dabei sind die Einnahmeverluste und die Maximallängen der Warteschlangen festzustellen und mit früheren Programmläufen (beim Vorhandensein nur eines Verkäufers) zu vergleichen.

5. Das ursprüngliche Programm (Abb. 13.3) ist so zu modifizieren, daß die durchschnittliche Wartezeit der die Geschäftsräume betretenden Personen errechnet werden kann.

14 Überblick über den Softwaremarkt

Bisher haben wir uns hauptsächlich damit beschäftigt, BASIC-Programme zur Lösung der verschiedensten Aufgabenstellungen abzufassen. Tatsächlich aber mangelt es den meisten Leuten an der Zeit, für alle Aufgaben, die sie dem Computer übertragen wollen, für die sie ihn letzten Endes angeschafft haben, Programme zu schreiben; vielfach besteht auch keine Neigung dazu. In solchen Fällen wird man sich nach Programmen umsehen, die andere geschrieben haben, und versuchen, diese käuflich zu erwerben. Gerade in den letzten Jahren ist der Markt für die kommerziell verfügbaren Programme im wahrsten Sinne des Wortes explodiert; in der Sprache der Datenverarbeiter und Computerfachleute spricht man vom sogenannten „Softwaremarkt". In diesem Kapitel wollen wir über die Argumente und Gegenargumente sprechen, die es beim Kauf oder Mieten von Software zu berücksichtigen und zu beachten gilt. Wir wollen damit beginnen, zunächst einen Überblick über die Klassen zu geben, nach denen die auf dem Markt befindliche Software eingeteilt werden kann.

14.1 Einteilung der Software

Es gibt bereits mehrere Tausend Programme auf dem Softwaremarkt, die man für den eigenen Personalcomputer käuflich oder in Ausnahmefällen auch auf Mietbasis erwerben kann. Täglich kommen neue hinzu. Um dem Leser bei der Sichtung dieses schier unübersichtlichen Marktes zu helfen, wollen wir die angepriesenen Programme grob in Kategorien einteilen. Beim erforderlichen Erwerb von Software solle man von diesen Kategorien ausgehen; nur so kann man den Kauf überflüssiger Software vermeiden. Wir haben uns zu der folgenden Kategorisierung entschlossen:

1. *Textprozessoren*
   Über die Bedeutung und die Wichtigkeit von Textprozessoren für die Textverarbeitung haben wir bereits gesprochen. Für die IBM Personalcomputer werden heute bereits mehr als ein reichliches Dutzend angeboten. Ein Textprozessor sollte für die täglichen Arbeiten (Schriftverkehr etc.) auf jeden Fall zur Verfügung stehen; also sollte man einen solchen auf die Kaufliste setzen.

2. *Dateiverwaltungsprogramme und Datenbankverwaltungssysteme (DBVS)*
   Solche Programme erlauben die Erzeugung, die Aufdatierung und das Durchsuchen von Dateien bzw. Datenbanken.

3. *Programme zur Ausbreitung von Tabellen (Tabellenkalkulation)*
   Hierbei handelt es sich um Programme, die den Computer in einen Arbeitsbogen (Arbeitsblatt) der Art verwandeln, wie ihn Konten- und Rechnungsführer in der Buchhaltung benutzen. Die Arbeitsweise dieser sehr populären Programme wollen wir im nächsten Abschnitt näher besprechen.

4. *Programmpakete für die Kontenführung*
   Man kann Programme oder Programmpakete, d.h. mehrere miteinander im Zusammenhang stehende Programme erwerben, die für die Führung verschiedener Kontenarten eingesetzt werden können, angefangen bei der Rechnungsstellung über die Verwaltung fälliger Zahlungen bzw. ausstehender Rechnungen bis hin zur Führung kompletter Lohnlisten und der Führung von Hauptbüchern.

5. *Programmpakete für die Steuern*
   Programme bzw. Programmpakete zur Berechnung von Steuern können ein weites Anwendungsspektrum umfassen. Sie reichen von der Steuerberechnung nach den geltenden Gesetzen und der Ausfüllung von Steuerformularen bzw. -erklärungen bis hin zur Planung von Steuerstrategien.

6. *Programmpakete für die graphische Datenverarbeitung*
   Diese Programmpakete werden bei der Schaffung und bei der Anzeige von Diagrammen bzw. von Zeichnungen der mannigfachsten Arten eingesetzt sowie bei der Ausgabe sogenannter Hartkopien derselben (Ausgabe auf Papier oder anderen beständigen Medien).

7. *Programme für die Inventar- bzw. Materialverwaltung*
   Diese Programme können zur Verwaltung des Inventars bzw. Materials von kleinen und mittleren Unternehmen (Firmen) eingesetzt werden.

8. *Programme zur Führung von Adreßverzeichnissen*
   Diese Programme können zur Führung und Verwaltung von Adreßverzeichnissen eingesetzt werden; mit ihnen kann auch das Beschriften (Drucken) von Adreßetiketten vorgenommen werden, entweder vom gesamten Adreßverzeichnis oder von ausgewählten Teilen desselben.

9. *Spielprogramme*
   Diese Programme dienen der Freizeitgestaltung der großen und der kleinen Kinder.

10. *Kommunikationsprogramme*
    Durch den Einsatz solcher Programme können Personalcomputer mit anderen Computern verbunden werden; damit ist eine Kommunikation zwischen Computern möglich.

11. *Dienst- und Hilfsprogramme*
    Durch diese Programme wird die Verwaltung aller Programme erleichtert, die zu einer Installation, zu einem System also, gehören. Sie können ebenso Hilfestellung bei der Programmierung leisten (Editoren).

12. *Lehrprogramme*
    Durch die Lehrprogramme können Erwachsene und Kinder sich Kenntnisse auf neuen Gebieten erwerben und vorhandene Kenntnisse vervollkommnen.

Die soeben aufgestellte Liste erhebt nicht den Anspruch auf Vollständigkeit. Aber sie kann zumindest als Anhaltspunkt dafür dienen, in welcher Richtung man seine eigene Programmbibliothek einrichten kann bzw. ergänzen muß.

14.2 Programme für die Ausbreitung von Tabellen (Tabellenkalkulation)

Zu den populärsten Softwareprodukten, die jemals geschaffen wurden, zählen ohne Zweifel die Programme für die Ausbreitung von Tabellen. Eines der ersten auf diesem Gebiet war VisiCalc[1], ein Finanzplanungsprogramm, das in Hunderten von Situationen eingesetzt werden kann, sowohl im geschäftlichen als auch im privaten Bereich. Zu der Klasse der Programme für die Ausbreitung von Tabellen gehört insbesondere auch das überaus leistungsfä-

[1] VisiCalc ist ein geschütztes Warenzeichen der *VisiCorp Inc.*, USA

hige und daher äußerst erfolgreiche Programm LOTUS 1-2-3[2], das inzwischen auf dem deutschen Markt ebenfalls Fuß gefaßt hat.

Wegen der vielfältigen Einsatzmöglichkeiten dieser Programme und ihrer daraus resultierenden Bedeutung wollen wir in diesem Abschnitt eine typische Anwendung derselben beschreiben, und zwar die Aufstellung eines häuslichen Budgets. Wir nehmen deshalb einmal an, daß die in der Abb. 14.1 zusammengestellten Daten die Einnahmen und die Ausgaben der Familie Schmitt beinhalten.

Einnahmen

Johann	3950.00 DM pro Monat
Wally	2862.50 DM pro Monat
Zinsen	700.00 DM im Januar, April, Juli, Oktober
Dividenden	8750.00 DM im Dezember

Ausgaben

Hypothek	1219.33 DM pro Monat
Autoabzahlung	468.75 DM pro Monat
Haushaltwaren	517.50 DM pro Monat
Kleidung	500.00 DM pro Monat
Hausreparatur	250.00 DM pro Monat
Nahrungsmittel	1600.00 DM pro Monat
Unterhaltung	300.00 DM pro Monat
Treibstoff	337.50 DM pro Monat
Lebensversicherung	1457.50 DM im Oktober
Urlaub	2500.00 DM im Juli
Krankenversicherung	712.50 DM pro Monat
Altersversorgung	1000.00 DM pro Monat

Abb. 14.1 Aufstellung der Einnahmen und der Ausgaben für ein häusliches Budget

Jeder Mensch hat wahrscheinlich irgendwann schon einmal eine Aufstellung dieser oder ähnlicher Art angefertigt. Nach dem Zusammentragen der in einer solchen Liste enthaltenen Fakten und Daten wird man sich sicher eine Reihe höchstwichtiger Fragen gestellt und nach ihrer Beantwortung gesucht haben. So könnte sich z.B. die Familie Schmitt mit folgenden Problemen konfrontiert sehen und sich mit ihnen auseinandergesetzt haben:

1. Sind die Einnahmen hoch genug, um alle Ausgaben abdecken zu können?
2. Gibt es Monate im Jahr, in denen man knapp bei Kasse ist?
3. Wie hoch sind die jährlichen Überschüsse, oder muß mit einem Defizit gerechnet werden?

Um solche und ähnliche Fragen, speziell die zweite, beantworten zu können, verfährt man am besten wohl so, daß man eine auf die einzelnen Monate bezogene Zusammenfassung der jeweiligen Einnahmen und Ausgaben fortschreibt (laufender Monatsabschluß). Eine solche Liste könnte man beispielsweise auf einem zum Rechnen eingerichteten Arbeitsbogen in der Fassung niederschreiben, die in der Abb. 14.2 dargestellt ist.

---

[2] LOTUS 1-2-3 ist ein geschütztes Warenzeichen der *Lotus Development Corp.*, USA

	Jan.	Feb.	März	...	Nov.	Dez.	Jahr
Einnahmen ⋮ Ausgaben ⋮							
Gesamteinkommen Gesamtausgaben							
Einkommen minus Ausgaben							
Laufender Monatsabschluß							

Abb. 14.2 Zum Rechnen vorbereiteter Arbeitsbogen für ein häusliches Budget

Wir wollen es dem Leser überlassen, die in der Abb. 14.1 zusammengestellten Daten auf den Arbeitsbogen (Abb. 14.2) zu übertragen und die einzelnen Rechenschritte durchzuführen. Nach deren Erledigung wird man feststellen können, daß die Familie Schmitt nicht mit ihren Einkünften auskommt. Während einer Reihe von Monaten entsteht ein Defizit; auch am Jahresende weist die Bilanz ein Minus auf. Da die Angehörigen der Familie Schmitt empfindsame Leute sind, wollen sie ihren Haushaltsplan so neu gestalten, daß sie nicht mehr ausgeben als sie einnehmen und daß sie in keinem Monat auf ein Defizit auflaufen.

Um dieses Ziel zu erreichen, beginnen sie damit, verschiedene Einschränkungen bei ihren Ausgaben ins Auge zu fassen und anschließend durchzuspielen. Für jede beabsichtigte Ausgabenbeschränkung müssen sie aber den Arbeitsbogen (siehe Abb. 14.2) neu durcharbeiten, um letzten Endes bestimmen zu können, ob alle monatlichen Defizite auch beseitigt sind. Unter Umständen können hierzu 15 oder gar 20 Ansätze erforderlich sein, bevor man zu einem Haushaltsplan gelangt, mit dem man leben kann und der frei von irgendwelchen Defiziten ist. Die hierbei durchzuführenden Berechnungen erweisen sich als ziemlich lästig, wenn sie von Hand durchgeführt werden müssen. Es ist jedoch gerade diese Art von Berechnungen, die relativ einfach von einem Programm für die Ausbreitung von Tabellen erledigt werden kann; deshalb hat man für diese Art von Programmen auch den Begriff „Programme für die Tabellenkalkulation" geprägt.

Ein Programm für die Ausbreitung von Tabellen verwandelt den Bildschirm des Computers in einen großen Arbeitsbogen, der dem entspricht, den wir bei der Budgetaufstellung in Abb. 14.2 benutzt haben. Man beachte hierbei, daß die Aufstellung des Budgets für die Familie Schmitt einen Bogen erforderte, der aus 20 Zeilen und 14 Spalten besteht. Eine durch ein Programm veranlaßte Tabellenausbreitung gestattet sogar das Anlegen von weitaus mehr Zeilen und Spalten als die bei unserem Beispiel vorkommenden. Die maximal möglichen Zeilen- und Spaltenzahlen hängen einmal vom benutzten Programm und zum anderen von der Hauptspeicherkapazität (RAM-Kapazität) des eingesetzten Rechners ab. In der Abb. 14.3 ist die Vorgabe für Arbeitsbögen dargestellt, wie sie auf dem Bildschirm durch die Tabellenkalkulationsprogramme gewöhnlich angezeigt werden. Der Positionsanzeiger (Cursor) ist dabei auf die erste Tabellenspalte der ersten Tabellenzeile, d.h. auf Position A1 eingestellt.

A1

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3											
4											
5											
6											
7											
8											
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											

Abb. 14.3 Anzeige einer typischen Vorgabe für eine ausgebreitete Tabelle (Arbeitsbogen) auf dem Bildschirm

	A	B	C
		Jan	Feb
1			
2	Einnahmen		
3	Johann		
4	Wally		
5	Zinsen		
6	Dividenden		
7			
8	Ausgaben		
9	Hypothek		
10	Autoabzahlung		
11	Haushaltwaren		
12	Kleidung		
13	Hausreparatur		
14	Nahrungsmittel		
15	Unterhaltung		
16	Treibstoff		
17	Lebensversicherung		
18	Urlaub		
19	Krankenversicherung		
20	Altersversorgung		
21	Gesamteinkommen		
22	Gesamtausgaben		
23	Einnahmen minus Ausgaben		
24	Laufender Monatsabschluß		

Abb. 14.4 Ausgebreitete Tabelle mit eingetragenen Überschriften (Titeln)

Unsere erste Aufgabe besteht nun darin, den Arbeitsbogen mit den Daten auszufüllen, die für das Budget der Familie Schmitt vorliegen. Dazu tragen wir zunächst die Überschriften (Titel) in die erste Zeile bzw. in die erste Spalte der Tabelle ein. In unserem Fall führen wir zuerst den Positionsanzeiger auf die Stellung B1 und tippen dort Jan als Abkürzung für den Monat Januar ein; danach schlagen wir die Eingabetaste an. Anschließend gehen wir mit dem Positionsanzeiger zur Stellung C1 über und tippen Feb ein, gefolgt vom Drücken der Eingabetaste usw. In ähnlicher Weise versorgen wir dann der Reihe nach die erste Spalte der einzelnen Zeilen mit den entsprechenden Merkmalen (Titeln), beginnend mit Einnahmen in der ersten Spalte der zweiten Zeile. Man beachte hierbei, daß linksbündig in der ersten Zeile jedes Bildes, das auf dem Bildschirm erscheint, die augenblickliche Stellung des Positionsanzeigers vermerkt ist; das gilt für nahezu alle Programme für die Ausbreitung von Tabellen.

Wir wollen jetzt einmal annehmen, daß die Eingabe aller Überschriften (Titel) beendet ist. Wir haben also den Zustand erreicht, wie er in der Abb. 14.4 dargestellt ist.

Nunmehr werden wir die vorliegenden Haushaltsdaten in die für die Monate Januar bis Dezember eingerichteten Tabellenspalten eintragen. Am besten beginnen wir mit der Tabellenspalte für den Monat Januar. Hierzu führen wir den Positionsanzeiger auf die Stellung B3 und tippen das Einkommen (die Einnahmen) von Johann ein, nämlich 3950.00 . Anschließend gehen wir mit dem Positionsanzeiger eine Zeile tiefer auf die Stellung B4 und füllen diese mit dem Einkommen von Wally für Januar, also mit 2862.50 , aus. So fahren wir fort, bis wir auf diese Weise alle Budgetdaten für Januar erfaßt haben.

Nunmehr müssen wir die Gesamteinnahmen (das Gesamteinkommen) und die Gesamtausgaben für Januar berechnen. Das Gesamteinkommen der Familie Schmitt erhalten wir, indem wir die Eintragungen auf den Positionen B3 bis B6 addieren. Die Arithmetik kann dabei durch das Programm bewerkstelligt werden. Die hierfür erforderlichen präzisen Befehle hängen jeweils von dem speziellen Programm ab, das wir benutzen.

Zur Berechnung der Gesamtausgaben für den Monat Januar ist die Bildung der Summe aus den Eintragungen auf den Tabellenpositionen B9 bis B20 erforderlich. Durch Eingabe eines entsprechenden Befehles sorgen wir dafür, daß vom Programm die Eintragung auf der Stellung B22 vorgenommen wird, wobei diese gleich der Summe aus den Eintragungen auf den Positionen B9 bis B20 ist. Die Eintragung auf der Stellung B24 ist die gleiche wie die auf der Stellung B23 . Damit ist die Ausfüllung der für den Monat Januar eingerichteten Tabellenspalte abgeschlossen.

Um die zweite Monatsspalte, die für den Monat Februar also, ist es im allgemeinen nicht nötig, die gleiche Mühsal auf sich zu laden wie bei der Ausfüllung der Januarspalte. In der Tat sind nämlich alle Eintragungen mit Ausnahme der Zinseintragung (700.00 DM in Zeile 5) gleich. Da im Februar für die Familie Schmitt keine Zinsen gutgeschrieben werden, ist die Zinseintragung in der Februarspalte auf 0 zu stellen. Die meisten Programme für die Tabellenkalkulation besitzen nun eine Einrichtung, die auf einfache Art und Weise die Wiederholung einer Reihe von Eintragungen zuläßt. Auf diese Einrichtung können wir zurückgreifen, um die Spalten C,D, E,· · ·,M auszufüllen.

Der Arbeitsbogen ist nunmehr mit Ausnahme der letzten Tabellenspalte N , die die Gesamtsummen der Einzeleinnahmen bzw. -ausgaben aufnehmen soll, und der letzten Zeile 24, in der die aufgelaufenen Summen (die Kontenstände also) ausgewiesen werden sollen, ausgefüllt. Wir wollen uns zunächst damit befassen, welche Eintragungen in die letzte Spalte aufgenommen werden müssen. Ein Beispiel genügt, um klarzumachen, welcher Rechengang ablaufen

muß, um den aufzunehmenden Wert zu ermitteln. In der Position N2 muß beispielsweise die Summe der Eintragungen von B2,C2,D2,···,M2 erscheinen.

In der letzten Zeile soll nun in der Tabellenposition C24 die Summe der Eintragungen der Positionen B24 und C23 aufgeführt sein; mit anderen Worten heißt das, daß die Summe aus Überschuß (bzw. Defizit) vom Monat Januar (Eintragung auf Stellung B24) und dem Überschuß (bzw. Defizit) vom Monat Februar (Eintragung auf Stellung C23) gebildet werden soll. Für jeden Folgemonat ist in ähnlicher Weise der aufgelaufene Überschuß bzw. das aufgelaufene Defizit zu errechnen, d.h. es soll der laufende Monatsabschluß (der Stand des Haushaltskontos am jeweiligen Monatsende also) ausgewiesen werden. Konkret ausgedrückt, der Kontenstand am Ende jeden Monats ergibt sich als Summe des Kontenstandes vom Vormonat und dem Überschuß bzw. dem Defizit des laufenden Monats. Also ist die Summe der Eintragungen +B24+C23 in die Position C24 zu bringen, +C24+D23 in die Position D24 usw. bis +L24+M23 in die Stellung M24 . Damit ist der Arbeitsbogen für das Budget der Familie Schmitt vervollständigt.

Nach der Herstellung eines solchen Arbeitsbogens (oder Arbeitsblattes) kann irgendeine Eintragung in demselben geändert werden. Als Folge davon können alle davon abhängigen Eintragungen (Summen usw.) unter Zugrundelegung der vorgegebenen Beziehungen neu errechnet werden. Auf diese Weise kann man z.B. die Wirkung beobachten, die sich aus einer Steigerung ergeben wird. Auch lassen sich die Folgen feststellen, die von einer Beschneidung bestimmter Ausgaben herrühren. Zusätzliche Einnahmen, wie beispielsweise Erbschaften, Steuerrückerstattungen usw., sowie zusätzlich Ausgaben lassen sich ebenfalls leicht in die ausgebreitete Tabelle einbauen und berücksichtigen.

