BASIC

BASIC

Hopfer · Müller

Einführung in das Programmieren

BASIC-Anweisungen

DATA	Datenlistendefinition
DEF	Funktionsdefinition
DIM	Dimensionierung von Feldern
END	Endeanweisung
FOR-NEXT	Laufanweisung
GOSUB	Unterprogrammaufruf
GOTO	Sprunganweisung
IF-THEN	Verzweigungsanweisung
INPUT	Dateneingabe
LET	Wertzuweisung
ON-GOSUB	Unterprogrammaufruf mit Auswahl
ON-GOTO	Verzweigungsanweisung mit Auswahl
OPTION	Vereinbarung der unteren Feldgrenzen
PRINT	Datenausgabe
RANDOMIZE	Startpunktvorgabe für Zufallsfunktion
READ	Datenliste lesen
REM	Kommentaranweisung
RESTORE	Datenzeiger rücksetzen
RETURN	Unterprogrammrücksprung
STOP	Haltanweisung

Vorwort

BASIC ist eine Programmiersprache, die international weite Verbreitung gefunden hat. Sie wird für die Programmierung von Klein- und Kleinstrechnern am häufigsten eingesetzt. Dies läßt sich vor allem darauf zurückführen, daß BASIC infolge seines einfachen Aufbaus sehr leicht erlernt werden kann, dennoch sehr leistungsfähig ist und sich zur Lösung von Problemstellungen aus den verschiedensten Gebieten, wie z. B. Wissenschaft, Technik, Wirtschaft und Unterhaltung, eignet. Die Programmiersprache BASIC bietet deshalb auch für den Nichtinformatiker einen einfachen Zugang zum Programmieren und ermöglicht ihm damit die effektive Nutzung leistungsfähiger Computer bei der Lösung von Aufgaben seines Fach- oder Interessengebietes. Heute stehen ihm dazu neben großen und mittleren Rechenanlagen vor allem Klein- und Kleinstrechner, wie Bürocomputer, programmierbare Tischrechner, Personalcomputer und Heimcomputer, zur Verfügung.

Die Sprache BASIC (**B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode – symbolische Allzwecksprache für Anfänger) wurde ursprünglich (1963/64) für Anfänger auf dem Gebiet der Programmierung entwickelt. In der Folgezeit fand sie das Interesse verschiedener Computerhersteller, die BASIC auf ihren Anlagen für die Arbeit im Dialogbetrieb einführten und mit unterschiedlichen Zielvorstellungen modifizierten. Diese Entwicklung führte dazu, daß der Name "BASIC" heute genau genommen keine einheitliche Sprache, sondern eine Vielzahl von "Sprachdialekten" repräsentiert. Um eine Vereinheitlichung zu erreichen, wurde 1984 (Entwurf 1979) unter der Bezeichnung "Minimal-BASIC" ein Sprachumfang international standardisiert, der die wichtigsten Anweisungen und Datentypen gebräuchlicher BASIC-Versionen umfaßt (International Standard ISO 6373).

Das vorliegende Buch führt in das praktische Programmieren mit BASIC ein.

Zunächst werden grundlegende Kenntnisse zu Aufbau und Programmierung eines Computers vermittelt. Es folgen Hinweise zur Algorithmierung, zur Arbeit mit einem BASIC-Interpreter und zur Programmerstellung. Den Hauptteil der Ausführungen bildet jedoch die vollständige Darstellung der Programmiersprache BASIC gemäß dem Standard für Minimal-BASIC. Es sind alle Sprachelemente, Datenarten und Anweisungen detailliert ent-

halten und anhand von Beispielen erläutert. Außerdem werden wesentliche Spracherweiterungen beschrieben, die in unterschiedlichen und häufig genutzten BASIC-Versionen Eingang gefunden haben. Daran schließen sich einige Hinweise zu Programmierfehlern an. Eine zusammenfassende Darstellung der Syntax der Programmiersprache BASIC enthält der Anhang, dem die im ISO-Standard 6373 verwendete Beschreibungsform zugrunde liegt.

Die im Buch benutzten Beispiele sowie deren Lösungen sind nach didaktischen Gesichtspunkten ausgewählt, wobei besonderer Wert auf leicht verständliche Algorithmen und einen übersichtlichen Programmaufbau gelegt wird. Zur graphischen Darstellung der Algorithmen wurden alternativ sowohl Programmablaufpläne als auch Struktogramme verwendet. Die Auswahl erfolgte vorrangig unter dem Aspekt einer geeigneten Vermittlung beider Beschreibungsformen und weitgehend unabhängig von Effektivitätskriterien. Die aufgeführten Programme wurden auf einem Bürocomputer mit CPM-Betriebssystem und Mikrosoft-BASIC-Interpreter entwickelt sowie getestet.

Das Hauptanliegen der Broschüre besteht darin, eine geschlossene Darstellung der Lösung von einfachen Problemstellungen der Informationsverarbeitung mit Hilfe der Programmiersprache BASIC zu geben. Darüber hinaus sind einige praktische Hinweise für die Handhabung eines entsprechenden Computers enthalten. Die Darstellung des Stoffes ist so gewählt, daß auch Programmieranfänger sich im Selbststudium die Programmiersprache BASIC aneignen können. Mit dem vorliegenden Buch soll ein breiter Kreis von Lesern angesprochen werden.

An dieser Stelle möchten wir uns bei Herrn Doz. Dr. sc. W.-G. Matthäus für zahlreiche Hinweise und Vorschläge zur Verbesserung des Manuskripts bedanken. Außerdem gilt unser Dank dem VEB Fachbuchverlag Leipzig für die sehr angenehme Zusammenarbeit.

Die Verfasser

Inhaltsverzeichnis

1. Einführung

1.1. Aufbau eines Computers

Zum Grundaufbau eines Computers gehören die Eingabe-, die Ausgabe- und die Rechnereinheit (Bild 1.1). Die Eingabeeinheit ist in der Regel eine Tastatur, die Ausgabeeinheit heute meist ein Bildschirm. Die Rechnereinheit selbst setzt sich aus folgenden wesentlichen Funktionseinheiten zusammen (Bild 1.2):

- Prozessor
- Hauptspeicher
- Ein- und Ausgabesystem
- BUS-System.

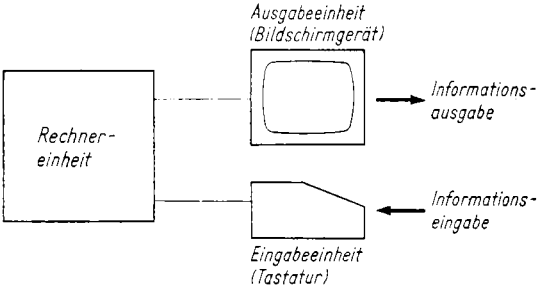


Bild 1.1. Grundaufbau eines Computers

Der Prozessor, der Hauptspeicher und Komponenten des Ein- und Ausgabesystems (E/A-System) liegen meist als mikroelektronische digitale Schaltkreise vor und sind schaltungstechnisch durch ein BUS-System (ein System von Verbindungsleitungen) miteinander verbunden. Über dieses System findet der Informationsaustausch zwischen den einzelnen Funktionseinheiten der Rechnereinheit statt.
Eine sehr komplexe Funktionseinheit ist der Prozessor, der in der mikroelektronischen Realisierungsform auch als Mikroprozessor bezeichnet wird. Er koordiniert und steuert alle Vorgänge im Computer und ist damit gewissermaßen "Steuerzentrale" und "Verarbeitungszentrum". Man nennt ihn auch Zentrale Verarbeitungseinheit (ZVE) oder CPU (Central processing unit). Eine weitere wichtige Einheit ist der Hauptspeicher, in dem die auszu-

führenden Programme und die zu verarbeitenden Daten gespeichert sind. Im allgemeinen gliedert er sich in zwei Bereiche:

- Nur-Lese-Speicher (read only memory-ROM)
- Schreib-Lese-Speicher (random access memory-RAM)

Im ROM-Bereich ist es nicht möglich, die eingespeicherten Informationen zu verändern, daher auch die Bezeichnung Festwertspeicher. Im RAM können beliebig oft Informationen abgespeichert und wieder gelesen werden.

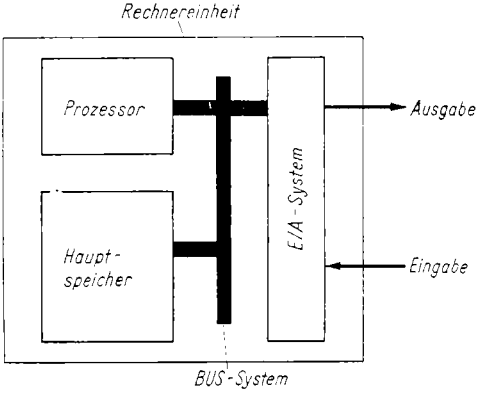


Bild 1.2. Grundstruktur der Rechnereinheit

Die notwendige Verbindung der Rechnereinheit zum Nutzer stellt das Ein- und Ausgabesystem her, das den Anschluß von E/A-Geräten oder anderen zu steuernden Einrichtungen an den Computer ermöglicht. Es besteht auch die Möglichkeit, außer den genannten Geräten andere an die Rechnereinheit anzuschließen. Beispielsweise gibt es neben dem Hauptspeicher als internem Speicher auch sogenannte externe oder periphere Speichereinheiten. Sie ergänzen den Hauptspeicher, erweitern das Speichervolumen und erhöhen damit die Leistungsfähigkeit eines Computers um ein Vielfaches. Bei Kleincomputern werden z. B. Kassettengeräte (Kassettenrecorder) oder Floppy-Disk-Einheiten (Diskettenlaufwerke) eingesetzt. In Kassetteneinheiten dienen übliche Magnetbandkassetten für die Speicherung der Informationen, während bei Floppy-Disk-Einheiten kreisförmige, magnetisierbare Folien (Disketten) diese Aufgabe übernehmen.

Als weitere periphere Einheiten sind häufig zusätzliche Ein- und Ausgabegeräte wie etwa Drucker, Plotter (Zeichengerät) oder Fernschreiber bzw. elektronische Schreibmaschinen anzutreffen. Bild 1.3 zeigt die Blockdarstellung eines mit peripheren Einheiten erweiterten Computers.

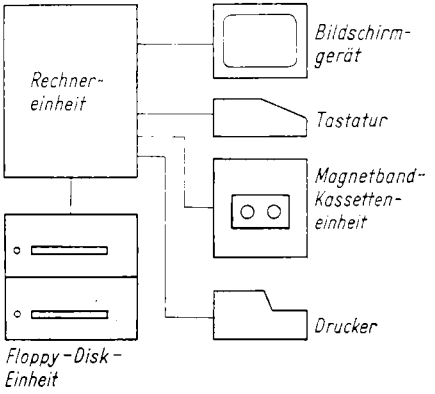


Bild 1.3. Computer mit mehreren peripheren Einheiten

1.2. Programmierung eines Computers

Ein Computer ist zur Lösung vielfältiger Aufgabenstellungen geeignet. Dazu muß ihm zunächst ein Programm übergeben oder eingegeben werden. Ein solches Programm besteht aus einer Folge von Anweisungen und beschreibt die Aktionen, die der Computer auszuführen hat, wie z. B. die Eingabe von Daten, die Ausführung von Berechnungen, die Verarbeitung nichtnumerischer Daten (Texte) oder gar die Steuerung von Geräten bzw. Anlagen. Das Programm ist während der Programmentwicklung vom Programmierer zu entwerfen und muß, wenn es vom Computer ausgeführt werden soll, im Hauptspeicher abgespeichert sein.
Man unterscheidet *frei programmierbare* und *nicht frei programmierbare* Computer. Nicht frei programmierbare Computer sind oft in Geräte eingebaut und für ganz spezielle Zwecke vorgesehen. Sie verfügen über ein Programm oder Programmsystem, daß nachträglich nicht mehr änderbar ist. Frei programmierbare Rechner

sind dadurch gekennzeichnet, daß die verschiedenartigsten Programme eingegeben und ausgeführt und damit auch unterschiedlichste Aufgabenstellungen gelöst werden können. Im weiteren werden stets frei programmierbare Computer vorausgesetzt.
Zu einem arbeitsfähigen Computer gehören demnach technische Einrichtungen *und* Programme. In der rechentechnischen Praxis wird die Gesamtheit aller gerätetechnischen Elemente eines Computers auch als *Hardware* bezeichnet. Alle beim Programmieren entstehenden Produkte, wie die Programme und die Programmdokumentationen sowie alle programmtechnischen Mittel zur Unterstützung der Programmierung sind unter der Bezeichnung *Software* zusammengefaßt.

Programmiersprachen

Es wurde bereits erläutert, daß ein Computer erst dann eine Aufgabe lösen kann, wenn die Lösungsvorschrift als Programm im Computer gespeichert ist. Zur Formulierung bzw. Notierung dienen Programmiersprachen. Im Unterschied zu natürlichen Sprachen, die historisch in einem langen Entwicklungsprozeß und unter verschiedenen Einflußfaktoren entstanden sind, werden Programmiersprachen vom Menschen künstlich geschaffen. Ihr Aufbau unterliegt exakten Regeln. Mit Hilfe der Programmiersprache werden im Computer die Daten (Verarbeitungsgrößen, Ein- und Ausgabegrößen) und die Verarbeitungsvorgänge (z.B. arithmetische Operationen, Ein- und Ausgabeprozesse) symbolisch beschrieben. Programmiersprachen können in ihrer Leistungsfähigkeit und Darstellungsmöglichkeit sehr unterschiedlich sein. Grundlage dafür ist immer, daß sich jeder informationsverarbeitende Prozeß in eine Folge von Grundoperationen auflösen läßt, die von der Programmiersprache beschrieben wird. Erst auf diese Weise gelingt es, die Problemstellung "rechentechnisch" zu formulieren und dem Computer als Aufgabe zu übertragen.
In der Praxis hat man drei Klassen von Programmiersprachen zur Auswahl: Maschinensprachen, maschinenorientierte (maschinennahe) und problemorientierte (höhere) Sprachen. Da die Formulierung der Programme in Maschinen- und maschinenorientierten Sprachen sehr umständlich und unhandlich ist, und zudem eine solche Sprache schwierig erlernbar ist, wurden höhere Programmiersprachen entwickelt, die eine Programmierung wesentlich vereinfachen. Darüber hinaus sind höhere Sprachen weit-

gehend unabhängig vom Rechnertyp, d.h., die Programme sind auf verschiedenen Computern ausführbar. Es gibt eine sehr große Zahl von höheren Programmiersprachen mit unterschiedlichsten Leistungsmerkmalen und Verwendungszweck. Auch die Sprache BASIC gehört dazu.

BASIC ist eine leicht verständliche Programmiersprache. Sie benutzt einfache englische Wörter sowie allgemein übliche alphanumerische Zeichen und Operatoren. Wie jede Programmiersprache unterliegt sie grammatikalischen Regeln, die äußerst einfach gehalten sind. Heute existiert eine Vielzahl von BASIC-Sprachversionen, die sich mehr oder weniger stark voneinander unterscheiden.

Ein internationaler Standard existiert bisher nur für *Minimal*-BASIC [1], [2]. Die Standardisierung ermöglicht die Ausführung von Programmen auf verschiedenen Computern ohne oder mit nur geringen Programmänderungen. Der Name "Minimal-BASIC" besagt, daß der entsprechende Standard festlegt, welche Sprachelemente die Sprache eines realisierten BASIC-Systems "minimal" enthalten soll. Minimal-BASIC ist der Grundstock für die gebräuchlichsten BASIC-Versionen.

Zahlreiche BASIC-Systeme realisieren einen Sprachumfang, der gegenüber Minimal-BASIC erweitert ist (Erweiterte BASIC-Sprachversionen). Sie enthalten zusätzliche, z.T. sehr leistungsfähige Sprachkomponenten, z.B. Anweisungen für Matrizenoperationen, Graphikarbeit und für Echtzeitbetrieb. Diese BASIC-Spracherweiterungen sind jedoch nicht standardisiert und unterscheiden sich bei den einzelnen Systemen teilweise erheblich. Daher sind Programme, die Elemente einer erweiterten BASIC-Version benutzen, oft ohne wesentliche Änderungen auf anderen BASIC-Systemen nicht ausführbar. Außerdem benötigen erweiterte Sprachversionen in der Regel einen leistungsfähigeren Computer. Es gibt internationale Bestrebungen [3], auch eine erweiterte BASIC-Version zu standardisieren.

Als Beispiel einer Sprache, die nur für sehr kleine Rechner wie Hobby- oder Spielcomputer Anwendung findet, sei Tiny-BASIC genannt (engl.: tiny – "winzig", d.h. BASIC mit sehr kleinem Speicherplatzbedarf) [9]. Sprachvereinfachungen, die allerdings die Leistungsfähigkeit dieser Sprachversion entscheidend mindern, machen es möglich, Tiny-BASIC in nur wenigen Stunden zu erlernen. Für die Praxis hat Tiny-BASIC jedoch geringere Bedeutung.

Programmiersysteme

Ein Computer kann nur solche Programme ausführen, die in Maschinensprache vorliegen. Ein BASIC-Programm muß daher wie jedes in einer höheren Programmiersprache geschriebene Programm in Maschinensprache übersetzt werden. Hierzu ist ein spezielles Programm (Übersetzerprogramm) notwendig, das der Computer selbst ausführt. Die in einer Programmiersprache geschriebenen Programme, außer solchen, die in Maschinensprache formuliert sind, nennt man *Quellprogramme*. Ein Übersetzerprogramm wandelt ein Quellprogramm in ein *Zielprogramm* um. Nicht immer ist das Zielprogramm sofort ein Maschinenprogramm. Häufig wird zunächst eine Übersetzung in eine sogenannte Zwischensprache vorgenommen und in einem weiteren Übersetzungsvorgang das Maschinenprogramm erzeugt.
Neben dem Übersetzerprogramm erfordert eine effektive Programmierarbeit noch weitere Programme. Dazu gehören beispielsweise solche, die die Ausführung des übersetzten Nutzerprogramms steuern und kontrollieren (Laufzeitsystem) und auch Programme, die Aufbereitung und Korrektur eines Quellprogramms ermöglichen (Editor). Die Gesamtheit aller dieser Programme oder Programmkomponenten, die zur Anwendung einer höheren Sprache benötigt werden, bezeichnet man als *Programmiersystem* (auch Sprachverarbeitungssystem). Ein Programmiersystem hat außerdem die Aufgabe, etwaige Programmierfehler automatisch festzustellen und dem Programmierer anzuzeigen.
In der Praxis der BASIC-Programmierung finden zwei Arten von Programmiersystemen Anwendung:

- Interpreter-Systeme
- Compiler-Systeme

Beide unterscheiden sich vor allem in der Art der Übersetzung und der Aufbereitung des entsprechenden Maschinenprogramms. Die Interpreter-Systeme haben für die BASIC-Programmierung die größere Bedeutung. Die Interpreterarbeitsweise ist dadurch gekennzeichnet, daß ein BASIC-Programm schrittweise Anweisung für Anweisung in Maschinensprache übersetzt und jede übersetzte Anweisung auch sofort ausgeführt wird ("interpretieren" entspricht hier "übersetzen und ausführen"). Bei einem Interpreter-System sind demnach Übersetzung und Ausführung eines Programms untrennbar miteinander verbunden. Das entsprechende Programmsystem nennt sich auch BASIC-Interpreter oder

einfach *Interpreter*. In einem Interpreter sind die obengenannten Programmkomponenten (Übersetzerprogramm, Laufzeitsystem, Korrektursystem u.a.) zu einem gemeinsamen Programmsystem zusammengefaßt. Erkennt der Interpreter während der Übersetzung oder Ausführung eines Programms einen Programmierfehler, so gibt er eine entsprechende Mitteilung aus. Der Fehler kann dann sofort korrigiert und anschließend die Programmausführung erneut gestartet werden.

Bei Compiler-Systemen wird das BASIC-Programm als eine Einheit aufgefaßt. Folgende Schritte werden durchlaufen:

- Übersetzen des Quellprogramms mit Hilfe eines Übersetzerprogramms (*Compiler*) in ein Objektprogramm
- Binden (Linken), also Zusammenfassen eines oder mehrerer Objektprogramme mit entsprechenden Betriebsprogrammen (Laufzeitsystem) zu einem Maschinenprogramm mit Hilfe eines Bindeprogramms (*Verbinder*)
- Laden des Maschinenprogramms in den Hauptspeicher mit Hilfe eines Ladeprogramms (*Lader*)
- Start und Ausführen des Maschinenprogramms

Programmierfehler werden während der Übersetzung erkannt. Nach der Korrektur des Quellprogramms mit Hilfe des Editors ist allerdings das gesamte Programm erneut zu übersetzen. Werden bestimmte Programmierfehler erst in der Ausführphase festgestellt, sind bei Compiler-Systemen nach vorgenommener Korrektur sogar alle aufgeführten Schritte noch einmal abzuarbeiten.

Der Vorteil eines *Interpreter-Systems* liegt in seiner sehr benutzerfreundlichen und einfachen Arbeitsweise. Mit ihm können sehr günstig BASIC-Programme erstellt, getestet, korrigiert und ausgeführt werden. Darüber hinaus gestattet der Interpreter entsprechend seiner Funktionsweise einen Dialog zwischen Benutzer und Computer. Der Nachteil der Interpreterarbeitsweise besteht darin, daß die BASIC-Programme eine größere Rechenzeit benötigen. Bei vielen Programmen macht sich jedoch diese Tatsache weniger störend bemerkbar.

Häufig wird bei BASIC-Interpretern mit einer laufzeit- und speicherplatzgünstigen Zwischensprache gearbeitet. Während der Eingabe des BASIC-Programms wird es bereits Anweisung für Anweisung in die Zwischensprache übersetzt und in dieser Form im Hauptspeicher abgelegt. Die Interpretation kann dann wesentlich schneller erfolgen.

Bei *Compiler-Systemen* erreicht man im Gegensatz zur Interpreterarbeitsweise kürzere Rechenzeiten der BASIC-Programme. Die Verwendung eines Compiler-Systems ist daher besonders bei der Forderung nach möglichst kurzer Programmlaufzeit zu empfehlen. Es besteht auch die Möglichkeit, ein BASIC-Programm zunächst mit Hilfe des BASIC-Interpreters zu entwickeln, zu testen und fehlerfrei zu machen und anschließend in ein Compiler-System zu überführen. Viele kleinere Computer realisieren jedoch nur die Interpreterarbeitsweise. In den weiteren Ausführungen wird auch nur sie vorausgesetzt.

Betriebssystem

Um einen Computer in seiner Gesamtheit rationell und effektiv zu betreiben, sind besondere Betriebs- und Steuerprogramme erforderlich. Diese haben allgemeingültigen Charakter, da sie nicht nur der Lösung einer einzigen Aufgabe dienen, sondern Voraussetzungen zur Lösung aller auftretenden Probleme am jeweiligen Computer schaffen. Sie werden zu einem besonderen Programmsystem, dem Betriebssystem, zusammengefaßt. Das Betriebssystem übernimmt die Steuerung und die Kontrolle aller Hard- und Softwarekomponenten. Es enthält eine Vielzahl von Programmroutinen, z.B. Programmodule für die Ein- und Ausgabe. Im allgemeinen verwaltet das Betriebssystem auch eine Reihe von Systemprogrammen, unter anderem Programmiersysteme für die unterschiedlichsten Programmiersprachen sowie Datenbestände und Anwenderprogramme.

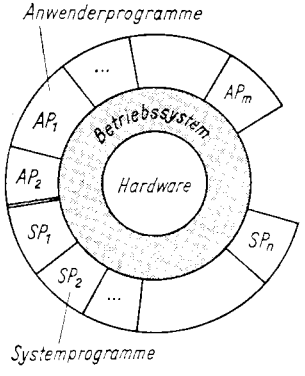


Bild 1.4. Verwaltungsbereich eines Betriebssystems

Das Betriebssystem und eine Reihe von Systemprogrammen werden in der Regel vom Hersteller des Computers mitgeliefert. Den schematischen Aufbau eines vollständigen Softwaresystems zeigt Bild 1.4. Bei sehr kleinen Rechnern ist das Betriebssystem oft sehr einfach aufgebaut und wird als Steuer- bzw. Monitorprogramm oder einfach als *Monitor* bezeichnet.
Zusammenfassend sei festgestellt, daß für das Arbeiten mit einem BASIC-Interpreter im Hauptspeicher eines Computers folgende Programme bzw. Programmsysteme Platz finden müssen (Bild 1.5): das Betriebssystem, der BASIC-Interpreter und ein oder mehrere BASIC-Anwenderprogramme.