Die Idee einer solchen Tabellenausbreitung ist, im Grunde genommen, sehr einfach. Sie kann aber eine wirkungsvolle unbezahlbare Hilfe bei der Bewältigung aller Arten von finanziellen Vorhaben und Plänen sein. Nach der einmaligen Definition aller Beziehungen zwischen den verschiedenen in die Tabelle aufgenommenen Größen können die Werte einiger Größen jederzeit geändert werden. Damit läßt sich ohne weiteres feststellen, wie diese Änderungen die zu bestimmenden Werte beeinflussen.

14.3 Der Erwerb von Software (Programmausrüstung)

Der Leser wird inzwischen schon bemerkt haben, daß es nicht immer leicht ist, ein komplexes Programm so zu entwickeln, daß man es sofort fehlerfrei ablaufen lassen kann. Um ein verwikkeltes Programm zu schreiben und fehlerfrei zu gestalten, ist eine gehörige Portion von Verstand und verbissener Entschlußkraft erforderlich; man sollte auch die Zeit nicht außer acht lassen, die man in ein solches Unterfangen investieren muß. Zusätzlich muß man bedenken, daß eine erhebliche technische Erfahrung mitgebracht werden muß, um die verschiedenen Einrichtungen des Computers sinnvoll nutzen zu können. Viele Leute wollen ihren Computer nur zur Erledigung und zur Vereinfachung ihrer täglich wiederkehrenden Arbeiten einsetzen. Sie sind hingegen nicht daran interessiert, größere Anwendungsprogramme von Grund auf zu entwickeln. Für diesen Personenkreis ist eine ständig anwachsende Menge von Programmen entstanden, die auf dem Softwaremarkt verfügbar ist; gehandelt werden die Programme in Computergeschäften und teilweise sogar schon im Versandhandel.

Kommerziell verfügbar sind heute Programme für beinahe jeden denkbaren Bedarf. Dazu gehören u.a.:

- *Computerspiele*
- *Textverarbeitungssysteme*
- *Systeme für die Materialverwaltung*
- *Bestell- und Verwaltungssysteme für Selbstständige* (Ärzte, Zahnärzte, Rechtsanwälte usw.)
- *Buchhaltungssysteme für kleinere, mittlere und größere Geschäfte*

usw.

Leider führte die rapide Einführung neuer Produkte und die große Anzahl verfügbarer Programme dazu, daß der Erwerb von geeigneter Software allmählich selbst zu einer gewissen Anstrengung, Mühe und Überlegung geriet. In diesem Abschnitt wollen wir deshalb einige Punkte besprechen, die den Leser helfen sollen, sich durch den „Softwaredschungel" hindurchzufinden.

Die nachfolgenden Unterabschnitte befassen sich mit einigen Fragen, die man sich selbst stellen sollte, wenn man sich mit dem Gedanken trägt, Software zu erwerben.

14.3.1 Wird das Programm auf dem eigenen Computer laufen?

Damit ein Programm auf dem eigenen PC ablaufen kann, muß dieser über genügend Speicherplatz verfügen und mit dem geeigneten Betriebssystem ausgerüstet sein.

Die Programmbeschreibungen enthalten üblicherweise auch den Speicherbedarf des betreffenden Programmes. Offensichtlich kann man ein Programm, das zu seiner Ausführung 64 k Speicherplätze benötigt, nicht auf einem PC mit nur 16 k ablaufen lassen. (Unter *nk* Speicherplätzen sind bekanntlich $n \cdot 1024$ Speicherplätze zu verstehen).

Programme sind stets so geschrieben, daß sie zu ihrer Ausführung ein bestimmtes Betriebssystem benötigen. In diesem Buch haben wir nur über das Betriebssystem DOS der IBM gesprochen. Neben diesem gibt es aber noch weitere Betriebssysteme. Die IBM selbst hat z. B. eine Version des weit verbreiteten Betriebssystems CP/M (registrierte Handelsmarke von *„Digital Research Corp."*) für ihren Personalcomputer entwickelt. Andere Betriebssysteme werden sicher bald folgen. Beim Erwerb eines Anwendungsprogrammes gilt es deshalb unbedingt zu überprüfen, ob das betreffende Programm auch unter dem eigenen Betriebssystem ausgeführt werden kann.

14.3.2 Wird das Programm die gewünschten Operationen durchführen?

Beim Erwerb eines Programmes für eine bestimmte Anwendung sieht sich der Benutzer sehr vielen Auswahlmöglichkeiten gegenübergestellt. Wie kann er nun unter diesen eine für seine Zwecke geeignete Wahl treffen? Offen gesagt, hier liegt für jeden Benutzer ein echtes Problem vor.

Im Softwareauswahlverfahren sollte deshalb der Benutzer so frühzeitig wie möglich seine eigenen Bedürfnisse definieren, und zwar so eingehend und genau, wie er es nur kann. Um den Umfang bestimmen zu können, bis zu dem ein Softwareprodukt die eigenen Bedürfnisse befriedigen kann, ist ein beträchtliches Maß an Schürfarbeit zu verrichten. Nahegelegene Computergeschäfte, versierte Berater, Artikel in Computerfachzeitschriften und Softwaredokumentationen erweisen sich dabei als recht gute und aufschlußreiche Informationsquellen.

Das örtliche Computerfachgeschäft bzw. ein örtlich tätiger Berater mit einem sachlich fundierten gründlichen Wissen sollten die ersten Stellen sein, bei denen man um Auskünfte nach-

sucht. Oft wird man dadurch in die Lage versetzt, aufgrund der erteilten Auskünfte selbst eine Auswahl zu treffen. Viele Computerzeitschriften enthalten Besprechungen über ganze Anwendungsgebiete und die sie abdeckenden Softwareprodukte. Besonders nützlich sind dabei solche Artikel, die dem Vergleich mehrerer ähnlicher Programme gewidmet sind.

Eine andere Quelle vergleichender Informationen ist die Softwaredokumentation selbst. Man kann nämlich oft die Dokumentation unabhängig von der Software selbst erwerben. Zwar entstehen dadurch Unkosten, aber oft erweist sich dieser Weg als der einzige, auf dem man sich darüber vergewissern kann, ob das in Betracht zu ziehende Programm auch das leistet, was man sich wünscht. Wenn man so sorgfältig wie möglich die Auswahlmöglichkeiten bis auf zwei anscheinend gleichwertige Programme eingeengt hat, warum soll man dann sich nicht die Handbücher der fraglichen Programme beschaffen und ihre Eigenschaften von einem engeren Spielraum her ansehen?

Nachträglich erweist es sich bestimmt als billiger, gewisse Beträge in die Dokumentation eines Programmes investiert zu haben, das später nicht zum Einsatz kommt, als Hunderte von DM in ein Programm, das nicht im entferntesten das leistet, was man sich vorgestellt hat.

14.3.3 Was bekommt man für den Betrag, den man für Software ausgibt?

Beim Erwerb von Software bekommt als Minimum das Programm auf einem bestimmten Datenträger (Speichermedium), wie Kassette oder Diskette; zusätzlich wird die Dokumentation geliefert. Was also könnte man vom Verkäufer sonst noch erwarten?

1. Ist der Verkäufer bereit und willens, telefonische Auskünfte zu erteilen, wenn man Hilfe bei der Installation des Programmes oder für den Ablauf desselben braucht?
2. Wie hoch sind die Kosten für die neuen Versionen der Software, d.h. für die auf den neuesten Stand gebrachte Software?
3. Informiert der Verkäufer automatisch den Käufer über größere bzw. schwerwiegende Fehler, die bei der erworbenen Software festgestellt worden sind?
4. Wie gut ist die Dokumentation? Ist sie leicht verständlich? Enthält sie klare, übersichtliche Beispiele? Kann sie auf einfache Weise auch als Nachschlagehandbuch verwendet werden?
5. In welchem Maße und in welchem Umfang kann und darf die Software erforderlichenfall gepflegt und gewartet werden?
6. Wie teuer kommt es zu stehen, wenn defekte Programmkopien ersetzt werden müssen? Viele Softwarehändler händigen augenblicklich noch dem Kunden nur zwei Programmkopien aus, die nicht kopiert werden können. Um Ersatz zu beschaffen, wenn diese Kopien beschädigt und daher nicht mehr brauchbar sind, muß man dann zahlen. Andere Programme hingegen kann man vom Original so oft kopieren, wie man will.

Man sollte sich durch die soeben gemachten Ausführungen nicht einschüchtern lassen. Die meisten Softwarehändler gehen sehr behutsam vor. Wie in jedem anderen Handelszweig muß man jedoch das Informiertsein bezahlen. Hoffen wir also, daß unsere Erörterungen und Anregungen auf fruchtbaren Boden gefallen sind und den Eigentümern von IBM Personalcomputern helfen, so manchen DM-Betrag zu sparen!

15 Weitere Einsatzmöglichkeiten des IBM Personalcomputers

In diesem Kapitel wollen wir über einige zusätzliche Einsatzmöglichkeiten des IBM Personalcomputers sprechen. Wir sind dabei selbstverständlich nur in der Lage, einen knappen Überblick über diese Anwendungen vorzustellen. Eine vollständige Diskussion würde uns über den Rahmen dieses Buches hinausführen. Da die von uns anzureißenden Anwendungen von enormer Wichtigkeit sind, sollte man zumindest in den Grundzügen davon etwas gehört haben.

15.1 Computerverbindungen (Anschluß von externen Geräten an Computer)

Irgendwann kann einmal der Wunsch aufkommen, daß man seinen persönlichen Computer mit externen Geräten, auch periphere Geräte genannt, verbinden will. Im Handel werden heute bereits viele solcher Geräte angeboten; die Einführung weiterer Geräte wird mit einem erschreckenden, nicht zu bremsenden Tempo vonstatten gehen. Einige der wichtigsten, augenblicklich verfügbaren peripheren Geräte sind nachstehend aufgeführt:

- *Datensichtgeräte für graphische Anwendungen*
- *Lichtstifte*
- *Zeichengeräte (Plotter)*
- *Geräte für die Stimmerzeugung*
- *Geräte für die Erzeugung von Musik*
- *Temperaturfühler*

Unsere Aufstellung ist selbstverständlich nicht vollständig; wir haben bewußt nur einige der vielfältigen Möglichkeiten erwähnt. Neben den aufgeführten Geräten befinden sich natürlich auch noch andere Geräte im Einsatz.

Neben dem Anschluß peripherer Geräte will man u.U. auch den eigenen PC mit anderen Computern verbinden, um den Programmaustausch sowie den Datenaustausch mit anderen Anwendern zu ermöglichen.

In diesem Abschnitt wollen wir uns mit den Grundlagen der Computerverbindungen beschäftigen. Wir wollen damit nicht bezwecken, den Leser zum Fachmann auf diesem Gebiet auszubilden. Uns steht der Sinn nur nach einer Einführung in dieses Wissensgebiet, um den Leser mit dem Gedankengut und den Begriffsbildungen vertraut zu machen. Dadurch wollen wir den Leser in die Lage versetzen, die sich mit diesem Thema beschäftigenden Artikel in den Fachzeitschriften lesen und verstehen zu können.

In den meisten Fällen ist es unmöglich, zwei elektronische Geräte direkt miteinander zu verbinden. Man muß ein Zwischengerät einschalten, das die Aufgabe hat, die von dem einen Gerät ausgehenden elektronischen Signale so umzuformen, daß sie von dem anderen Gerät verstanden werden können. Ein solches Zwischengerät wird „Schnittstelle" oder „Interface" genannt. Die Aufgabe, Geräte elektronisch miteinander zu verbinden, wird als „Verbinden über eine Schnittstelle" bezeichnet. Für Verbindungen, in die Mikrocomputer eingehen, gibt es eine Standardschnittstelle; das entsprechende Gerät heißt „RS 232-C Schnittstelle". Diese Schnittstelle ermöglicht die gegenseitige Verbindung zweier Geräte; die Verbindung erfolgt über ein 25-drähtiges Kabel. Jeder Draht überträgt dabei ein Signal, dem eine feststehende standardisierte Bedeutung zukommt. Man kann eine solche Schnittstelle für den IBM Personalcomputer jederzeit über den Handel beziehen. In der Terminologie des IBM PC wird die Schnittstelle

RS 232-C als „Schnittstelle für asynchrone Verbindungen" bezeichnet. Bei Benutzung dieser Schnittstelle kann der IBM Personalcomputer mit einer großen Vielfalt an peripherer Ausrüstung ausgestattet werden. Dabei spielt es keine Rolle, ob die peripheren Geräte von der IBM oder von anderen Firmen hergestellt wurden.

Bevor wir fortfahren, sollte noch ein Wort der Vorsicht, der Warnung an die Eigner von Personalcomputern gerichtet werden. Für viele Geräte wird mit dem Slogan „Mit eingebauter RS 232-C Schnittstelle" Reklame gemacht, auch findet man oft den Hinweis „Verträglich mit RS 232-C". Der kritiklose Erwerb solcher Geräte kann zu riesigen Anstrengungen führen, ehe sie zusammen mit dem Personalcomputer einwandfrei arbeiten. Die Ursache hierfür kann die folgende sein: Obgleich alle RS 232-C Schnittstellen mit 25-drähtigen Kabeln ausgerüstet sind, brauchen nicht alle Drähte unbedingt benutzt zu werden. Daher kann es der Fall sein, daß der Computer (bzw. die Programme) ein Signal erwartet, das nicht gesendet wurde, oder es sendet ein Signal nicht aus, das am anderen Ende, d.h. beim Gerät, benötigt wird. – Aus solchen Vorkommnissen ist die Quintessenz zu ziehen, daß man Erwerb peripherer Geräte, die man an den eigenen Computer anschließen will, Vorsicht walten läßt. Man sollte sich daher unbedingt vergewissern, ob der Hersteller, der Lieferant oder der Verkäufer auch willens ist, beim Anschließen des Gerätes Unterstützung zu gewähren, oder zumindest bereit ist, ein Gerät umzutauschen, mit dem der Eigentümer des PC nicht zurecht kommt.

In diesem Abschnitt verfolgen wir u.a. auch den Zweck, den Leser in die Grundgedanken und in die Terminologie der Computerkommunikation einzuführen. Zu Beginn dieser Erörterungen wollen wir zunächst mehr Einzelheiten über die Form der Datenspeicherung bringen als wir bisher vorgestellt haben.

Eine *Dualzahl,* in den Computerwissenschaften meist als *Binärzahl* bezeichnet, ist, von einem anderen Standpunkt aus betrachtet, nichts weiter als eine Folge von Nullen und Einsen. Nachfolgend geben wir ein typisches Beispiel für eine Binärzahl:

10110100011010001111010010 1

Eine *Dualziffer (Binärziffer),* d.h. 0 oder 1, wird auch Bit genannt. Eine Folge von acht aufeinanderfolgenden Bits heißt Byte. Hierfür folgen jetzt zwei Beispiele:

10110010 und 01110101

Im Computer werden alle Daten und Instruktionen in Form von Binärzahlen ausgedrückt. Was die Zeichen (Buchstaben usw.) anbelangt, so werden diese durch eine in einem Byte enthaltene Binärzahl verschlüsselt.

Es gibt zwei fundamental unterschiedliche Typen von Computerverbindungen:

- parallele Kommunikation
- serielle Kommunikation

Bei der parallelen Verbindung werden alle acht Bits eines Bytes gleichzeitig gesendet. Man erreicht das dadurch, daß man zur Übertragung der acht Bits acht Drähte heranzieht. Ein über den Draht gehendes Signal entspricht der Binärziffer 1; fehlt das Signal, so kommt dieses Fehlen der Binärziffer 0 gleich. Der Punktmatrixdrucker der IBM bedient sich dieser Technik. Bei der seriellen Kommunikation werden die einzelnen Bits eines Bytes nacheinander über den gleichen Draht gesendet. Viele Drucker bedienen sich dieser Übertragungstechnik. Außerdem kommt diese Technik bei der Datenübertragung von einem Computer zu einem anderen mittels Telefonleitungen zum Zuge. Die Schnittstellen für die genannten beiden Kom-

munikationstechniken unterscheiden sich völlig. Der IBM Personalcomputer ist mit einer Schnittstelle für die parallele Kommunikation ausgerüstet; an diese kann der Drucker angeschlossen werden, d.h. das entsprechende Verbindungskabel eingesteckt werden. Wenn man jedoch die serielle Kommunikationstechnik benutzen muß oder will, so ist es nötig, die RS 232-C Schnittstelle dazwischenzuschalten.

Bei der Einrichtung von Computerverbindungen muß man eine Anzahl verschiedener Parameter betrachten und berücksichtigen. Zuerst muß man einen Blick auf die Übertragungsgeschwindigkeit werfen, die standardmäßig in *Baud* gemessen wird. Bei altmodischen Fernschreibern wurde mit einer Übertragungsgeschwindigkeit von 110 Baud gearbeitet, d.h. es wurden etwa 11 Zeichen in der Sekunde über die Leitung gesendet. Die Datenübertragungsgeschwindigkeiten zwischen einem IBM Personalcomputer und einem Drucker reichen von 300 Baud bis zu 1200 Baud. Hochgeschwindigkeitsleitungen lassen heute Übertragungsgeschwindigkeiten von bis zu 9600 Baud zu. Man kann die in Baud gemessene Übertragungsgeschwindigkeit der RS 232-C Schnittstelle mittels eines Computerbefehles auf den Wert einstellen, der im speziellen Falle benötigt wird.

Alle Verbindungen sind Störungen unterworfen, die in erster Linie durch statische Elektrizität auf den Leitungen verursacht werden. Es ist nun aber außerordentlich wichtig, daß die Datenverbindungen bei Computer akkurat übertragen. Man stelle sich bloß einmal die Verheerung vor, die bei einer falschen Übertragung von nur einigen wenigen Ziffern in einem Finanzbericht entstehen würde! Um sich gegen solche Fehler zu schützen, wird bei vielen Datenübertragungen ein gesondertes Bit zusätzlich benutzt, das an jedes Byte angehängt wird. Dieses Bit wird Prüfbit oder Paritätsbit genannt. Der Wert des Paritätsbits hängt von der Summe der Bits ab, die zu dem betreffenden Byte gehören. Im voraus muß klar sein, ob die Summe der Bits in einem Byte (einschließlich des Paritätsbits) gerade oder ungerade sein soll (gerade bzw. ungerade Parität). Um das Paritätsbit zu setzen, bestimmt der Computer zunächst die Summe der Bits, die zum Byte gehören. Nehmen wir einmal an, daß die Summe ungerade ist, die Parität hingegen gerade. In diesem Fall wird das Paritätsbit auf 1 gesetzt (damit ist die Summe, wie verlangt, gerade). Das empfangende Gerät überprüft das Paritätsbit, um seine Korrektheit zu bestimmen. Bei Feststellung eines Fehlers wird gewöhnlich um eine erneute Übertragung nachgesucht. Diese Vorgänge geschehen gänzlich automatisch. Man muß jedoch die Übertragungswege so einstellen, daß sie der erwarteten Parität angepaßt sind. Man erreicht diese Anpassung durch Hinausschickung eines Computerbefehles an die RS 232-C Schnittstelle.

Zu guter Letzt erweist es sich mitunter als notwendig, wenn über den Datenverkehr auf den Verbindungswegen Protokoll geführt wird. In manchen Fällen nämlich ist es sinnvoll, die Daten mit einer höheren Geschwindigkeit zu übertragen, als sie der Empfänger annehmen kann. Zu diesem Zweck sendet der Computer die Daten in sogenannten Bündeln oder Blöcken. In der Wartezeit zwischen den einzelnen Bündeln kann der Computer nutzbringend für die Erledigung anderer Arbeiten eingesetzt werden. Beim Empfänger wird jedes eintreffende Bündel zunächst temporär in einem Speicher untergebracht, der Puffer genannt wird. In diesem wird das Datenbündel solange festgehalten, bis der Empfänger die Möglichkeit besitzt, sich dieses Bündels anzunehmen. Bei einem solchen Verfahren der Datenübertragung ist es notwendig, zwischen dem Absender und dem Empfänger ein Signalpaar auszutauschen. Der Absender muß nämlich dem Empfänger mitteilen, wenn weitere Daten abgeschickt wurden. Andererseits muß der Empfänger auch dem Absender sagen können, wenn er wieder zum Empfang von Daten bereit ist. Ein solcher Signalaustausch wird als Verbindungsprotokoll bezeichnet. Im allgemeinen ist eine Anzahl verschiedenartiger Protokolle im Gebrauch. Wie die-

se Protokolle aussehen, ist nicht so wichtig, aber es ist entscheidend, daß Sender und Empfänger das gleiche Protokoll benutzen. Man kann unter den verschiedenen möglichen Protokollen eine Auswahl treffen; hierzu dient ein Computerbefehl. Man beachte unbedingt, daß ein Verbindungsprotokoll nur in den Fällen benötigt wird, in denen die Übertragungsgeschwindigkeit der Daten zu schnell für den Empfänger ist. So ist es z. B. kennzeichend für die hier angeschnittene Problematik, daß für Drucker kein Protokoll erforderlich ist, wenn die Übertragungsrate bei 300 Baud oder geringer liegt. Man muß jedoch auf 1200 Baud übergehen, wenn man die höchstmögliche Druckgeschwindigkeit gewisser verfügbarer Druckertypen herausholen will.

Über Einzelheiten bezüglich der Computerverbindungen und der Anschlußtechnik informiert die einschlägige Literatur. Diese wird im Bedarfsfall zum eingehenden Studium empfohlen, insbesondere die entsprechenden Handbücher der IBM.

15.2 Erweiterte Graphik

Wir haben bisher (Kap. 8) nur einige Statements für die graphische Datenverarbeitung kennengelernt. Bekanntlich stehen diese dann zur Verfügung, wenn der IBM Personalcomputer mit der Schnittstelle für Farbe/Graphik ausgerüstet ist. Darüberhinaus gibt es weitere Statements. Durch sie können komplexe Figuren gezeichnet und Bildschirmregionen „bemalt" werden. Eine vollständige Besprechung dieser Statements enthält die Fortsetzung dieses Buches, d. h. das Werk

Larry Goldstein
Advanced BASIC and beyond: Techniques for the IBM PC
Robert J. Brady Co. 1983

Viele Drucker weisen einen Graphikmodus auf, der die Erzeugung von Hartkopien der Bildschirmgraphiken zuläßt. Das wird üblicherweise durch Drucken von Punkten erreicht, wobei ein hohes Auflösungsvermögen vorliegt. Noch schärfere Graphiken auf Hartkopien werden beim Einsatz von Zeichengeräten (Plottern) erzielt, von denen sich inzwischen eine Vielzahl auf dem Markt befindet. Die am weitesten entwickelten derartigen Geräte ermöglichen die wahrheitsgetreue Darstellung von Entwürfen, technischen Zeichnungen, Wetterkarten und sonstigen Abbildungen.

Mikrocomputer können heutzutage sogar mit graphischen Tafeln ausgerüstet werden. Darunter versteht man eine Einrichtung, die die Eingabe von Bildern in den Computer gestattet. Hierbei braucht man, im Grunde genommen, nur das Bild auf einem speziellen Tisch mit einem elektronischen Stift aufzuzeichnen. Das Bild wird dann in eine Folge von Punkten übergeführt und dem Computer über eine Verbindungsschnittstelle übermittelt.

15.3 Verbindungen von Mikrocomputern mit der Umwelt

Man kann den IBM Personalcomputer mit der Umwelt verbinden. Man benötigt hierzu einmal eine RS 232-C Schnittstelle und zum anderen ein besonderes Verbindungsgerät, das Modem genannt wird.

Ein Modem setzt die elektronischen Signale des Computers in Signale um, die über Telefonleitungen übertragen werden können. Ein Modem wird an den Computer über eine Schnittstelle RS 232-C angeschlossen. Zum Aufbau der Telefonverbindung zu einem anderen Com-

puter hat man wie üblich zunächst die Telefonnummer desselben anzuwählen. Nach Herstellung der Verbindung legt man den Telefonhörer in die Gabel des Modems[1]. (Die Schnittstelle RS 232-C muß dabei selbstverständlich eingeschaltet sein.) Auf diese Weise wird also eine Verbindung des eigenen PC zur Umwelt eingerichtet.

Man kann derartige Verbindungswege für die unterschiedlichsten Zwecke benutzen. An erster Stelle der Möglichkeiten ist der Nachrichtenaustausch mit anderen Mikrocomputerbenutzern zu nennen; mit diesen kann man dann spielen, Daten austauschen, Programmideen verwirklichen usw. Man kann sogar den Computer als eine Art „elektronisches Postamt" ansehen und einen Briefwechsel mit den Partnern aufziehen. In der Tat verspricht diese spezielle Anwendung von Computer-zu-Computer-Verbindungen den Geschäftsbetrieb in der nächsten Dekade zu revolutionieren. Anstelle der Versendung von Briefen und Drucksachen wird man die entsprechenden Daten über die entsprechenden Verbindungswege übertragen. Wenn die zu versendenden Informationen vertraulich zu behandeln sind, könnte man den Zugang zu ihnen entweder über Paßwörter (Schutzwörter) regeln oder die Informationen nach einem bestimmten Code verschlüsseln. Gewiß entstehen hierdurch gewisse Erschwernisse, doch denke man einmal gründlich und gerecht über alle Aspekte nach und bewerte auch die positiven! Es gibt keine Verzögerungen mehr bei der Postzustellung, keine verlorengegangenen Mitteilungen, und es gibt auch die verschiedenen anderen Probleme des konventionellen Nachrichtenaustauschs nicht mehr. Als Mikrocomputerbenutzer kann man zu den ersten gehören, die sich der Vorteile dieser neuen Nachrichtenwege bedienen können.

Man kann die Möglichkeit der Computer-zu-Computer-Verbindungen auch dazu benutzen, um sich mit dem PC an ein Großcomputersystem anzuhängen, zu dem man eine Zugriffserlaubnis besitzt. Dadurch verschafft man sich Zugang zu den ungleich weit größeren Kapazitäten und Fähigkeiten größerer Datenverarbeitungsanlagen, u.a. zum Beispiel zu den Programmbibliotheken eines Realzeitsystems.

Schließlich wäre noch zu erwähnen, daß man den eigenen PC auch in irgendein Informationsnetzwerk einbinden kann, beispielsweise in *Btx (Bildschirmtext).* Für die Teilnahme an solchen Netzwerken ist im allgemeinen ein monatlicher Beitrag zu zahlen. Dafür hat man dann Zugriff zu den neuesten Nachrichten, zu den neuesten Marktangeboten, zu den aktuellen Aktienkursen und zu verschiedenen anderen Informationen. Darüberhinaus wird meist auch die Benutzung umfangreicher Programmbibliotheken angeboten. Solche Informationsdienstleistungen sind z.Z. gerade im Entstehen begriffen. Man kann mit Sicherheit davon ausgehen, daß sie in den nächsten Jahren gewaltig anwachsen werden, ihr Angebotsspektrum wird auch breiter und sicher auch durchdachter.

---

[1] Bei Benutzung eines Modems, der für Direktverbindungen konstruiert ist, ist der Computer direkt mit einer Telefonleitung (Festverbindung) verbunden; die Verwendung eines Telefonhörers entfällt hierbei.

16 Vertiefung und Erweiterung der Kenntnisse

Wir haben uns bemüht, dem Leser diejenigen Kenntnisse zu vermitteln, die er für das Arbeiten mit dem IBM Personalcomputer braucht. Unsere Wissensvermittlung bezog auch die Programmiersprache BASIC ein, deren Kenntnis es dem Leser ermöglicht, eigne Programme zu verfassen und ausführen zu lassen. Natürlich konnten wir dabei nur die Oberfläche der Computerwissenschaften ankratzen; bei den Anwendungen mußten wir uns ebenfalls auf die Grundlagen beschränken. Deshalb wollen wir in diesem letzten Kapitel einige Worte über die von uns bisher nicht berührten Sachgebiete verlieren und in unseren Diskussionen auch die Punkte streifen, die richtungweisend für die Weiterarbeit zur Vertiefung und Ergänzung des Wissens maßgebend sein können.

16.1 Programmierung in der Assemblierersprache (AS)

Bei unserer gesamten bisherigen Programmierung haben wir uns ausschließlich der Programmiersprache BASIC bedient. Dem IBM Personalcomputer liegt jedoch eine viel primitivere Sprache zugrunde, nämlich die 8088-Maschinensprache. Tatsächlich ist der BASIC-Interpretierer selbst ein Programm, das in dieser Maschinensprache geschrieben ist. In der Tat sind viele kommerziell genutzte Programme direkt in der Maschinensprache abgefaßt.

Die soeben genannte Maschinensprache besteht aus Instruktionen, die unmittelbar vom 8088-Mikrobaustein (Chip) ausgeführt werden können. Diese Instruktionen sind weitaus primitiver als die Statements, die wir bei der Behandlung der höheren Programmiersprache BASIC kennengelernt haben. Diese Aussage bedeutet nichts anderes, als daß eine BASIC-Anweisung im allgemeinen vielen Instruktionen der Maschinensprache entspricht. In gewissem Sinne ist das ziemlich unvorteilhaft, da man gewissermaßen ein Programm aus sehr, sehr vielen kleinen Schritten zusammenstellen muß. Allerdings sind die auf diese Weise aufgebauten Programme wirkungsvoller, sie laufen auch schneller als die in BASIC geschriebenen ab. Das gilt selbstverständlich allgemein nur dann, wenn erfahrene und kenntnisreiche Programmierer mit der Programmierung betraut wurden. Wird in der Maschinensprache programmiert, kommt hinzu, daß man dann besser die Vorgänge verstehen wird, die im Innern eines 8088-Mikrobausteins bei einer diesem zur Ausführung übertragenen Instruktion ablaufen.

Nach erfolgter Einarbeitung in die Programmiersprache BASIC könnte man sich in der nächsten Phase mit dem Studium der Maschinensprache beschäftigen. Zur Hilfe beim Start in dieses neue Wissensgebiet wollen wir kurzzeitig darüber sprechen, wie die Maschinensprache aussieht und arbeitet. Zunächst wollen wir eine von uns bereits früher getätigte Feststellung wiederholen: Alle internen Operationen eines Computers werden binär ausgeführt. Das betrifft auch die Instruktionen der Maschinensprache. Es ist jedoch außerordentlich schwierig, Programme in der binären Form zu schreiben; sie würden ja nichts anderes darstellen als eine lange Kette von Nullen und Einsen. Um den Programmierer von dieser geradezu schrecklichen Bürde zu befreien, hat man die Möglichkeit geschaffen, die Instruktionen der Maschinensprache durch mnemotechnische Begriffe auszudrücken. Mnemotechnische Begriffe sind Wörter, denen eine gewisse Bedeutung zukommt, so wie wir es für die Wörter kennengelernt haben, aus denen die BASIC-Statements gebildet werden. Die Sprache, die durch die Benutzung mnemotechnischer Begriffe entstanden ist, gehört zur Klasse der sogenannten maschinenorientierten Programmiersprachen.

Ein in einer maschinenorientierten Sprache geschriebenes Programm muß noch umgewandelt werden, und zwar in das Programm in der Maschinensprache. Diese Umwandlung wird durch ein Programm besorgt, das Assemblierer (englisch: Assembler) genannt wird; infolgedessen werden maschinenorientierte Programmiersprachen oft auch als Assemblierersprachen (englisch: Assembler languages) bezeichnet. Für das Umwandlungsprogramm ist das in der maschinenorientierten Sprache geschriebene Programm das sogenannte Quellenprogramm (englisch: source program), das nach der Umwandlung entstandene Programm das Objektprogramm (englisch: object program). Wenn man einmal ein Objektprogramm auflistet, wird man erkennen, daß es außerordentlich schwierig zu lesen ist, was bei einer Folge, die nur aus Nullen und Einsen besteht, nicht verwunderlich ist. Um auch diese Bürde vom Programmierer zu nehmen, hat man in der Computerwissenschaft eine Schreibweise eingeführt, die auf dem sedezimalen Zahlensystem (englisch: hexadecimal system) basiert. Im Sedezimalsystem werden 16 Symbole verwendet, nämlich 0 bis 9 sowie A bis F. Auflistungen von Objektprogrammen werden vorzugsweise unter Benutzung dieses Systems durchgeführt. Darüberhinaus werden auch alle Speicheradressen in diesem Zahlensystem ausgedrückt. Wegen der direkten Beziehung dieses Zahlensystems zum dualen Zahlensystem (bekanntlich ist $2^4 = 16$) ist die sedezimale Schreibweise auch für den Computer unmittelbar verständlich.

Für die Programmumwandlung durch den Assemblierer muß vorgegeben werden, welchen Speicherplätzen das Objektprogramm zugeordnet werden soll. Um diese zusätzliche Erschwernis braucht man sich bei der Programmierung in BASIC nicht zu kümmern. BASIC selbst sucht entsprechende freie Speicherplätze auf; außerdem führt BASIC Buch über die Speicherplätze, in denen die verschiedenen Programmteile untergebracht sind. Bei der Programmierung in der maschinenorientierten Sprache oder gar in der Maschinensprache selbst liegt die Verantwortung über die Belegung der Speicherplätze und die Buchführung über die Belegung beim Programmierer selbst. Nach der Assemblierung kann das Objektprogramm in den RAM geladen und danach ausgeführt werden.

Nach dieser Betrachtung wird man sich mit Recht fragen, ob die Programmierung in der maschinenorientierten Sprache der Mühe wert ist, die gemäß der obigen Erörterungen anfällt. Das wird mit Sicherheit dann nicht der Fall sein, wenn das zu erstellende Programm nur ein einziges Mal oder nur wenige Male ausgeführt werden muß oder aber nur sporadisch (in größeren Zeitabständen) ablaufen soll. Plant man jedoch den Entwurf eines Programmes, das ständig, zumindest öfters benutzt werden soll oder muß, beispielsweise als Unterprogramm in zahlreichen BASIC-Programmen, dann wird es sich möglicherweise lohnen, das Mehr an Zeit zu investieren, das man braucht, um ein Programm statt in BASIC in der Assemblierersprache zu schreiben. Als Entscheidungskriterium sollte dabei gelten, daß die Vorteile der Programmierung in der Assemblierersprache die geschilderten Nachteile aufheben. Als Vorteile sind dabei anzusehen:

1. Die Ausführungszeit des Programmes wird kürzer, d. h. das Programm läuft schneller ab.
2. Es kann die Ausführung von Operationen vorgesehen werden, die in BASIC entweder nur sehr plump oder aber überhaupt nicht ausgedrückt werden können; hier sind insbesonders diejenigen Aktionen gemeint, die spezielle Funktionen von Datensichtgeräten, Tastaturen und Druckern betreffen.

Über die Programmierung des IBM Personalcomputers in der Assemblierersprache gibt es eine ganze Reihe von Schriften, die wir dem Leser zum Studium empfehlen können.

16.2 Andere Programmiersprachen und Betriebssysteme

BASIC ist nur eine der zahlreichen heute existierenden Computersprachen. Auch für die IBM Personalcomputer stehen neben BASIC noch weitere Programmiersprachen zur Verfügung. Die Beherrschung einer dieser Sprachen oder gar mehrerer kann durchaus als Ziel ergänzender Studien gelten.

Viele der heute auch für Mikrocomputer zur Verfügung stehenden Programmiersprachen wurden ursprünglich für die Verwendung auf großen Computersystemen entworfen und verwirklicht. Als aber sich der Einsatz der Mikrocomputer immer mehr verbreiterte, wurden diese Sprachen auch für die Mikrocomputer übernommen. Jede Aufstellung von Programmiersprachen, die für die IBM Personalcomputer verfügbar sind, wird höchstwahrscheinlich bereits zu dem Zeitpunkt unvollständig sein, an dem dieses Buch in den Druck geht. Nichtsdestotrotz wollen wir zumindest die Programmiersprachen erwähnen, die in der Praxis häufig eingesetzt werden und bereits zur Verfügung stehen oder demnächst verfügbar sind.

Als klassische, inzwischen schon mehrere Male standardisierte problemorientierte Programmiersprache ist insbesondere FORTRAN zu erwähnen. Diese ausdrucksstarke, mächtige Sprache wird besonders für mathematisch-technische und für wissenschaftliche Anwendungen eingesetzt.

Für die FORTRAN-Sprache steht ein Kompilierer bereit; bekanntlich arbeitet BASIC mit einem Interpretierer. Hinsichtlich ihrer Arbeitsweise besteht ein gravierender Unterschied. Bei einem Interpretierer wird das eingegebene Programm direkt ausgeführt. Das geschieht dadurch, daß das Interpretierprogramm die zum eingegebenen Programm gehörenden Anweisungen nacheinander durchgeht, untersucht und sofort danach durch entsprechende Maschineninstruktionen ausführen läßt. Genauer gesagt, um eine Anweisung auszuführen, ruft der Computer ein Unterprogramm in der Maschinensprache auf, das die durch die Anweisung beabsichtigte Aktion interpretiert und ausführt. Bei Vorhandensein eines Kompilierers wird das in der entsprechenden Programmiersprache vorliegende Programm genau so eingegeben wie beim Vorhandensein eines Interpretierers. Jedoch muß das Programm vor seiner Ausführung erst kompiliert werden. Das bedeutet nichts anderes, als daß zunächst ein spezielles Programm, der Kompilierer, ablaufen muß. Der Kompilierer wandelt die Anweisungen des vom Benutzer geschriebenen Programmes, des Quellenprogrammes also, in Instruktionen der Maschinensprache um: Aus dem Quellenprogramm wird also zunächst ein Objektprogramm erzeugt. Das Objektprogramm kann dann ausgeführt werden. Ein kompiliertes Programm ist in der Ausführung ohne Zweifel weitaus effizienter als ein Programm, das mit Hilfe eines Interpretierers ausgeführt wird. Abhängig vom Inhalt eines Programmes ergaben sich Ausführungszeiten, die bei vorhergehender Kompilierung um den Faktor 5 bis 50 schneller waren als bei reiner Interpretierung. Bei dieser Zeitanalyse muß man freilich beachten, daß die Kompilierung eine zusätzliche Zeit erfordert. Außerdem ist für die Speicherung des Objektprogrammes zusätzlicher Speicherplatz bereitzustellen.

Man kann sich für den eigenen IBM Personalcomputer neben dem BASIC-Interpretierer auch noch einen BASIC-Kompilierer beschaffen. Damit erreicht man, daß man nach wie vor mit der Programmiersprache arbeiten kann, die man bereits kennt. Natürlich hat man dann sein Wissen um die Fakten zu erweitern, die bei der Benutzung des Kompilierers zu beachten sind. Durch Einsatz eines BASIC-Kompilierers können somit die soeben geschilderten Vorteile kompilierter Programme voll ausgenutzt werden.

COBOL und PASCAL sind zwei populäre Programmiersprachen. COBOL ist augenscheinlich die am häufigsten in der Praxis eingesetzte Programmiersprache für kaufmännische (kommerzielle) Anwendungen. Sie ist bereits beim Entwurf darauf abgestellt worden, die Programmierung von Managementberichten und Finanzplänen bzw. -ergebnissen möglichst einfach zu gestalten. PASCAL ist eine beachtlich mächtige Programmiersprache, die bei der Programmierung allgemeiner Probleme eingesetzt werden kann. Ihre Sprachelemente erlauben, daß komplizierte Programme mit sehr wenigen Statements niedergeschrieben werden können.