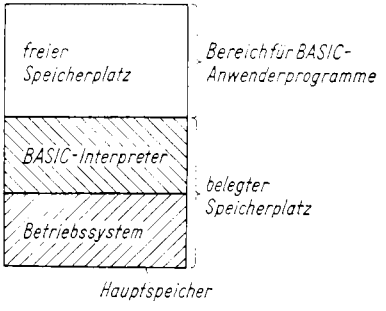


Bild 1.5. Hauptspeicherbelegung eines Computers mit BASIC-Interpreter

1.3. Algorithmen und ihre Darstellung

Eine wesentliche Etappe auf dem Weg von der Aufgabenstellung zum Programm ist die Entwicklung eines geeigneten Algorithmus, d.h. einer Lösungsvorschrift. Allgemein versteht man unter einem Algorithmus eine Vorschrift, nach der Handlungen in einer bestimmten Reihenfolge auszuführen sind und nach der man alle Aufgaben eines Typs lösen kann. Das Entwerfen eines Algorithmus für eine gegebene Problemstellung wird als *Algorithmierung* bezeichnet [4], [5].
Zwei Beispiele sollen die Aufstellung von Algorithmen verdeutlichen.

Beispiel:

Zunächst sei eine sehr einfache Aufgabe zu lösen. Mit Hilfe eines Computers sind für zwei Dreiecke, deren Flächen sowie die Summe beider Flächen zu berechnen und die Werte auszugeben (Bild 1.6).

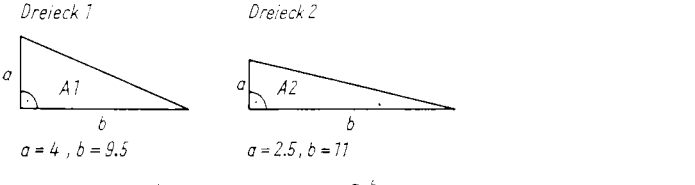


Bild 1.6. Berechnung der Fläche zweier Dreiecke

Die Zerlegung dieser Aufgabe in schematisch ausführbare Handlungsschritte führt zu folgendem Algorithmus:

0 BEGINN des Algorithmus, gehe zu Zeile 1
1 Berechne Fläche von Dreieck 1, gehe zu Zeile 2
2 Berechne Fläche von Dreieck 2, gehe zu Zeile 3
3 Addiere beide Flächen, gehe zu 4
4 Ausgabe von Fläche 1, Fläche 2 und Gesamtfläche, gehe zu 5
5 ENDE des Algorithmus

Jeder Zeile entspricht eine in sich abgeschlossene algorithmische Einheit. Außerdem ist jede Zeile numeriert, um später Beziehungen zwischen den einzelnen Einheiten herzustellen.
Es ist zweckmäßig, diesen bisher in natürlich-sprachlicher Form vorliegenden Algorithmus in eine Darstellung zu überführen, die der jeweils verwendeten Programmiersprache besser angepaßt ist. Weiter wird für die im folgenden verwendete Notierung der Algorithmen festgelegt, daß die Schritte eines Algorithmus in der Reihenfolge ihrer Notierung ausgeführt werden. Nur Abweichungen von dieser Folge sind deshalb gesondert anzugeben. Diesen Gesichtspunkten entspricht folgende Notationsform:

```
0 BEGINN
1 A1=4*9.5/2
2 A2=2.5*11/2
3 A=A1+A2
4 Ausgabe von A1, A2, A
5 ENDE
```

Möglichkeiten der graphischen Darstellung dieses Algorithmus werden später gezeigt.

Beispiel:

Die Summe von n Zahlen a_i $(i = 1, \ldots, n)$ ist zu berechnen:

$$S = \sum_{i=1}^{n} a_i \quad (n < \infty)$$

Der Wert n und die einzelnen Zahlen a_i sollen in den Computer eingegeben und die berechnete Summe S ausgegeben werden. Dafür ist ein Algorithmus zu formulieren.

Unproblematisch ist die Lösung dieser Aufgabe, wenn für n ein ganz bestimmter und nicht zu großer Wert vorgegeben wird. Beispielsweise kann man für $n = 4$ die Rechenvorschrift in der Form

$$S = a_1 + a_2 + a_3 + a_4$$

notieren. Komplizierter wird es jedoch, wenn die Anzahl der zu addierenden Werte groß ist oder wenn die Vorschrift, wie im vorliegenden Beispiel, für eine variable Zahl von Summanden formuliert werden soll. Dann ist es sinnvoll, die Aufgabe in mehreren Schritten zu lösen. So kann man beispielsweise die Berechnung von S_n in folgende n Schritte zerlegen:

1. Schritt: $S_1 = a_1$
2. Schritt: $S_2 = S_1 + a_2$
3. Schritt: $S_3 = S_2 + a_3$
$\vdots$
n. Schritt: $S_n = S_{n-1} + a_n$

Die Formulierung des Algorithmus läßt sich unter Zuhilfenahme der Zuweisungsgleichung vereinfachen. Diese hat die allgemeine Form

$$\boxed{x := \text{Ausdruck}}\,.$$

Sie sagt aus, daß der Wert des rechts stehenden Ausdrucks zu berechnen und der links stehenden Größe (Variable) x zuzuweisen ist. So bedeutet beispielsweise "$S:= S+a_1$", daß zum Wert von S der Wert von a_1 zu addieren und die Summe der Variablen S zuzuordnen ist (neuer Wert von S). Das Zeichen ":=" wird Ergibtzeichen genannt. "$S:= S+a_1$" liest man als "S ergibt sich aus $S+a_1$".
Durch Einführung der Zuweisungsgleichung kann damit die im vorliegenden Rechenschema vorgenommene Indizierung von S entfallen. Es entsteht folgende Darstellung für die Summation:

$$
\begin{array}{|l|}
\hline
S:=0 \\
S:=S+a_1 \\
S:=S+a_2 \\
S:=S+a_3 \\
\vdots \\
S:=S+a_n \\
\hline
\end{array}
$$

Jedem der Schritte (außer dem ersten) liegt dabei die allgemeine Vorschrift

$$S := S + a_i$$

zugrunde. Somit wird es möglich, diese Schritte durch wiederholte (zyklische) Ausführung einer einzigen Vorschrift zu realisieren.
Ausgehend von diesem prinzipiellen Lösungsweg soll der vollständige Algorithmus für die vorliegende Aufgabenstellung entwickelt werden. Dazu wählt man folgendes Vorgehen. Nach Eingabe des Wertes n wird die Summe S auf Null und der Zählindex i auf eins gesetzt (Anfangswerte). In weiteren Schritten folgt dann im Zyklus die Eingabe einer Zahl a_i, die Addition dieses Wertes zum bisherigen Wert der Summe S und die Erhöhung des Zählindex i. Außerdem wird am Ende eines jeden Zyklus geprüft, ob i den Wert n bereits überschritten hat. Ist dies der Fall, dann ist S die zu ermittelnde Summe, die ausgegeben wird. Das vorliegende Beispiel erhält somit folgende Notationsform des Algorithmus:

0	BEGINN
1	Eingabe n
2	$S:=0$
3	$i:=1$

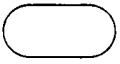
```
4  Eingabe a_i
5  S:=S+a_i
6  i:=i+1
7  Falls i≤n, gehe zu 4, sonst zu 8
8  Ausgabe S
9  ENDE
```

Der in dieser Form dargestellte Algorithmus kann mit Hilfe einer höheren Programmiersprache leicht in ein Programm umgesetzt werden.

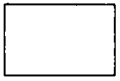
Graphische Darstellung von Algorithmen

Algorithmen werden zur besseren Veranschaulichung oft graphisch dargestellt. Besondere Bedeutung haben dafür in der praktischen Programmierung Programmablaufplan und Struktogramm. Beide geben den Ablauf in einem informationsverarbeitenden System in Abhängigkeit von den jeweils vorhandenen Daten wieder.

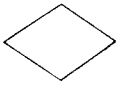
Die wichtigsten Sinnbilder zur Darstellung von Programmablaufplänen sind im Bild 1.7 angegeben [6], [7].



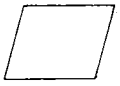
Grenzstelle; Anfang, Ende, Unterbrechung im Programmablauf



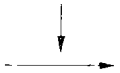
Allgemeine Darstellung des Bearbeitens, Darstellung einer Operation oder einer Gruppe von Operationen (Verarbeitungsvorgang)



Verzweigen oder Verzweigung auf Grund eines Vergleiches (Entscheidung)



Allgemeine Darstellung einer Eingabe oder Ausgabe



Flußlinie (mit Richtungsangabe)

Konnektor; Übergang zu einer anderen Stelle oder von einer anderen Stelle

Bild 1.7. Sinnbilder für Programmablaufpläne (Auszug [6])

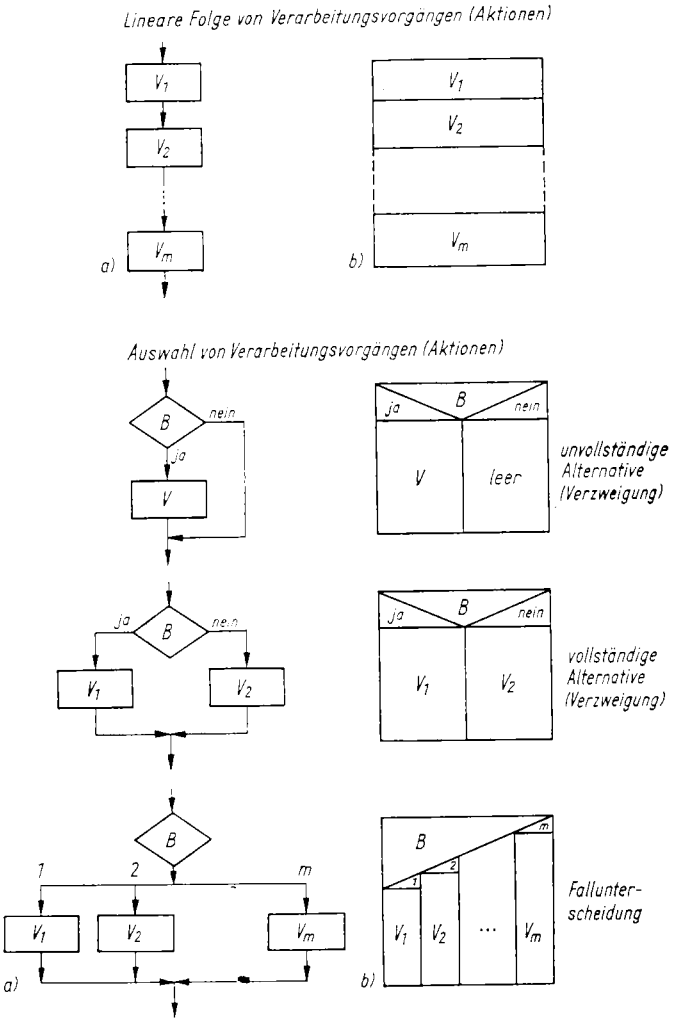
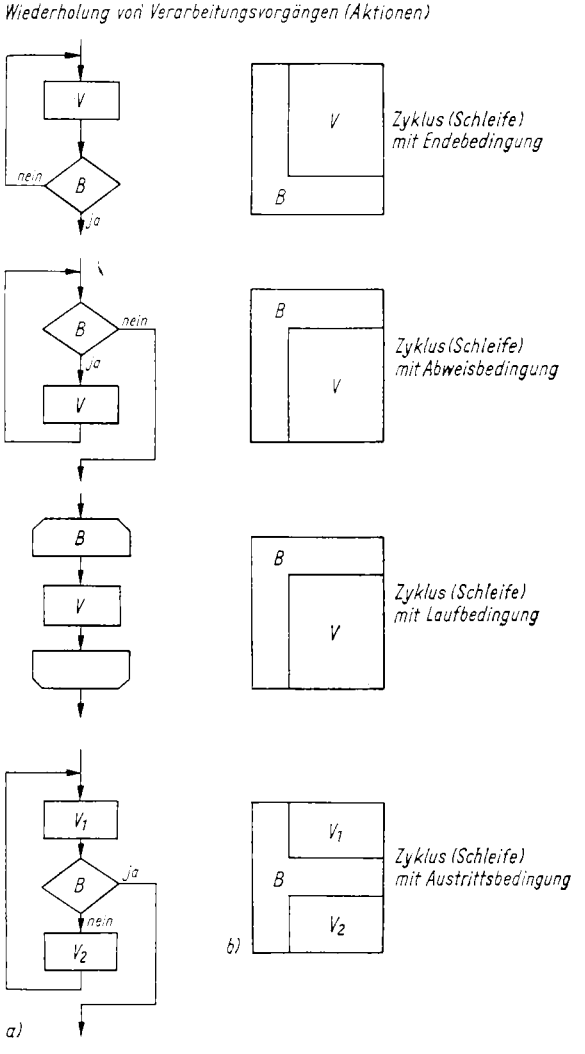


Bild 1.8. Algorithmische Grundstrukturen in der Darstellung als Programmablaufplan (a) und Struktogramm (b); (V Verarbeitungsvorgang oder Folge von Verarbeitungsvorgängen, B Bedingung)



Fortsetzung Bild 1.8

Programmablaufpläne machen den Algorithmus anschaulich sichtbar. Diese Darstellungsform kann aber infolge ihrer relativ großen strukturellen Freizügigkeit auch zum Entwurf schlecht strukturierter, unübersichtlicher Algorithmen verleiten. Beispiele für einfache Programmablaufpläne sind in Bild 1.9a und 1.10a enthalten.
Struktogramme dagegen beschreiben die Algorithmen unter Verwendung von sogenannten Strukturblöcken [4]. Sie repräsentieren algorithmische Grundstrukturen (Bild 1.8b). Jeder Strukturblock besitzt nur einen Eingang (obere Kante) und einen Ausgang (untere Kante). Durchlaufen wird ein Strukturblock bzw. Struktogramm stets von oben nach unten. Struktogramme unterstützen den Entwurf gut strukturierter Algorithmen bzw. Programme, indem sie beim Entwurf zur Verwendung vorgegebener algorithmischer Grundstrukturen zwingen. Die Bilder 1.9b und 1.10b zeigen Beispiele für Struktogramme.
Grundsätzlich läßt sich jeder Algorithmus auf eine Kombination der drei algorithmischen Grundstrukturen zurückführen. Bild 1.8 zeigt diese mit Hilfe von Programmablauf- und Struktogramm-

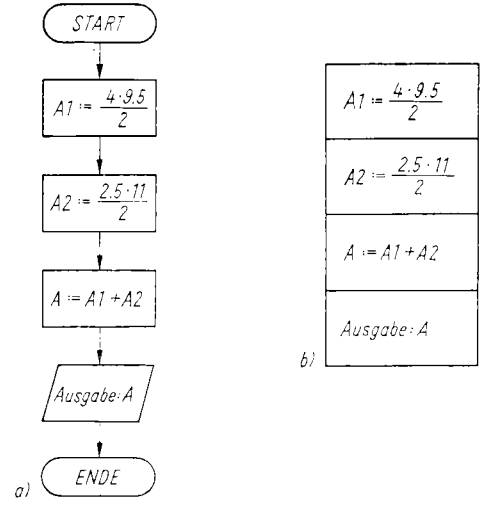


Bild 1.9. Graphische Darstellung des Algorithmus zur Berechnung der Fläche zweier Dreiecke (Bild 1.6)
a) Programmablaufplan b) Struktogramm

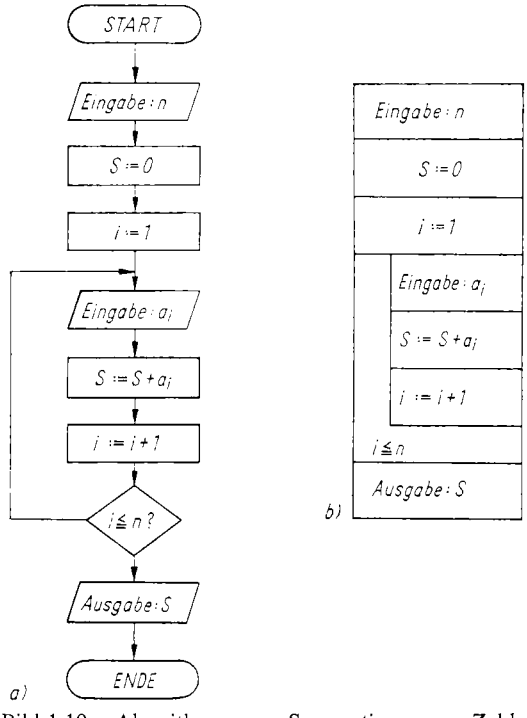


Bild 1.10. Algorithmus zur Summation von n Zahlen a_i
a) Programmablaufplan b) Struktogramm
(Schleife wird durchlaufen bis Bedingung $i \leqq n$ nicht mehr erfüllt)

Sinnbildern in verschiedenen Modifikationen. Die drei Grundstrukturen sind:

– *Folge von Verarbeitungsvorgängen* (lineare Folge, Sequenz)
Die einfachste algorithmische Struktur besteht aus einer Folge von Verarbeitungsvorgängen. Diese laufen in der angegebenen Reihenfolge nacheinander ab.

– *Auswahl zwischen Verarbeitungsvorgängen* (Verzweigung, Alternative)
In Abhängigkeit von einer Bedingung B wird einer der möglichen Verarbeitungsvorgänge V ausgewählt.

– *Wiederholung von Verarbeitungsvorgängen* (Schleife, Zyklus)
Ein Verarbeitungsvorgang V wird solange wiederholt, bis eine Bedingung B erfüllt bzw. nicht mehr erfüllt ist.

Unter Verarbeitungsvorgängen werden dabei sowohl rechnerinterne Prozesse als auch Ein- und Ausgabeprozesse verstanden. Bei einem Programmentwurf sollte man immer versuchen, die genannten algorithmischen Grundstrukturen zu verwenden, um einen übersichtlichen Aufbau der entsprechenden Programme zu erreichen. Mehrere Beispiele werden dies später noch verdeutlichen.

1.4. Notation von BASIC-Programmen

Ein BASIC-Programm setzt sich aus einer Folge von BASIC-Programmzeilen zusammen. Eine Programmzeile besteht aus Zeilennummer, BASIC-Anweisung und einem Zeilenendezeichen. Jede BASIC-Anweisung beschreibt dabei eine bestimmte Aktion, d.h. einen Verarbeitungsvorgang im Computer. Das Programm wird im Normalfall in der Reihenfolge der Zeilennummern ausgeführt. Die letzte Anweisung eines BASIC-Programms ist in der Regel die END-Anweisung. Aufbau und Wirkungsweise der einzelnen Programmanweisungen werden in mehreren Abschnitten im Verlaufe der weiteren Ausführungen noch ausführlich erläutert. Hier sollen zunächst nur einige elementare Anweisungen sowie zwei sehr einfache Programmbeispiele aufgezeigt werden, um die Notationsform von BASIC-Programmen zu veranschaulichen.

Beispiel:
Eine der wichtigsten Anweisungen in BASIC ist die PRINT-Anweisung (engl.: to print – drucken, ausgeben). Sie bewirkt die Ausgabe von berechneten Größen auf dem Bildschirm. Beispielsweise wird mit Hilfe der Programmanweisung

```
20 PRINT 5+9
```

die Summe 5+9 berechnet und der berechnete Wert auf dem Bildschirm ausgegeben. Die Zahl 20 ist die Zeilennummer. In ähnlicher Weise wird bei Ausführung der folgenden Anweisung

```
10 PRINT "BEGINN"
```

auf dem Bildschirm die Zeichenfolge BEGINN ausgegeben. Mit beiden Anweisungen kann bereits ein sehr einfaches BASIC-

Programm aufgestellt werden:

```
10 PRINT "BEGINN"
20 PRINT 5+9
30 END
```

Die END-Anweisung schließt das Programm ab. Bei Ausführung dieses Programms werden auf dem Bildschirm des Computers zunächst das Wort BEGINN und anschließend der Wert 14 ausgegeben.

Beispiel:
Eine weitere wichtige BASIC-Programmanweisung ist die LET-Anweisung (engl.: to let – mache, führe aus). Sie bewirkt, daß die in ihr angegebene Verarbeitungsvorschrift ausgeführt wird. Das folgende Programmbeispiel zeigt die Berechnung der Fläche zweier verschiedener Dreiecke entsprechend Bild 1.6.

```
10 LET A1=4*9.5/2
20 LET A2=2.5*11/2
30 LET A=A1+A2
40 PRINT A1,A2,A
50 END
```

In Zeile 10 und 20 werden die Flächen A1 und A2 der beiden Dreiecke berechnet. Zeile 30 bewirkt die Berechnung der Gesamtfläche A, und bei Ausführung der Zeile 40 werden die ermittelten Werte A1, A2 sowie A ausgegeben.

2. Arbeit mit BASIC-Interpreter

2.1. Inbetriebnahme

Zu Beginn der Arbeit am Computer muß der BASIC-Interpreter, falls er nicht als ständig im Hauptspeicher gespeichertes Programm vorliegt, in den Hauptspeicher geladen und anschließend gestartet werden. In der Regel erfolgt dies durch ein bestimmtes Betriebssystem-Kommando (nicht zu verwechseln mit den noch

zu behandelnden BASIC-Kommandos). Es gibt aber auch BASIC-Systeme, die mit dem Einschalten des Computers ihre Arbeit automatisch aufnehmen. Nach dem Start des BASIC-Interpreters meldet dieser mit einer Bildschirmausschrift, daß er betriebsbereit ist. Bei anspruchsvolleren Systemen wird außerdem die Größe des verfügbaren Speicherbereichs angegeben. Als Mitteilung für die Arbeitsbereitschaft dient im allgemeinen die Zeichenfolge "OK" oder "READY" (*Fertigmeldung*). Sie zeigt an, daß sich der Computer auf der *Kommandoebene* befindet, d.h., daß der Benutzer Eingaben über die Tastatur vornehmen kann. Alle Eingabeinformationen des Nutzers werden zur Kontrolle auf dem Bildschirm mitgeschrieben. Auf die Eingaben, die immer mit einem Schlußzeichen (Zeilenendezeichen) abzuschließen sind, antwortet der Computer durch Ausgabe entsprechender Zeichen oder Mitteilungen auf dem Bildschirm. So entwickelt sich zwischen Benutzer und Computer ein Dialog, daher auch die Bezeichnung Dialogbetrieb.

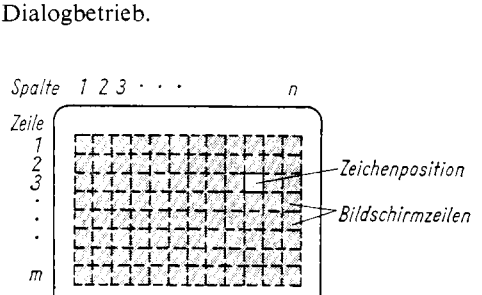


Bild 2.1. Geometrische Einteilung eines Bildschirmes (*m* Zeilenanzahl, *n* Zeichenanzahl je Zeile)

Die Ein- und Ausgabe von Informationen erfolgt bei BASIC zeilenorientiert. Jeder Bildschirm ist in eine feste Anzahl von Zeilen eingeteilt, wobei jede Zeile wiederum eine bestimmte Anzahl von Zeichenpositionen enthält (Bild 2.1). Alle für die Informationsdarstellung verwendbaren Zeichen sind auf der Eingabetastatur des Computers vorhanden. Neben den üblichen Zeichen (Buchstaben, Ziffern, Sonderzeichen) befinden sich auf der Tastatur zusätzlich einige Tasten, die zur Realisierung von Steuerfunk-

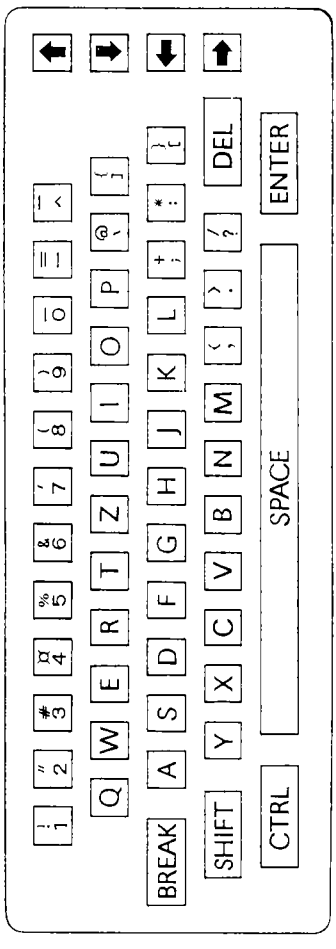


Bild 2.2. Zeichenbelegung einer Computer-Eingabetastatur (Beispiel)

BREAK Abbruch eines Programms, SHIFT Umschaltung Groß-/Kleinbuchstaben, CTRL Erzeugung Steuerzeichen, SPACE Zwischenraum, ENTER Zeilenende, DEL Löschen eines Zeichens, Cursorsteuerung ↑ höher, ↓ tiefer, ← links, → rechts

tionen (Steuerzeichen) nötig sind (Bild 2.2). Wichtigstes Steuerzeichen ist das Zeilenendezeichen. Es wird benutzt, wenn eine über die Tastatur eingegebene und auf dem Bildschirm dargestellte Zeile abgeschlossen und auf den nächsten Zeilenanfang übergegangen werden soll. Auf der Tastatur ist diese Taste in der Regel mit "ENTER" (ET), "NEWLINE" (NL) oder "RETURN" (CR) markiert. Im Buch wird die Bezeichnung "ENTER" verwendet.

Andere Steuerzeichen, die jedoch in den einzelnen BASIC-Systemen sehr unterschiedlich gebraucht werden, sind beispielsweise Zeichen zum Löschen eines fehlerhaft eingegebenen Zeichens (DEL-Taste), Zeichen für Umschaltung Groß-/Kleinbuchstaben (SHIFT-Taste), Zeichen für den Abbruch eines laufenden Programms (BREAK-Taste) sowie das Zeichen für die Erweiterung des Steuerzeichenvorrates (CTRL-Taste). Letzteres Zeichen dient dazu, um alle die Steuerzeichen zu erzeugen, für die nicht extra eine Taste auf der Tastatur enthalten ist (s. Anhang A).

2.2. Kommandos und Programmanweisungen

Folgende Arten von Informationen kann der Nutzer auf der Kommandoebene des BASIC-Interpreters über die Tastatur eingeben:

Kommandos

Kommandos steuern die vom Computer durchzuführenden Verarbeitungsvorgänge im Zusammenhang mit der BASIC-Programmerstellung und Programmausführung. Beispiele dafür sind das Auslisten eines abgespeicherten Programms (LIST-Kommando), der Start eines BASIC-Programms (RUN-Kommando) oder das Löschen eines abgespeicherten Programms (NEW-Kommando). Alle Kommandos werden nach ihrer Eingabe vom Computer sofort ausgeführt. Der gesamte Vorrat der Kommandos wird als Interpreter-*Steuersprache* bezeichnet. Die Steuersprache der einzelnen BASIC-Systeme ist sehr unterschiedlich und unterliegt bisher keinem Standard. Die wichtigsten Kommandos sind jedoch relativ einheitlich aufgebaut. Sie werden in den nächsten Abschnitten eingehender behandelt.

BASIC-Programmanweisungen

Das sind Anweisungen aus der Sprache BASIC, denen je eine Zeilennummer vorangestellt ist. Programmanweisungen werden nach deren Eingabe zunächst nur abgespeichert und bilden in ihrer Gesamtheit ein BASIC-Programm. Erst nach Eingabe eines Startkommandos (Kommando RUN) kommen sie zur Ausführung. Während des Laufs eines BASIC-Programms nimmt der BASIC-Interpreter die sogenannte *Programmebene* ein. Nach Beendigung der Ausführung eines Programms wird automatisch die Programmebene verlassen und zur *Kommandoebene* zurückgekehrt

(Bild 2.3). Den Prozeß der Programmerstellung und -ausführung in der hier beschriebenen Form bezeichnet man als *Programmbetrieb* oder *Indirektbetrieb.*

BASIC-Sofortanweisungen

BASIC-Sofortanweisungen sind *nicht* numerierte Anweisungen aus dem Sprachvorrat von BASIC. Sie werden nicht abgespeichert, sondern nach deren Eingabe vom Computer sofort ausgeführt. Mit Hilfe dieser Anweisungen lassen sich z.B. "Sofort"-Rechnungen durchführen oder im Computer gespeicherte Werte von Variablen ausgeben. Kleinere Berechnungen sind in einfacher Art und Weise durchführbar, ohne erst ein Programm aufstellen zu müssen. Eine solche Arbeitsweise wird als *Direktbetrieb* bezeichnet. Weil man diese Betriebsweise mit der eines (nicht programmierbaren) Taschenrechners vergleichen kann, ist dafür auch die Bezeichnung Taschenrechnermodus gebräuchlich. Manche BASIC-Systeme lassen die Verwendung von Sofortanweisungen jedoch nicht zu. Damit ist dann auch kein Direktbetrieb verfügbar.

Alle drei Informationsarten werden dem Interpreter zeilenweise übergeben, wobei jede Zeile mit einem Zeilenendezeichen abzuschließen ist. Es kann beliebig zwischen Kommando-, Programmanweisungs- und Sofortanweisungs-Eingabe gewechselt werden.

Für die praktische Nutzung eines Computers hat der Programmbetrieb weitaus größere Bedeutung als der Direktbetrieb. Der Direktbetrieb ist andererseits beim Programmtest vorteilhaft anwendbar (s. Abschn. 11.). Deshalb wird im folgenden das Arbeiten im Direktbetrieb anhand von Beispielen kurz beschrieben.

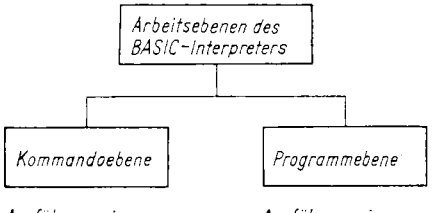


Bild 2.3. Arbeitszustände eines BASIC-Interpreters

2.3. Direktbetrieb

Der BASIC-Interpreter muß sich bei dieser Betriebsweise auf der Kommandoebene befinden (Bild 2.4). Zur Demonstration werden die bereits eingeführte PRINT- und LET-Anweisung benutzt.

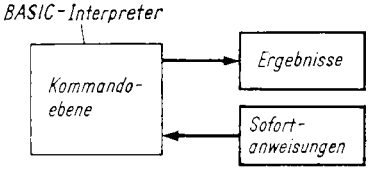


Bild 2.4. Rechnen auf Kommandoebene

Beispiel:
Die Anweisung PRINT 5+9 wird eingegeben und mit "ENTER" abgeschlossen.

```
PRINT 5+9 [ENTER]
14
OK
```

Der Computer gibt das Ergebnis 14 aus und quittiert mit "OK" ("Fertig").

Beispiel:
Für den Radius $r=2{,}5$ ist die Kreisfläche $A=\pi \cdot r^2$ zu ermitteln.

```
PRINT 3.14159 *2.5 *2.5 [ENTER]
19.6349
OK
```

Beispiel:
Es sind die Einzelflächen und die Gesamtfläche von zwei Dreiecken gemäß Bild 1.6 zu berechnen.

```
LET A1=4 *9.5/2 [ENTER]
OK
```

```
LET A2=2.5*11/2 ENTER
OK
LET A=A1+A2 ENTER
OK
PRINT A1,A2,A ENTER
19      13.75      32.75
OK
```

Zunächst wird die Fläche des ersten Dreiecks berechnet und in der Variablen A1 (ein Speicher mit dem Namen A1) zwischengespeichert. Gleiches wird mit dem zweiten Dreieck vorgenommen und das Ergebnis der Variablen A2 zugeordnet. Anschließend wird die Gesamtfläche berechnet und danach die Ergebnisse ausgegeben.
Die aufgezeigten Beispiele machen deutlich, daß man mit Hilfe des Direktbetriebes eine große Zahl von einfachen Aufgaben lösen kann. Es ist jedoch zu beachten, daß die eingegebenen Anweisungen (ohne Zeilennummern) nach deren Ausführung wieder "vergessen" sind. Sie werden vom Computer nicht abgespeichert. Eine Programmierung ist auf diese Weise also nicht möglich. Dazu muß mit Programmanweisungen und im Programmbetrieb gearbeitet werden. In den weiteren Ausführungen wird darauf näher eingegangen.

2.4. Programmeingabe

Die BASIC-Programmierung erfolgt in zwei Etappen: Programmerstellung und Programmausführung (Bild 2.5). Die Programmerstellung entspricht der Eingabe von Programmzeilen. Der Start der Programmausführung erfolgt durch das Startkommando. Während der Programmausführung werden die Eingabedaten über die Tastatur eingegeben und die berechneten Ergebnisse über den Bildschirm ausgegeben. Ist ein Programm ausgeführt, teilt dies der Computer durch eine "OK"-Fertigmeldung mit. Im folgenden sind diese Schritte ausführlicher dargestellt.
Die Eingabe der numerierten Programmzeilen eines BASIC-Programms kann prinzipiell in beliebiger Reihenfolge geschehen. Der Computer ordnet die Programmzeilen immer nach aufsteigenden Zeilennummern. Es ist üblich, die Zeilennummern mit Zehner-

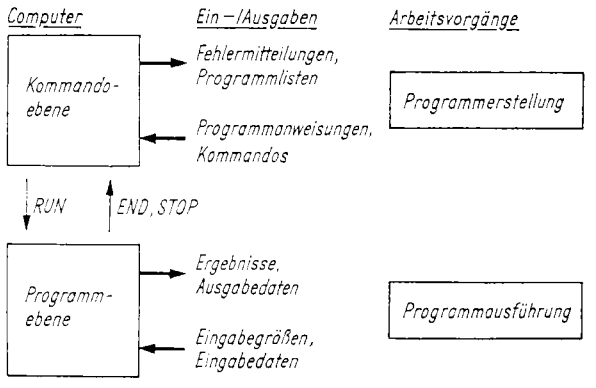


Bild 2.5. Arbeitsvorgänge in der BASIC-Programmierung

schritten zu notieren. Dadurch lassen sich bei Änderungen und Korrekturen eines Programms in einfacher Weise noch zusätzliche Anweisungen einfügen. Es sei noch einmal vermerkt, daß jede eingegebene Programmzeile mit einem Zeilenendezeichen ("ENTER") abzuschließen ist. Erst dann wird sie vom BASIC-Interpreter übernommen. In den weiteren Ausführungen wird auf diesen Sachverhalt nicht mehr hingewiesen. Auch in den Programmbeispielen (außer im folgenden) wird dieses Zeichen nicht mit dargestellt.
Die maximale Anzahl der Programmanweisungen, die man in einen Computer eingeben kann, hängt von dessen Speichergröße ab. Ist die maximale Speichergrenze erreicht, wird dies dem Benutzer des Computers über eine entsprechende Bildschirmausschrift mitgeteilt. Liegt ein BASIC-Programm abgespeichert vor, dann ist die Programmzeile mit der kleinsten Zeilennummer der Programmbeginn und die Zeile mit der größten Nummer in der Regel das Programmende.
An einem einfachen Beispiel soll die Programmeingabe erläutert werden.

Beispiel:

Es wird ein Programm (s. 1.4.) eingegeben, das insgesamt drei Zeilen umfaßt:

Eingabe 1. Anweisung:

```
20 PRINT 5+9 ENTER
```

Der Computer speichert die Programmzeile ab und erwartet weitere Eingaben.
Eingabe der 2. Anweisung:

```
10 PRINT "BEGINN" ENTER
```

Diese Programmzeile wird vor der zuerst eingegebenen Zeile eingeordnet, weil sie die Zeilennummer 10 besitzt.
Eingabe der 3. Anweisung:

```
30 END ENTER
```

Diese Anweisung wird als letzte eingeordnet. Damit steht im Hauptspeicher des Computers folgendes Programm:

```
10 PRINT "BEGINN"
20 PRINT 5+9
30 END
```

Natürlich hätte man die einzelnen Programmzeilen auch in der normalen Reihenfolge oder in einer beliebigen anderen eingeben können, es wäre immer das gleiche Programm entstanden.

2.5. Ausgabe des Programms als Liste [LIST]

Ein eingegebenes und im Hauptspeicher des Computers abgelegtes Programm kann durch das LIST-Kommando zur Kontrolle wieder auf dem Bildschirm ausgegeben werden. Es hat die Form:

```
LIST
```

Bei Eingabe dieses Kommandos antwortet der Computer mit der Ausgabe des gesamten im Hauptspeicher stehenden Programms oder Teilprogramms. Soll nur eine einzelne Zeile ausgegeben wer-

den, ist das LIST-Kommando durch die entsprechende Zeilennummer zu ergänzen. Das Kommando hat dann die Form:

LIST zeilennummer

Als Beispiel möge das in 2.4. eingegebene Programm dienen. Nach Ausführung des Kommandos LIST erscheint die Bildschirmanzeige:

```
LIST
10  PRINT "BEGINN"
20  PRINT 5+9
30  END
OK
```

Soll nur die Zeile 20 ausgegeben werden, realisiert diese Aufgabe das LIST-Kommando mit nachgestellter Zeilennummer:

```
LIST 20
20  PRINT 5+9
OK
```

2.6. Programmausführung [RUN]

Ist ein Programm oder Teilprogramm in den Hauptspeicher des Computers gespeichert, kann dieses durch folgendes Kommando gestartet werden:

RUN

Es bewirkt, daß der BASIC-Interpreter von der Kommandoebene in die Programmebene übergeht.
Die Programmausführung beginnt mit der Programmzeile, die die kleinste Zeilennummer enthält, und endet mit der END- oder STOP-Anweisung (s. Abschnitt 6.).
Als Beispiel dient das bereits im vorhergehenden Abschnitt benutzte Programm. Mit Hilfe des RUN-Kommandos wird es gestartet, wobei sich nachstehende Anzeige auf dem Bildschirm ergibt:

```
RUN
BEGINN
14
OK
```

Ein BASIC-Programm steht auch nach der Ausführung unverändert im Hauptspeicher zur Verfügung, so daß es prinzipiell beliebig oft mit Hilfe des RUN-Kommandos gestartet und abgearbeitet werden kann.
Während der Programmausführung prüft der Computer, ob Programmierfehler vorliegen. Ist dies der Fall, wird der Programmlauf unterbrochen und eine Mitteilung über die Art des Fehlers auf dem Bildschirm angezeigt. Hierbei schaltet der BASIC-Interpreter automatisch von der Programmebene in die Kommandoebene. Wie dann weiter zu verfahren ist, soll anschließend erläutert werden.

2.7. Korrektur eines Programms

Nicht selten besteht die Notwendigkeit, ein gespeichertes Programm durch Ändern, Einfügen oder Streichen von Programmzeilen zu korrigieren oder zu modifizieren.

Einfügen einer Zeile

Für die einzufügende Programmanweisung wählt man eine Zeilennummer, die der gewünschten Position im Programm entspricht. Um den Sachverhalt zu demonstrieren, sei für das bereits mehrfach verwendete Beispielprogramm

```
10 PRINT "BEGINN"
20 PRINT 5+9
30 END
```

angenommen, daß vor der END-Anweisung eine weitere PRINT-Anweisung eingeschoben werden soll. Mit ihr sei die Zeichenfolge "ENDE" auszugeben. Wir wählen die Zeilennummer 25 und geben ein:

```
25 PRINT "ENDE"
```

Zur Kontrolle wird das Programmbeispiel noch einmal als Liste ausgegeben und dann ausgeführt. Dabei ergibt sich nachstehendes Protokoll auf dem Bildschirm:

```
LIST
10  PRINT "BEGINN"
20  PRINT 5+9
25  PRINT "ENDE"
30  END
RUN
BEGINN
14
ENDE
OK
```

Streichen einer Zeile

Eine Programmzeile wird gestrichen, wenn man die betreffende Zeilennummer ohne weitere Zeichen, nur durch das Zeilenendezeichen abgeschlossen, eingibt.

Korrigieren einer Zeile

Die Korrektur einer Zeile erfolgt, indem man die neue, korrigierte Zeile – mit der "alten" Zeilennummer versehen – eingibt. Dabei wird die "alte" durch die "neue" Zeile überschrieben.

Beispiel:

Im bisher benutzten Beispielprogramm sei angenommen, daß die Programmzeile 25 zu korrigieren ist, und zwar soll der auszugebende Text von "ENDE" in "ENDE DES PROGRAMMS" geändert werden. Zur Korrektur dient die Eingabe:

```
25  PRINT "ENDE DES PROGRAMMS"
```

Die bisherige Zeile 25 wird durch die eingegebene neue Programmzeile ersetzt.

Zur Kontrolle wird das Programm noch einmal als Liste ausgegeben:

```
LIST
10  PRINT "BEGINN"
20  PRINT 5+9
```

```
25 PRINT "ENDE DES PROGRAMMS"
30 END
```

2.8. Löschen eines Programms [NEW]

Vor der Eingabe eines neuen Programms muß das im Hauptspeicher befindliche gelöscht werden. Hierfür dient das Kommando

NEW

Es bewirkt das Rücksetzen des BASIC-Systems und das Löschen aller gespeicherten Programmzeilen.
Außerdem sind alle bisher vorliegenden Variablenwerte (s. Abschn. 3.) nach NEW nicht mehr verfügbar.

Aufstellung der Kommandos

Zusammenfassend sind noch einmal alle behandelten Kommandos aufgeführt:
LIST – Ausgabe des Programms als Liste auf Bildschirm
RUN – Start eines Programms
NEW – Löschen eines Programms
Bei Computern mit zusätzlichem Drucker hat man oft auch folgendes Kommando zur Verfügung:
LLIST – Ausgabe des Programms als Liste auf Drucker
Die Verwendung dieses Kommandos verläuft in Analogie zum LIST-Kommando.

3. BASIC-Sprachelemente und Programmaufbau

3.1. Zeichen

Eine Programmiersprache wird durch deren Syntax und Semantik festgelegt. Die Syntax beinhaltet die grammatikalischen Regeln, nach denen die Sprachkonstruktionen aufzubauen sind. Die Semantik dagegen beschreibt den inhaltlichen Aspekt, also die Bedeutung der jeweiligen Sprachkonstruktion. In den weiteren Aus-

führungen werden die syntaktischen Regeln der Sprache BASIC schrittweise eingeführt und entsprechende Anwendungsbeispiele aufgezeigt. Im Anhang A ist außerdem die Syntax für Minimal-BASIC in einer Übersicht angegeben. Eine semantische Beschreibung der Sprache wird im Verlaufe der Behandlung der einzelnen BASIC-Anweisungen vorgenommen.
Allen Programmiersprachen einschließlich BASIC ist gemeinsam, daß eine gegebene Menge von Grundzeichen – der *Zeichensatz* – zum Aufbau der Sprachkonstruktionen dient.
In BASIC sind folgende Zeichen vereinbart:

– *Buchstaben*	A, B, C, ..., Z
– *Ziffern*	0, 1, 2, ..., 9
– *Sonderzeichen*	+ − * / ^ , . : ; ? ! " % & ( ) = < > # ○ ' _ Leerzeichen

Der Zeichensatz ist Grundlage für die Darstellung der Symbole der Programmiersprache wie Wortsymbole, Spezialsymbole, Zahlen, Zeichenketten, Namen und Schriftsymbole. Alle Zeichen werden im Computer als binäre Informationen (Binärworte) dargestellt. Die tabellarische Aufstellung der Zeichen und der entsprechenden Binärworte ist der sogenannte *Zeichencode*. Er ist international standardisiert [1] und im Anhang A dargestellt.

3.2. Wortsymbole

BASIC-Anweisungen enthalten als Wortsymbol ein sogenanntes Schlüsselwort. In ihm ist die entsprechende Steuerfunktion für den Computer verschlüsselt. Es gibt darüber Auskunft, welcher spezielle Verarbeitungsvorgang ausgeführt werden soll, zeigt also an, "WAS zu tun" ist. Ein Schlüsselwort ist eine Zeichenfolge aus Buchstaben. Wichtige Schlüsselworte in BASIC sind beispielsweise:

PRINT – "Drucke oder gebe über Bildschirm aus"
LET – "Operation ausführen"
INPUT – "Eingabe Daten"
STOP – "Programmhalt"

Bei manchen BASIC-Systemen können in den Anweisungen bestimmte Schlüsselworte aus Gründen der Schreibersparnis verkürzt geschrieben werden oder ganz entfallen. Eine alphabetische Aufstellung aller in Minimal-BASIC verwendeten Schlüsselworte ist im Anhang A enthalten.

3.3. Spezialsymbole

In BASIC gibt es Zeichen und Zeichenkombinationen, die, ähnlich wie Schlüsselworte, eine besondere Bedeutung haben. Diese Spezialsymbole werden ebenfalls aus dem Zeichensatz gebildet, wobei hierzu vor allem die Sonderzeichen Anwendung finden. Einige Spezialsymbole sind:

+	Additionsoperator
>=	Vergleichsoperator (größer gleich)
,	Trennzeichen
.	Dezimalpunkt
"	Zeichenketten-Begrenzungszeichen
E	Zeichen für Dezimalexponent-Darstellung

Spezialsymbole werden benötigt, um Zahlen, Textdarstellungen, Rechen- und Textausdrücke sowie Operationen zu beschreiben. Die vollständige Aufstellung der Spezialsymbole für BASIC ist im Anhang A enthalten.

3.4. Konstanten und Variablen

Im Computer laufen solche Prozesse ab, durch die die zu verarbeitenden Größen, die Daten, transportiert, verknüpft und gespeichert werden. Man unterscheidet
- numerische Daten (Zahlen) und
- nichtnumerische Daten (Zeichenketten).

Daten können in einem Programm als Konstanten (feste Größen) und Variablen (veränderliche Größen) auftreten. Konstanten behalten während der gesamten Programmausführung ihren Wert. Variablen können im Verlaufe der Programmausführung verschiedene Werte annehmen. Sie werden in einem Programm durch

Namen bezeichnet. Am Beginn eines Programms, noch bevor einer Variablen ein Wert zugeordnet wird, ist ihr Wert undefiniert. Bei manchen BASIC-Systemen erhalten alle Variablen bei Programmstart einen vorgegebenen (implementationsabhängigen) Wert (meist Null bzw. Leer-Zeichenkette). Variable muß man sich in ihrer technischen Realisierung im Computer als einen "Speicherplatz" vorstellen, der mit dem entsprechenden Variablennamen versehen ist. Der Inhalt des Speicherplatzes selbst ist der Wert der Variablen. Dieser läßt sich durch entsprechende Computeranweisungen beliebig ändern. Die Variablennamen sind vom Nutzer entsprechend einzuhaltender Regeln frei wählbar. Im folgenden werden die Regeln für die Bildung der Konstanten und Variablen der einzelnen Datenarten angegeben.

Zahlen

Numerische Daten sind Elemente aus der Menge der reellen Zahlen (genauer einer Untermenge der reellen Zahlen). Für diese Art von Daten existieren in BASIC drei Darstellungsmöglichkeiten:

- ganze Dezimalzahlen (Ganzzahldarstellung)
- gebrochene Dezimalzahlen in Normaldarstellung (Festkommadarstellung)
- gebrochene Dezimalzahlen in Exponentialdarstellung (Gleitkommadarstellung)

In dieser Form können numerische Daten in den Computer eingegeben werden. Die gleichen Darstellungen werden vom Computer für die Ausgabe benutzt. Wichtig ist, daß in einer gebrochenen Dezimalzahl anstelle des üblichen Kommas ein Dezimalpunkt geschrieben werden muß. Folgende Beispiele sollen die Darstellungsformen veranschaulichen.

Ganze Zahlen:
50 7200 −223
Festkommazahlen:
7.1 0.05 −1201.373
Gleitkommazahlen:
3E−12 ($\triangleq 3 \cdot 10^{-12}$)
2.133E+07 ($\triangleq 2{,}133 \cdot 10^{7}$)
−7.115E−16 ($\triangleq -7{,}115 \cdot 10^{-16}$).