Bei der Anschaffung von Interpretierern oder Kompilierern für weitere Programmiersprachen muß man unbedingt darauf achten, daß die zu erwerbende Version mit dem auf dem eigenen Computer eingesetzten Betriebssystem verträglich ist. Als Betriebssystem für den IBM Personalcomputer wird in der Regel das Betriebssystem „IBM DOS" eingesetzt. Um andere Programmiersprachen oder bestimmte Anwendungsprogramme verwenden zu können, muß man mitunter auf andere Betriebssysteme ausweichen. Der gegenwärtige Softwaremarkt offeriert in solchen Fällen hauptsächlich die beiden folgenden Betriebssysteme:

- CP/M-86
  Die Bezeichnung CP/M-86 ist für „Digital Research Corporation" geschützt.
- UNIX
  Die Bezeichnung UNIX ist für „Bell Telephone Laboratories, Inc." geschützt.

16.3 Vorschläge für die eigene Weiterbildung

Über neue Erkenntnisse, technische Weiterentwicklungen, neue Geräte und über Anwendungsentwicklungen sollte man sich ständig durch das regelmäßige Studium der Fachzeitschriften informieren, die sich mit dem IBM Personalcomputer beschäftigen. Im deutschsprachigen Raum erscheint bereits eine ganze Reihe ausgezeichneter Fachorgane, deren Inhalt wertvolle Anregungen für den nutzbringenden Einsatz des eigenen Personalcomputers geben kann. Laufend kommen neue Zeitschriften hinzu.

Darüberhinaus können PC-Benutzer auch wertvolle Anregungen aus dem Studium allgemeiner Computer-Fachzeitschriften und Periodika gewinnen.

Der Bezug aller Fachzeitschriften kann über die üblichen Vertriebswege der deutschsprachigen Länder erfolgen. Viele Computerfachgeschäfte bieten ebenfalls Kaufmöglichkeiten für einige oder alle Fachzeitschriften an.

Für die Weiterbildung steht dem Leser bereits heute eine breitgefächerte Palette von Fachbüchern in englischer und deutscher Sprache zur Verfügung. Auskunft und Ratschläge wird man sicher einmal aus der Fachpresse, andererseits auch über die Fachgeschäfte bekommen. Die wichtigsten Bücher werden die meisten Fachgeschäfte auch in ihrem Sortiment führen.

Wir hoffen, daß dieses Buch das Interesse des Lesers an den Mikrocomputern soweit geweckt hat, daß er von sich aus nach einer Erweiterung seiner Kenntnisse strebt.

Lösungen ausgewählter Aufgaben

Aufgabengruppe 1

Die einzelnen Zeilen der Aufgaben sind nacheinander über die Tastatur einzugeben. Nach der Eingabe jeder Zeile ist die Eingabetaste zu betätigen.
Nach Durchführung jeder Aufgabe sind die beiden Tasten

Ctrl und *Home*

gemeinsam, d.h. gleichzeitig zu betätigen, um den Bildschirm zu löschen.

Aufgabengruppe 2

1.
```
10 PRINT 57+23+48
20 END
```

2.
```
10 PRINT 57.83*(48.27-12.54)
20 END
```

3.
```
10 PRINT 127.86/38
20 END
```

4.
```
10 PRINT 365/0.005+1.02^5
20 END
```

5.
```
10 PRINT 2^1,2^2,2^3,2^4
20 PRINT 3^1,3^2,3^3,3^4
30 PRINT 4^1,4^2,4^3,4^4
40 PRINT 5^1,5^2,5^3,5^4
50 PRINT 6^1,6^2,6^3,6^4
60 END
```

6.
```
10 PRINT "SCHIENE",45
20 PRINT "BEHANDLUNG",35
30 PRINT "MEDIKAMENTE",5
40 PRINT
50 PRINT "GESAMTSUMME",45+35+5
60 PRINT "ABZ. VERS.",0.8*(45+35+5)
70 PRINT "RECHN.-BETR.",0.2*(45+35+5)
80 END
```

7.
```
10 PRINT "TISCHLER",698+732+129+487
20 PRINT "HOFFMANN",148+928+246+201
30 PRINT "WEBER",379+1087+148+641
40 PRINT
50 PRINT "GESAMTSTIMMEN",698+732+129+487+148+928+246+201
                         +379+1087+148+641
60 END
```

<u>Anmerkung</u>: Man beachte, daß die Zeile mit der Zeilennummer 50 über eine Bildschirmzeile hinausgeht. Um eine solche Programmzeile einzugeben, darf man am Ende einer Bildschirmzeile nicht die Eingabetaste anschlagen, sondern muß mit dem Eingeben fortfahren (hier also fortlaufend die Leertaste betätigen).

8. 2

9.
```
SILBER        GOLD          KUPFER        PLATIN
327           448           1Ø52          2
```

10.
```
              LEBENSMITTEL  FLEISCH       DROGERIE
MON           1,245         2,348         2,531
DIE            248          3,459         2,148
```

11. 2.3E7

12. 1.7525E2

13. −2E8

14. 1.4E-4

15. −2.75E-10

16. 5.342E16

17. 159,000

18. −20,345,600

19. −.000000000007456

20. .00000000000000000239456

Aufgabengruppe 3

1. 10

2. 0

3. 50

4. 9 −7 18

5. HANS ABEL ALTER 38

6. 22
   57

7. Der Variablen A dürfen nur numerische Werte zugewiesen werden.

8. Dieses Statement weist keinen Fehler auf.

9. Der Variablen A$ dürfen nur Zeichenkettenkonstanten als Werte zugewiesen werden.

10. Es fehlt die Zeilennummer. Außerdem muß die Zeichenkettenkonstante in Anführungszeichen eingeschlossen werden.

11. Es liegt kein Fehler vor.

12. Ein Variablenname muß stets mit einem Buchstaben beginnen.

13.
```
10  LET  A=2.3758 :   B=4.58321 :   C=58.11
20  PRINT  A+B+C
30  PRINT  A*B*C
40  PRINT  A^2+B^2+C^2
50  END
```

14.
```
10  LET  A$="Büro" :   B$="DV-Abt." :   C$="Pressedienst"
20  LET  EA=346712 :   EB=459321 :      EC=367872
30  LET  AA=176894 :   AB=584837 :      AC=402195
40  PRINT  ,A$,B$,C$
50  PRINT  "Einkünfte",EA,EB,EC
60  PRINT  "Ausgaben",AA,AB,AC
70  LET  GA=EA-AA :    GB=EB-AB :       GC=EC-AC
80  PRINT  "Profit",GA,GB,GC
90  PRINT
100 PRINT  "Gesamter Profit aller drei Abteilungen: ",GA+GB+GC
110 END
```

Aufgabengruppe 4

1. Die Summe ergibt sich zu (A + B): 36.6
   Das Produkt ergibt sich zu (A*B): 334.25

5. Die eingefügte Zeile mit der Zeilennummer 25 führt zu der folgenden Ausgabe: 671.06

Aufgabengruppe 6

1.
```
10  LET  S = 0
20  FOR  J = 1  TO  25
30     LET  S = S + J^2
40  NEXT J
50  PRINT  S
60  END
```

2.
```
10  LET  S = 0
20  FOR  J = 0  TO  10
30     LET  S = S + (1/2)^J
40  NEXT J
50  PRINT S
60  END
```

3.
```
10  LET  S = 0
20  FOR  J = 1  TO  10
30     LET  S = S + J^3
40  NEXT J
50  PRINT S
60  END
```

4.
```
10  LET  S = 0
20  FOR  J = 1  TO  100
30     LET  S = S + 1/J
40  NEXT J
50  PRINT  S
60  END
```

5.
```
10  PRINT  "N","N^2","N^3",N^4"
20  FOR  N = 1  TO  12
30     PRINT  N,N^2,N^3,N^4
40  NEXT N
50  END
```

6.
```
10  PRINT  "Monat","Zinsen","Kontenstand"
20  B = 4000 :       Z = 125.33
30  FOR  J = 1  TO  12
40    I = 0.01*B :        'I --- Zinsbetrag pro Monat
50    T = Z - I  :        'T --- Tilgung pro Monat
60    B = B - T  :        'Errechnung des neuen Kontostandes
70    PRINT  J,I,B
80  NEXT J
90  END
```

7.
```
10  PRINT  "Jahresende","Kontenstand"
20  K = 1000
30  FOR  I = 1  TO  15
40     K = K + 1000 + 0.1*K :    'Addition der neuen Einlage
50                               'und der Zinsen
60     PRINT I,K
70  NEXT I
80  END
```

8.
```
10  LET  S = 3.5E7 :      P = 5.54E6
20  PRINT  "Jahresende","Verkauf","Gewinn"
30  FOR  J = 1  TO  3
40     LET  S = 1.2*S :      P = 1.3*P
50     PRINT  J,S,P
60  NEXT J
70  END
```

Aufgabengruppe 7

1.
```
10  J = 1
20  IF  J^2 >= 45000  THEN 100  ELSE 30
30  PRINT J,J^2
40  J = J + 1
50  GOTO 20
100 END
```

2.
```
10  PI = 3.14159
20  R = 1
30  IF  PI+R^2 <= 5000  THEN 40  ELSE 100
40  PRINT R,PI+R^2
50  R = R + 1
60  GOTO 30
100 END
```

3.
```
10  PRINT "Seitenlänge","Volumen"
20  S = 1
30  V = S^3
40  IF  V < 175000  THEN 50  ELSE 100
50  PRINT S,V
60  S = S + 1
70  GOTO 30
100 END
```

4.
```
10  FOR J=1 TO 10 :      'Schleife zum Lösen von 10 Aufgaben
20    INPUT "Eingabe zweier zweiziffrigen Zahlen: "; A,B
30    INPUT "Wie lautet ihr Produkt? "; C
40    IF  A*B=C  THEN 200
50    PRINT "Falsch --- Die richtige Antwort ist:  ",A*B
60    GOTO 500 :         'Übergang zur nächsten Aufgabe
200   PRINT "Richtig --- Gratulation"
210   LET  PZ = PZ + 1 :'Erhöhung der Punktzahl
500 NEXT J
600 PRINT "Erreicht wurden ",PZ,"Punkte von 10 möglichen"
700 PRINT "Zur Wiederholung bitte den Befehl  RUN  eingeben"
800 END
```

5.
```
10  FOR J=1 TO 10 :      'Schleife zum Lösen von 10 Aufgaben
20    PRINT "Welche Rechenart soll durchgeführt werden?"
30    PRINT "  Addition (A), Subtraktion(S) oder Multiplikation(M): "
40    INPUT A$
50    INPUT "Eingabe zweier zweiziffrigen Zahlen: "; A,B
60    IF  A$="A"  THEN 100 :     'Addition
70    IF  A$="S"  THEN 200 :     'Subtraktion
80    IF  A$="M"  THEN 300 :     'Multiplikation
100   INPUT "Wie lautet ihre Summe? "; C
110   IF  A+B=C  THEN 400
120   PRINT "Falsch --- Die richtige Antwort ist:  ",A+B
130   GOTO 500 :                 'Übergang zur nächsten Aufgabe
200   INPUT "Wie lautet ihre Differenz? "; C
210   IF  A-B=C  THEN 400
220   PRINT "Falsch --- Die richtige Antwort ist:  ",A-B
230   GOTO 500 :                 'Übergang zur nächsten Aufgabe
300   INPUT "Wie lautet ihr Produkt? "; C
310   IF  A+B=C  THEN 400
320   PRINT "Falsch --- Die richtige Antwort ist:  ",A+B
330   GOTO 500 :                 'Übergang zur nächsten Aufgabe
400   PRINT "Richtig --- Gratulation"
410   LET  PZ = PZ + 1 :         'Erhöhung der Punktzahl
500 NEXT J
600 PRINT "Erreicht wurden ",PZ,"Punkte von 10 möglichen"
700 PRINT "Zur Wiederholung bitte den Befehl  RUN  eingeben"
800 END
```

6. Das Lösungsprogramm ist leicht zu entwickeln, wenn man die Lösung der Aufgabe 8. betrachtet.

7. Das Lösungsprogramm ist leicht zu entwickeln, wenn man die Lösung der Aufgabe 8. betrachtet.

8.
```
10  INPUT "Anzahl der Zahlen eingeben: "; N
11  FOR  J = 1  TO  N
12    INPUT A
13    IF  J=1  THEN M=A
14    IF  A>M  THEN M=A
15  NEXT J
16  PRINT "Die größte eingegebene Zahl ist: ",M
17  END
```

9. Bei der Lösung von Aufgabe 8. ist die Zeile mit der Zeilennummer 14 durch die nachfolgende Zeile zu ersetzen:

```
14    IF  A<M  THEN M=A
```

10.
```
10  A80 = 5782 :     A81 = 6548 :     B80 = 4811 :     B81 = 6129
20  C80 = 3865 :     C81 = 4270 :     D80 = 7950 :     D81 = 8137
30  E80 = 4781 :     E81 = 4248 :     F80 = 6598 :     F81 = 7048
40  FOR  J = 1  TO  6
50    IF  J=1  THEN A=A80 :  B=A81
60    IF  J=2  THEN A=B80 :  B=B81
70    IF  J=3  THEN A=C80 :  B=C81
80    IF  J=4  THEN A=D80 :  B=D81
90    IF  J=5  THEN A=E80 :  B=E81
100   IF  J=6  THEN A=F80 :  B=F81
110   DIFF = B - A
120   IF  DIFF>0  THEN PRINT "Stadt",J,"hat ein Wachstum von",DIFF
130   GOTO 200
140   IF  DIFF<0  THEN PRINT "Stadt",J,"verzeichnet ein Minus von",DIFF
150   GOTO 300
200   IF  DIFF>500  THEN PRINT "  Das Wachstum ist größer als 500"
300 NEXT J
400 END
```

11.
```
10  REM    Simulation einer Registrierkasse
20  REM    ================================
30  PRINT "Dieses Programm simuliert eine Registrierkasse"
40  PRINT "Nach Erscheinen von Fragezeichen bitte den Betrag eintippen"
50  PRINT "  Nach Eingabe aller Beträge ist  -1  einzutippen"
60  INPUT "Eingabe von  J  , wenn fertig "; A$
70  IF  A$="J"  THEN  80  ELSE 30
80  CLS
90  INPUT "Betrag: "; A
100 IF  A=-1  THEN 200
110 SUMME = SUMME + A
120 GOTO 90
200 PRINT
210 PRINT "Summe:",SUMME
220 ST = 0.14*SUMME :          'Errechnung der Mehrwertsteuer
230 PRINT "Mehrwertst.:",ST
240 PRINT "Rechnungssumme beträgt:",SUMME+ST
250 INPUT "Zahlung: "; ZAHLUNG
260 PRINT "Wechselgeld:", ZAHLUNG-(SUMME+ST)
300 END
```

12.
```
10  INPUT "Kassenbestand: "; B
20  PRINT "Eingabe der im nächsten Monat eingehenden Beträge"
30  PRINT "  Nach dem letzten Betrag ist  -1  einzugeben"
40  INPUT "Erwarteter Betrag: "; A
50  IF  A=-1  THEN 100
60  C = C + A :     'C --- Gesamtbetrag der erwarteten Beträge
70  GOTO 40
100 PRINT
110 PRINT "Eingabe der im nächsten Monat zu zahlenden Beträge"
120 PRINT "  Nach dem letzten Betrag ist  -1  einzugeben"
130 INPUT "Zu zahlender Betrag: "; D
140 IF  A=-1  THEN 200
150 E = E + D :     'E --- Gesamtbetrag der zu zahlenden Beträge
160 GOTO 130
200 PRINT
210 PRINT "Kassenbestand: ",B
220 PRINT "Voraussichtlich eingehende Beträge: ",C
230 PRINT "Voraussichtlich zu zahlende Beträge: ",E
240 PRINT
250 PRINT "Voraussichtlicher Kassenbestand im nächsten Monat: ",
          B+C-E
260 END
```

Aufgabengruppe 8

1.
```
1000 'Ausgabe von 10 Sternzeichen
1010    LOCATE L,1
1020    PRINT "**********"
1030 RETURN
```

2.
```
1000 'Ausgabe von  m  Sternzeichen
1010    LOCATE L,1
1020    FOR J=1 TO M
1030      PRINT "*";
1040    NEXT J
1050 RETURN
```

3.
```
1000 'Ausgabe von  m  Sternzeichen an einer beliebigen Stelle
1010    LOCATE L,K
1020    FOR J=1 TO M
1030      PRINT "*";
1040    NEXT J
1050 RETURN
```

4.
```
10  REM Fall a)
20  K = 5:     L = 3:     M = 30
30  GOSUB 1000
40  REM Fall b)
50  K = 4:     L = 5:     M = 35
60  GOSUB 1000
70  REM Fall c)
80  K = 8:     L = 7:     M = 12
90  GOSUB 1000
100 END
```

Anmerkung: ***Den Zeilen dieses Hauptprogrammes müssen die Zeilen des Unterprogrammes (siehe Lösung der Aufgabe 3) folgen.***

5. Die Programmablaufsteuerung bewirkt eine Verzweigung zu den Zeilen mit den folgenden Zeilennummern:
   a) 200
   b) 500
   c) Keine Verzweigung: Sequentielles Weitergehen zur Zeile, die sich unmittelbar hinter der Zeile mit der Zeilennummer 10 befindet
   d) wie bei Teilaufgabe c)
   e) Die Programmausführung wird mit einer Fehlernachricht abgebrochen.

6. Die Statements auf den Zeilen mit den Zeilennummern 200 und 50.

Aufgabengruppe 9

1. DIM A(5)

2. DIM A(2,3)

3. DIM A$(3)

4. DIM A(3)

5. DIM A$(4),B(4)

6.
```
10    DIM A$(3),B(3,3),C$(3)
20    PRINT ,,"WARENEINGÄNGE IN DM"
30    C$(1)="Geschäft 1" :   C$(2)="Geschäft 2" :
      C$(3)="Geschäft 3"
40    A$(1)="01.1.-10.1." :   A$(2)="11.1.-20.1." :
      A$(3)="21.1.-31.1."
50    B(1,1)=57385.48 :   B(1,2)=89485.45 :   B(1,3)=38456.90
60    B(2,1)=39485.98 :   B(2,2)=76485.49 :   B(2,3)=40387.86
70    B(3,1)=45467.21 :   B(3,2)=71494.25 :   B(3,3)=37983.38
100   PRINT  ,C$(1),C$(2),C$(3)
200   FOR I=1 TO 3
210     PRINT A$(I),B(I,1),B(I,2),B(I,3)
220   NEXT I
1000  END
```

7. In das Lösungsprogramm von Aufgabe 6. sind die folgenden Anweisungen einzufügen:
```
5     DIM D(3)
240   FOR I=1 TO 3
250     D(I)=B(1,I)+B(2,I)+B(3,I)
260   NEXT I
270   PRINT "Summen:",D(1),D(2),D(3)
```

8. In das Lösungsprogramm von Aufgabe 7. sind die folgenden Anweisungen einzufügen:
```
6     DIM E(3)
300   FOR I=1 TO 3
310     E(I)=B(I,1)+B(I,2)+B(I,3)
320   NEXT I
330   PRINT
340   PRINT "Dekade","Gesamte Wareneingänge"
350   FOR I=1 TO 3
360     PRINT A$(I),E(I)
370   NEXT I
```

9.
```
10    DIM A$(4),B$(5),C(5,4)
20    A$(1)="Geschäft 1" :   A$(2)="Geschäft 2" :
      A$(3)="Geschäft 3" :   A$(4)="Geschäft 4"
30    B$(1)="Kühlschränke" :     B$(2)="Öfen" :
      B$(3)="Waschautom."  :     B$(4)="Wäschetrockn."
40    B$(5)="Klimaanlagen"
50    PRINT "Eingabe der augenblicklichen Lagerbestände"
60    FOR I=1 TO 4
70      PRINT A$(I)
80      PRINT
90      FOR J=1 TO 5
100       PRINT B$(J)
110       INPUT C(J,I)
120     NEXT J
130   NEXT I
200   REM  Ab hier:  Aufdatierung des Warenbestandes
210   PRINT "Auswahl unter folgenden Möglichkeiten:"
220   PRINT "   Warenbewegungen (W)"
230   PRINT "   Anzeige des augenblicklichen Bestandes (A)"
240   INPUT "Möglichkeit  W  oder  A  : "; M$
250   IF  M$="W"  THEN 300
260   IF  M$="A"  THEN 600  ELSE CLS: GOTO 200
300   CLS :           'Warenbewegungen
310   PRINT "Warenbewegungen"
320   INPUT "Eingabe der Geschäftsnummer (1 bis 4): "; NR
330   PRINT "Kennzahlen: Kühlschr. (1), Öfen (2), Waschaut. (3)," ;
            "            Wäschetr. (4), Klim.(5)"
340   INPUT I
350   INPUT "Anzahl: "; ANZ
360   C(I,NR) = C(I,NR) - ANZ
370   GOTO 200
600   CLS :           'Anzeige des Warenbestandes
610   PRINT ,A$(1),A$(2),A$(3),A$(4)
620   FOR I=1 TO 5
630     PRINT
640     PRINT B$(I),C(I,1),C(I,2),C(I,3),C(I,4)
650   NEXT I
660   GOTO 200
1000  END
```

Anmerkungen: *Es sei darauf hingewiesen, daß dieses Programm eine endlose Schleife enthält (ab der Zeile mit der Zeilennummer 200 bis zur Zeile mit der Zeilennummer 660). Bei dieser Art von Programmen erweist sich ein solches Vorgehen als außerordentlich nützlich. Man will vermeiden, daß zufällig das Programmende angesteuert wird, evtl. durch eine versehentliche falsche Eingabe.*
Dieses Programm kann nur durch die Betätigung der Unterbrechungstaste (Tastenkombination Ctrl *und* Break*) beendet werden.*

Aufgabengruppe 10

1. Den Bereichselementen A(1) bis A(9) werden der Reihe nach die folgenden Werte zugewiesen:
   2, 4, 6, 8, 10, 12, 14, 16, 18, 20

2. a) Den Bereichselementen A(0) bis A(3) werden der Reihe nach die folgenden Werte zugewiesen:
   1.1, 3.3, 5.5, 7.7
   b) Den Bereichselementen B(0) bis B(3) werden der Reihe nach die folgenden Werte zugewiesen:
   2.2, 4.4, 6.6, 8.8

3. a) Den Bereichselementen A(0) bis A(3) werden der Reihe nach die folgenden Werte zugewiesen:
   1, 2, 3, 4
   b) Den Bereichselementen B$(0) bis B$(3) werden der Reihe nach die folgenden Werte zugewiesen:
   "A", "B", "C", "D"

4. Den Bereichselementen A(0) bis A(3) bzw. B(0) bis B(3) werden der Reihe nach die folgenden Werte zugewiesen:
   Bereich A → 1, 3, 1, 3
   Bereich B → 2, 4, 2, 4

5. Den Bereichselementen werden folgende Werte zugewiesen:

1 5 9	2 6 10	3 7 11	4 8 12
A(I,1) mit I=1,2,3	A(I,2) mit I=1,2,3	A(I,3) mit I=1,2,3	A(I,4) mit I=1,2,3

6. Den Bereichselementen werden folgende Werte zugewiesen:

1 2 3	4 5 6	7 8 9	10 11 12
A(I,1) mit I=1,2,3	A(I,2) mit I=1,2,3	A(I,3) mit I=1,2,3	A(I,4) mit I=1,2,3

7. Fehlernachricht:

   Out of Data in 30

   Die Ursache hierfür ist darin zu suchen, daß in dieser Anweisung ein 5. Wert gelesen werden soll; das DATA-Statement auf der Zeile mit der Zeilennummer 50 enthält aber nur vier Werte.

8. Fehlernachricht:

   Typ Mismatch in 30

   Die Ursache hierfür ist darin zu suchen, daß in dieser Anweisung eine Zeichenkettenkonstante einer numerischen Variablen zugewiesen werden soll.

10.
```
10  DIM T(23)
20  DATA  10,10,9,9,8,11,15,18,20,25,31,35,38,39,40,40
30  DATA  42,38,33,27,22,18,15,12
40  FOR  I = 0  TO  23
50    READ T(I)
60    SUMME = SUMME + T(I)
70  NEXT I
80  CLS
90  PRINT "Die Durchschnittstemperatur des Tages betrug: ",
          SUMME/24
100 PRINT "Temperaturermittlung für eine bestimmte Tageszeit"
110 PRINT "--- Beendigung durch Eingabe eines Zahl größer als 23"
120 INPUT "Eingabe der Tageszeit (0 bis 23): "; ZEIT
130 IF  ZEIT>23  THEN 200
140 PRINT "Die erfragte Temperatur betrug in Grad Celsius: ",
          T(ZEIT)
150 GOTO 100
200 END
```

Aufgabengruppe 11

9.
```
1 PRINT "Programmende bei Eingabe von 0"
2 INPUT "Eingabe der zu rundenden Zahl: "; ZAHL
3 IF  ZAHL=0  THEN END
4 PRINT USING "######"; ZAHL
5 GOTO 2
```

Durch die Eingabe des Befehles RUN wurde ein Programmablauf veranlaßt, der die folgenden Resultate erbrachte:

```
Programmende bei Eingabe von 0                  → A
Eingabe der zu rundenden Zahl: ? 125.6          → A und E
   126                                          → A
Eingabe der zu rundenden Zahl: ? 125.4          → A und E
   125                                          → A
Eingabe der zu rundenden Zahl: ? -125.6         → A und E
  -126                                          → A
Eingabe der zu rundenden Zahl: ? -125.4         → A und E
  -125                                          → A
Eingabe der zu rundenden Zahl: ? 0              → A und E
Ok                                              → A (Programmende, d. h.
                                                    Rückkehr zu BASIC)
```

Hierbei bedeuten: *E* | *Eingabe*
A | *Ausgabe*

10. Das Lösungsprogramm zu Aufgabengruppe 7, Aufgabe 11., ist durch Statements zu ergänzen, die die eingegebenen Beträge sowie alle anderen Beträge nach Abschluß der Eingabe stellengerecht ausgeben (Dezimalpunkt unter Dezimalpunkt); hierzu sind PRINT-Anweisungen mit der Option USING zu verwenden.

11. Die Zeileneinteilung ist neu zu entwerfen und das Programm (Aufgabe 6.) entsprechend abzuändern. Dabei sollte nicht vergessen werden, die Anweisung

    WIDTH 40

    ins Programm aufzunehmen.

Aufgabengruppe 12

1. 100*RND
2. 100+RND
3. INT(50*RND+1)
4. INT(4+77*RND)
5. 2*INT(25*RND+1)
6. 50+50*RND
7. 3*INT(9*RND+1)
8. 1+3*INT(7*RND+1)
9.

```
10  PRINT "Auswahl der durchzuführenden Rechenoperation"
20  PRINT "  Addition (A),  Subtraktion (S),  Multiplikation (M)"
30  INPUT A$
40  REM  Bestimmung der Operanden auf Zufallsbasis
50  REM  (Wertebereich 1 bis 9, ganzzahlig)
60  A = INT(10*RND) :          B = INT(10*RND)
70  IF  A$="A"  THEN 100
80  IF  A$="S"  THEN 200
90  IF  A$="M"  THEN 300
100 CLS :                  'Addition
110 PRINT "Welches Ergebnis ergibt sich bei ";A;"+";B;" ?"
120 INPUT ERGEBNIS
130 C = A + B
140 GOTO 400
200 CLS :                  'Subtraktion
210 PRINT "Welches Ergebnis ergibt sich bei ";A;"-";B;" ?"
220 INPUT ERGEBNIS
230 C = A - B
240 GOTO 400
300 CLS :                  'Multiplikation
310 PRINT "Welches Ergebnis ergibt sich bei ";A;"*";B;" ?"
320 INPUT ERGEBNIS
330 C = A * B
400 REM Überprüfung der Rechenergebnisse auf Richtigkeit
410 IF  ERGEBNIS = C  THEN 420  ELSE 440
420 PRINT "--- Richtig ---"
430 GOTO 450
440 PRINT "--- Falsch, das richtige Ergebnis lautet: "; C
450 INPUT "Soll eine weitere Aufgabe gelöst werden? (J/N): "; A$
460 IF  A$="J" OR A$="j"  THEN 10
470 PRINT
480 PRINT "Programmende"
490 END
```

Frage: Angenommen, es wird beim INPUT-Statement auf der Zeile mit der Zeilennummer 30 keine richtige Kennung eingegeben. Welche Rechenoperation wird in diesem Fall dann ausgeführt?

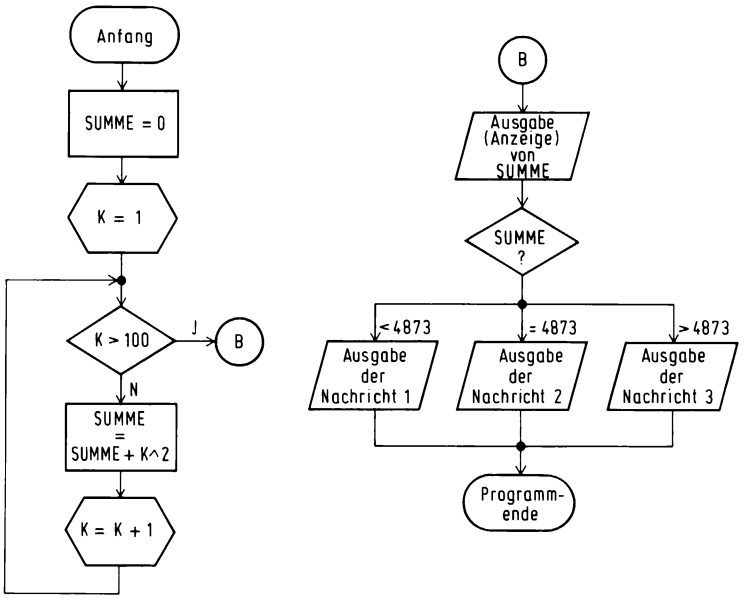
10.
```
10  CLS
20  DIM NAMEN$(10)
30  FOR  J = 1  TO  10
40     READ NAMEN$(J)
50  NEXT J
60  FOR  J = 1  TO  4
70     PRINT NAMEN$(10*RND)
80  NEXT J
100 DATA  ...
110 DATA  ...
 :
200 END
```

Anmerkung: ***Die Namen der der Auswahl unterliegenden Personen sind in die ab der Zeile mit der Zeilennummer 100 beginnenden DATA-Statements zu stellen.***

Aufgabengruppe 13

1. Die Aufgabenstellung erfordert keine Eingabe.

PAP



Aufgabengruppe 14

1. Es sind die nachfolgenden Berichtigungen vorzunehmen:

   a) Zeile 10
   Wegen der nicht übereinstimmenden Datentypen muß "0" durch 0 ersetzt werden.

   b) Zeile 20
   Der Endwert der Laufvariablen J muß in 50 geändert werden

   c) Zeile 30
   Das Zeichen) muß durch das Zeichen ∧ ersetzt werden.

   d) Zeile 40
   K muß durch J ersetzt werden.

   e) Zeile 80
   Das Statement muß richtig
   80 NEXT J
   lauten.

   f) Zeile 90
   Diese Zeile muß gestrichen werden.

   g) Zeile 100
   Das Produkt ST muß richtig zu S*T geschrieben werden.

   h) Zeile 110
   Diese Zeile muß richtig wie folgt lauten:

```
110 PRINT "ALS LOESUNG ERGIBT SICH ",A
```

   Das korrigierte Programm nimmt also die folgende Gestalt an:

```
1Ø  LET  S = Ø
2Ø  FOR  J=1 TO 5Ø
3Ø    S = S + J^2
4Ø  NEXT J
5Ø  LET  T = Ø
6Ø  FOR  J=1 TO 2Ø
7Ø    LET  T = T + J^3
8Ø  NEXT J
1ØØ LET  A = S*T
11Ø PRINT  "ALS LOESUNG ERGIBT SICH ",A
12Ø END
```

2. Es sind die nachfolgenden Berichtigungen vorzunehmen:

   a) Zeile 20
   Das Größer-als-Zeichen ist durch das Kleiner-als-Zeichen zu ersetzen.

   b) Zeile 30
   Es fehlt die Spezifikation von N

c) Zeile 110
Das Verzweigungsziel muß die Zeile mit der Zeilennummer 20 sein.

d) Es ist außerdem die Zeile
```
40 GOTO 200
```
in das Programm einzufügen.

Das korrigierte Programm nimmt also die folgende Gestalt an:

```
1Ø  LET  N = Ø
2Ø  IF  N^2 < 175263  THEN 1ØØ
3Ø  PRINT  "DAS KLEINSTE N IST GLEICH ",N
4Ø  GOTO 2ØØ
1ØØ N = N + 1
11Ø GOTO 2Ø
2ØØ END
```

Aufgabengruppe 15

1. CAS1:KOSTEN
2. A:EDITH
3. B:NIEDER.001
4. CAS1:FORM1040.82
5. gültiger Dateiname
6. Dateiname zu lang (mehr als acht Zeichen)
7. unzulässige Sonderzeichen in der Zeichenkette; infolgedessen liegt kein Dateiname vor
8. ungültiger Dateiname (GRAPH.001 könnte als Dateiname aufgefaßt werden, wenn ihm der Gerätename A: vorausgehen würde)
9. COPY B:TEST LPT1:
10. COPY A:*.COM B:
11. ERASE A:TEST
12. COPY A:TEST B:TEST3
13. COPY A:D??? B:

Aufgabengruppe 16

1.
```
10  DATA  5.7, -11.4, 123, 485, 49
20  OPEN "ZAHLEN" FOR OUTPUT AS #1
30  FOR  K = 1  TO  5
40    READ A
50    WRITE #1, A
60  NEXT K
70  CLOSE #1
80  END
```

2.
```
10  OPEN "ZAHLEN" FOR INPUT AS #1
20  FOR  K = 1  TO  5
30    INPUT #1, A
40    PRINT  A
50  NEXT K
60  CLOSE #1
70  END
```

3.
```
10  DATA  5, 78, 4.79, -1.27
20  OPEN "ZAHLEN" FOR APPEND AS #1
30  FOR  K = 1  TO  4
40    READ A
50    WRITE  #1, A
60  NEXT K
70  CLOSE #1
80  END
```

4.
```
10  OPEN "ZAHLEN" FOR INPUT AS #1
20  FOR  K = 1  TO  9
30    INPUT #1, A
40    PRINT  A
50  NEXT K
60  CLOSE #1
70  END
```

5.
```
5   OPEN "SCHECKS" FOR OUTPUT AS #1
10  CLS
20  PRINT  "EINGABE DER SCHECKDATEN"
30  INPUT "SCHECKNUMMER "; A
40  INPUT "AUSSTELLUNGSDATUM IN DER FORM TT.MM.JJ "; B$
50  INPUT "ZAHLUNGSEMPFAENGER "; C$
60  INPUT "BETRAG "; D
70  INPUT "VERWENDUNGSZWECK "; E$
80  WRITE #1, A,B$,C$,D,E$
90  PRINT  "SIND WEITERE SCHECKS VORHANDEN? (J/N):"
100 INPUT F$
110 IF  F$ = "J"  THEN 10  ELSE 200
200 WRITE #1, -9999
210 CLOSE #1
300 END
```

<u>Anmerkung</u>: Als letzter Satz wird in die Datei SCHECKS ein Satz mit der „künstlichen" Schecknummer -9999 hineingestellt. Diese Schecknummer kann dann in Folgeprogrammen anstelle einer EOF-Bedingung abgefragt werden; sie signalisiert hier also für Folgeprogramme eine leicht abzufragende Dateiendebedingung.

6.
```
5   SUMME = 0
10  OPEN "SCHECKS" FOR INPUT AS #1
20  INPUT #1, A
30  IF  A = -9999  THEN 200  ELSE 40
40  INPUT #1, B$,C$,D,E$
50  SUMME = SUMME + D
60  GOTO 20
200 PRINT  "GESAMTSUMME DER AUSGESTELLTEN SCHECKBETRAEGE: ", SUMME
210 CLOSE
300 END
```

7.
```
10   'Einfache Lagerverwaltung fuer Einzelhandelsgeschaefte
20   'Hauptprogramm
30   GOSUB 1010:          'Anzeige des Aktivitaetenmenues
40   ON REPLY  GOSUB 2010,3010,4010
50   INPUT "SOLL EINE NEUE AKTIVITAET ERFOLGEN? (J/N):"; Y$
60   IF  Y$ = "J"  THEN 30  ELSE 70
70   END
71   '-------------------------------------------------------------
1000 'Anzeige des Aktivitaetenmenues
1010     CLS
1020     PRINT  "LAGERVERWALTUNG"
1030     PRINT:   PRINT
1040     PRINT  "AUSWAHL DER AKTIVITAETEN"
1050     PRINT
1060     PRINT  "1. ANZEIGE DES LAGERBESTANDES EINES ARTIKELS"
1070     PRINT  "2. WARENEINGANG EINES ARTIKELS"
1080     PRINT  "3. VERKAUF EINES ARTIKELS"
1100     PRINT
1110     INPUT  "GEWUENSCHTE AKTIVITAET (1 BIS 3) EINGEBEN"; REPLY
1200 RETURN
1201 '-------------------------------------------------------------
2000 'Anzeige des Lagerbestandes
2010    CLS
2020    BEST = 0:              EPR = 0
2030    OPEN "LAGER" FOR INPUT AS #1
2040    INPUT #1, B$
2050    IF  B$ = "ENDE"  THEN 2200  ELSE 2060
2060    INPUT "BITTE ARTIKELBEZEICHNUNG EINGEBEN: "; A$
2070    INPUT #1, P,M
2080    IF  A$ = B$  THEN 2090  ELSE 2040
2090    BEST = BEST + M:       EPR = P
2100    GOTO 2040
2200    PRINT  A$,BEST,EPR
2300    CLOSE #1
2400 RETURN
2410 '-------------------------------------------------------------
3000 'Wareneingang eines Artikels
3010    CLS
3020    OPEN "LAGER" FOR APPEND AS #1
3030    INPUT B$,P,M
3040    WRITE #1, B$,P,M
3050    CLOSE #1
3100 RETURN
3110 '-------------------------------------------------------------
```

```
4000 'Verkauf eines Artikels
4010    CLS
4020    OPEN "LAGER" FOR APPEND AS #1
4030    INPUT B$,P,M
4035    M = -M
4040    WRITE #1, B$,P,M
4050    CLOSE #1
4100 RETURN
4110 '-----------------------------------------------------------
5000 END
```

Anmerkung: Pro Artikel können mehrere Sätze in der Datei vorliegen. Bei Verkäufen wird die Verkaufsmenge durch ein Minuszeichen gekennzeichnet, da sie einen Lagerabgang darstellt.