In allen Zahlendarstellungen kann das positive Vorzeichen bei der Eingabe entfallen. Es wird auch bei Computerausgaben im all-

gemeinen nicht geschrieben, sondern durch ein Leerzeichen ersetzt. Die Gleitkommadarstellung entspricht der in der Mathematik und Physik oft benutzten Schreibweise $K \cdot 10^n$ von Zahlen mit Dezimalexponent. Anstelle von "mal 10 hoch" wird "E" notiert. Gleitkommazahlen sind besonders bei sehr großen oder sehr kleinen Zahlen vorteilhaft zu verwenden. BASIC wechselt bei der Ausgabe von der Normaldarstellung der Dezimalzahlen zur Gleitkommadarstellung und umgekehrt immer dann, wenn es notwendig ist. Bei Ausgabe von sehr kleinen oder sehr großen Werten wird automatisch die Gleitkommadarstellung benutzt. Ansonsten gibt der Computer ganze Zahlen oder Dezimalzahlen mit Dezimalpunkt aus.

Im Computer lassen sich aus technischen Gründen Zahlen nicht beliebig groß und nicht beliebig genau darstellen. Im allgemeinen wird in BASIC mit einer Genauigkeit von 6 bis 7 Dezimalstellen gerechnet. Bei den meisten BASIC-Systemen ist der Betrag der größten darstellbaren Zahl 10^{+38} und der der kleinsten (von Null verschiedenen) Zahl 10^{-38}.

Die *numerischen Konstanten* werden in einem BASIC-Programm in den erwähnten Zahlendarstellungen notiert. Der entsprechende Wert wird als ganze Zahl, als Festkommazahl oder Gleitkommazahl dargestellt.

Die *numerischen Variablen* werden durch Namen bezeichnet, die aus einem oder zwei Zeichen bestehen. Das erste Zeichen muß ein Buchstabe, das zweite eine Ziffer sein. Folgende Beispiele zeigen richtige und fehlerhafte numerische Konstanten und Variablennamen.

Konstanten		Variablen (in Minimal-BASIC)	
zulässig	nicht zulässig	zulässig	nicht zulässig
30	3 0	A	1A
−120.50	−120,50	B	AB
2.5	2 .5	C	A?
6E−12	6E −12	A1	A 1
1.235E+06	1.235+E06	Z9	A11

In vielen BASIC-Systemen sind auch Variablennamen mit einer größeren Anzahl von Zeichen zugelassen. Es darf dann allerdings ein Variablenname nicht mit einem BASIC-Schlüsselwort (reservierte Worte) übereinstimmen.

Zeichenketten

Nichtnumerische Daten sind Elemente aus der Menge der darstellbaren Zeichenketten. Sie werden auch Strings (engl.: string – Kette) genannt. Zeichenketten bestehen aus einer Folge von Einzelzeichen aus dem gesamten BASIC-Zeichensatz. Dazu gehören auch alle Sonderzeichen. Die zulässige Länge einer Zeichenkette kann verschieden sein, jedoch im allgemeinen nicht länger als eine Eingabezeile. Verschiedene Zeichenketten (z.B. unterschiedliche Länge oder Zeichenbelegung) entsprechen verschiedenen Werten. Folgende Beispiele sollen den Aufbau von Zeichenketten zeigen:

```
ABCD
ERGEBNIS=
! ! ! ! ! ! ! !
AUSGABE ?
```

Die erste Zeichenkette im Beispiel umfaßt 4 Zeichen, während die drei folgenden genau 9 Zeichen enthalten. Auch das Leerzeichen zwischen "AUSGABE" und "?" sowie das Fragezeichen selbst gehören mit zur Zeichenkette des letzten Beispiels. Zeichenketten finden vor allem breite Anwendung in der Textverarbeitung.
Eine STRING-*Konstante* wird in einem Programm notiert, indem die entsprechende Zeichenkette in hochgestellte Anführungszeichen (") als Begrenzungszeichen eingeschlossen wird.
Die Namen der STRING-*Variablen* werden aus zwei Zeichen gebildet, wobei das erste ein Buchstabe und das zweite (bzw. das letzte bei längerem Namen in erweiterten BASIC-Versionen) das Währungszeichen (□ oder $) sein muß. Folgende Beispiele veranschaulichen STRING-Konstanten und -Variablen.

Konstanten	Variablen (in Minimal-BASIC)	
	zulässig	nicht zulässig
"ABCD"	A□	B
"ERGEBNIS="	B□	1□
"AUSGABE ?"	Z□	B1□

Felder

Die bisher beschriebenen Arten von Daten sind sogenannte einfache Datentypen oder Grundtypen. Ihre Elemente werden als kleinste unteilbare Einheiten angesehen. In BASIC gibt es jedoch auch einen strukturierten Datentyp, die Felder (ARRAYs). Sie sind aus den genannten einfachen Datentypen aufgebaut. Mit Feldern kann man sehr effektiv programmieren und besonders größere Datenmengen handhaben. Sie werden jedoch erst an späterer Stelle behandelt.

3.5. Operationen, Ausdrücke

Die Verknüpfung von Daten (Operanden) über Operationszeichen (Operatoren) erfolgt in sogenannten Ausdrücken. Man unterscheidet:

- numerische Ausdrücke
- Zeichenkettenausdrücke
- Vergleichsausdrücke

Als Operanden können sowohl Konstanten, Variablen und Funktionsaufrufe als auch wieder Ausdrücke auftreten.

Numerische Ausdrücke

Um numerische Daten miteinander zu verknüpfen, sind in BASIC eine Reihe von Operationen definiert:

Operator	Operation
+	Addition
−	Subtraktion
*	Multiplikation
/	Division
^	Potenzieren

Numerische Ausdrücke werden in einer von der normalen mathematischen Darstellung abweichenden Form notiert, da sämtliche Operanden und Operationszeichen zeilenorientiert anzuordnen sind. Wenn notwendig, können Klammern benutzt werden. Einige Beispiele sollen die Schreibweise in BASIC verdeutlichen.

Normaldarstellung	Darstellung in BASIC
$\frac{3,6+2,1}{0,117}$	(3.6+2.1)/0.117
$\frac{A}{B+C}$	A/(B+C)
N^3	N^3
$\frac{Z1-Z2}{X1+X2}$	(Z1−Z2)/(X1+X2)
3MN	3*M*N

Ein numerischer Ausdruck wird in folgender Reihenfolge vom Computer abgearbeitet (Rangfolge):
- Klammerberechnung (höchster Rang)
- Funktionsaufruf (s. Abschn. 5.)
- Potenzieren
- Multiplikation/Division
- Addition/Subtraktion (niedrigster Rang)

Ansonsten wird ein Ausdruck immer in der Reihenfolge von links nach rechts berechnet.

Zeichenkettenausdrücke

Zur Verknüpfung von nicht numerischen Daten existiert in BASIC eine Verkettungsoperation. Sie entspricht einem Aneinanderfügen von Zeichenketten-Operanden:

Operator	Operation
+	Verkettung

Bei manchen BASIC-Systemen wird anstelle des "+"-Zeichens auch ein anderes Zeichen benutzt.
Die Beispiele zeigen einige Möglichkeiten der Notation von Zeichenkettenausdrücken.

Zeichenketten-ausdruck	Erläuterung
"ROT"+"BLAU"	Verketten von Zeichenkettenkonstanten

Zeichenketten-ausdruck	Erläuterung
A$+B$+C	Verketten von Zeichenkettenvariablen
X$+"?"	Verketten von Variable und Konstante

Vergleichsausdrücke

Der Vergleich von Operanden erfolgt in Vergleichsausdrücken. Er kann nur zwischen im Datentyp gleichen (verträglichen) Operanden als
- numerischer Vergleich oder
- Zeichenkettenvergleich

vorgenommen werden. Für Vergleiche sind die folgenden Operationen definiert:

Operator	Operation
=	gleich
<>	ungleich
>	größer
<	kleiner
>=	größer oder gleich
<=	kleiner oder gleich

Das Ergebnis einer Vergleichsoperation ist eine logische Größe, die nur die Werte "wahr" oder "falsch" annehmen kann. Auch hier sind einige Beispiele zur Verdeutlichung angegeben.

Vergleichsausdruck	Erläuterung
X=5	numerischer Vergleich
A>=B+10	numerischer Vergleich
Z$="EIN"	Zeichenkettenvergleich
A$<=B$	Zeichenkettenvergleich

3.6. Anweisungen

Eine wichtige Sprachkonstruktion in BASIC ist die *Anweisung*. Sie bewirkt im Computer eine bestimmte in sich abgeschlossene

Aktion (Verarbeitungsvorgang). BASIC-Anweisungen werden aus den bisher behandelten Sprachelementen (Symbolen) nach vorgegebenen Regeln (Syntax) aufgebaut. Jede Anweisung beginnt mit einem Schlüsselwort, dem weitere BASIC-Symbole folgen. Jede BASIC-Anweisung hat ihren speziellen Aufbau (s. Anhang A). Im weiteren wird zur syntaktischen Beschreibung der einzelnen Anweisungen folgende allgemeine Darstellung benutzt: Die Schlüsselworte der Anweisungen werden immer mit Großbuchstaben geschrieben. Es sind Wortsymbole, also Sprachelemente von BASIC. Alle *variablen* Elemente einer Anweisung, d.h. alle Teile, die erst bei der konkreten Anwendung der Anweisung eine spezielle Belegung erhalten, werden mit Kleinbuchstaben notiert. Folgendes Beispiel der PRINT-Anweisung zeigt dies deutlich:

PRINT ausdruck

Die Zeichenfolge "PRINT" ist das Schlüsselwort, der variable Teil der Anweisung ist "ausdruck". Für "ausdruck" läßt sich wahlweise eine Konstante, eine Variable oder allgemein ein Rechen- oder Zeichenkettenausdruck einsetzen. Für die allgemeine Darstellung werden außerdem geschweifte Klammern ({, }) benutzt. Sie dienen der Aufzählung von Möglichkeiten, von denen genau eine ausgewählt werden muß.

3.7. Aufbau eines BASIC-Programms

Ein BASIC-Programm besteht aus einer Folge von numerierten Programmzeilen. Die Zeilennummern legen die Reihenfolge für die Ausführung der Anweisungen fest. Diese wird bei der Anweisung mit der niedrigsten Zeilennummer begonnen und mit der nächstgrößeren fortgesetzt. Spezielle Anweisungen erlauben es aber, die normale Ausführungsfolge zu verlassen. Jede BASIC-Programmzeile hat folgende Struktur:

zeilennummer anweisung zeilenendezeichen

Die Zeilennummer ist eine in der Regel maximal vier- oder fünfstellige Dezimalzahl. Als kleinste Zeilennummer ist im allgemeinen die "1" zulässig. Üblicherweise werden die Zeilennummern

bei der Notierung eines Programms jedoch nicht in der Reihenfolge der natürlichen Zahlen, also 1, 2, 3, ... gewählt, sondern im Fünfer- oder Zehnerschritt. Dadurch lassen sich auf einfache Weise Programmkorrekturen durch Einfügen zusätzlicher Programmzeilen realisieren. Das Zeilenendezeichen schließt jede Programmzeile ab. Bei der Notierung von BASIC-Programmen wird es jedoch nicht angegeben. Manche BASIC-Systeme erlauben es, auch mehrere Anweisungen in einer Programmzeile zu schreiben.
Im Normalfall wird ein BASIC-Programm mit einer END-Anweisung abgeschlossen. Sie kann jedoch bei vielen BASIC-Systemen weggelassen werden. Dann beendet der Computer nach der Ausführung der letzten Anweisung das Programm.

4. Programmieren mit einfachen Anweisungen

4.1. Ergibtanweisung [LET]

Im folgenden werden die BASIC-Anweisungen in systematischer Weise eingeführt und anhand von Beispielen veranschaulicht. Zunächst sind relativ einfach aufgebaute Anweisungen Gegenstand der Betrachtungen.
Die Ergibtanweisung (LET-Anweisung) ist eine der wichtigsten Anweisungen in BASIC. Sie hat folgende allgemeine Form:

LET variable=ausdruck

Die Wirkung der LET-Anweisung besteht darin, daß zunächst der Ausdruck berechnet und dann der errechnete Wert der Variablen zugewiesen wird. Der Ausdruck kann auch nur aus einer Konstanten oder Variablen bestehen. Eine Zuweisung darf nur zwischen gleichen Arten von Daten vorgenommen werden. Ist der Ausdruck beispielsweise ein numerischer Ausdruck, dann muß die Variable auch eine numerische Variable sein. Entsprechendes gilt für Zeichenkettengrößen.
Die nachfolgenden Beispiele sollen die Wirkung der LET-Anweisung zunächst bei numerischen Größen veranschaulichen.

Beispiel:

```
10 LET A=3.5
20 LET C=A
30 LET B=9+C
40 LET E=B+7/2-30
```

Erläuterungen:

Zeile 10: Der Variablen A wird der Wert 3.5 zugeordnet.
Zeile 20: Die Variable C erhält den Wert der Variablen A (= 3.5).
Zeile 30: B bekommt den Wert des Rechenausdrucks 9+C (=12.5) zugewiesen.
Zeile 40: E erhält den Wert −14.

Man beachte, daß numerische Ausdrücke nach den bereits erläuterten Vorrangregeln berechnet werden.

Beispiel:

Zu berechnen sei in Programmzeile 120 die Gleichung

$$y = \frac{a(z_1 - z_2)}{b(x_1 + x_2)^2} + 5{,}2.$$

Als Lösung ergibt sich folgende Programmzeile:

```
120 LET Y=A*(Z1-Z2)/(B*(X1+X2)^2)+5.2
```

Beim Verarbeiten von Zeichenketten können diese mit Hilfe der LET-Anweisung aneinandergefügt (verkettet) werden. Als Operationszeichen (Operator) fungiert das "+"-Zeichen. Sowohl Zeichenkettenkonstanten als auch Zeichenkettenvariablen lassen sich verketten.

Beispiel:

```
10 LET A$="ROT"
20 LET B$="BLAU"
30 LET C$=A$
40 LET D$=B$+"GELB"
50 LET E$=A$+B$
```

Erläuterung:

Zeile 10: Der Zeichenkettenvariablen A$ wird die Zeichenkettenkonstante "ROT" zugeordnet.
Zeile 20: Der Variablen B$ wird "BLAU" zugewiesen.
Zeile 30: Der Variablen C$ wird der Wert von der Variablen A$ (="ROT") zugeordnet.
Zeile 40: Der Variablen D$ wird die Verkettung des Wertes von B$ und "GELB" zugeordnet. Damit erhält D$ den Wert "BLAUGELB".
Zeile 50: E$ wird die Verkettung von A$ und B$ zugewiesen. E$ erhält den Wert "ROTBLAU".

4.2. Ausgabeanweisung [PRINT]

In BASIC sorgt die Ausgabeanweisung (PRINT-Anweisung) für die Ausgabe aller Größen wie Zahlen, Zeichen, Zeichenketten auf dem Bildschirm des Computers. Die einfachste Form der PRINT-Anweisung ist:

PRINT ausdruck

Der Ausdruck kann im einfachsten Fall nur eine Konstante oder eine Variable sein. Der Wert des Ausdrucks wird berechnet und auf dem Bildschirm ausgegeben. Nach Abschluß der Ausgabe wird auf eine neue Bildschirmzeile übergegangen.

Beispiel:

```
10 LET A=15
20 LET Z$="ZEICHEN"
30 PRINT "ERGEBNIS"
40 PRINT A
50 PRINT Z$
60 PRINT A+3*7.5-40
```

Die Wirkung der Ausgabeanweisung für verschiedene Ausgabegrößen wird anhand des Ausgabeprotokolls veranschaulicht. Es zeigt die auf dem Bildschirm ausgegebenen Größen nach Ausführung der entsprechenden Anweisungen.

Ausgabeprotokoll:

```
 ERGEBNIS
   15
 ZEICHEN
 -2.5
```

Das positive Vorzeichen wird als Leerzeichen ausgegeben.
Mit Hilfe der PRINT-Anweisung können auch mehrere Zahlen bzw. Zeichenketten auf eine Bildschirmzeile ausgegeben werden, wie die folgende allgemeine Darstellung zeigt:

PRINT liste von ausdrücken

Ein Beispiel soll dies verdeutlichen:

```
60 PRINT "WERT=";A,A+B,(A+B)/2
```

Trennzeichen zwischen den Ausdrücken sind Komma oder Semikolon. Ihre Wahl beeinflußt das Ausgabeformat. Die Verwendung des Kommas bewirkt die Ausgabe im Spaltenformat, während bei Semikolon die Ausgabe formatfrei ist. Zur Bildung des Spaltenformats wird jede Bildschirmzeile in mehrere Ausgabebereiche (Zonen) eingeteilt. Die Anzahl der Zonen je Zeile und die je Zone aufnehmbare Anzahl der Zeichen hängt von der Darstellungskapazität einer Bildschirmzeile (in der Praxis 24 bis 80 Zeichen je Zeile) und der entsprechenden BASIC-Version ab. Häufig wird eine Einteilung in 4 oder 5 Zonen zu je 14 oder 15 Zeichenpositio-

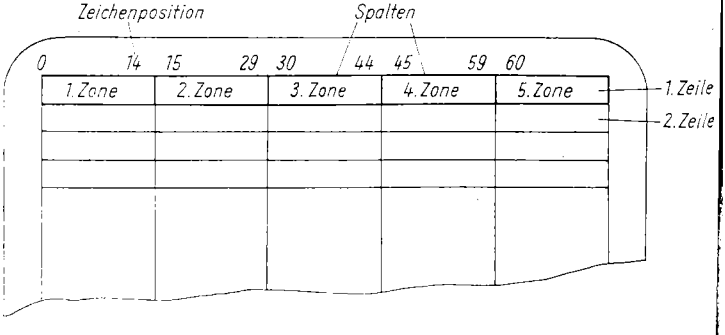


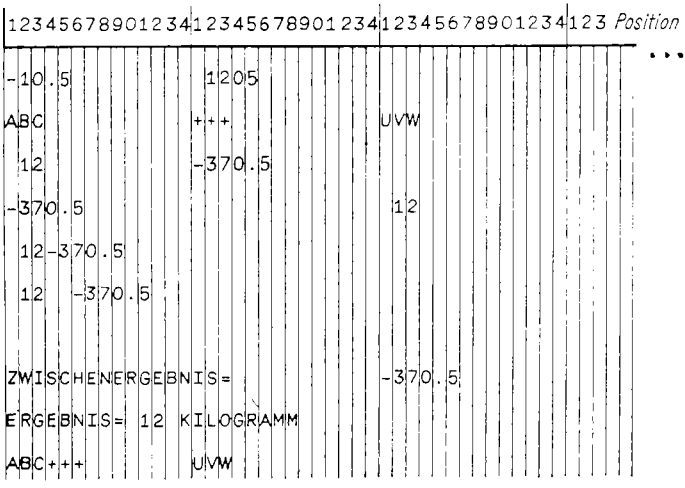
Bild 4.1. Einteilung einer Ausgabezeile in Ausgabezonen

nen vorgenommen (Bild 4.1). Die Ausgabe der einzelnen Werte erfolgt bei Spaltenformat der Reihe nach in diesen Zonen linksbündig zur Zonengrenze.
Es gelten folgende Regelungen für die Ausgabe: Steht ein Komma als Trennzeichen, so wird die Ausgabe des Wertes des nachfolgenden Ausdrucks in der nächsten Zone der gleichen Zeile vorgenommen. Sind alle Zonen einer Zeile bereits belegt und folgen noch weitere Ausgaben, dann wird bei den meisten BASIC-Systemen automatisch auf die nächste Zeile zur Zone 1 übergegangen. Außerdem ist ein Überspringen von Zonen durch das Setzen von unmittelbar aufeinanderfolgenden Kommas möglich.
Wird dagegen ein Semikolon als Trennzeichen verwendet, dann wird der Wert des nachfolgenden Ausdrucks ohne Berücksichtigung von Ausgabezonen unmittelbar an das letzte Zeichen der vorhergehenden Ausgabe angefügt.
Kommas und Semikolons können auch gemischt in einer PRINT-Anweisung als Trennzeichen verwendet werden.
Steht hinter dem letzten Ausdruck in der PRINT-Anweisung kein Trennzeichen, dann wird zum Abschluß der Ausgaben der Übergang auf die nächste Bildschirmzeile ausgeführt.
Die Verwendung der PRINT-Anweisung ohne Ausgabegrößen bewirkt die Ausgabe einer Leerzeile. Man hat damit die Möglichkeit, das Ausgabebild von Daten sinnvoll zu gestalten.
Einige PRINT-Programmanweisungen sollen den Sachverhalt verdeutlichen.

Beispiel:

```
10  PRINT -10.5,1205
20  PRINT "ABC","+++","UVW"
30  LET X=12
40  LET Y=-370.5
50  PRINT X,Y
60  PRINT Y,,X
70  PRINT X;Y
80  PRINT X;" ";Y
90  PRINT
100  PRINT "ZWISCHENERGEBNIS=",Y
110  PRINT "ERGEBNIS=";X;" KILO";
115  PRINT "GRAMM"
120  PRINT "ABC";"+++","UVW"
```

Ausgabeprotokoll (schematisch):



Im Normalfall werden die Ausgaben jeder PRINT-Anweisung auf je eine Zeile ausgegeben. Es ist aber auch möglich, die Ausgaben von zwei oder mehreren PRINT-Anweisungen auf eine einzige Zeile zu positionieren. In diesem Fall muß hinter der letzten Ausgabegröße der entsprechenden Anweisungen ein Trennzeichen (Komma oder Semikolon) stehen. Damit wird verhindert, daß nach Ausführung der betreffenden PRINT-Anweisung der Übergang auf eine neue Zeile vorgenommen wird.

Beispiel:

In einem vorhergehenden Programmteil haben S1 den Wert 3.5 und S2 den Wert 71.25 erhalten. Nun sollen beide Werte ausgegeben werden.

```
120 PRINT "SUMME1=";S1,
130 PRINT "SUMME2=";S2
```

Die Größen werden auf einer einzigen Zeile geschrieben:

```
SUMME1=  3.5    SUMME2=  71.25
```

Mit Hilfe der LET- und der PRINT-Anweisung ist es bereits möglich, kleinere Programme zu schreiben.

Beispiel: Volumen der Kugel

Entsprechend der Beziehung $V = \frac{4\pi \cdot r^3}{3}$ ist das Volumen einer Kugel mit dem Radius $r = 2{,}5$ zu ermitteln.

```
10  LET R=2.5
20  LET V=4*3.141592*R^3/3
30  PRINT "VOLUMEN="
40  PRINT V
50  END
RUN
VOLUMEN=
65.4498
OK
```

In Zeile 10 wird der Radius der Kugel festgelegt, anschließend in Zeile 20 das Volumen berechnet und der Variablen V zugewiesen. Die Größe π ist im Interesse einer möglichst hohen Genauigkeit mit entsprechend vielen Stellen einzugeben. Nach der Ausgabe "VOLUMEN=" folgt in der nächsten Zeile die Ausgabe des Wertes von V.
Veränderungen am Programm:
Sollen die beiden Ausgaben auf einer Ausgabezeile liegen, dann sind sie zweckmäßig in nur einer PRINT-Anweisung zusammenzufassen. Falls die Volumenberechnung für einen anderen Wert des Radius (z.B. $r = 5$) vorgenommen werden soll, wäre außerdem Zeile 10 zu verändern. Beide Korrekturen werden durchgeführt. Das geänderte Programm wird zur Kontrolle aufgelistet und dann ausgeführt.

```
LIST
10  LET R=5
20  LET V=4*3.141592*R^3/3
30  PRINT "VOLUMEN=";V
50  END
RUN
VOLUMEN= 523.598
OK
```

Der fortgeschrittene Leser wird bemerken, daß sich das Ergebnis des vorliegenden Programms durch eine einzige Zeile erreichen läßt:

```
30 PRINT "VOLUMEN=";4*3.141592*5^3/3
```

Der Nachteil des Kugelprogramms, daß jeweils nur die Berechnung für einen Wert des Radius möglich ist, kann erst bei Verwendung von Eingabeanweisungen umgangen werden.