8.
```
10  OPEN "REZEPT" FOR OUTPUT AS #1
20  Z = 1
30  CLS
40  WRITE #1, "REZEPT ",Z
50  PRINT  "REZEPTZEILEN EINGEBEN"
60  PRINT  "ALS LETZTE ZEILE DIE ZEICHENKETTE  REZEPTENDE  EINGEBEN"
70  INPUT A$
80  WRITE #1, A$
90  IF  A$ = "REZEPTENDE"  THEN 100  ELSE 70
100 WRITE #1, "---> REZEPTENDE <---"
110 Z = Z + 1
120 INPUT "SOLL EIN WEITERES REZEPT EINGEGEBEN WERDEN? (J/N):"; Z$
130 IF  Z$ = "J"  THEN 30
140 PRINT "PROGRAMM BEENDET"
150 CLOSE #1
200 END
```

9.
```
10  OPEN "SCHULE" FOR OUTPUT AS #1
20  DIM Z(8)
30  FOR  J=1 TO 8
40    Z(J) = 0
50  NEXT J
60  CLS
70  INPUT "NAME:            "; N$
80  INPUT "KLASSE:          "; K$
90  INPUT "UNTERRICHTSFACH:"; U$
100 INPUT "FEHLTAGE:        "; F
110 WRITE #1, N$,K$,U$,F, Z(1),Z(2),Z(3),Z(4),Z(5),Z(6),Z(7),Z(8)
120 INPUT "SIND WEITERE DATEN EINZUGEBEN? (J/N):"; Z$
130 IF  Z$ = "J"  THEN 60
140 PRINT "PROGRAMM BEENDET"
150 CLOSE #1
200 END
```

10.
```
10  OPEN "KREDIT" FOR OUTPUT AS #1
20  INPUT "KREDITKARTE NR.: "; K$
30  INPUT "NAME:            "; N$
40  INPUT "ORTSTEIL:        "; T$
50  INPUT "STRASSE:         "; S$
60  INPUT "POSTLEITZAHL     "; P
70  INPUT "ORT:             "; O$
80  WRITE #1, K$,N$,T$,S$,P,O$
90  PRINT "PROGRAMM BEENDET"
100 CLOSE #1
200 END
```

Aufgabengruppe 17

1. "MY","DOG","SAM", 1234<E>

2. MY DOG SAM 1234<E>

3. MY,DOG,SAM, 1234<E>

4. "MY DOG, ",SAM, 1234<E>

5. MY

6. "MY","DOG","SAM", 1234

7. MY DOG SAM 1234 → *Aufg. 2)*
   MY → *Aufg. 3)*
   MY DOG, → *Aufg. 4)*

8. Die gewünschte Ausgabe wird durch das nachfolgende Programm erzielt:

```
10 OPEN ··· FOR INPUT AS =1
20 INPUT =1, A$,B$,C
30 PRINT A$;B$
40 PRINT C
50 CLOSE
60 END
```

Aufgabengruppe 21

1. a)
```
10   'Berechnung der Quadrate von 1 bis 50
20   FOR  J = 1  TO  50
30     PRINT  J,J^2
40   NEXT J
1000 END
```

   b) SAVE "QUADRATE",A

2. a)
```
100  'Berechnung der Kubikzahlen von 1 bis 30
200  FOR  J = 1  TO  30
300    PRINT  J,J^3
400  NEXT J
1000 END
```

Nach dem Eintippen des Programmes ist dieses wegen der Aufgabe 3. in einer Datei sicherzustellen. Hierzu könnte man beispielsweise den Befehl

```
SAVE "KUBIK",A
```

benutzen.

b) `MERGE "QUADRATE"`

c) `LIST`

d) `RUN`

e) `SAVE "KOMBIN",A`

3. `LOAD "KUBIK"`

4. Entweder

`ERASE "QUADRATE.BAS"`	*unter dem Betriebssystem DOS*
oder	
`KILL "QUADRATE.BAS"`	*unter BASIC*

Aufgabengruppe 22

1. 65,97, 66,98, 67,99, 68,100

2. Wenn die ASCII-Verschlüsselung eines Großbuchstabens gleich n ist, dann ist die ASCII-Verschlüsselung des entsprechenden Kleinbuchstabens gleich n+32 .

Aufgabengruppe 23

1.
```
100  DIM B$(10)
110  DATA Gerechtigkeit, Mitte, Probe, Charakter, Kapital
120  DATA Suchen, Ersetzen, Folgen, Passwort, Erweitern
130  'Aufbau des Bereiches B$
140  FOR  J=1 TO 10
150    READ B$(J)
160  NEXT J
170  'Vergleichsschleifen
180  FOR J=1 TO 9
190    FOR K=1 TO 9
200      IF  B$(K+1) < B$(K)  THEN 210  ELSE 250
210      'Vertauschen von B$(K) mit B$(K+1)
220      C$ = B$(K)
230      B$(K) = B$(K+1)
240      B$(K+1) = C$
250    NEXT K
260  NEXT J
270  'Anzeige der Resultate
280  FOR J=1 TO 10
290    PRINT B$(J)
300  NEXT J
1000 END
```

Aufgabengruppe 29

1.
```
10  COLOR 5,1
```

2.
```
10  COLOR 12,0
```

3.
```
10  PSET (200,80), 1
```

4.
```
10  COLOR 3,0
20  PSET (100,100), 2
```

5.
```
10  PSET STEP (-200,-100), 3
```

6.
```
10  PSET STEP (100,0)
```

Aufgabengruppe 30

1.
```
10  LINE (20,50)-(40,100)
```

2.
```
10  LINE -(250,150),2
```

3.
```
10  LINE (125,50)-STEP(100,75),1
```

4.
```
10  LINE (10,20)-(200,150),,B
```

5.
```
10  LINE (10,20)-(200,150),3,BF
```

6. `10 CIRCLE (30,50),20`

7. `10 CIRCLE (30,50),20,, 1.5,3.1`

8. `10 CIRCLE (30,50),20,, -1.5,-3.1`

Aufgabengruppe 35

1. A

2. c

3. Beim vorgelegten Lösungsprogramm wird das Programmende durch die Eingabe des Prozentzeichens angesteuert.

```
10  A$ = INKEY$
20  IF  A$ = "%"  THEN 50
30  IF  A$ <> ""  THEN PRINT A$;
40  GOTO 10
50  END
```

Aufgabengruppe 36

1. `KEY 5, ""`

2. `KEY 1, "LLIST"+CHR$(13)`

3. `10 KEY 5, "CLS"+CHR$(13)+"NEW"+CHR$(13)`

4. Jedes Unterprogramm ist durch eine zusätzliche Zeile einzuleiten und durch eine weitere zusätzliche Zeile abzuschließen.
   a) Die zusätzliche Einleitungszeile kann wie folgt lauten:

      `x001 FOR J=1 TO 4: KEY(J) STOP: NEXT J`

      Unter x ist hier der Reihe nach 1,2,3,4 zu verstehen.
   b) Die zusätzliche Schlußzeile muß jeweils vor derjenigen Zeile eingefügt werden, die das RETURN-Statement enthält. Sie könnte wie folgt codiert werden:

      `x099 FOR J=1 TO 4: KEY(J) ON: NEXT J`

      Unter x ist hier der Reihe nach 1,2,3,4 zu verstehen.

5. KEY(11) ON

Aufgabengruppe 37

1.
```
10   ON ERROR  GOTO 1000
 :
1000 RESUME
 :
```

2. Das Programm kann wie folgt umrissen werden:

```
10    ON ERROR  GOTO 1000
:
1000  IF  ERR = 13  THEN 1010  ELSE END
1010  IF  ERL = 500  THEN 1020  ELSE END
1020  PRINT  "KEINE TYPUEBEREINSTIMMUNG IN ZEILE 500"
            +" (FEHLER)"
1030  RESUME 600
:
```

Anmerkung: Die Fehlerbehandlungsroutine beginnt in diesem Fall auf der Zeile mit der Zeilennummer 1000.

Aufgabengruppe 38

1. `CHAIN MERGE "L"`

2. Das Programm „A" muß mit dem Statement

   `CHAIN "B"`

   enden, das Programm „B" mit dem Statement

   `CHAIN "C"`

Aufgabengruppe 39

1. Zunächst ist der Befehl

   `DATE$=`„*mm/tt/jj*"

   einzugeben. Hierbei bedeuten:

tt	Tag
mm	Monat
jj	Jahr (letzte zwei Ziffern)

   Anschließend ist der Befehl

   `TIME$=`„*hh*:*mm*:*ss*"

   einzugeben. Hierbei bedeuten:

hh	Stunden der augenblicklichen Tageszeit
mm	Minuten der augenblicklichen Tageszeit
ss	Sekunden der augenblicklichen Tageszeit

   Nach jeder Eingabe ist die Eingabetaste zu betätigen.

2. Nach Eingabe des Befehles

   `PRINT TIME$`

   ist die Eingabetaste zu drücken.

3. Die Aufgabenstellung kann durch das folgende Programm befriedigt werden:

```
10  S1$ = RIGHT$(TIME$,2)
20  S$ = RIGHT$(TIME$,2)
30  IF  S$<>S1$  THEN 40  ELSE 20
40  S1$ = S$
50  CLS
60  PRINT DATE$
70  PRINT TIME$
80  GOTO 10
90  END
```

4. Die Aufgabenstellung kann durch das folgende Programm befriedigt werden:

```
10  M1$ = MID$(TIME$,4,2)
20  M$ = MID$(TIME$,4,2)
30  IF  M$<>M1$  THEN 40  ELSE 20
40  M1$ = M$
50  CLS
60  PRINT DATE$
70  PRINT TIME$
80  GOTO 10
90  END
```

Aufgabengruppe 40

1. Die Zeile mit der Zeilennummer 290 des ursprünglichen Programmes (siehe Abb. 11.7) ist abzuändern.

2. Die Zeile mit der Zeilennummer 290 des ursprünglichen Programmes (siehe Abb. 11.7) ist abzuändern.

Aufgabengruppe 41

1. (10,10) – (89,49)

2. (0,0) – (639,15) bei Vorliegen des graphischen Modus hohen Auflösungsvermögens

3. DIM Z%(212)

4. DIM Z%(327)

5. Es ist die Statementfolge zu benutzen, die im Abschnitt 11.3 aufgeführt wurde.

6. Es ist die Statementfolge zu benutzen, die im Abschnitt 11.3 aufgeführt wurde.

7.
```
10  SCREEN 0
20  CLS
30  PRINT CHR$(2);
40  DIM Z(9)
50  GET (0,0)-(7,7),Z
```

8.
```
60  SCREEN 1
70  PUT (0,0),Z
80  PUT (50,50),Z
90  PUT (0,100),Z
...
...
...
```

Anmerkung: Es wird vorausgesetzt, daß die Statements auf den Zeilen mit den Zeilennummern 60 bis 90 im Anschluß an die Statementfolge der Lösung der Aufgabe 6) in einem gemeinsamen Programm ablaufen.

9.
```
10  SCREEN 0
20  CLS
30  PRINT CHR$(2);
40  DIM Z(9)
50  GET (0,0)-(7,7),Z
60  SCREEN 1
70  FOR  J=0 TO 312
80    PUT (J,72),Z
90    PUT (J,72),Z
100 NEXT J
110 END
```

10.
```
10  SCREEN 0
20  CLS
30  PRINT CHR$(2);
40  DIM Z(9)
50  GET (0,0)-(7,7),Z
60  SCREEN 1
70  FOR  J=0 TO 312
80    PUT (J,200*J/320),Z
90    PUT (J,200*J/320),Z
100 NEXT J
110 END
```

Aufgabengruppe 44

```
 1)   3                  2)   2.370000          3)   578000.0
 4)   2.0000000000000000
 5)   3.000000           6)   -4.100000         7)   -4.100000
 8)   3500.684           9)   217.60000000000000
10)   -5940000000000.0000
11)   3.5869504003837265
12)   -2.3454238374621E10
13)   -2367000000000000000000
14)   4.570000E17       15)   46                16)   0.5000000
17)   0.60000000000000000
18)   1.600000         19)   0.66666666666666666
20)   1.1966666666666666
```

```
21)   0.6666666        22)   1.196666        23)   4963.000
24)   1749.9999900000000
25)   46;   0.5;   0.6#;   1.6#;   0.66666666666666666;
      1.1966666666666666;   0.6666666;   1.19666;   4963;
      1749.99999#
26)   333.3108                (genau bis auf die 4. Ziffernstelle)
27)   3.3333333333333330      (Rundungsfehler auf der 17. Ziffern-
                               stelle)
```

Aufgabengruppe 45

1. Durch die folgende Statementfolge ergibt sich die Lösung:

```
10  PRINT (5.87+3.85-12.07)/11.98
20  END
```

2. Durch die folgende Statementfolge ergibt sich die Lösung:

```
10  PRINT (15.1+11.9)^4/12.88
20  END
```

3. Durch die folgende Statementfolge ergibt sich die Lösung:

```
10  PRINT (32485+9826)/(321.5-87.6^2)
20  END
```

4. Im Lösungsprogramm zur Aufgabe 1. ist hinter alle Konstanten das Anhängsel # zu setzen!

5. Im Lösungsprogramm zur Aufgabe 2. ist hinter alle Konstanten das Anhängsel # zu setzen!

6. Im Lösungsprogramm zur Aufgabe 3. ist hinter alle Konstanten das Anhängsel # zu setzen!

7. Das folgende Programm löst die gestellte Aufgabe:

```
10  INPUT X%
20  IF  X%<0  THEN X%=X%-1
30  PRINT X%
40  END
```

8. −5 9. 5 10. −11 11. 1.780000

12. 1.7800000000000000 13. 32.65343 14. 4.252345E21

15. −1.234568E−32 16. 3.2836464930292736

17. −5.7400000000000000

Aufgabengruppe 46

1.
```
10  PRINT EXP(1.54)
20  END
```

2.
```
10  PRINT EXP(-2.376)
20  END
```

3.
```
10  PRINT LOG(58)
20  END
```

4.
```
10  PRINT LOG(9.75E-5)
20  END
```

5.
```
10  PRINT SIN(3.7)
20  END
```

6.
```
10  PRINT COS(0.017453*45)
20  END
```

7.
```
10  PRINT ATN(1)
20  END
```

8.
```
10  PRINT TAN(0.682)
20  END
```

9.
```
10  PRINT 57.29578*ATN(2)
20  END
```

10.
```
10  PRINT LOG(18.9)/LOG(10)
20  END
```

11.
```
10  FOR  X = -5.0  TO 5.0  STEP 0.1
20    PRINT  X, EXP(X)
30  NEXT X
40  END
```

12.
```
10  DATA  1.7, 3.1, 5.9, 7.8, 8.4, 10.1
20  FOR  J = 1  TO  6
30    READ X
40    PRINT  X, 3*X^(1/4)*LOG(5*X)+EXP(-1.8*X)*TAN(X)
50  NEXT J
60  END
```

15.
```
10  INPUT X
20  PRINT "DER GEBROCHENE ANTEIL VON ";X;" BETRAEGT ";
          ABS(X-FIX(X))
30  END
```

Aufgabengruppe 47

1.
```
10  DEF  FNA(X) = X^2 - 5*X
```

2.
```
10  DEF  FNA(X) = 1/X - 3*X
```

3.
```
10  DEF  FNA(X) = 5*EXP((-2*X)
```

4.
```
10  DEF  FNA(X) = X*LOG(X/2)
```

5.
```
10  DEF  FNA(X) = TAN(X)/X
```

6.
```
10  DEF  FNA(X) = COS(2*X) + 1
```

7.
```
10  DEF  FNA$(C$) = RIGHT$(C$,2)
```

8.
```
10  DEF  FNA$(A$,J) = MID$(A$,J,4)
```

9.
```
10  DEF  FNA$(B$) = MID$(B$,INT(LEN(B$)/2)+1,1)
```

10.
```
10  DEF  FNA(X) = 5*EXP(-2*X)
20  FOR  X = 0  TO 10  STEP 0.1
30    PRINT  X, FNA(X)
40  NEXT X
50  END
```

Anhang A

Verzeichnis ausgewählter Fachwörter (englisch-deutsch)

A

American standard code for information interchange	Amerikanischer Standard-Code für Informationsaustausch
Argument	Argument; Aktualparameter
assemble	assemblieren
Assembler	Assemblierer
Assembler Language	Assemblierersprache

B

Background	Hintergrund
Bar chart	Balkendiagramm, Säulendiagramm
Bit	Bit
Blank	Leerzeichen, Zwischenraumzeichen; Blank
Buffer	Puffer
builtin	eingefügt, eingebaut
builtin function	eingefügte Funktion, eingebaute Funktion
Burst	Bündel, Block
Byte	Byte

C

Cartridge	Kassette; Patrone; Register
Cassette	Kassette
Central processing unit	Zentraleinheit
Chain	Kette; Verketten, Verkettung
chain	anhängen
change	Ändern, Änderung
Character	Zeichen
Character set	Zeichenvorrat, Zeichenmenge
Chip	Mikrobaustein, Chip
Column	Spalte
Command	Befehl, Kommando
Communication	Verbindung, Kommunikation; Nachrichtenverbindung; Informationsaustausch
compile	kompilieren
Compiler	Kompilierer
Computer	Computer, Rechner, Rechenanlage
Constant	Konstante
Control	Steuerung; steuern
Control program	Steuerprogramm
Cursor	Positionsanzeiger, Cursor

D

Data	Daten
Data Base	Datenbank
Data integrity	Datensicherheit, Datenintegrität
Data item	Datenelement
Data medium	Datenträger
Data privacy	Datenschutz
Data processing	Datenverarbeitung
Data security	Datensicherheit
Data set	(geordnete) Datenmenge; Datei; Datenbestand
Device	Gerät
Digit	Ziffer
Disk	Platte
Disk operating system	Plattenbetriebssystem
Diskette	Diskette
Diskette operating system	Diskettenbetriebssystem
double precision	doppelte (erhöhte) Genauigkeit

E

edit	edieren
Editor	Editor
Enter	Eingabe
Entry	Eintragung; Eingang
Error	Fehler
Event	Ereignis

F

Feature	Einrichtung; Zusatzeinrichtung; Ausrüstung; Sonderausrüstung
feed	zuführen, Zuführung; Transport; Vorschub
File	Datei
fixed point	Festpunkt, Festkomma
floating point	Gleitpunkt, Gleitkomma, Fließkomma
Flowchart	Datenflußplan
Font	Schriftart
Foreground	Vordergrund
Function	Funktion

G

graphic character	graphisches Zeichen

H

Hardware	Maschinenausrüstung; Geräteausrüstung; (technische) Bestandteile einer Rechenanlage; Hardware
hexadecimal	sedezimal, (hexadezimal)
hexadecimal system	Sedezimalsystem, (Hexadezimalsystem)

I

Index	Verzeichnis, Sachwörterverzeichnis; Index
Input	Eingabe
Instruction	Instruktion
Integer	ganze Zahl; ganzzahlig
Interface	Schnittstelle, Anschluß; Interface
interprete	interpretieren, auslegen
Interpreter	Interpretierer
Item	Element; Größe

K

Key	Taste; Schlüssel
Keyboard	Tastatur

L

Letter	Buchstabe
Library	Bibliothek
Line	Zeile
locate	lokalisieren; Lokalisierung; Lagebestimmung

M

Machine language	Maschinensprache
Machine oriented language	maschinenorientierte Sprache
Main program	Hauptprogramm
Main storage	Hauptspeicher
Medium	Datenträger
Memory	Speicher
Message	Nachricht, Meldung

N

numeric	numerisch

O

Object	Objekt
Object language	Zielsprache
Operating system	Betriebssystem
Output	Ausgabe
Overflow	Überlauf

P

Parameter	Parameter
Parity bit	Prüfbit, Paritätsbit
Password	Paßwort, Schutzwort, Kennwort
Personal Computer	Personalcomputer; Persönlicher Computer; Arbeitsplatzcomputer, Computer am Arbeitsplatz

Plotter	Zeichengerät, Plotter
Precision	Genauigkeit
print	drucken; ausgeben
Printer	Drucker
Priority	Vorrang, Priorität
Privacy	Vertraulichkeit
Problem oriented language	problemorientierte Sprache
Procedure	Prozedur, Verfahren
Processing	Verarbeitung
Program	Programm
Program flowchart	Programmablaufplan
Programming language	Programmiersprache
Prompting	Bedienerführung
Prompting message	Systemanfrage; Systemmeldung

Q

Queue	Warteschlange

R

read	lesen
Reader	Leser
Record	Aufzeichnung; Satz
Request	Anforderung; Anfrage
Restart	Wiederanlauf
run	ausführen, ablaufen

S

Session	Sitzung
Set	Menge
single precision	einfache Genauigkeit
Software	Programmausrüstung; Software
Source	Quelle
Source language	Quellensprache
Special character	Sonderzeichen
Specification	Spezifikation
Statement	Statement; Anweisung
String	Kette
Subprogram	Unterprogramm
Subroutine	Unterroutine, Subroutine
Subset	Untermenge
Substring	Teilkette
System unit	Systemeinheit

T

Teleprocessing	Datenfernverarbeitung
Track	Spur

U

Unit	Einheit
Update	aufdatieren; Aufdatierung; aktualisieren; Aktualisierung

V

Variable	Variable

W

write	schreiben; aufzeichnen

Anhang B

Deutsche und englische Abkürzungen

A

ASCII	American standard code for information interchange
AL	Assembler language
AS	Assemblierersprache

B

BASIC	Beginners all purpose symbolic instruction code
BS	Betriebssystem
Btx	Bildschirmtext

C

COBOL	Common oriented business language
CP	Control program
CPU	Central processing unit

D

DA	Direct access
DB	Data base
DB	Datenbank; Datenbasis
DBS	Datenbeschreibungssprache; Diskettenbetriebssystem
DBVS	Datenbankverwaltungssystem
DFP	Datenflußplan
DFV	Datenfernverarbeitung
DOS	Disk operating system; Diskette operating system
DP	Data processing

E

EA	Eingabe/Ausgabe

F

FORTRAN	Formula translation

H

HAPRO	Hauptprogramm
HP	Hauptprogramm

I

I/O	Input/Output
IBM	International Business Machines

IBM	Internationale Büromaschinen
ISO	International Organization for Standardization

M

Modem	Modulator/Demodulator

O

OS	Operating system

P

PAP	Programmablaufplan
PBS	Plattenbetriebssystem
PC	Personal Computer
PC	Personalcomputer; Persönlicher Computer
Pixel	Picture element
PL/I	Programming language / one
PL/1	Programming language / one
POP	Programmorganisationsplan

R

RAM	Random access memory
ROM	Read only memory

S

SA	Sequential access

T

TP	Teleprocessing

U

UP	Unterprogramm
UPRO	Unterprogramm

Z

ZE	Zentraleinheit

Anhang C

In der Schlußphase der Generierung des Betriebssystemes DOS wird bekanntlich festgelegt, mit welchen Zeichen die einzelnen Tasten der Tastatur belegt werden sollen. Durch diese Auswahlmöglichkeit kann man die Tastaturbelegung den nationalen Erfordernissen anpassen (nationale Versionen des Betriebssystemes DOS). Die Tabelle der druckbaren Zeichen in der Abb. 7.1 unterscheidet sich folglich bei den einzelnen nationalen Versionen. In Abb. 7.1 ist natürlich die deutsche Version wiedergegeben. Durch das nachfolgende, einfache BASIC-Programm kann man sich jederzeit eine entsprechende Tabelle drucken lassen und ist dadurch in der Lage, sich auf die Belange der jeweiligen Betriebssystemversion in der Programmierung einzustellen.

```
10 REM
12 REM   ********************************************************
14 REM   * Tabelle der druckbaren Zeichen (ASCII-Codes 32 bis 127) *
16 REM   * ------------------------------------------------- *
18 REM   *   Programmiert von R. Gritsch              12.3.1985   *
20 REM   ********************************************************
22 REM
24 LPRINT CHR$(12):     LPRINT:     LPRINT
26 LPRINT TAB(6) "ASCII-Code   Zeichen     ASCII-Code   Zeichen";
28 LPRINT TAB(56) "ASCII-Code   Zeichen"
30 LPRINT
32 LPRINT TAB(9) 32, TAB(19) "(Blank)", TAB(34) 64, TAB(47) CHR$(64);
34 LPRINT TAB(59) 96, TAB(72) CHR$(96)
36 FOR  J=33 TO 63
38   LPRINT TAB(9) J, TAB(22) CHR$(J), TAB(34) J+32, TAB(47) CHR$(J+32);
40   LPRINT TAB(59) USING"###";J+64;
42   LPRINT TAB(72) CHR$(J+64)
44 NEXT J
46 LPRINT:     LPRINT
48 LPRINT TAB(4)
          "Anmerkung:   Unter  Blank  ist hier das Leerzeichen zu verstehen"
50 LPRINT TAB(4)
          "---------"
52 LPRINT CHR$(12)
54 END
```

Sachregister

Das Sachregister wurde unter Benutzung von Prozeduren erstellt, die auf dem Datenbankverwaltungssystem dBASE II beruhen.
Bei der Erstellung des Sachregisters blieben die Vorworte, die Aufgabengruppen, die Anhänge und die Lösungen unberücksichtigt.

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

PC professionell
Arbeitsplatzrechner in Ausbildung und Praxis

Herausgeber: Prof. Dr.-Ing. Werner Heinzel, Fulda

In dieser Reihe sind bereits erschienen:

Banahan/Rutter, UNIX – Lernen, verstehen, anwenden

Feuer, Das C-Puzzle Buch

Fuchs, Einführung in BTX-Anwendungen

Gillner, Datenbanken auf Arbeitsplatzrechnern

Heinzel, Arbeitsplatzrechner

Kernighan/Ritchie, Programmieren in C

Meißner, Arbeitsplatzrechner im Verbund

Norton, MS-DOS und PC-DOS

Norton, Die verborgenen Möglichkeiten des IBM PC

Partosch, PASCAL mit Arbeitsplatzrechnern

Pomberger, Lilith und Modula-2

Schirmer, Die Programmiersprache C

Shoup, Numerische Verfahren für Arbeitsplatzrechner

Wolfe/Koelling, BASIC-Programme aus Naturwissenschaft und Technik

Carl Hanser Verlag, 8000 München 86

Fehlermeldungen bei BASIC

Die nachfolgende Tabelle enthält die Fehlermeldungen, die von BASIC[1] auf dem Bildschirm angezeigt werden. In der ersten Spalte sind die Fehlernummern aufgeführt; diese sind gleichzeitig die Werte, die der Variablen ERR nach dem Auftreten des Fehlers zugewiesen werden. In der zweiten Spalte ist die dazugehörige Fehlerkurzbeschreibung genannt.

ERR	Fehlerkurzbeschreibung
1	Next ohne FOR
2	Syntaxfehler
3	RETURN ohne GOSUB
4	Zuwenig Daten
5	Ungültiger Funktionsaufruf
6	Überlauf
7	Nicht ausreichender Hauptspeicher (RAM)
8	Nicht definierte Zeilennummer
9	Index außerhalb des Bereiches
10	Doppelte Definition
11	Division durch Null
12	Ungültiger Direktmodus
13	Keine Typübereinstimmung
14	Platz für Zeichenkette zu Ende
15	Zeichenkette zu lang
16	Formel für Zeichenkette zu komplex
17	Programmausführung kann nicht fortgesetzt werden
18	Nicht definierte Benutzerfunktion
19	Keine Wiederaufnahme (RESUME) des Programmes
20	RESUME ohne Fehler
22	Fehlender Operand
23	Überlauf des Zeilenpuffers
24	Gerätezeitsperre (Ein/Ausgabe-Zeitsperre)
25	Gerätefehler (Einheitenfehler)
26	FOR ohne NEXT
27	Papierende (kein Papier im Drucker)
29	WHILE ohne WEND
30	WEND ohne WHILE
50	Überlauf von FIELD
51	Interner Fehler
52	Falsche Dateinummer
53	Datei nicht gefunden
54	Falscher Dateityp
55	Datei schon eröffnet
57	EA-Fehler bei einem Gerät
58	Datei existiert schon
61	Platte (Diskette) voll
62	Eingabe nach (logischem) Ende
63	Falsche Satznummer
64	Falscher Dateiname
66	Anweisung für die direkte Betriebsart (Direktmodus) in der Datei
67	Zu viele Dateien
68	Gerät nicht verfügbar
69	Überlauf des Verbindungspuffers (Datenfernverarbeitungspuffers)
70	Schreibschutz bei Platte (Diskette)
71	Platte (Diskette) nicht betriebsbereit
72	Fehler beim Plattenmedium (Diskettenfehler)
73	(Benutzung einer) Einrichtung für das erweiterte BASIC
74	Umbenennung über Platten (Disketten) hinweg
75	Fehler beim Zugriff zur Datei/Pfad
76	Pfad nicht gefunden
—	Nicht druckbarer Fehler
~	Inkompatible DOS-Version

[1] Tabelle gilt für BASIC 2.00

DOS 2.00: Überblick über die Befehle

Befehlswort	Erklärung	Format (Syntax)
ASSIGN	Neuer Leitweg für Diskettenbefehle	ASSIGN [#]lfn=neulfn[, . . .]
BACKUP	Erzeugung einer Auswichdatei von einer Datei auf Hardplatte	BACKUP [d:] [pfad] [ds] [d:] [/S] [/M] [/A] [D: mm-tt-jj]
BREAK	Aktivierung der Prüfung von Ctrl-Break	BREAK [ON|OFF]
CHDIR oder CD	Änderung des Verzeichnisses	CHDIR [[d:]pfad]
CHKDSK	Prüfen Plattenzustand (Dateiformatprüfung)	CHKDSK [d:] [ds] [/F] [/V]
CLS	Löschen Bildschirm	CLS
COMP	Überprüfen von Dateien auf Gleichheit (Datenvergleich)	COMP [d:] [pfad] [ds] [d:] [pfad] [ds]
COPY	Kopieren von Dateien	COPY [/A] [/B] [d:] [pfad] [ds] [d:] [pfad] [ds] [/A] [/B] [/V]
DATE	Eingabe des Datums (u-eingeschener Form)	DATE [mm-tt-jj]
DIR	Anzeige des (Inhalts-)verzeichnisses	DIR [d:] [ds] [/P] [/W]
DISKCOMP	Vergleichen zweier Disketten auf Gleichheit (Diskettenvergleich)	DISKCOMP [d:] [d:] [/1] [/8]
DISKCOPY	Kopieren einer Diskette auf eine andere (Diskettenkopieren)	DISKCOPY [d:] [d:] [/1]
ERASE oder DEL	Löschen einer Datei (Dateilöschung)	ERASE [d:] [pfad] [ds]
FORMAT	Formatieren einer Diskette	FORMAT [d:] [/S] [/1] [/8] [/V] [/B]
(GRAPHICS)	Drucken des Bildschirminhaltes im graphischen Modus	Drücken der Tasten ↑ und PrtSc
MODE	Festlegung der Betriebsarten: a) für Drucker b) für Bildschirm c) für den Verbindungsadapter Umleiten der Ausgabe vom Drucker zum Adapter	MODE LPT # : [n] [,[m][,P]] MODE [n],[m],T] MODE COMn:baud [,par [,db [,sb [,P]]]] MODE LPT # : = COMn
PATH	Definieren von Suchpfaden für das (Inhalts-)verzeichnis	PATH [d:]pfad[[;d:]pfad] . . .]

PC BASIC – Nachschlageübersicht

Zeichenerklärungen: 1. [e] Das in eckigen Klammern stehende Element e ist ein optionales Element, d.h. es kann, aber muß nicht verwendet werden.

2. . . . Drei Punkte zeigen an, daß das vorhergehende Element beliebig oft wiederholt werden darf.

1. BASIC-Befehle (Direktmodus)

Schlüsselwort	Erklärung	Format (Syntax)
AUTO	Automatische Erzeugung von Zeilennummern	AUTO [,] [nummer] [, [schrittweite]]
CHDIR	Änderung von (Inhalts-)Verzeichnisses	CHDIR [[d:]pfad]
CONT	Fortsetzung eines durch eine Unterbrechung unterbrochenen Programmes	CONT
DELETE	Löschung von bestimmten Programmzeilen	DELETE [,] [zeile 1] [-zeile 2]
EDIT	Aufbereiten (Editieren): Anzeige einer Zeile	EDIT [,] [zeile]
FILES	Anzeige der Namen derjenigen Dateien, die die Dateispezifikation dis besitzen; bei Fehlen von ds werden die Namen aller Dateien auf aktuellen Diskettenkette angezeigt	FILES [ds]
KILL	Löschung der Dateien) mit der gegebenen Dateispezifikation ds	KILL [ds]
LIST	Auflistung von Programmzeilen auf dem Bildschirm	LIST [zeile 1] [-[zeile 2]] [,ds]
LLIST	Auflistung von Programmzeilen über den Drucker	LLIST [zeile 1] [-[zeile 2]]
LOAD	Laden eines Programmes von Diskette oder Kassette	LOAD ds [,R]
MERGE	Mischen einer Programmdatei mit dem laufenden Programm	MERGE ds
MKDIR	Erzeugung eines (Inhalts-)verzeichnisses	MKDIR ds
NAME	Umbenennung der Datei mit der Dateispezifikation ds	NAME ds AS dateiname
NEW	Löschen des laufenden Programmes im RAM (Löschen des Hauptspeichers)	NEW
RENUM	Neunummerierung von Programmzeilen	RENUM [neuenummer] [,[altenummer] [,schrittweite]]
RESET	Abschließen aller Dateien und Löschen des Systempuffers	RESET
RMDIR	Zurücklübertragung eines (Inhalts-)Verzeichnisses	RMDIR pfad
SAVE	Sicherstellung des laufenden Programmes	SAVE ds[,A][,P]
SYSTEM	Rückkehr zum Betriebssystem DOS	SYSTEM
TRON	Anzeige der Zeilennummern der ausgeführten Zeilen (Programmablaufverfolgung ein - Trace on)	TRON
TROFF	Umwirksammachung der Programmablaufverfolgung (Trace off)	TROFF

2. Programmsteuerung

END	Beendigung der Ausführung des laufenden Programmes	END
FOR . . . NEXT	Wiederholung der Statements zwischen FOR und NEXT solangeg, wie der Wert der Laufvariablen lv kleiner als oder gleich y ist. Die Werte der Laufvariablen beginnen bei x und werden bei jeder Wiederholung um z erhöht.	FOR lv=x TO y [STEP z] . . . NEXT [lv] [,lv] . . .
GOSUB . . . RETURN	Verzweigung der Ausführung zur Zeile zeile. Bei Erreichen der Anweisung RETURN: Rückspringung zur Zeile, die auf die GOSUB enthaltende Zeile folgt.	GOSUB zeile . . . RETURN
GOTO	Übertragung der Programmablaufsteuerung zur gegebenen Zeile zeile	GOTO zeile
IF . . . THEN . . . ELSE	Wenn der Ausdruck ausdr den Wert „wahr" aufweist, dann wird die Klausel kla1 ausgeführt (oder es erfolgt eine Verzweigung zur Zeile zeile), andernfalls wird die Klausel kla2 ausgeführt.	IF ausdr THEN kla1 [ELSE kla2] IF ausdr THEN GOTO zeile [ELSE kla2]
ON . . . GOSUB	Der Wert des Ausdruckes ausdr wird in einen ganze Zahl n umgewandelt; danach erfolgt die Übertragung der Programmablaufsteuerung an daseinge Unterprogramm, das auf der Zeile begonnen, deren Zeilennummer znr an der n-ten Stelle in der Liste der Zeilennummern steht.	ON ausdr GOTO znr [,znr] . . .
ON . . . GOTO	Der Wert des Ausdruckes ausdr wird in eine ganze Zahl n umgewandelt; danach erfolgt die Übertragung der Programmablaufsteuerung zur Zeile mit der Zeilennummer znr , die an der n-ten Stelle in der Liste der Zeilennummern steht.	ON ausdr GOTO znr [,znr] . . .
STOP	Stoppen der Programmausführung	STOP
SYSTEM	Verlassen von BASIC und Rückkehr zum Betriebssystem DOS	SYSTEM
WHILE . . . WEND	Ausführung der Statements zwischen WHILE und WEND solange, bis der Ausdruck ausdr den Wert „falsch" aufweist	WHILE ausdr . . . WEND

5. Steuerung der Tastatur und des Bildschirmes (Textmodus)

CLS	Löschen des Bildschirmes	CLS
COLOR	Setzen der Farben für den Vordergrund, für den Hintergrund und für den Bildschirmrand	COLOR [vordergrund] [,[hintergrund] [,rand]]
CSRLIN	Übergabe der Nummer der Zeile, auf der der Positionsanzeiger (Cursor) augenblicklich steht	x= CSRLIN
INKEY$	Rückgabe des über die Tastatur eingegebenen Zeichens	x$ = INKEY$
INPUT	Lesen der über die Tastatur eingegebenen Zeichen	INPUT [;] [„anfrage";] variable [,variable] . . .
INPUT$	Rückgabe einer Zeichenkette von n Zeichen, die über die Tastatur eingelesen wurden	x$ = INPUT$(n)
KEY	Zuweisung von Funktionen zu den Funktionstasten bzw. Anzeige der den Funktionstasten zugewiesenen Funktionen	KEY n,x$ KEY LIST KEY ON KEY OFF
KEY(n)	Wirksammachen bzw. Unwirksammachen der Unterbrechung für die angegebene Taste (Funktionstasten bzw. Tasten für die Cursorbewegung)	KEY(n) ON KEY(n) OFF KEY(n) STOP
LINE INPUT	Eingabe einer gesamten Textzeile und Zuweisung derselben bei der Zeichenkettenvariablen z$	LINE INPUT [;] [„anfrage";] x$
LOCATE	Setzen des Positionsanzeigers (Cursors) auf die Stelle in der Spalte x und der Zeile y	LOCATE [zeile] [,[spalte] [,[cursor] [,[start] [,stopp]]]]
ON KEY	Festlegung der Zeile, an der verzweigt werden soll, falls die angegebene Funktionstaste bzw. Cursorbewegungstaste betätigt wird und sowmit eine Unterbrechung hervorruft → Aktivierung durch KEY(n) ON	ON KEY(n) GOSUB zeile
POS	Rückgabe der horizontalen Position des Cursors (n ist dabei Scheinargument, d.h. ohne Bedeutung)	POS(n)
PRINT	Anzeige von Daten auf dem Bildschirm bei Angabe einer Liste von Ausdrücken	PRINT [liste]
PRINT USING	Ausgabe von formatierten (numerischen) Daten bei Angabe einer Liste von Ausdrücken	PRINT # dateinummer [USING x$;] liste
SCREEN	Rückgabe des ASCII-Codes, wenn z „falsch" (gleich 0) ist, oder der Farbe, wenn z „wahr" ist, des Zeichens auf der Stelle mit den Koordinaten (zeile, spalte)	SCREEN (zeile, spalte [,z])
(PRINT) SPC	Einfügung von n Leerzeichen vor dem Drucken	PRINT SPC(n)
(PRINT) TAB	Übergang zur Spalte n vor dem Drucken	PRINT TAB(n)
WIDTH	Festlegung der Zeilenbreite breite auf dem Bildschirm	WIDTH breite
WRITE	Anzeige von Daten auf dem Bildschirm (mit Kommas zwischen den Datenelementen und einschließenden Anführungszeichen bei Zeichenketten). Unter liste ist eine Liste von Ausdrücken zu verstehen.	WRITE liste