4.3. Eingabeanweisung [INPUT]

Mit Hilfe der Eingabe- oder INPUT-Anweisung lassen sich während der Ausführung eines Programms Daten über die Tastatur eingeben. Die einfachste Form ist:

INPUT variable

Die in der Anweisung angegebene Variable übernimmt den Eingabewert. Dieser muß in der Datenart mit der Variablen übereinstimmen. Der Ablauf einer Eingabe ist folgender: Wird während der Ausführung des Programms eine INPUT-Anweisung erreicht, gibt der Computer ein Fragezeichen auf dem Bildschirm aus (bei manchen BASIC-Systemen auch ein anderes Zeichen) und wartet dann auf die Eingabe. Der Benutzer muß einen Wert eingeben und mit dem Zeilenendezeichen abschließen. Daraufhin setzt der Computer seine Arbeit fort.

Beispiel:
Das dargestellte Programm realisiert die Ein- und Ausgabe einer numerischen und einer Zeichenkettengröße:

```
10 INPUT A
20 INPUT Z$
30 PRINT A;" ";Z$
40 END
```

Programmausführung:

```
?27.5
?"ABCDE"
 27.5  ABCDE
```

Erläuterung:

Zeile 10: Der eingegebene Wert 27.5 (muß eine numerische Größe sein) wird der Variablen A zugewiesen.

Zeile 20: Die eingegebene Zeichenkette "ABCDE" wird der Zeichenkettenvariablen Z$ zugeordnet. Bei manchen BASIC-Systemen kann eine Zeichenkette auch ohne hochgestellte Anführungszeichen eingegeben werden.

Zeile 30: Die Werte beider Variablen werden auf einer Zeile (mit Zwischenraum) ausgegeben.

Zur Programmausführung sei noch bemerkt, daß nicht bei jedem BASIC-System, so wie hier, nach einer INPUT-Anweisung auf den Anfang der nächsten Zeile übergegangen wird. Unter Umständen muß nach jeder INPUT-Anweisung ein Übergang auf eine neue Zeile durch eine zusätzliche PRINT-Anweisung (ohne Ausgabegröße) programmiert werden, um das gleiche Ausgabebild zu erreichen.

In einer INPUT-Anweisung können auch mehrere Variable notiert werden. Dann hat sie folgende Form:

INPUT variable,variable,...,variable

In diesem Fall ist dann bei Ausführung der Anweisung genau die programmierte Anzahl der Eingabewerte, durch Komma getrennt, über die Tastatur einzugeben. Der Abschluß der Eingaben wird durch das Zeilenendezeichen erreicht.

Beispiel:

```
10 INPUT A,B,C$
```

Eingabeprotokoll:

```
? 15,12.05,"KILOGRAMM" (NL)
```

Der erste einzugebende (numerische) Wert wird der Variablen A, der zweite (numerische) Wert B und schließlich der dritte einzugebende Wert (eine Zeichenkette) wird C$ zugeordnet.

Beispiel: Volumen- und Oberflächenberechnung einer Kugel

Für einen beliebigen einzugebenden Wert des Radius *R* sind das Volumen und die Oberfläche einer Kugel zu berechnen. Hierzu

wird das bereits in 4.2. behandelte Programm zur Kugelberechnung um die Eingabe des Radius und um die Oberflächenberechnung erweitert.
Der Ablauf des zu realisierenden Programms ist im Bild 4.2 ver-

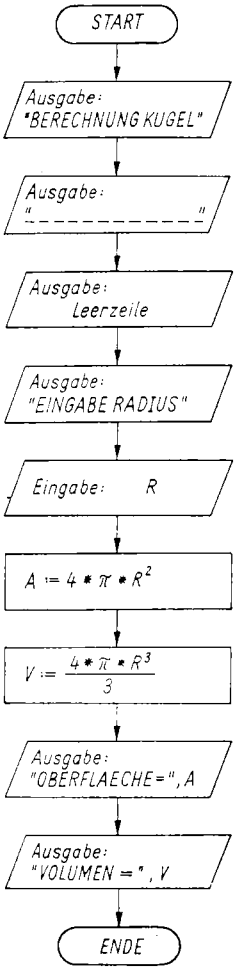


Bild 4.2. Graphische Darstellung des Programmablaufs zur Kugelberechnung

anschaulicht. Um eine Dialogunterstützung des Benutzers während der Ausführung des Programms zu erreichen, wird ein entsprechender Text als Eingabeaufforderung ausgegeben. Das BASIC-Programm und ein Ausführungsprotokoll zeigt die folgende Übersicht.

Programm:

```
10 PRINT "BERECHNUNG KUGEL"
20 PRINT "- - - - - - - - - - - - - - - -"
30 PRINT
40 PRINT "EINGABE RADIUS:"
50 INPUT R
60 LET A=4*3.141592*R^2
70 LET V=4*3.141592*R^3/3
80 PRINT "OBERFLAECHE=",A
90 PRINT "VOLUMEN=",V
100 END
```

Ausführung des Programms:

```
BERECHNUNG KUGEL
- - - - - - - - - - - - - - - -

EINGABE RADIUS.
?5.1
OBERFLAECHE=  326.851
VOLUMEN=      555.647
```

Während der Ausführung des Programms wurde als Radius der Wert 5.1 eingegeben.

4.4. Kommentaranweisung [REM]

Die Kommentar- oder REM-Anweisung (engl.: to remark – bemerken) ist für das Einfügen von erklärenden Texten innerhalb eines Programms gedacht. Der Programmierer kann sie beliebig oft und an jeder Stelle des Programms verwenden. Eine REM-Anweisung dient nur der Erläuterung und Erhöhung der Lesbarkeit des Programms. Sie hat keine Wirkung auf die Programmaus-

führung. Die REM-Anweisung hat einen einfachen Aufbau:

REM kommentar

Der Kommentar ist ein Text mit beliebigen Zeichen aus dem gesamten Zeichensatz von BASIC.

Beispiel:

```
10  REM PROGRAMM ZUR EINGABE
15  REM UND AUSGABE
20  INPUT A
 .
 .
 .
```

Man beachte, daß REM-Anweisungen in einem BASIC-Programm (Quellprogramm) Speicherplatz belegen, obwohl sie während der Ausführung keine Wirkung verursachen.

5. Standardfunktionen

5.1. Numerische Funktionen

In der praktischen Anwendung begegnet man vielfach Problemstellungen, in denen mathematische Funktionen eine größere Rolle spielen. Solche Aufgaben sind nicht ohne weiteres über algebraische Ausdrücke mit Hilfe der Addition, Subtraktion, Multiplikation und Division zu berechnen. Die Wurzel von x oder trigonometrische Funktionen $\sin x$ und $\cos x$ sind solche Beispiele. Derartige Funktionen lassen sich nur mit Hilfe der numerischen Mathematik berechnen, wobei die entsprechenden Programme recht umfangreich und kompliziert sind. Die Programmiersprache BASIC stellt jedoch eine ganze Reihe solcher Funktionen dem Anwender als fertige Programme zur Verfügung. Sie sind selbst Bestandteil der Programmiersprache und werden als Standardfunktionen bezeichnet. Ihre Verwendung erfolgt in einem BASIC-Programm einfach durch Aufruf des entsprechenden Funktionsnamens.

Beispiel:

```
20 PRINT SIN(X)
```

In Tabelle 5.1 sind die üblichen BASIC-Standardfunktionen im Überblick aufgeführt. Für das Argument kann eine Konstante, eine Variable oder ein Ausdruck stehen. Die Wirkungsweise des Funktionsaufrufes besteht darin, daß zunächst der als Argument stehende Ausdruck berechnet, anschließend der Funktionswert bestimmt und das Ergebnis dem Funktionsnamen (eine Variable) zugewiesen wird. Es sei darauf hingewiesen, daß in BASIC keine rekursive Verwendung von Funktionen möglich ist. Das bedeutet, daß sich eine Funktion nicht selbst wieder aufrufen kann. Es ist jedoch zulässig, daß im Argument einer Funktion ein Ausdruck steht, der wieder beliebige Funktionsaufrufe enthält.

Tabelle 5.1. Numerische Standardfunktionen

Funktionsname	Erläuterung	mathematische Bezeichnung
SQR(X)	Quadratwurzel von X; $X \geqq 0$	$\sqrt{x}$
ABS(X)	Betrag von X	$\lvert x \rvert$
SIN(X)	Sinusfunktion; X im Bogenmaß	$\sin x$
COS(X)	Cosinusfunktion; X im Bogenmaß	$\cos x$
TAN(X)	Tangensfunktion; X im Bogenmaß	$\tan x$
ATN(X)	Arcustangensfunktion; Ergebnis im Bogenmaß	$\arctan x$
EXP(X)	Exponentialfunktion	e^x
LOG(X)	Natürlicher Logarithmus	$\ln x$
SGN(X)	Signumfunktion	$\operatorname{sgn}(x)$
INT(X)	Berechnung des ganzzahligen Teils von X	Integerfunktion
RND	Berechnung einer Zufallszahl im Bereich $0 \leqq \text{RND} < 1$	Zufallsfunktion

Beispiel:

```
80 LET Y=SIN(A+B+C*EXP(Z))
```

Anschließend werden noch einige Hinweise für die Benutzung der Standardfunktionen gegeben.

Quadratwurzel SQR(X)

Man beachte, daß X keine negative Zahl sein darf. Für solche Werte bricht das Programm mit einer Fehlermeldung des Computers ab.

Beispiel:

Es wird die Wurzel $y = \sqrt{x}$ einer beliebigen einzugebenden positiven Zahl x berechnet.

```
10 PRINT "EINGABE X:"
20 INPUT X
30 LET Y=SQR(X)
40 PRINT "WURZEL=";Y
50 END
RUN
EINGABE X:
?25
WURZEL= 5
OK
```

Signumfunktion SGN(X)

Die Signumfunktion liefert in Abhängigkeit vom Vorzeichen des Arguments X einen ganzzahligen Wert. Es gilt:

$$\mathrm{SGN(X)} = \begin{cases} +1 \text{ für } X > 0 \\ 0 \text{ für } X = 0 \\ -1 \text{ für } X < 0 \end{cases}$$

Beispiel:

```
20 LET Y=SGN(X)
```

X	25.1	−7.31	−0.002	0
Y	1	−1	−1	0

Integerfunktion INT(X)

Die Funktion INT(X) liefert die ganze Zahl kleiner oder gleich X, die X am nächsten liegt. Für positive Argumente entspricht das der Berechnung des ganzzahligen Teils von X.

Beispiel:

```
20 LET Y=INT(X)
```

X	2.171	16.8	−16.8	16
Y	2	16	−17	16

Die INT-Funktion wird unter anderem zur Rundung von gebrochenen Zahlen verwendet.

Beispiel:

```
20 LET Y=INT(X+0.5)
```

X	6.4	6.5	−18.0
Y	6	7	−18

Allgemein gilt beim Runden einer Variablen X auf die D-te Dezimalstelle nach dem Komma:

```
Y=INT(X*10^D+0.5)/10^D
```

Beispiel:

```
D=3
20 LET Y=INT(X*10^3+0.5)/10^3
```

X	0.36361	−2.77145
Y	0.364	−2.771

Zufallsfunktion RND

Bei Aufruf dieser Funktion wird während des Programmlaufs vom Computer eine zufällige (genauer eine pseudozufällige) Zahl im Bereich von 0 bis 0.999 ... erzeugt.

Beispiel:

```
10  REM ERZEUGEN VON 3 ZUFALLS-
15  REM ZAHLEN
20  PRINT RND,RND,RND
30  END
RUN
0.37135        0.81132        0.12592
OK
```

Mit Hilfe der RND-Funktion lassen sich auch Zufallszahlen für beliebige andere Zahlenintervalle (z.B. ganze Zufallszahlen) programmtechnisch erzeugen.
Es sei darauf hingewiesen, daß die mit der RND-Funktion berechneten Zufallszahlen zwar gleichverteilt sind, die Folge der Zahlen jedoch rechnerabhängig ist. Normalerweise tritt bei jedem Programmlauf die gleiche Zufallsfolge auf, wenn keine besonderen Vorkehrungen getroffen sind. Aus diesem Grund wird die RANDOMIZE-Anweisung benutzt. Sie besteht nur aus dem Schlüsselwort:

RANDOMIZE

Diese Anweisung nimmt auf die Berechnung der Zufallsfolge Einfluß, indem bei ihrer Ausführung durch den Computer ein neuer Startwert für die Zufallszahlenberechnung erzeugt bzw. angefordert wird. Sie bewirkt demnach, daß bei jedem Programmlauf andere Zufallszahlen entstehen.

Beispiel:

Es sollen bei jedem Programmlauf zwei ganze Zufallszahlen aus dem Bereich von 1 bis 100 erzeugt werden.

```
10  RANDOMIZE
20  PRINT INT(100*RND+1),
25  PRINT INT(100*RND+1)
30  END
RUN
21        30
OK
```

```
RUN
73        6
OK
```

Die von der Funktion RND erzeugten Zufallszahlen ($0 \leq$ Zufallszahl < 1) werden mit Hilfe der INT-Funktion und einer entsprechenden Umrechnungsvorschrift in ganze Zufallszahlen ($1 \leq$ Zufallszahl ≤ 100) transformiert. Würde man die RANDOMIZE-Anweisung weglassen, so wären nach dem zweiten Programmstart die gleichen Zufallszahlen berechnet worden wie im ersten Durchlauf.
Es sei noch vermerkt, daß in manchen BASIC-Systemen aus Gründen einer einheitlichen syntaktischen Form aller Standardfunktionen die RND-Zufallsfunktion mit der Schreibweise RND(X) benutzt werden muß. Das Argument X kann in diesem Fall einen beliebigen Wert haben (Wert bleibt unberücksichtigt). Es gibt aber auch Systeme, wo mit Hilfe von X eine zusätzliche Steuerung der Zufallszahlenerzeugung ähnlich der RANDOMIZE-Anweisung vorgenommen werden kann.

5.2. Tabellierungsfunktion [TAB]

Die TAB-Funktion dient der programmtechnischen Steuerung des Ausgabegerätes. Sie darf nur in einer PRINT- bzw. LPRINT-Anweisung benutzt werden und hat folgende Form:

TAB(X)

Die TAB-Funktion bewirkt, daß als neue Ausgabeposition auf der aktuellen Bildschirm- oder Druckerzeile die durch den Wert von X bezeichnete Position eingestellt wird (bei manchen BASIC-Systemen auch die darauf folgende).
Für das Argument X kann eine Zahl, eine numerische Variable oder allgemein ein numerischer Ausdruck stehen. Ergibt sich für X ein gebrochener numerischer Wert, so wird vom Computer auf eine ganze Zahl gerundet. TAB(1) entspricht der äußersten linken Position einer Zeile.
In einer PRINT-Anweisung können sich TAB-Aufrufe, numerische Größen und Zeichenkettengrößen beliebig abwechseln. Als Trennzeichen gelten Komma und Semikolon entsprechend den in 4.2. angegebenen Regeln.

Beispiel:

```
10  PRINT "FLAECHE";TAB(10); "VOLUMEN";
15  PRINT TAB(20);"GEWICHT"
20  PRINT "-------------------------"
.
.
.

130  PRINT A,TAB(10);V;TAB(20);G
.
.
.

Ausgabeprotokoll:

FLAECHE     VOLUMEN    GEWICHT

-----------------------------

 2              250          3020
```

Außer den behandelten numerischen Standardfunktionen gibt es in vielen erweiterten BASIC-Systemen auch Zeichenkettenfunktionen. Sie unterstützen vor allem eine leistungsfähige Textverarbeitung. Möglichkeiten zur Berechnung anderer mathematischer Funktionen mit Hilfe der BASIC-Standardfunktionen sind im Anhang B aufgeführt.

6. Steuerung des Programmablaufs

6.1. Aufgabe der Steueranweisungen

Die Anweisungen eines BASIC-Programms werden normalerweise in der Reihenfolge der Zeilennummern ausgeführt. Mit Hilfe von Anweisungen zur Steuerung des Programmablaufs ist es jedoch möglich, diese Folge zu verlassen oder die Programmausführung zu beenden. So kann man auch komplizierte algorithmische Strukturen, wie Verzweigungen (zu unterschiedlichen Verarbeitungsschritten) und Schleifen (wiederholte Ausführung von Anweisungen), die in praktischen Problemlösungen sehr häufig vorkommen, programmieren.

Zur Steuerung des Programmablaufs stehen in BASIC folgende Anweisungen zur Verfügung:
- Sprunganweisung (GOTO)
- Verzweigungsanweisung (IF-THEN)
- Verzweigungsanweisung mit Auswahl (ON-GOTO)
- Laufanweisung (FOR,NEXT)
- Haltanweisung (STOP)
- Programmendeanweisung (END)
- Unterprogrammanweisung (GOSUB, RETURN)

Ihre Notierungsform, Wirkung und Anwendung wird in den folgenden Abschnitten beschrieben.

6.2. Sprunganweisung [GOTO]

Die Sprung- oder GOTO-Anweisung (engl.: go to – gehe nach) hat die Form

GOTO zielzeilennummer.

Sie bewirkt, daß als nächste jene Anweisung ausgeführt wird, deren Zeilennummer hinter dem Schlüsselwort GOTO notiert ist (Sprungziel).

Beispiel:

```
        .
        .
        .
    110 LET A=3
    120 GOTO 140
    130 LET A=5
    140 PRINT A
        .
        .
        .
```

Bei der Ausführung dieser Anweisungsgruppe wird durch die GOTO-Anweisung die Programmzeile 130 übergangen (übersprungen.) Die folgende PRINT-Anweisung bewirkt deshalb die Ausgabe der Zahl "3".
Wird als Sprungziel eine Zeile festgelegt, die sich im Programmablauf vor der GOTO-Anweisung befindet, so entsteht eine Schleife.

Beispiel:

```
10 LET N=1
20 PRINT "SCHLEIFENDURCHLAUF";N
30 LET N=N+1
40 GOTO 20
 .
 .
 .
```

Die Sprunganweisung führt in diesem Beispiel immer wieder zur Zeile 20 zurück, und die Programmzeilen 20, 30 und 40 werden deshalb zyklisch durchlaufen. Auf dem Bildschirm erscheint die Anzeige

```
SCHLEIFENDURCHLAUF 1
SCHLEIFENDURCHLAUF 2
SCHLEIFENDURCHLAUF 3
 .
 .
 .
```

Hier handelt es sich speziell um eine Endlosschleife. Die Ausführung dieses Programms kann der Anwender deshalb nur durch das Drücken der Unterbrechungstaste ("BREAK") beenden. Soll die Schleife ohne Eingriff des Anwenders verlassen werden, muß man eine der im folgenden beschriebenen Steueranweisungen einsetzen.

6.3. Verzweigungsanweisung [IF-THEN]

Oft sind Algorithmen so gestaltet, daß in Abhängigkeit vom Ergebnis eines Vergleichs eine von zwei verschiedenen Anweisungsfolgen auszuführen ist (Programmverzweigung). Verzweigungen lassen sich in BASIC mit der IF-THEN-Anweisung (engl.: if...then – wenn...dann) realisieren. Sie hat die Form

IF ausdruck vergleichsoperator ausdruck **THEN** zielznr

Zwischen den Schlüsselworten IF und THEN steht ein Vergleichsausdruck. *Wenn* dieser den Wert "wahr" hat, *dann* wird die Programmausführung mit der durch die Zielzeilennummer (zielznr)

bezeichneten Zeile fortgesetzt. Anderenfalls hat diese Anweisung keine Wirkung, d.h., als nächste wird die Anweisung der folgenden Programmzeile ausgeführt.
In Tabelle 6.1 sind die möglichen Vergleichsoperatoren sowie Beispiele für die Notierung von IF-THEN-Anweisungen angegeben. Die Ausdrücke, aus denen der Vergleichsausdruck gebildet wird, müssen entweder beide numerische Ausdrücke oder beide Zeichenkettenausdrücke sein, weil eine Zahl nur mit einer Zahl und eine Zeichenkette nur mit einer Zeichenkette verglichen werden kann.

Tabelle 6.1. Vergleichsoperatoren

Mathem. Symbol	Vergleichs-operator	Beispiel
=	=	IF A=2*B THEN 120
≠	<>	IF B$ <>"JA" THEN 120
<	<	IF ABS(X)<1 THEN 120
≦	<=	IF A<=Z1 THEN 120
>	>	IF A>B THEN 120
≧	>=	IF A*B>=0 THEN 120

Beispiel:

```
10 INPUT A
20 IF A<0 THEN 40
30 PRINT "NICHT ";
40 PRINT "NEGATIV"
50 END
```

Im Beispiel wird in Abhängigkeit vom Vorzeichen einer eingegebenen Zahl A der Text "NEGATIV" oder "NICHT NEGATIV" ausgegeben. Ist A kleiner als Null, so hat der Vergleichsausdruck in Anweisung 20 den Wert "wahr" und die Programmausführung wird deshalb auf Programmzeile 40 fortgesetzt. Anderenfalls bleibt die IF-THEN-Anweisung ohne Wirkung, und vor der Anweisung 40 kommt die Anweisung 30 zur Ausführung.
Weil die IF-THEN-Anweisung wie eine GOTO-Anweisung wirkt, sofern der Vergleichsausdruck den Wert "wahr" hat, wird sie

manchmal auch als *bedingte Sprunganweisung* bezeichnet. Der Vergleichsausdruck beschreibt die Sprungbedingung.
Im Gegensatz zum Zahlenvergleich ist der Vergleich von Zeichenketten nicht allgemein bekannt und wird deshalb im folgenden erläutert.
Minimal-BASIC sieht für den Vergleich von Zeichenketten nur den Gleichheits- und den Ungleichheitsoperator vor. Es gilt: Zwei Zeichenketten sind nur dann gleich, wenn beide die gleiche Länge (Zeichenzahl) haben und in jedem Zeichen übereinstimmen. Im folgenden Beispiel wird der Zeichenkettenvergleich zur Steuerung des Programmlaufs benutzt.

Beispiel:

```
      .
      .
      .
180   PRINT "PROGRAMMFORTSETZUNG ?"
190   PRINT "'JA' ODER 'NEIN' EINGEBEN!"
200   INPUT X$
210   IF X$="JA" THEN 30
220   IF X$<>"NEIN" THEN 190
230   END
```

Wird die Frage nach der Programmfortsetzung mit der Eingabe der Zeichenkette "JA" beantwortet, dann bewirkt Anweisung 210 einen Sprung zur Zeile 30. Im Falle einer Fehleingabe, d.h., wenn weder "JA" noch "NEIN", sondern beispielsweise "J" eingegeben wird, führt Anweisung 220 zurück zur Programmzeile 190 und damit zur Wiederholung der Eingabeaufforderung. Bei Eingabe von "NEIN" hat keiner der Vergleichsausdrücke den Wert "wahr", und die Programmausführung wird deshalb beendet.
Häufig erlauben BASIC-Systeme, den gesamten Vorrat der Vergleichsoperatoren entsprechend Tabelle 6.1 für den Zeichenkettenvergleich zu nutzen. Die Wirkung von Vergleichsausdrücken mit den Operatoren "kleiner als" oder "größer als" wird sichtbar, wenn man sich den Ablauf des Zeichenkettenvergleichs im Computer verdeutlicht. Dieser ordnet nach einem Code jedem Zeichen einen bestimmten Zahlenwert zu. In Tabelle 6.2 sind diese Zahlenwerte für Ziffern und Buchstaben angegeben, und Anhang A zeigt

die Verschlüsselung aller BASIC-Zeichen. Ein Vergleich von Zeichen bzw. Zeichenketten erfolgt im Computer, indem diese zugeordneten Zahlenwerte verglichen werden. So gilt beispielsweise

"B">"A", weil 66>65
"5"<"Z", weil 53<90.