4. Handhabung (Manipulation) von Dateien

CLOSE	Abschließen von Dateien	CLOSE [[#] dateinummer] [,[#] dateinummer] . . .]
CVD bzw. CVI bzw. CVS	Umwandlung einer Zeichenkette z$ in eine ganze Zahl (I), in eine Zahl einfacher Genauigkeit (S) bzw. in eine Zahl doppelter Genauigkeit (D)	x= CVD(z$) x= CVI (z$) x= CVS(z$)
EOF	Rückgabe des Wertes „wahr" (–1), wenn das Dateiende vorliegt, andernfalls des Wertes „falsch" (0)	EOF(dateinummer)
GET	Übertragen von Daten zum Puffer zu einer Datei mit wahlfreiem (direkten) Zugriff in einen Puffer	GET [#] dateinummer [,satznummer]
INPUT #	Lesen von Datenelementen von einem sequentiellen Eingabegerät bzw. von einer sequentiellen organisierten Datei	INPUT # dateinummer, variable[,variable] . . .
LINE INPUT #	Lesen einer gesamten Datenzeile von einer Datei in eine Zeichenkettenvariable zk$	LINE INPUT # dateinummer, zk$
LSET	Übertragen von Daten zum Puffer zu einer Datei mit wahlfreiem (direkten) Zugriff; die Übertragung erfolgt linksbündig	LSET zk$ = x$
MKD$ bzw. MKI$ bzw. MKS$	Umwandlung eines numerischen Wertes x in eine ganze Zahl (INT), in eine Zahl einfacher Genauigkeit (SNG) bzw. in eine Zahl doppelter Genauigkeit (DBL)	zk$ = MKI$(ganzezahl) zk$ = MKS$(zahleinfgen) zk$ = MKD$(zahldoppgen)
OPEN	Eröffnen einer Datei	OPEN ds [FOR modus] AS # dateinummer [LEN = satzlaenge]
PRINT #	Schreiben von Daten in eine sequentielle (sequentiell organisierte) Datei aufgrund einer Liste von Ausdrücken	PRINT # dateinummer, liste
PRINT # USING	Schreiben von formatierten Daten in eine sequentielle (sequentiell organisierte) Datei aufgrund einer Liste von Ausdrücken	PRINT # dateinummer, [USING x$;] liste
PUT	Eintragen eines Satzes in einen Puffer zum Schreiben desselben in eine Datei mit wahlfreiem (direkten) Zugriff	PUT [#] dateinummer [,satznummer]
RSET	Übertragen von Daten zum Puffer zk$ einer Datei mit wahlfreiem (direkten) Zugriff; die Übertragung erfolgt rechtsbündig	RSET zk$ = x$
WRITE #	Schreiben von Daten in eine sequentielle Datei aufgrund einer Liste von Ausdrücken.(mit Kommas zwischen den Datenelementen und einschließenden Anführungszeichen bei Zeichenketten)	WRITE # dateinummer,liste

5. Handhabung (Manipulation) von Variablen und Konstanten

CDBL bzw. CINT bzw. CSNG	Umwandlung (Konvertierung) einer Zahl x in eine ganze Zahl (INT), in eine Zahl einfacher Genauigkeit (SNG) bzw. in eine Zahl doppelter Genauigkeit (DBL)	CDBL(x) CINT(x) CSNG(x)
DATA	Identifizierung der Daten, die durch READ-Statements gelesen werden sollen	DATA konstante [,konstante] . . .
DEFDBL bzw. DEFINT bzw. DEFSNG bzw. DEFSTR	Definieren des Typs typ von Variablen: a) ganzzahlige Variablen (INT) b) Variablen einfacher Genauigkeit (SNG) c) Variablen doppelter Genauigkeit (DBL) d) Zeichenkettenvariablen (STR)	DEFtyp buchstabe[-buchstabe] [buchstabe[-buchstabe]] . . .
LET	Zuweisung eines Wertes zu einer Variablen	[LET] variable = ausdruck
READ	Zuweisung von Werten zu Variablen unter Benutzung von Datenelementen in DATA-Statements	READ variable[,variable] . . .
SWAP	Austausch der den Variablen v1 und v2 zugewiesenen Werte	SWAP v1,v2

6. Bereiche

CLEAR	a) Setzen aller numerischen Variablen auf 0 b) Setzen aller Zeichenkettenvariablen auf die Nullkette c) Definieren des Umfanges des Speicherplatzes speicher für den Benutzer d) Definieren des Stapelbereiches stapel	CLEAR [,[speicher] [,stapel]]
DIM	Definieren von Umfängen für die Bereiche bereich	DIM bereich(dimensionen) [,bereich(dimensionen)] . . .
ERASE	Eliminieren von Bereichen aus einem Programm	ERASE bereich[,bereich] . . .
OPTION BASE	Definieren des ersten Indexwertes n für Bereiche (0 oder 1)	OPTION BASE n

7. Graphik und Tonerzeugung

BEEP	Aktivieren des Tonerzeugers	BEEP
CIRCLE	Zeichnen eines Kreises, einer Ellipse oder eines Sektors mit dem Mittelpunkt (xmittelp, ymittelp)	CIRCLE (xmittelp, ymittelp), radius[,farbe [,anfang, ende [,ansicht]]]
COLOR	Festlegung der laufenden Hintergrundfarbe und der Palette	COLOR [hintergrund] [,[palette]]
DRAW	Zeichnen einer Figur, die durch eine Zeichenkette zk beschrieben ist	DRAW zk
GET	Speicherung der Farbe eines gegebenen Rechteckes in einem Bereich	GET (x1,y1)–(x2,y2), bereichname
LINE	Zeichnen einer geraden Linie oder eines Rechtecks	LINE [(x1, y1)]–(x2,y2), [,[farbe] [,[B[F]] [,stil]]
ON PLAY	Ermöglicht tonfaszende Musik während des Programmablaufes	ON PLAY(note) GOSUB zeile
PAINT	Ausmalen einer Bildschirmregion	PAINT (x,y)[,farbe][,grenze][,hintergrund]]
PLAY	Spielen von Musik entsprechend des in der Zeichenkette zk beschriebenen Notensatzes	PLAY zk
PMAP	Umwandlung eines Ausdruckes in physikalische Koordinaten	PMAP (ausdruck,n)

POINT	Rückgabe der Farbe (0 bis 3) des Pixels (x,y)	**POINT (x,y)**
PSET	Zuweisung einer Farbe zum Pixel (x,y)	**PSET (x,y)[,farbe]**
PRSET	Setzen des Pixels (x,y) auf die Hintergrundfarbe (Farbe 0 bei Fehlen der Farbangabe)	**PRSET (x,y)[,farbe]**
PUT	Anzeige des im Bereich bereich gespeicherten Abbilds an der Stelle (x,y)	**PUT (x,y),bereich[,aktion]**
SCREEN	Setzen der Bildschirmattribute für die nachfolgenden Anweisungen	**SCREEN [modus][,[aneiloas] [,[aseite] [,vseite]]]**
SOUND	Erzeugung von Tönen über den Lautsprecher	**SOUND frequenz,dauer**
VIEW	Definieren einer Gesichtshaltung innerhalb des Bildschirmes	**VIEW [[SCREEN] [(x1,y1)–(x2,y2) [,[farbe] [,[grenze]]]]]**
WINDOW	Neudefinition von Koordinaten auf dem Bildschirm	**WINDOW [SCREEN] (x1,y1)–(x2,y2)**

8. Verbindungen (Kommunikationen)

CLOSE	Abschließen einer Kommunikationsdatei	**CLOSE [#]dateinummer**
COM	Aktivieren bzw. Inaktivieren des Auffangens der Eingaben vom Verbindungsadapter n	**COM(n) ON** **COM(n) OFF** **COM(n) STOP**
ON COM	Definieren der Auffangzeile, wenn Eingaben vom Verbindungsadapter n vorliegen	**ON COM(n) GOSUB zeile**
OPEN	Eröffnen der mit dem Verbindungsadapter n verknüpften Verbindungsdatei unter der Dateinummer nr ; dabei können gegeben sein: a) Übertragungsgeschwindigkeit geschw in Bound b) Parität par c) Datenbits für Absendern/Empfangen d) Stoppbits für Absendern/Empfangen Vor nr kann das Zeichen # codiert werden	**OPEN „COMn: [geschw] [,par] [,datenbits] [,stoppbits]" AS nr**

9. Arithmetische Funktionen

ABS	Als Funktionswert ergibt sich der absolute Wert des Arguments x	**ABS(x)**
ATN	Als Funktionswert ergibt sich der Arkustangens von x (arctan)	**ATN(x)**
COS	Als Funktionswert ergibt sich der Kosinus von x	**COS(x)**
DEF FN . . .	Definition der Benutzerfunktion FNname mit der Parameterliste liste in Form eines von den Parametern abhängigen arithmetischen Ausdrucks ausdruck	**DEF FNname[liste] = ausdruck**
EXP	Als Funktionswert ergibt sich exp(x) — Exponentialfunktion	**EXP(x)**
FIX	Als Funktionswert ergibt sich der ganzzahlige Anteil von x	**FIX(x)**
INT	Als Funktionswert ergibt sich die größte ganze Zahl, die kleiner als oder gleich x ist	**INT(x)**
RANDOMIZE	Als Funktionswert ergibt sich eine freigewählte Zahl, die als Ausgangszahl oder Anfangswert für den Zufallszahlengenerator dient	**RANDOMIZE [n]** **RANDOMIZE timer**
RND	Als Funktionswert ergibt sich eine Zufallszahl	**RND[(x)]**
SGN	Als Funktionswert ergibt sich das Vorzeichen von x	**SGN(x)**
SIN	Als Funktionswert ergibt sich der Sinus von x	**SIN(x)**
SQR	Als Funktionswert ergibt sich die Quadratwurzel von x	**SQR(x)**
TAN	Als Funktionswert ergibt sich der Tangens von x	**TAN(x)**
TIMER	Als Funktionswert ergibt sich die Anzahl der Sekunden, die seit Mitternacht oder seit dem Neustart des Systems vergangen sind	**TIMER**
VAL	Als Funktionswert ergibt sich der numerische Wert, der in der Zeichenkette x$ enthalten ist (nach Entfernung von Leerzeichen, Tabulatorzeichen, Zeilenvorschubzeichen und anderen nichtnumerischen Zeichen)	**VAL(x$)**

10. Zeichenkettenfunktionen

ASC	Als Funktionswert ergibt sich der ASCII-Code des ersten Zeichens der Zeichenkettenvariablen x$	**ASC(x$)**
CHR$	Als Funktionswert ergibt sich das Zeichen, dessen ASCII-Code gleich n ist	**CHR$(n)**
DATE$	Als Funktionswert ergibt sich das augenblickliche Datum in der (englischen) Form mm-tt-jjjj. Hierbei bedeuten: tt	Tag mm	Monat jjjj	Jahr	**DATE$**
INSTR	Als Funktionswert ergibt sich die Position, auf der zum ersten Mal die Zeichenkette y$ innerhalb der Zeichenkette x$ erscheint, wobei die Untersuchung ab der Stelle n beginnt	**INSTR(x$,y$[,n])**
LEFT$	Als Funktionswert ergibt sich die Zeichenkette, die aus den n linksbündigen Zeichen der Zeichenkette x$ besteht	**LEFT$(x$,n)**
LEN	Als Funktionswert ergibt sich die Länge der Zeichenkette x$	**LEN(x$)**
HEX$	Als Funktionswert ergibt sich die Zeichenkette, die gleich dem sedezimalen (hexadezimalen) Wert von n ist	**HEX$(n)**
MID$	Als Funktionswert ergibt sich die Zeichenkette, die aus den an der Stelle n beginnenden m Zeichen von x$ besteht	**MID$(x$,n[,m])**
OCT$	Als Funktionswert ergibt sich die Zeichenkette, die gleich dem oktalen Wert von n ist	**OCT$(n)**
RIGHT$	Als Funktionswert ergibt sich die Zeichenkette, die aus den n rechtsbündigen Zeichen der Zeichenkette x$ besteht	**RIGHT$(x$,n)**
SPACE$	Als Funktionswert ergibt sich die Zeichenkette, die aus n Leerzeichen besteht	**SPACE$(n)**
STR$	Als Funktionswert ergibt sich die Zeichenkette, die durch Umwandlung der Zahl x entsteht	**STR$(x)**
STRING$	Als Funktionswert ergibt sich die Zeichenkette, die aus den folgenden Zeichen besteht a) n-malige Wiederholung des Zeichens mit dem ASCII-Code m (1. Form) b) n-malige Wiederholung des ersten Zeichens der Zeichenkette x$ (2. Form)	**STRING$(n,m)** **STRING$(n,x$)**
TIME$	Als Funktionswert ergibt sich die augenblickliche Tageszeit in der Form hh:mm:ss Hierbei bedeuten: hh	Stunden mm	Minuten ss	Sekunden	**TIME$**

11. Externe Geräte (Einheiten)

INP	Rückgabe des von der Anschlußstelle n gelesenen Bytes	**INP(n)**
ON PEN	Definition einer Zeile, zu der bei einer Eingabe mit Lichtstift verzweigt werden soll	**ON PEN GOSUB zeile**
ON STRIG	Definition einer Zeile, zu der bei Bedienung eines Spielpulttastens (n=0, n=2, n=4 oder n=6) verzweigt werden soll	**ON STRIG(n) GOSUB zeile**
OUT	Übertragung eines Bytes zur Anschlußstelle n	**OUT(n)**
PEN	Aktivieren bzw. Inaktivieren der Lesefunktion des Lichtstiftes bzw. von der vom Lichtstift ausgehenden Unterbrechung	**PEN ON** **PEN OFF** **PEN STOP**
PEN(n)	Lesen der Koordinaten des Lichtstiftes unter Beeinflussung durch den numerischen Ausdruck n (im Wertebereich von 0 bis 9)	**PEN(n)**
STICK	Rückgabe der x-Koordinate oder der y-Koordinate eines Spielpultes; unter n ist ein numerischer Ausdruck im Wertebereich von 0 bis 3 zu verstehen	**STICK(n)**
STRIG	Aktivieren bzw. Inaktivieren der Eingabefunktion der Spielpulttaste bzw. der von ihnen ausgehenden Unterbrechung	**STRIG ON** **STRIG OFF**
STRIG(n)	Rückgabe des Zustandes (Status) eines Spielpulttastes	**STRIG(n)**
WAIT	Warten auf Eingabe von einer Anschlußstelle	**WAIT anschluss,n[,m]**

12. Auffangen von Fehlern

ERR	Rückgabe des Fehlercodes des zuletzt aufgetretenen Fehlers	**ERR**
ERL	Rückgabe der Nummer der Programmzeile, die den zuletzt aufgetretenen Fehler verursachte	**ERL**
ERROR	Simulation des Auftretens des Fehlers n	**ERROR n**
ON ERROR	Übertragung der Programmablaufsteuerung zur Zeile mit der Zeilennummer znr bei Auftreten eines Fehlers	**ON ERROR GOTO znr**
RESUME	Löschen der Fehlerbedingung und Übertragung der Programmablaufsteuerung entweder zur Zeile nach der den Fehler enthaltenden Zeile (NEXT) oder zur Fehlerzeile selbst (0) oder zur Zeile mit der Zeilennummer znr	**RESUME [0]** **RESUME NEXT** **RESUME znr**

13. Schnittstelle zur Maschinensprache

BLOAD	Laden einer Binärdatei von der Diskette in den Hauptspeicher (RAM), beginnend bei der angegebenen relativen Stelle (Offset)	**BLOAD ds[,offset]**
BSAVE	Sicherstellen des bei der relativen Stelle (Offset) beginnenden Hauptspeicherinhalts in einer Länge von laenge Bytes	**BSAVE ds,offset,laenge**
CALL	Aufruf eines in der Maschinensprache vorliegenden Unterprogrammes, dessen Startadresse im Hauptspeicher der numerischen Variablen nvar zugewiesen ist, mit den Argumenten arg	**CALL nvar[(arg[,arg] . . .)]**
DEF SEG	Definieren des laufenden Segmentes im Hauptspeicher (RAM)	**DEF SEG [= adresse]**
DEF USR	Definieren der Anfangsadresse eines Unterprogrammes in der Maschinensprache	**DEF USR[ziffer] = offset**
PEEK	Rückgabe des Inhalts der angegebenen Hauptspeicherstelle	**PEEK(adresse)**
POKE	Speicherung des Wertes w in die angegebene Hauptspeicherstelle	**POKE adresse,w**
USR	Aufruf des vom Benutzer definierten Unterprogrammes in der Maschinensprache mit dem Argument x	**USR[ziffer](x)**
VARPTR	Rückgabe der relativen Adresse (Offset) der angegebenen Variablen bzw. des angegebenen Dateisteuerblockes der Datei mit der Dateinummer nr	**VARPTR(variable)** **VARPTR(# nr)**

14. Drucker

LLIST	Auflisten von Programmzeilen über den Drucker	**LLIST [zeile 1] [-[zeile 2]]**
LPRINT	Drucken von Konstanten und der Werte von Variablen bei Angabe einer Liste von Ausdrücken	**LPRINT [liste]**
LPRINT USING	Drucken von formatierten Daten aufgrund einer Liste von Ausdrücken	**LPRINT USING X$ liste[;]**

15. Verschiedene sonstige Statements und Funktionen

CHAIN	Übergabe der Programmablaufsteuerung an ein angehängtes Programm (unter ber ist ein Zeilenbereich zu verstehen)	**CHAIN [MERGE] ds[,[zeile] [,[ALL] [,DELETE ber]]]**
COMMON	Spezifizierung von Variablen, die an ein angehängtes Programm weitergegeben werden sollen	**COMMON variable[,variable] . . .**
MOTOR	Ein- bzw. Ausschalten des Kassettenmotors (abhängig vom Status)	**MOTOR [status]**
ON TIMER	Übertragung der Programmablaufsteuerung zur Zeile mit der Zeilennummer znr nach Ablauf einer gegebenen Zeitperiode	**ON TIMER GOTO znr**
REM	Bemerkung, daher ohne Bedeutung für die Programmausführung	**REM bemerkung** **'bemerkung**
{PRINT}	Hineinstellen von Dateien in eine Warteschlange zum Drucken	**PRINT [[d:][dad]/[T]]/[C]]/P] . . .]**
RECOVER	Wiederherstellung gelöschter Dateien	**RECOVER [d:] [pfad] [ds]**
REM	Anzeige von Bemerkungen in einer Stapeldatei	**REM [bemerkungen]**
RENAME oder REN	Umbenennen einer Datei	**RENAME [d:] [pfad] ds neuebezeichnung**
RESTORE	Zurückspeichern einer Datei von einer Sicherungskopie	**RESTORE d: [d:] [pfad] [ds] [/S] [/P]**
RMDIR oder RD	Löschen eines (Inhalts-)verzeichnisses	**RMDIR [d:]pfad**
SYS	Kopieren von DOS auf eine Diskette	**SYS d:**
TIME	Eingabe der augenblicklichen Zeit	**TIME [hh:mm:ss.xx]**
TREE	Anzeige der augenblicklichen Baumstruktur eines (Inhalts-)verzeichnisses	**TREE [d:] [/F]**
TYPE	Anzeige des augenblicklichen Inhalts einer Datei	**TYPE [d:] [pfad]ds**
VER	Anzeige der Versionsnummer des DOS	**VER**
VERIFY	Einrichten oder Außerkraftsetzen der Nachprüfung	**VERIFY [ON|OFF]**
VOL	Anzeige des Kennsatzes des Datenträgers	**VOL [d:]**

Bemerkungen In der letzten Spalte sind folgende Abkürzungen verwendet worden:

diffw	jedes Laufwerk
d:	Laufwerksbezeichnung
db	Datenbits
dn	Dateiname
ds	Dateispezifikation
#	Nummer eines Gerätes
hh:mm:ss.xx	Stunden:Minuten:Sekunden:Sekundenbruchteile
j	ja
mm	Monat
neuffw	neues Laufwerk
par	Parität
sb	Stoppbits
tt	Tag
|	„oder"

Beilage zu **Goldsteins IBM PC-Buch**

Mit diesem Buch liegt die deutsche Ausgabe des berühmten „Goldstein" vor, das die amerikanischen Computer-Zeitschriften als besonderes Ereignis feiern und das im englischen Sprachraum ein Bestseller ersten Ranges ist. Wer seinen IBM PC richtig und intensiv nutzen will, braucht dieses Buch mit einer Fülle von Anwendungsbeispielen.

Das Goldstein-Buch bietet:

- eine klare und prägnante Darstellung dessen, was ein IBM PC ist und wie er arbeitet
- eine gründliche Einführung in BASIC mit vielen nützlichen Ratschlägen und Tips für das Programmieren
- viele Anwendungsmöglichkeiten für Beruf, Grafikerzeugung, Textverarbeitung und Spiele
- übersichtliche Tabellen, Schaubilder, Anhänge

ISBN 3-446-13956-7