Beim Vergleich zweier Zeichenketten werden, von links nach rechts gehend, die jeweils stellengleichen Zeichen verglichen. Demnach wird das erste Zeichen der ersten Zeichenkette dem ersten Zeichen der zweiten gegenübergestellt. Sind beide gleich, so folgt der Vergleich der zweiten Zeichen beider Zeichenketten. Das wird fortgeführt, bis zwei unterschiedliche Zeichen ermittelt werden oder bis das Ende einer Zeichenkette erreicht ist. Beispielsweise ergibt sich so

"LAUS">"LAUB",

weil die ersten 3 Zeichen beider Zeichenketten zwar übereinstimmen, aber beim 4. Zeichen "S">"B" gilt. Es ist aber auch

"HAUSMANN">"HAUS",

da die rechte Zeichenkette zwar mit dem ersten Teil der linken identisch ist, die linke aber mehr Zeichen umfaßt.
Für die Anwendung des Zeichenkettenvergleichs ist eine Aussage wichtig, die aus obigen Erläuterungen und aus der Systematik der Zeichencodierung (Tabelle 6.2) folgt: Sind A$ und B$ zwei Variable, so gilt A$<B$, wenn die A$ zugeordnete Zeichenkette in der alphabetischen Folge vor derjenigen steht, die durch B$ bezeichnet wird. Zeichenkettenvergleiche kann man daher auch für die alphabetische Sortierung nutzen.
Die Eigenschaft der IF-THEN-Anweisung, Entscheidungen zu treffen bzw. die Programmabarbeitung entscheidungsabhängig zu steuern, erlaubt es, unterschiedliche Algorithmenstrukturen in BASIC-Programme umzusetzen. Die Bilder 6.1 bis 6.4 zeigen häufig vorkommende Strukturen in allgemeiner Form als Programmablaufpläne, Struktogramme und BASIC-Anweisungsfolgen. In den Programmen sind zn_1, zn_2, zn_3 Zeilennummern, die der Bedingung $zn_1 < zn_2 < zn_3$ unterliegen. Welchen Einfluß die Formulierung des Vergleichsausdruckes der IF-THEN-Anwei-

Tabelle 6.2. Zeichencode

Zeichen	Dezimal-wert	Zeichen	Dezimal-wert	Zeichen	Dezimal-wert
0	48	C	67	O	79
1	49	D	68	P	80
2	50	E	69	Q	81
3	51	F	70	R	82
4	52	G	71	S	83
5	53	H	72	T	84
6	54	I	73	U	85
7	55	J	74	V	86
8	56	K	75	W	87
9	57	L	76	X	88
A	65	M	77	Y	89
B	66	N	78	Z	90

sung auf das Programm haben kann, wird aus der Gegenüberstellung der Anweisungsfolgen in den Bildern 6.1 und 6.2 ersichtlich. Die Struktur mit leerem Ja-Zweig ist offensichtlich günstiger, denn sie erfordert keine zusätzliche GOTO-Anweisung. Bei der im Bild 6.3 gezeigten Struktur ist weder der Ja- noch der Nein-Zweig leer. Im entsprechenden Programm werden die Aktionen beider

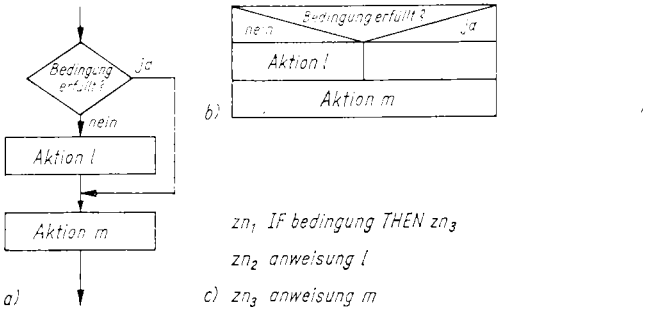
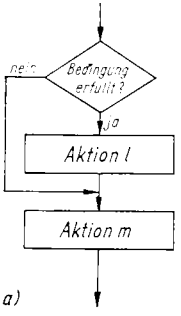
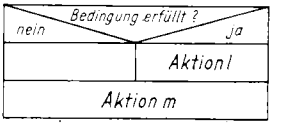


Bild 6.1. Verzweigung mit leerem Ja-Zweig





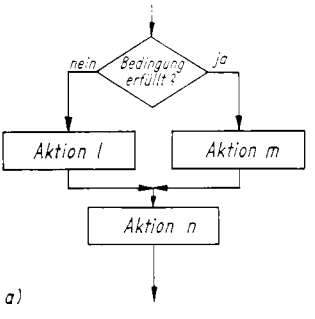
zn_1 *IF bedingung THEN* zn_3

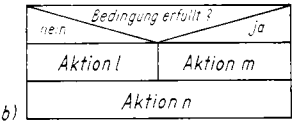
zn_2 *GOTO* zn_4

zn_3 *anweisung l*

c) zn_4 *anweisung m*

Bild 6.2. Verzweigung mit leerem Nein-Zweig





zn_1 *IF bedingung THEN* zn_4

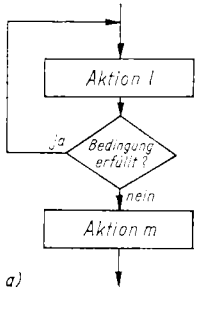
zn_2 *anweisung l*

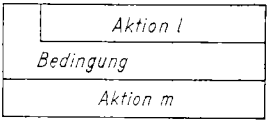
zn_3 *GOTO* zn_5

zn_4 *anweisung m*

c) zn_5 *anweisung n*

Bild 6.3. Verzweigung





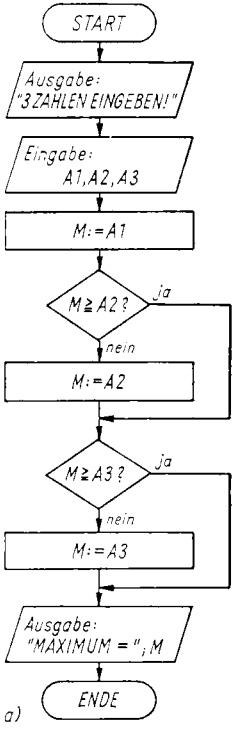
zn_1 *anweisung l*

zn_2 *IF bedingung THEN* zn_1

c) zn_3 *anweisung m*

Bild 6.4. Schleife

Zweige hintereinander notiert. Deshalb muß auf die Anweisung des Nein-Zweiges eine GOTO-Anweisung zum Überspringen des Ja-Zweiges folgen.
Nach diesen allgemeinen Ausführungen soll anhand weiterer Beispiele die Verwendung der IF-THEN-Anweisung zur Programmierung von Algorithmen unterschiedlicher Struktur verdeutlicht werden.



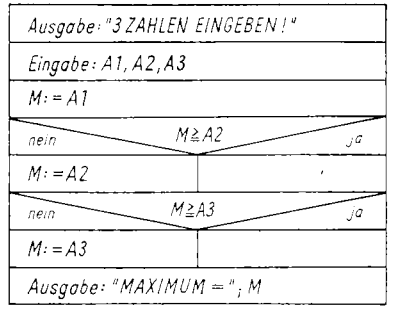


Bild 6.5. Maximum dreier Zahlen
a) Programmablaufplan
b) Struktogramm

Beispiel: Maximum dreier Zahlen

Von drei über die Tastatur eingegebenen beliebigen Zahlen A1, A2 und A3 wird vom Rechner die größte ermittelt und auf dem Bildschirm angezeigt. Der dem Programm zugrunde liegende Algorithmus ist im Bild 6.5 als Programmablaufplan und als Struktogramm dargestellt. Nach der Eingabe der Zahlen wird (willkürlich) als vorläufiger Größtwert der Wert A1 angenommen, d.h., der Variablen M wird der Wert A1 zugewiesen. Anschließend folgt der Vergleich von M und A2. Sofern die Bedingung $M \geqq A2$ nicht erfüllt ist, wird als neuer (vorläufiger) Größtwert A2 festgelegt, d.h., die Variable M nimmt den Wert A2 an. Danach folgt analog der Vergleich von M mit A3 und im Falle $M < A3$ die

```
10  REM MAXIMUM DREIER ZAHLEN
20  PRINT "3 ZAHLEN EINGEBEN !"
30  INPUT A1,A2,A3
40  LET M=A1
50  IF M>=A2 THEN 70
60  LET M=A2
70  IF M>=A3 THEN 90
80  LET M=A3
90  PRINT "MAXIMUM=";M
100 END
RUN
3 ZAHLEN EINGEBEN !
? 22,37,15
MAXIMUM= 37
OK
```

Programm 6.1. Maximum dreier Zahlen

Zuweisung des Wertes A3 an M. Die Umsetzung des Programmablaufplanes bzw. Struktogramms in BASIC-Anweisungen führt zum Programm 6.1.

Beispiel: Lösungen einer quadratischen Gleichung

Berechnet werden die reellen Lösungen der quadratischen Gleichung $x^2 + px + q = 0$ nach der Formel

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$$

Den vollständigen Algorithmus beschreibt Bild 6.6. Nach der Eingabe der beiden Parameter P und Q wird der Wert D (Aus-

druck unter der Wurzel) berechnet. Das Vorzeichen von D entscheidet darüber, ob die Gleichung reelle Lösungen besitzt. Ist D nicht kleiner als Null, so werden X1 und X2 berechnet und ausgegeben. Anderenfalls wird der Text "KEINE REELLEN LOESUNGEN" angezeigt. Die Realisierung des Algorithmus in BASIC zeigt das Programm 6.2.
Die Programmierung von Schleifen mit Hilfe der IF-THEN-Anweisung zeigt das Bild 6.4 in allgemeiner Form und das folgende Beispiel in der praktischen Anwendung.

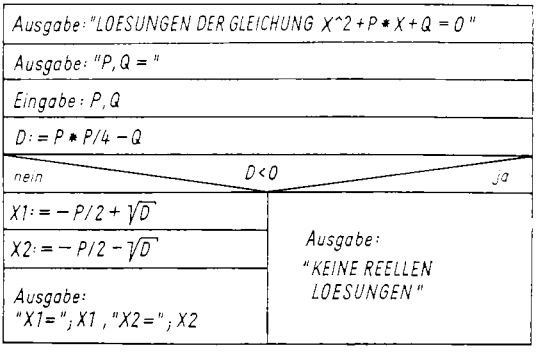


Bild 6.6. Lösungen einer quadratischen Gleichung

```
10  PRINT "LOESUNGEN DER GLEICHUNG X^2+P*X+Q=0"
20  PRINT "P,Q=";
30  INPUT P,Q
40  LET D=P*P/4-Q
50  REM TEST AUF KOMPLEXE LOESUNGEN
60  IF D<0 THEN 110
70  LET X1=-P/2+SQR(D)
80  LET X2=-P/2-SQR(D)
90  PRINT "X1=";X1,"X2=";X2
100 GOTO 120
110 PRINT "KEINE REELLEN LOESUNGEN"
120 END
RUN
LOESUNGEN DER GLEICHUNG X^2+P*X+Q=0
P,Q=? 2,-3
X1= 1          X2=-3
OK
```

Programm 6.2. Lösungen einer quadratischen Gleichung

Beispiel: Summe von n Zahlen a_i $(i = 1, 2, \ldots, n)$
Der Algorithmus zur Lösung dieser Aufgabe wurde bereits in 1.3. entwickelt und im Bild 1.10 dargestellt. Seine Umsetzung führt zum BASIC-Programm 6.3.

```
10  PRINT "SUMME VON N ZAHLEN"
20  PRINT "N=";
30  INPUT N
40  LET S=0
50  LET I=1
60  PRINT "A(";I;")=";
70  INPUT A
80  LET S=S+A
90  LET I=I+1
100 IF I<=N THEN 60
110 PRINT "SUMME=";S
120 END
RUN
SUMME VON N ZAHLEN
N=? 6
A( 1 )=? 25
A( 2 )=? 18.5
A( 3 )=? 33
A( 4 )=? 11
A( 5 )=? 5.4
A( 6 )=? 7
SUMME= 99.9
OK
```

Programm 6.3. Summe von n Zahlen

Modifikationen der IF-THEN-Anweisung

Abschließend sollen noch Modifikationen der IF-THEN-Anweisung beschrieben werden, die in erweiterten, leistungsfähigen BASIC-Sprachversionen häufig zu finden sind.

IF ausdruck **THEN** { zeilennummer / anweisung }

Auf das Schlüsselwort THEN kann bei dieser Form entweder eine Zeilennummer oder eine Anweisung folgen. Hat der Ausdruck den Wert "wahr", so wird zur angegebenen Zeilennummer verzweigt bzw. die hinter THEN stehende Anweisung ausgeführt. Anderenfalls hat die Anweisung keine Wirkung.
Wahlweise kann man außerdem oft die IF-THEN-Anweisung durch Angabe einer Alternative ergänzen, die mit dem Schlüsselwort ELSE (engl.: else – sonst) beginnt und dann wirksam wird,

wenn der Ausdruck den Wert "falsch" hat. Hinter ELSE kann wiederum entweder eine Zeilennummer oder eine Anweisung notiert werden.

IF ausdruck **THEN** { zeilennr. / anweisung } ELSE { zeilennr. / anweisung }

Beispiel:

```
10 INPUT A
20 IF A<0 THEN PRINT "NEGATIV"
   ELSE PRINT "NICHT NEGATIV"
30 END
```

Ist der eingegebene Wert A<0, so folgt die Textausgabe "NEGATIV". Bei A $\geqq$ 0 wird "NICHT NEGATIV" ausgegeben.
Bei einigen BASIC-Versionen können durch Verwendung logischer Operatoren (NOT,AND,OR) mehrere Vergleiche in einer IF-THEN-Anweisung zusammengefaßt werden. Bei der Auswertung eines Ausdrucks werden die Vergleichsoperationen vor den logischen Operationen ausgeführt. Für die logischen Operationen ist im allgemeinen die Rangfolge NOT,AND,OR festgelegt. Ihre Ergebnisse bei unterschiedlichen Werten der Operanden zeigt die Tabelle 6.3.

Beispiel:
Ein über die Tastatur einzugebender Wert X soll im Bereich $0 \leqq X \leqq 9999$ liegen. Er ist deshalb nach der Eingabe auf Einhaltung dieser Grenzen zu prüfen. Die beiden Vergleichsausdrücke X>=0 und X<=9999 werden mit dem Operator AND zu einem Ausdruck verbunden.

```
10 INPUT X
20 IF X>=0 AND X<=9999 THEN 50
30 PRINT "UNZULAESSIGER WERT"
40 GOTO 10
50 PRINT "ZULAESSIGER WERT"
 .
 .
 .
```

Im Beispiel werden zuerst die Vergleiche X$\geqq$0 und X$\leqq$9999 ausgeführt. Jeder der beiden kann das Ergebnis "wahr" oder "falsch"

liefern. Die Verknüpfung dieser Resultate durch den Operator AND liefert dann den Wert des Ausdrucks (Tabelle 6.3b). Dieser nimmt nur dann den Wert "wahr" an, wenn beide Bedingungen erfüllt sind.

Tabelle 6.3. Logische Operationen

a) NOT-Operation (Negation)

X	NOT X
wahr	falsch
falsch	wahr

b) AND-Operation (UND)

X	Y	X AND Y
wahr	wahr	wahr
wahr	falsch	falsch
falsch	wahr	falsch
falsch	falsch	falsch

c) OR-Operation (ODER)

X	Y	X OR Y
wahr	wahr	wahr
wahr	falsch	wahr
falsch	wahr	wahr
falsch	falsch	falsch

6.4. Verzweigungsanweisung mit Auswahl [ON-GOTO]

Die ON-GOTO-Anweisung, auch berechnete Sprunganweisung genannt, verzweigt in Abhängigkeit vom Wert eines numerischen Ausdrucks zu einer von mehreren möglichen Programmzeilen. Sie hat die allgemeine Form

ON numausdruck **GOTO** zielznr,zielznr,...,zielznr

Hinter dem Schlüsselwort GOTO steht eine Folge von Zielzeilennummern. Die Entscheidung, bei welcher der aufgeführten Programmzeilen die Programmausführung fortgesetzt wird, ist vom Zahlenwert des zwischen ON und GOTO stehenden numerischen Ausdrucks abhängig. Hat dieser beispielsweise den Wert 2, so wird die Programmausführung auf der zweiten der hinter GOTO notierten Zeilen fortgesetzt, hat er den Wert 4, bei der vierten. Allgemein formuliert gilt: Hat der auf ON folgende Ausdruck den Wert i, so wird durch die ON-GOTO-Anweisung als Sprungziel die i-te der hinter GOTO notierten Zeilennummern ausgewählt.

Beispiel:

```
10 INPUT A
20 ON A GOTO 220,270,340,390
```

Eingabewert A	1	2	3	4
Sprungziel Zeile	220	270	340	390

Ist der Wert des Ausdrucks nicht ganzzahlig, so wird er vor der Auswertung auf eine ganze Zahl gerundet. (Bei manchen BASIC-Versionen wird der gebrochene Teil der Zahl unabhängig von seiner Größe abgeschnitten.) Wenn der gerundete Wert des Ausdruckes kleiner als 1 oder größer als die Anzahl der hinter GOTO aufgeführten Sprungzeile ist, dann hat die ON-GOTO-Anweisung keine Wirkung auf den Programmablauf.

Beispiel: Programm-Menü

Auf dem Bildschirm werden drei verschiedene Spiel-Programme zur Auswahl angeboten (Menü). Welches ausgeführt werden soll, ist dem Computer durch Eingabe eines Kennzeichens (Ziffer 1, 2 oder 3) mitzuteilen. Der Computer wertet das Kennzeichen aus und verzweigt zum gewählten Spielprogramm. Bild 6.7 zeigt die algorithmische Struktur. Folgende Anweisungsgruppe löst die Aufgabe:

```
10 PRINT "+SPIEL-MENUE+"
20 PRINT "-WUERFELSPIEL: 1"
30 PRINT "-NIMM-SPIEL:   2"
40 PRINT "-HALMA-SPIEL:  3"
50 PRINT
```

```
60   PRINT "AUSWAHL DURCH:";
65   PRINT "EINGABE DER NUMMER:";
70   INPUT K
80   ON K GOTO 110,310,560
90   PRINT "SPIEL NICHT VORHANDEN"
100  GOTO 780
110  REM WUERFELSPIEL
 .
 .
 .
300  GOTO 780
310  REM NIMM-SPIEL
 .
 .
 .
550  GOTO 780
560  REM HALMA-SPIEL
 .
 .
 .
780  END
```

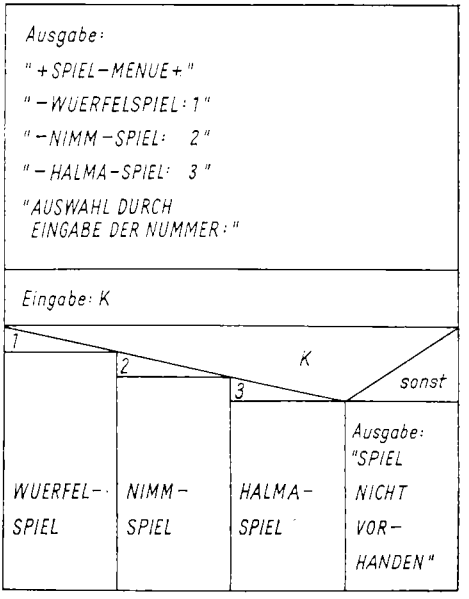


Bild 6.7. Spiel-Menü

In diesem Spiel beginnen demnach das Würfelspiel auf Zeile 110, das Nimm-Spiel auf Zeile 310 und Halma auf Zeile 560. Auf die Eingabe einer Zahl, der in der ON-GOTO-Anweisung keine Zeilennummer zugeordnet ist, folgt die Ausgabe der Mitteilung "SPIEL NICHT VORHANDEN" durch die Anweisung 90 und danach der Sprung zum Programmende. Obige Aufgabe läßt sich auch dürch Kombination mehrerer IF-THEN-Anweisungen lösen. Das Programm ist dann jedoch weniger übersichtlich.

6.5. Laufanweisung [FOR, NEXT]

Wie das Programm 6.3 zeigt, kann man Schleifen mit Hilfe der IF-THEN-Anweisung programmieren. BASIC verfügt aber außerdem über eine spezielle Sprachkonstruktion zur Formulierung von Programmschleifen: die Laufanweisung. Sie hat die Form

```
FOR variable=anfangswert TO endwert STEP schrittweite

anweisung[en]
NEXT variable
```

Mit der FOR-Anweisung (engl.: for...to – für...bis) wird festgelegt, welche Werte die angegebene (numerische) Variable, Laufvariable genannt, beim wiederholten Schleifendurchlauf annehmen soll. Anfangswert, Endwert und Schrittweite können beliebige numerische Ausdrücke sein. Auf die Programmzeile mit der FOR-Anweisung folgt die Gruppe der zyklisch auszuführenden Anweisungen (Schleifenkörper), die durch die NEXT-Anweisung abgeschlossen wird. Hinter dem Schlüsselwort NEXT muß die Laufvariable der zugehörigen FOR-Anweisung angegeben sein.
Die Laufanweisung bewirkt folgendes:

(1) Aus der FOR-Anweisung werden der Anfangswert, der Endwert und die Schrittweite der Laufvariablen berechnet. Der Laufvariablen wird der Anfangswert zugewiesen.
(2) Es wird geprüft, ob der Wert der Laufvariablen den Endwert überschreitet. Ist das nicht der Fall, werden die Anweisungen des Schleifenkörpers ausgeführt. Wenn der Wert der Laufvariablen größer als der Endwert ist, wird zu der Programmzeile übergegangen, die auf die NEXT-Anweisung folgt.

(3) Die NEXT-Anweisung addiert den Wert der Schrittweite zum Wert der Laufvariablen und schließt die Schleife. Es folgt die unter (2) beschriebene Prüfung des Wertes der Laufvariablen.

In der FOR-Anweisung kann der Teil "STEP schrittweite" entfallen, wenn die Schrittweite den Wert 1 hat.

Beispiel:

```
10  FOR I=2 TO 5
20  PRINT I
30  NEXT I
40  END
RUN
 2
 3
 4
 5
OK
```

Beim Eintritt in die Laufanweisung wird der Laufvariablen I der Anfangswert 2 zugewiesen. Das erstmalige Ausführen der Anweisung auf Zeile 20 bewirkt deshalb die Ausgabe des Wertes 2. NEXT I erhöht dann den Wert der Laufvariablen um 1 auf 3. Weil dieser Wert den angegebenen Endwert 5 nicht überschreitet, wird die Anweisung 20 erneut ausgeführt. So werden nacheinander die Werte 3, 4 und 5 ausgegeben. Die anschließende Erhöhung des Wertes der Laufvariablen auf 6 beendet die Ausführung der Laufanweisung.

Soll der Wert der Laufvariablen um Schrittweiten ungleich 1 geändert werden, so ist die erweiterte Form der FOR-TO-Anweisung mit der zusätzlichen Angabe der Schrittweite (STEP) zu verwenden. Die Schrittweite kann auch negativ sein.

Beispiel:

```
10  FOR X=11 TO 4 STEP -2
20  PRINT X,
30  NEXT X
40  END
RUN
11        9         7         5
OK
```

In diesem Beispiel wird nach jedem Durchlauf der Schleife −2 zum aktuellen Wert der Variablen X addiert. Zum letzten Mal wird die Ausgabeanweisung mit dem Wert X=5 wirksam, weil die nachfolgende Addition der Schrittweite −2 zur Unterschreitung des Endwertes 4 führt.
Obige Beschreibung macht deutlich, daß man mit Hilfe der FOR-NEXT-Anweisung nur Schleifen eines bestimmten Typs, sogenannte Zählschleifen, programmieren kann. Für die graphische Darstellung dieser Schleifen sind in der Literatur die im Bild 6.8 dargestellten Sinnbilder gebräuchlich.

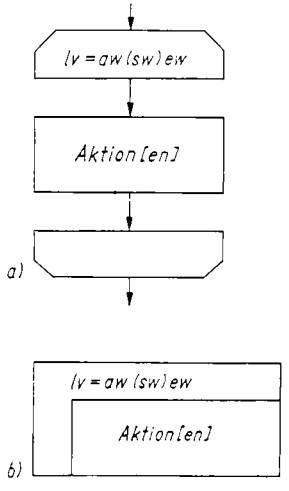


Bild 6.8. Sinnbilder für Zählschleife im Programmablaufplan (a) und im Struktogramm (b)
lv Laufvariable aw Anfangswert
ew Endwert sw Schrittweite

Wie bereits erwähnt, können Anfangswert, Endwert und Schrittweite in der FOR-TO-Anweisung nicht nur als Konstanten, sondern ganz allgemein als numerische Ausdrücke angegeben werden. Folgendes Beispiel soll dies demonstrieren.

Beispiel: Tabellierung von $\sin^2 x$

Die Funktion $y=\sin^2 x$ wird im Bereich $a \leqq x \leqq e$ in Schritten von $\Delta x = d$ tabelliert. Den Programmablauf verdeutlicht das Struktogramm im Bild 6.9. Zwecks größerer Übersichtlichkeit der Algo-

rithmendarstellung werden nun Textausgaben, die der Erläuterung nachfolgender Eingaben oder der Beschreibung von Datenausgaben dienen, im Struktogramm bzw. Programmablaufplan nicht mehr oder nur noch verbal (z.B. Tabellenüberschrift) dar-

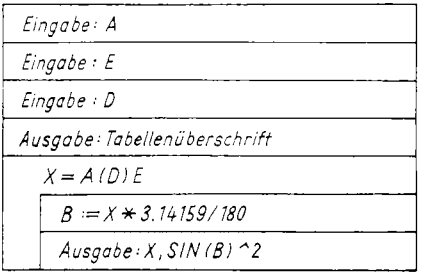


Bild 6.9. Tabellierung von $\sin^2 x$
A Anfangswert E Endwert D Schrittweite B Bogenmaß

```
10  PRINT "TABELLIERUNG VON SIN(X)^2"
20  PRINT "ANFANGSWERT (IN GRAD)";
30  INPUT A
40  PRINT "ENDWERT (IN GRAD)";
50  INPUT E
60  PRINT "SCHRITTWEITE (IN GRAD)";
70  INPUT D
80  PRINT
90  PRINT "  X","SIN(X)^2"
100 PRINT "-------------------------"
110 FOR X=A TO E STEP D
120 LET B=X*3.14159/180
130 PRINT X,SIN(B)^2
140 NEXT X
150 END
RUN
TABELLIERUNG VON SIN(X)^2
ANFANGSWERT (IN GRAD)? 30
ENDWERT (IN GRAD)? 180
SCHRITTWEITE (IN GRAD)? 30

  X            SIN(X)^2
-------------------------
 30             .25
 60             .75
 90             1
 120            .75
 150            .25
 180            .5 61023E-13
OK
```

Programm 6.4. Tabellierung von $\sin^2 x$

gestellt. Auf die Eingabe von Anfangswert, Endwert und Schrittweite folgt gemäß Bild 6.9 die Ausgabe der Tabellenüberschrift und dann der Eintritt in die Schleife. X durchläuft darin die Werte A bis E mit der Schrittweite D. Im Schleifenkörper wird zum Winkel X im Gradmaß der entsprechende Wert B im Bogenmaß berechnet und anschließend der Wert X sowie der berechnete Funktionswert SIN(B)^2 ausgegeben. Die Realisierung des Algorithmus in BASIC und das Laufprotokoll zeigt Programm 6.4.

Schachtelung von Laufanweisungen

Steht in der Anweisungsgruppe zwischen einer FOR- und der zugehörigen NEXT-Anweisung ein weiteres FOR-NEXT-Anweisungspaar, so wird das als Schachtelung von Laufanweisungen bezeichnet. Beispiele für entsprechende Konstruktionen sind schematisch im Bild 6.10 dargestellt. Zu beachten ist dabei, daß für innere und äußere Schleifen unterschiedliche Laufvariablen benutzt werden müssen. Unzulässig sind auch Überlappungen von Schleifen (Bild 6.11).

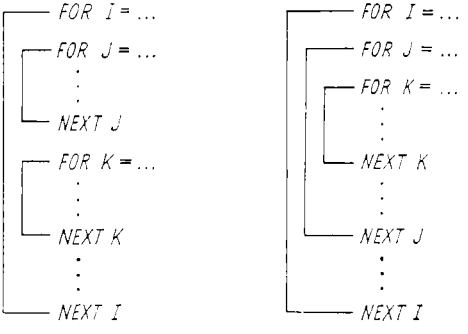


Bild 6.10. Schachtelung von Schleifen

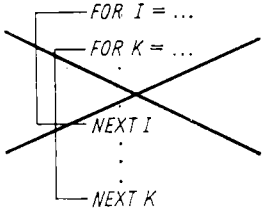


Bild 6.11. Unzulässige Schleifenüberlappung

Beispiel:

```
10  FOR I=1 TO 3
20  FOR J=1 TO 4
30  PRINT I*J,
40  NEXT J
50  PRINT
60  NEXT I
70  END
RUN

1       2       3       4
2       4       6       8
3       6       9       12

OK
```

Bei jedem Durchlauf der äußeren Schleife (Laufvariable I) wird die in diese eingebettete Laufanweisung (Laufvariable J) einmal vollständig ausgeführt, nimmt also die Laufvariable J nacheinander die Werte 1,2,3 und 4 an. Insgesamt wird die äußere Schleife dreimal, die innere zwölfmal durchlaufen. So entsteht das oben gezeigte Ausgabebild. (Die Spaltenabstände sind hier kleiner dargestellt, als es der üblichen Ausgabeform entspricht.)

Beispiel: Lottozahlengenerator

Um die Qual der Wahl beim Ausfüllen der Lotto-Tippscheine zu vermeiden, wird die Erzeugung entsprechender Zahlenfolgen dem Computer übertragen. Man gibt dazu folgende Daten über die Tastatur ein:

- die Anzahl der Werte je Tippschein (M)
- den Zahlenbereichsendwert (E)
- die Anzahl der zu erzeugenden Zahlenfolgen (N)

Sind beispielsweise 10 Tippreihen für das Spiel "6 aus 49" zu erzeugen, so ist die Zuweisung

M=6 ; E=49 ; N=10

vorzunehmen.

Diese Aufgabe läßt sich durch die Schachtelung zweier Schleifen lösen (Bild 6.12). Bei jedem der N Durchläufe der äußeren Schleife wird die innere Schleife M-mal ausgeführt und eine Folge von M Zufallszahlen aus dem vorgegebenen Bereich 1...E erzeugt. Im Programm 6.5 sind die beiden Schleifen durch FOR-NEXT-An-

weisungen realisiert. Die Berechnung der Zufallszahlen wird mit Hilfe der Standardfunktion RND ausgeführt. Diese liefert Zufallszahlen im Bereich von 0 bis 0,999..., die durch den Ausdruck INT(E*RND+1) auf ganze Zahlen im Bereich 1...E umgerech-

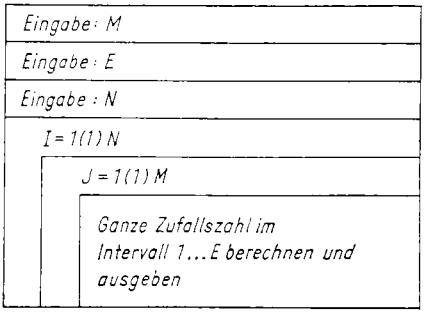


Bild 6.12. Lottozahlengenerator
M Anzahl der Werte je Folge
E Zahlenbereichsendwert
N Anzahl der Folgen

```
10  PRINT "LOTTOZAHLENGENERATOR"
20  PRINT "********************"
30  PRINT "ZAHL DER WERTE JE TIPPSCHEIN";
40  INPUT M
50  PRINT "ZAHLENBEREICHSENDWERT";
60  INPUT E
70  PRINT "ZAHL DER TIPPFOLGEN";
80  INPUT N
90  RANDOMIZE
100 FOR I=1 TO N
110 FOR J=1 TO M
120 PRINT INT(E*RND+1);"   ";
130 NEXT J
140 PRINT
150 NEXT I
160 END
RUN
LOTTOZAHLENGENERATOR
********************
ZAHL DER WERTE JE TIPPSCHEIN? 6
ZAHLENBEREICHSENDWERT? 49
ZAHL DER TIPPFOLGEN? 2
 15     12     39     27     16     4
 48     38     11     47     29     15
OK
```

Programm 6.5. Lottozahlengenerator

net werden. Die Anweisung RANDOMIZE bewirkt, daß sich bei einer Wiederholung der Programmausführung die erzeugte Folge von Zufallszahlen nicht wiederholt.
Es ist auch möglich, die Ausführung einer Laufanweisung schon vor dem Erreichen des Endwertes der Laufvariablen abzuschließen, indem der Zyklus mit Hilfe einer anderen Steueranweisung (z.B. IF-THEN) verlassen wird.
Der aktuelle Wert der Laufvariablen bleibt am Ende der Ausführung einer Laufanweisung erhalten und steht für die weitere Verarbeitung zur Verfügung. Im Falle der Beendigung durch Überschreiten des Endwertes hat die Laufvariable den Wert lv + sw. Dabei ist sw die Schrittweite und lv der Laufvariablenwert, mit dem der Schleifenkörper letztmalig durchlaufen wurde.
Der Eintritt in eine FOR-NEXT-Schleife ist in jedem Fall nur über die FOR-Anweisung zulässig, d.h., Sprünge "in eine Schleife" sind nicht erlaubt.
Schleifenanweisungen haben vor allem für die Arbeit mit Feldern Bedeutung. Weitere Anwendungsbeispiele sind daher im Abschn. 7. zu finden.

6.6. Haltanweisung [STOP]

Die Haltanweisung hat die Form

STOP .

Sie bewirkt, daß die Programmausführung gestoppt wird. In einem Programm dürfen beliebig viele Haltanweisungen stehen. Wird die Programmausführung durch eine Haltanweisung unterbrochen, so bleiben die Werte aller Variablen erhalten. Es ist somit auch möglich, diese Werte dann im Direktbetrieb mit Hilfe der PRINT-Anweisung ausgeben zu lassen (s. Abschn. 2.). Durch ein spezielles Kommando (oft CONTINUE oder CONT) kann die Programmausführung gegebenenfalls wieder fortgesetzt werden.

Beispiel:

```
      :
      :
120   IF X>0 THEN 140
```

```
130 STOP
140 LET B=LOG(X)
 .
 .
 .
```

In diesem Beispiel kommt es zum Programmhalt, wenn X Werte kleiner oder gleich Null annimmt und damit die Bedingung X>0 für die Berechnung des Funktionswertes LOG(X) verletzt.
Die Haltanweisung wird auch verwendet, um bei der schnell aufeinanderfolgenden Anzeige größerer Datenmengen auf dem Bildschirm die Programmausführung nach dem Beschreiben einer bestimmten Anzahl von Bildschirmzeilen zu unterbrechen und so dem Anwender Gelegenheit zu geben, die angezeigten Daten zu erfassen.

6.7. Programmendeanweisung [END]

Die Form der Programmendeanweisung ist

END .

Die END-Anweisung bewirkt die Beendigung der Programmausführung und die Rückkehr zur Kommandoebene des BASIC-Interpreters. Sie bildet zusammen mit einer Zeilennummer die Schlußzeile eines BASIC-Programms. Innerhalb des Programms darf die END-Anweisung (im Gegensatz zur STOP-Anweisung) nicht stehen. Abweichend von dieser Regel erlauben jedoch einige BASIC-Versionen, daß END-Anweisungen an beliebiger Stelle im Programm angeordnet, mehrfach verwendet und am (physischen) Programmende wahlweise eingesetzt werden dürfen.

7. Felder

7.1. Der Feldbegriff

Alle Variablen, die in den vorangegangenen Abschnitten benutzt wurden, repräsentieren Einzeldaten. Jeder dieser Variablen ist nur ein Wert zuordenbar. Man bezeichnet sie daher auch als einfache

oder skalare Variablen. Häufig ist es aber für die Verarbeitung zweckmäßig, eine Vielzahl von Daten zu einer geordneten Menge zusammenzufassen und einer einzigen Variablen zuzuordnen. Man nennt eine solche Menge gleichartiger Daten Feld (engl.: array) und eine Variable, der ein Datenfeld zugeordnet werden kann, *Feldvariable*. Eine Feldvariable wird, wie eine einfache Variable, durch einen Namen (Feldnamen) bezeichnet.
Der Zugriff auf ein bestimmtes Element eines Feldes ist durch Angabe des Feldnamens mit einer in Klammern angehängten Indexfolge möglich. Die Indizes bestimmen die Position des Elementes im Feld. Die Kombination von Feldname und Indexfolge bildet demnach den Namen einer Variablen, die (wie eine einfache Variable) einen einzelnen Wert vertritt und *indizierte Variable* genannt wird. Die Anzahl der zur Bezeichnung eines Feldelementes erforderlichen Indizes entspricht der Dimension des Feldes.

Eindimensionale Felder

Bei diesen wird zur Beschreibung der Ordnung (der Reihenfolge der Feldelemente) ein Index verwendet. In Bild 7.1a ist ein eindimensionales Feld schematisch dargestellt. In diesem Feld beginnt die Indexzählung mit Null, wie das in BASIC meist üblich ist. Eindimensionale Felder nennt man auch Vektoren.

Zweidimensionale Felder

Zweidimensionale Felder werden dargestellt, indem man deren Datenelemente in Zeilen und Spalten ordnet. Die Position eines Feldelements wird durch zwei Indizes gekennzeichnet, von denen in der Regel der erste die Zeile, der zweite die Spalte benennt. Beispielsweise ist die indizierte Variable A(2,3) das Element aus Zeile 2 und Spalte 3 des Feldes A (Bild 7.1b). Für zweidimensionale Felder ist auch die Bezeichnung Matrizen gebräuchlich.

Mehrdimensionale Felder

Außer ein- und zweidimensionalen Feldern können auch solche mit höherer Dimension gebildet werden. In einem n-dimensionalen Feld wird jedes Element durch n Indizes bezeichnet. Für den zulässigen Höchstwert von n existieren in den einzelnen Sprachversionen unterschiedliche Festlegungen.

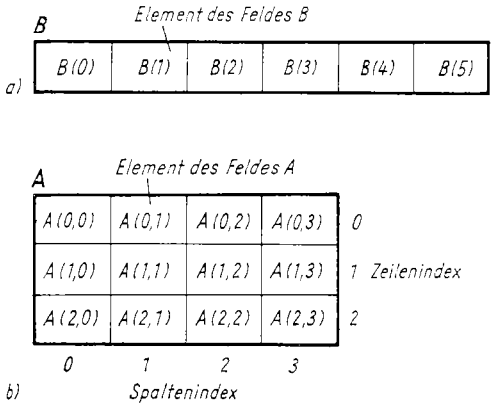


Bild 7.1. Felder (Beispiele)
a) eindimensionales Feld mit 6 Elementen
b) zweidimensionales Feld mit 3 Zeilen und 4 Spalten

7.2. Darstellung der Felder in BASIC [DIM, OPTION]

Felder werden in Minimal-BASIC nur durch einen einzelnen Buchstaben benannt. Sie können ein- oder zweidimensional sein und dürfen nur numerische Daten (Zahlen) enthalten. Viele BASIC-Systeme lassen jedoch Zeichenkettenfelder zu, unterscheiden sich aber bezüglich der Behandlung dieser Felder zum Teil erheblich. In manchen Systemen sind auch Felder höherer Dimension und längere Feldnamen erlaubt. Größe und Struktur eines Feldes kann man mit einer Dimensionierungsanweisung festlegen.
Feldelemente werden durch den Feldnamen mit angehängten Indizes, die man in (runden) Klammern hinter dem Namen notiert, bezeichnet. Die Indizes können auch als numerische Ausdrücke angegeben werden. Korrekte Bezeichnungen indizierter Variabler sind demnach
- bei eindimensionalen Feldern: A(3), B(X), Y(I*N+J)
- bei zweidimensionalen Feldern: D(4,5), E(I,K), M(3,2*B+1).

Enthält eine Anweisung indizierte Variablen, deren Indizes als numerische Ausdrücke angegeben sind, so werden beim Ausführen der Anweisung erst die Werte der Indizes bestimmt und dann folgt der Zugriff auf die indizierten Variablen.

Dimensionierung von Feldern

Für die Speicherung der Feldelemente muß im Computer eine entsprechende Zahl von Speicherplätzen reserviert werden. Dazu dient die Dimensionierungsanweisung. Sie hat die Formen

- für eindimensionale Felder:

```
DIM feldname(konstante)
```

- für zweidimensionale Felder:

```
DIM feldname(konstante,konstante)
```

Die hinter dem Feldnamen in Klammern angegebenen ganzzahligen numerischen Konstanten bezeichnen die Größtwerte der entsprechenden (stellengleichen) Indizes. Der Anfangswert der Indexzählung ist im allgemeinen auf Null festgelegt. In einer DIM-Anweisung können (durch Komma getrennt) mehrere Felder dimensioniert werden. Zu beachten ist, daß Variablen für die Festlegung der Feldgrenzen in der Regel nicht zugelassen sind.

Beispiel:

```
10 DIM B(5),A(2,3),X(100),T□(40)
```

Diese Anweisung vereinbart:

- ein eindimensionales Feld B mit den sechs numerischen Elementen
  B(0), B(1),..., B(5)
- eine Matrix A mit den 12 numerischen Elementen
  A(0,0), A(0,1), A(0,2), A(0,3)
  A(1,0), A(1,1), A(1,2), A(1,3)
  A(2,0), A(2,1), A(2,2), A(2,3)
- einen Vektor X mit den numerischen Elementen X(0), X(1),..., X(100)
- ein eindimensionales Zeichenkettenfeld T□ mit den 41 Elementen T□(0),...,T□(40)

Werden die Grenzen eines Feldes nicht in einer DIM-Anweisung festgelegt, dann benutzt der Computer eine Standardvereinbarung

DIM feldname(10) bzw. DIM feldname(10,10). Damit sind für ein eindimensionales Feld 11 und für ein zweidimensionales $11 \cdot 11 = 121$ Speicherplätze reserviert. Werden die Grenzen eines Feldes in einer DIM-Anweisung vereinbart, so muß diese im Programm vor dem ersten Zugriff auf ein Element des betreffenden Feldes stehen.

OPTION-Anweisung

Der kleinste Wert für die Indizierung aller Feldvariablen ist Null (Standardvereinbarung). Mit der OPTION-Anweisung kann die untere Indexgrenze neu festgelegt werden. Sie hat die Form

OPTION BASE n .

Dabei ist n entweder 0 oder 1. Durch die Anweisung OPTION BASE 1 wird als kleinster Indexwert Eins vereinbart.

Beispiel:

```
10 OPTION BASE 1
20 DIM B(5),A(2,3)
```

Diese Anweisungen vereinbaren:

- einen Vektor B mit den 5 Elementen
  B(1), B(2),..., B(5)
- eine Matrix A mit den 6 Elementen
  A(1,1), A(1,2), A(1,3)
  A(2,1), A(2,2), A(2,3).

7.3. Verarbeitung von Feldern

Die Verarbeitung von Feldern erfolgt durch die entsprechende Verarbeitung der Feldelemente mit Hilfe der bisher beschriebenen Anweisungen. Dabei treten an die Stelle der einfachen Variablen indizierte Variable. Die folgenden Beispiele sollen das verdeutlichen.

Beispiel:

Die Elemente A(1), A(2) und B(2) zweier eindimensionaler Felder bekommen mit Hilfe einer LET- bzw. INPUT-Anweisung Werte

zugewiesen. Anschließend wird ein aus diesen Elementen gebildeter numerischer Ausdruck berechnet, das Ergebnis dem Feldelement C(2) zugeordnet und ausgegeben.

```
110 LET A(1)=5
120 INPUT A(2),B(2)
130 LET C(2)=(A(2)+B(2))*A(1)
140 PRINT C(2)
```

Beispiel:

Die 10 Elemente X(0), ..., X(9) eines eindimensionalen Feldes werden mit Null belegt.

```
10 FOR I=0 TO 9
20 LET X(I)=0
30 NEXT I
40 END
```

Der Index I durchläuft die Werte von 0 bis 9, so daß nacheinander allen Feldelementen X(0) bis X(9) der Wert 0 zugewiesen wird. An diesem Beispiel erkennt man den Vorteil, den die Zusammenfassung von Einzeldaten in Feldern selbst bei kleinen Datenmengen bietet. Dadurch wird es möglich, Operationen, die nacheinander in gleicher Weise mit verschiedenen Größen auszuführen sind, in einer Programmschleife durch die gleiche Anweisung bzw. Anweisungsfolge zu realisieren.

Beispiel:

Die Zeilen 3 und 5 eines zweidimensionalen Zahlenfeldes mit 8 Spalten werden vertauscht.

```
10 FOR K=0 TO 7
20 LET Z=A(3,K)
30 LET A(3,K)=A(5,K)
40 LET A(5,K)=Z
50 NEXT K
```

Bei jedem der 8 Durchläufe der Schleife werden die Elemente in einer Spalte vertauscht. Die jeweilige Spaltennummer wird durch die Laufvariable K bezeichnet.

Beispiel:
Alle Elemente einer Matrix A mit 5 Zeilen und 8 Spalten sind mit dem Zahlenwert "1" zu belegen.

```
10 FOR I=0 TO 4
20 FOR K=0 TO 7
30 LET A(I,K)=1
40 NEXT K
50 NEXT I
60 END
```

Operationen mit allen Elementen eines zweidimensionalen Feldes erreicht man durch die Schachtelung von zwei Laufanweisungen, von denen im Beispiel die äußere den Zeilenindex, die innere den Spaltenindex festlegt. Die Wertzuweisung erfolgt aus diesem Grunde zeilenweise, d.h., nach einer vollständigen Ausführung der inneren Schleifenanweisung ist eine Zeile mit "1" belegt worden und der Zeilenindex wird erhöht.

Beispiel:
Aus den Elementen zweier Felder A und B mit je 6 Zeilen und 10 Spalten sind die Elemente eines Feldes C nach der Vorschrift

$$
\begin{aligned}
C(0,0) &= A(0,0) + B(0,0)\\
C(1,0) &= A(1,0) + B(1,0)\\
C(2,0) &= A(2,0) + B(2,0)\\
&\vdots\\
C(I,K) &= A(I,K) + B(I,K)\\
&\vdots\\
C(5,9) &= A(5,9) + B(5,9)
\end{aligned}
$$

zu berechnen (Matrizenaddition). Diese Aufgabe wird durch folgendes Programm gelöst:

```
10 FOR K=0 TO 9
20 FOR I=0 TO 5
30 LET C(I,K)=A(I,K)+B(I,K)
40 NEXT I
50 NEXT K
60 END
```

Dieses Programm hat die gleiche Struktur wie das vorangegangene (zwei geschachtelte Laufanweisungen). Das resultiert aus der weitgehenden Übereinstimmung der Aufgabenstellungen, nämlich den Elementen eines zweidimensionalen Feldes einen Wert zuzuordnen. Im vorliegenden Beispiel ist dieser nicht konstant, sondern wird berechnet. Unterschiedlich ist auch die Reihenfolge der Wertzuweisungen. Die äußere FOR-NEXT-Anweisung gibt den Spaltenindex vor, deshalb werden die Werte der Feldelemente spaltenweise berechnet.

Felder haben für das Arbeiten mit BASIC sehr große Bedeutung, weil viele praktische Aufgaben nur mit dieser Form der Datenorganisation effektiv gelöst werden können. Im Anschluß an die einführenden Beispiele sollen deshalb weitere Programme die Möglichkeiten der Feldverarbeitung demonstrieren. Die folgenden vier Beispiele werden als Einzelaufgaben beschrieben, sind jedoch inhaltlich aufeinander abgestimmte Teile der komplexen Aufgabenstellung "Prüfungsauswertung". Die Teilaufgaben entsprechen den Verarbeitungsstufen

- Datenfeldeingabe
- Berechnung der Durchschnittsnoten
- Alphabetische Sortierung
- Datenfeldausgabe.

Diese sind bei der Prüfungsauswertung in der genannten Reihenfolge zu durchlaufen. Die zugehörigen Programme 7.1 bis 7.4 sind daher eigentlich Teilprogramme. Bis auf das Programm 7.1 "Datenfeldeingabe" arbeiten alle anderen mit Daten, die von den vorangegangenen bereitgestellt werden.

Beispiel: Datenfeldeingabe

Für jeden Schüler einer Gruppe liegen Q Prüfungsnoten vor. Zwecks nachfolgender Verarbeitung durch den Computer sollen die Namen der Schüler in einem eindimensionalen Feld N$ und die Noten in einem zweidimensionalen Feld Z gespeichert werden. Die vorgesehene Form der Datenspeicherung ist schematisch im Bild 7.2 dargestellt. Der Index bzw. Zeilenindex gibt die Zuordnung von Namen und Noten an.

Den Algorithmus für die Dateneingabe zeigt Bild 7.3. Die Anzahl der Schüler ist nicht vorgegeben. Sie wird ermittelt, indem der Wert einer Zählvariablen L, der anfangs auf Null gesetzt ist, nach jeder Eingabe eines Namens um eins erhöht wird. Vor dem Weiterstellen der Variablen L werden in einer Schleife die Q Einzelnoten

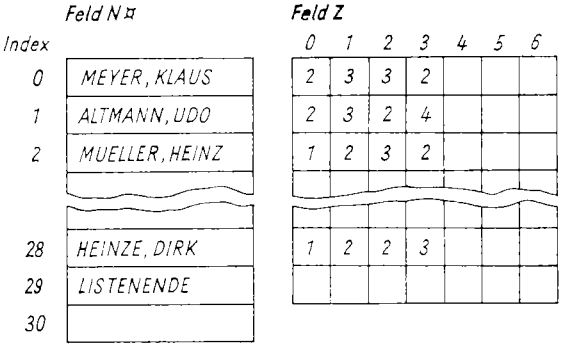


Bild 7.2. Aufbau der Felder N$() und Z

```
Eingabe: Q
L := 0
Eingabe: N$(L)
    K = 0 (1) Q - 1
        Eingabe: Z(L,K)
    L := L + 1
    Eingabe: N$(L)
N$(L) <> "LISTENENDE"
```

Bild 7.3. Datenfeldeingabe

Q	Anzahl der Noten	N$(L)	Name
L	Schüleranzahl	Z(L,K)	Note

des Schülers übernommen und im Feld Z gespeichert. Das Ende der Dateneingabe wird dadurch gekennzeichnet, daß statt eines Schülernamens die Zeichenkette "LISTENENDE" eingegeben wird.

Im zugehörigen Programm 7.1 werden zu Beginn die Feldgrenzen vorgegeben. Es ist nicht möglich, die Anzahl der für die Felder N$() und Z zu reservierenden Speicherplätze in Abhängigkeit von der tatsächlichen Zahl der Namen und Noten festzulegen, weil dafür nur numerische Konstanten zugelassen sind. Aus diesem Grunde werden in der DIM-Anweisung Feldgrößen vereinbart, die die bei der praktischen Nutzung des Programms möglichen

Grenzfälle berücksichtigen. Im vorliegenden Beispiel wird von maximal 30 Schülern und 6 Prüfungsnoten (Einzelnoten) je Schüler ausgegangen und im Feld Z außerdem eine Spalte für die Durchschnittsnote jedes Schülers reserviert (s. nächstes Beispiel).

```
10  REM DATENFELDEINGABE
20  DIM N$(30),Z(29,6)
30  PRINT "ZAHL DER NOTEN JE SCHUELER";
40  INPUT Q
50  LET L=0
60  PRINT "SCHUELER";
70  INPUT N$(L)
80  PRINT "NOTEN:"
90  FOR K=0 TO Q-1
100 INPUT Z(L,K)
110 NEXT K
120 LET L=L+1
130 PRINT "SCHUELER";
140 INPUT N$(L)
150 IF N$(L)<>"LISTENENDE" THEN 80
160 END
RUN
ZAHL DER NOTEN JE SCHUELER? 4
SCHUELER? "MEYER,KLAUS"
NOTEN:
? 2
? 3
? 3
? 2
SCHUELER? "ALTMANN,UDO"
NOTEN:
? 2
? 3
? 2
? 4
SCHUELER? "MUELLER,HEINZ"
NOTEN:
? 1
? 2
? 3
? 2
SCHUELER? "HEMPEL,REINER"
NOTEN:
? 1
? 2
? 1
? 1
SCHUELER? LISTENENDE
OK
```

Programm 7.1. Datenfeldeingabe

Demnach sind für das Feld N$ 31 Elemente (30 Namen und die Zeichenkette "LISTENENDE") und für das Feld Z 30 Zeilen und 7 Spalten vorzusehen. Die Belegung der Felder N$ und Z nach Ausführung des Programms ist für einen konkreten Fall (29 Schüler und 4 Einzelnoten je Schüler) im Bild 7.2 dargestellt.
Im vorliegenden Beispiel wird davon ausgegangen, daß die Namen bei der Eingabe in Begrenzerzeichen (Anführungszeichen) eingeschlossen sind. Prinzipiell besteht die Möglichkeit, darauf zu verzichten und somit die Eingabe zu vereinfachen. Dabei ist jedoch zu bedenken, daß das Komma dann als Trennzeichen wirkt, d.h. die Eingabe einer Zeichenkette abschließt. Familienname und Vorname müßte man deshalb in diesem Fall als einzelne Zeichenketten übernehmen und zwei Variablen zuweisen. Erst danach könnten sie zwecks einfacherer Verarbeitung im Computer zu einer Zeichenkette verknüpft werden. Programm 7.1 wäre wie folgt zu ändern:

```
70 INPUT F$,V$
75 LET N$(L)=F$+","+V$
```

Beispiel: Berechnung von Durchschnittsnoten

Die Prüfungsnoten einer Gruppe von L Schülern sind in einem Feld Z zusammengefaßt. Für jeden Schüler existieren Q Noten, die in einer Zeile des Feldes Z gespeichert sind (Bild 7.5). Aus diesen Daten soll die Durchschnittsquote jedes Schülers und die

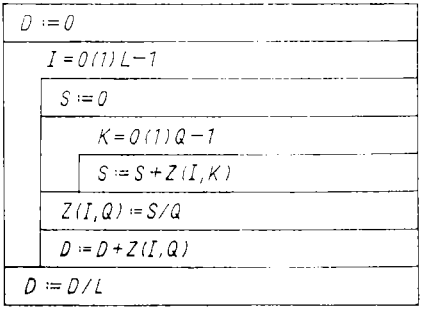


Bild 7.4. Durchschnittsnotenberechnung

L	Anzahl der Schüler	Z(I,K)	Note
Q	Anzahl der Noten	Z(I,Q)	Durchschnittsnote
D	Durchschnittsnote der Gruppe		

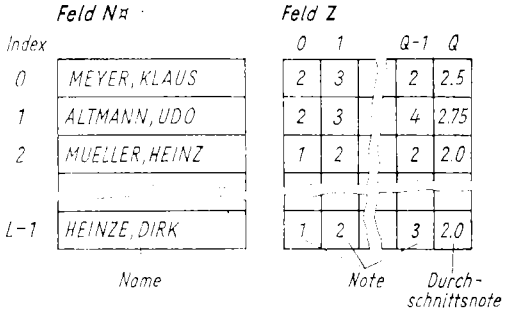


Bild 7.5. Belegung der Felder N○ und Z vor der Sortierung (Beispiel)

```
160 REM BERECHNUNG DER
170 REM DURCHSCHNITTSNOTEN
180 LET D=0
190 FOR I=0 TO L-1
200 LET S=0
210 FOR K=0 TO Q-1
220 LET S=S+Z(I,K)
230 NEXT K
240 LET Z(I,Q)=S/Q
250 LET D=D+Z(I,Q)
260 NEXT I
270 LET D=D/L
280 END
```

Programm 7.2. Durchschnittsnotenberechnung

Durchschnittsnote der gesamten Gruppe berechnet werden. Der Algorithmus, der im wesentlichen aus zwei geschachtelten Schleifen besteht, ist im Bild 7.4 dargestellt. In der inneren Schleife wird die Summe S der Noten Z(I,0) bis Z(I,Q−1) eines Schülers gebildet. Daraus wird in der äußeren Schleife die Durchschnittsnote bestimmt und ebenfalls im Feld Z (Zeile I, Spalte Q) abgelegt. Die Summierung der einzelnen Durchschnittsnoten Z(I,Q) dient der Berechnung der Durchschnittsnote D der Gruppe.
Das zugehörige Programm 7.2 baut auf dem Programm 7.1 auf. Es wird davon ausgegangen, daß die Variablenwerte L und Q sowie die Felder N○ und Z durch dieses Programm bereitgestellt werden. Auch in der Wahl der Zeilennummern ist die Abhängigkeit sichtbar.

Beispiel: Alphabetische Sortierung

Diesem Beispiel liegt der im Bild 7.5 dargestellte Feldaufbau zugrunde. Im Feld N□ sind die Namen von L Schülern einer Gruppe zusammengefaßt. Das Feld Z enthält Daten (Noten), die diesen Namen zeilenweise zugeordnet sind. Zum Feldelement N□(I) gehören demnach die Daten, die in der Zeile I des Feldes Z stehen. Die Felder N□ und Z sind so zu sortieren, daß im Feld N□ die alphabetische Reihenfolge der Daten hergestellt wird, aber trotzdem die Zuordnung von Namen und Noten erhalten bleibt. Dazu wird folgendes Verfahren angewendet: Alle Zeichenketten, die den Feldelementen N□(0) bis N□(L−1) zugeordnet sind, werden verglichen. Auf diese Weise erhält man das Element N□(X), dessen Zeichenkette die erste in der alphabetisch sortierten Folge der Werte der Feldelemente N□(0) bis N□(L−1) ist. Sofern N□(X) nicht mit N□(0) identisch ist, werden die Daten der Zeilen X und 0 in beiden Feldern N□ und Z vertauscht. Im weiteren Sortierablauf müssen nur noch die Werte der Feldelemente N□(1) bis N□(L−1) verglichen werden. Wieder wird unter diesen das Element N□(X) bestimmt, dessen Zeichenkette in der alphabetischen Folge am Anfang steht (jetzt bezogen auf die Feldelemente N□(1) bis N□(L−1)!). Wenn N□(X)≠N□(1) gilt, dann werden die Zeilen X und 1 vertauscht. Der Wert des Elementes N□(1) des sortierten Feldes ist somit festgelegt, und es wird bei der folgenden Sortierung ebenfalls nicht mehr berücksichtigt.

Dieser Ablauf ist als Algorithmus im Bild 7.6a angegeben und für ein konkretes Beispiel im Bild 7.7 dargestellt. Vereinfachend werden im Beispiel als Namen einzelne Buchstaben verwendet. Außerdem wird nur das Feld N□ betrachtet. Die durch den Wert der Laufvariablen I vorgegebene linke Sortiergrenze (Algorithmus im Bild 7.6a) ist im Bild 7.7 mit LS bezeichnet. Im Anfangszustand ist I=0. Es wird daher unter allen Feldelementen N□(0) bis N□(4) jenes Element N□(X) gesucht, dessen Wert in der alphabetischen Folge aller Zeichenketten am Anfang steht. Die Suche führt auf N□(4) mit dem Wert "A", d.h., X=4. Da X≠I, werden im nächsten Schritt (Algorithmus Bild 7.6a) die Werte der Feldelemente N□(4) und N□(0) vertauscht. Es ergibt sich der Zustand 2 im Bild 7.7. Nach der Erhöhung von I auf Eins wird der Sortierzyklus erneut durchlaufen, wobei jetzt unter den Feldelementen N□(1) bis N□(4) jenes mit dem kleinsten Wert zu suchen ist. Das ermittelte Element N□(X)=N□(1) hat den

Wert "B". Weil X=I=1, wird keine Vertauschung vorgenommen. (Die alphabetische Reihenfolge ist an dieser Stelle schon vorhanden.) Es existiert nun Zustand 3 nach Bild 7.7. I wird wiederum erhöht und nimmt den Wert 2 an. Es folgt die Suche von N□(X) im Bereich N□(2) bis N□(4) usw.
Die Umsetzung des Algorithmus nach Bild 7.6b in BASIC-An-

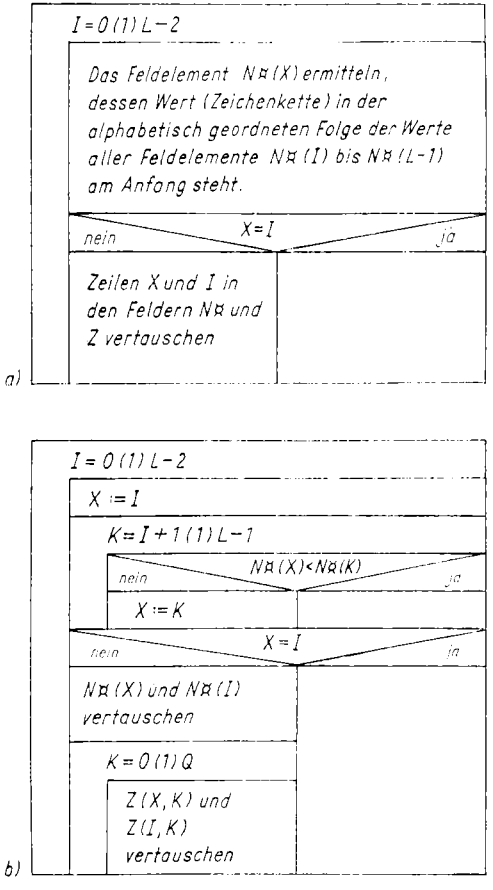


Bild 7.6. Alphabetische Sortierung in (a) Grob- und (b) Feindarstellung

L	Anzahl der Schüler	N□(I)	Name
Q	Anzahl der Noten	Z(I,K)	Note

weisungen führt zum (Teil-)Programm 7.3. Es schließt an die (Teil-)Programme 7.1 und 7.2 an und arbeitet mit den Variablenwerten, die von diesen Programmen bereitgestellt werden.

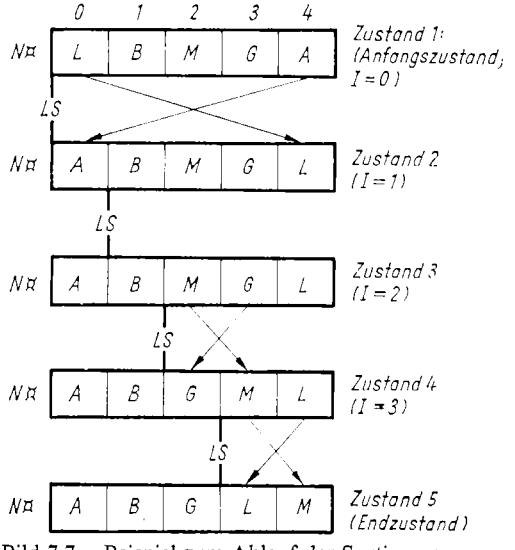


Bild 7.7. Beispiel zum Ablauf der Sortierung
(LS Linke Sortiergrenze)

```
280 REM ALPHABETISCHE SORTIERUNG
290 FOR I=0 TO L-2
300 LET X=I
310 FOR K=I+1 TO L-1
320 IF N¤(X)<N¤(K) THEN 340
330 LET X=K
340 NEXT K
350 IF X=I THEN 450
360 REM VERTAUSCHUNG
370 LET T¤=N¤(I)
380 LET N¤(I)=N¤(X)
390 LET N¤(X)=T¤
400 FOR K=0 TO Q
410 LET H=Z(I,K)
420 LET Z(I,K)=Z(X,K)
430 LET Z(X,K)=H
440 NEXT K
450 NEXT I
460 END
```

Programm 7.3. Alphabetische Sortierung

Beispiel: Datenfeldausgabe

Die Daten der Felder N$ und Z (Bild 7.5) sollen in folgender Form ausgegeben werden:

NR.	NAME	NOTEN	DURCH-SCHNITT
1	ALTMANN,UDO	2 3 2 4	2.75
2	.		.
.	.		.
.	.		.

Programm 7.4 löst diese Aufgabe gemäß dem Algorithmus im Bild 7.8. Die Einteilung des Ausgabebildes in Spalten erfolgt mit Hilfe der TAB-Funktion. Wegen der variablen Anzahl der Einzelnoten wird die Position der Spalte "DURCHSCHNITT" in Abhängigkeit von der Notenzahl berechnet. Bei der vorgesehenen Maximalzahl von 30 Schülern ist obige Tabelle nicht in jedem Fall geschlossen auf dem Bildschirm darstellbar (begrenzte Zeilenzahl !). Das berücksichtigen die Programmzeilen 570 und 580. Die STOP-Anweisung unterbricht die Programmausführung, nachdem die Noten von 15 Schülern ausgegeben wurden. Auf die Eingabe des Fortsetzungskommandos (bei vielen BASIC-Interpretern CONTINUE oder CONT) folgt dann die Ausgabe des zweiten Teils der Tabelle.

Die Programme 7.1 bis 7.4 bilden zusammen das Programm "Prüfungsauswertung", dessen Ausführungsprotokoll im Bild 7.9 wiedergegeben ist.

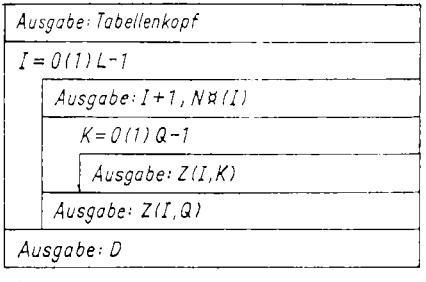


Bild 7.8. Datenfeldausgabe

L Anzahl der Schüler N$(I) Name
Q Anzahl der Noten Z(I,K) Note
D Durchschnittsnote der Gruppe

```
460 REM DATENAUSGABE
470 PRINT
480 PRINT "NR.";TAB(6);"NAME";TAB(28);
490 PRINT "NOTEN";TAB(33+3*Q);"DURCHSCHN."
500 PRINT
510 FOR I=0 TO L-1
520 PRINT I+1;TAB(6);N$(I);TAB(27)
530 FOR K=0 TO Q-1
540 PRINT Z(I,K);" ";
550 NEXT K
560 PRINT TAB(33+3*Q);Z(I,Q)
570 IF I<>14 THEN 590
580 STOP
590 NEXT I
600 PRINT
610 PRINT TAB(14+3*Q);"GESAMTDURCHSCHNITT=";D
620 END
```

Programm 7.4. Datenfeldausgabe

```
RUN
ZAHL DER NOTEN JE SCHUELER? 4
SCHUELER? "MEYER,KLAUS"
NOTEN:
? 2
? 3
? 3
? 2
SCHUELER? "ALTMANN,UDO"
NOTEN:
? 2
? 3
? 2
? 4
SCHUELER? "MUELLER,HEINZ"
NOTEN:
? 1
? 2
? 3
? 2
SCHUELER? "HEMPEL,REINER"
NOTEN:
? 1
? 2
? 1
? 1
SCHUELER? LISTENENDE

NR.  NAME                 NOTEN              DURCHSCHN.

 1   ALTMANN,UDO          2   3   2   4      2.75
 2   HEMPEL,REINER        1   2   1   1      1.25
 3   MEYER,KLAUS          2   3   3   2      2.5
 4   MUELLER,HEINZ        1   2   3   2      2

                      GESAMTDURCHSCHNITT= 2.125
OK
```

Bild 7.9. Ausführungsprotokoll des Programms „Prüfungsauswertung“

8. Wertzuweisung über Datenliste

8.1. Definition der Datenliste [DATA]

Neben den bisher behandelten Formen der Wertzuweisung bietet BASIC die Möglichkeit, den Variablen Werte aus einer Datenliste zuzuordnen. Die Folge der Elemente der Datenliste wird mit Hilfe von DATA-Anweisungen festgelegt. Die DATA-Anweisung hat die Form

DATA konstante,konstante,...,konstante

Beispiel:

```
10 DATA 1,2,3,-7.5
20 DATA 100,200,300
```

Diese beiden Anweisungen definieren in der Datenliste die Folge von Datenelementen 1,2,3,−7.5,100,200,300. Die Anordnung der Datenlistenelemente stimmt dabei mit der Reihenfolge ihrer Notierung im Programm bzw. in den DATA-Anweisungen überein.
In gleicher Weise können auch Zeichenkettenkonstanten als Elemente einer Datenliste definiert werden.

Beispiel:

```
100 DATA "JANUAR","KOCH,REINER","5"
```

Diese Anweisung erzeugt die Datenfolge:

"JANUAR","KOCH,REINER","5"

Die Begrenzerzeichen von Zeichenkettenkonstanten können bei der Notierung in DATA-Anweisungen im allgemeinen dann entfallen, wenn die Konstanten keine anderen Zeichen als Ziffern, Buchstaben, Punkt, Plus- und Minuszeichen enthalten. Für obiges Beispiel ist unter diesen Bedingungen auch die Notierung

```
100 DATA JANUAR,"KOCH,REINER",5
```

zulässig.

DATA-Anweisungen können an jeder beliebigen Stelle des Programms stehen, und es ist auch nicht erforderlich, daß alle DATA-Anweisungen direkt aufeinander folgen. Meist werden sie jedoch am Ende oder am Anfang des Programms notiert, weil diese Anordnung am übersichtlichsten ist und somit evtl. Änderungen der Datenfolge einfach zu verwirklichen sind.

8.2. Lesen der Datenliste [READ]

Mit der READ-Anweisung können die Elemente der Datenliste Variablen zugewiesen werden. Man sagt dazu auch, die Elemente der Datenliste werden gelesen (engl.: read – lesen). Die READ-Anweisung hat die Form

READ variable,variable,...,variable

Sie ordnet der ersten Variablen das erste Element der Datenliste, der zweiten Variablen das zweite Element usw. zu.

Beispiel:

```
10 DATA 1,7,-11,-5,37
20 READ A,B,C
```

Diese Anweisungsfolge erzeugt die Zuordnung A=1; B=7; C=−11.
In einem Programm können mehrere READ-Anweisungen stehen. Jede der READ-Anweisungen setzt dann das Lesen an der Stelle der Datenliste fort, an der es die vorhergehende beendete. Die im Laufe der Programmausführung jeweils aktuelle Leseposition markiert der Computer durch einen sogenannten Zeiger. Dieser Zeiger wird vom Start-Kommando RUN auf das erste Listenelement gesetzt und nach jeder Ausführung einer READ-Anweisung um soviel Positionen weitergestellt, wie Variable gelesen wurden.

Beispiel:

```
10  DATA 3,19
20  READ A,B,X1
 .
 .
 .
150  READ Y(1),Y(2)
160  DATA 0.01,3.5,2,1.5E+16
 .
 .
 .
```

Nach Ausführung der READ-Anweisungen haben die Variablen folgende Werte angenommen:

A	B	X1	Y(1)	Y(2)
3	19	0.01	3.5	2

Wird nach der Zuordnung des letzten Wertes der Datenliste eine weitere READ-Anweisung wirksam, so gibt der Computer eine Fehlermitteilung aus. Ein Fehler wird auch angezeigt, wenn beim Zugriff auf die Datenliste der Typ des Elements (Zahl bzw. Zeichenkette) nicht zum Typ der Variablen paßt. Zahlen können dabei, wie das folgende Beispiel zeigt, je nach Art der Variablen als Zahlenwert oder als Zeichenkette zugewiesen werden.

Beispiel:

```
10  DATA "MUELLER,MAX",2,1949,15,APRIL
20  READ N$,K,B,T$,M$
```

Folgende Zuweisung wird damit realisiert:

N$	K	B	T$	M$
MUELLER,MAX	2	1949	15	APRIL

8.3. Rücksetzen des Datenzeigers [RESTORE]

Die Anweisung

RESTORE

setzt den Zeiger der Datenliste auf deren erstes Element, unabhängig davon, an welcher Position er sich befindet. Damit können die gleichen Daten mehrmals Variablen zugewiesen werden.

Beispiel:

```
10 DATA 1,2,3
20 DATA 10,20,30
30 READ A,B
40 RESTORE
50 READ U,V,W,X,Y
```

Nach Ausführung dieser Anweisungen besteht die Zuordnung

A	B	U	V	W	X	Y
1	2	1	2	3	10	20

Beispiel: Berechnung von Prüfungsnoten

Die bei einer Prüfung von den Studenten erreichten Punkte sollen in Noten umgerechnet werden. Dafür ist folgender Schlüssel vorgegeben:

Mindestpunkte D %	96	80	60	40	0
Note N	1	2	3	4	5

Die angegebenen Mindestpunkte D sind Prozentwerte der erreichbaren maximalen Punktzahl M. Bild 8.1 beschreibt den zur Lösung dieser Aufgabe verwendeten Algorithmus. Dieser setzt voraus, daß die Prozentzahlen der Mindestpunkte in einer Datenliste in der Reihenfolge 96, 80, 60, 40, 0 notiert sind. Auf die Eingabe einer Punktzahl P folgt die Berechnung des entsprechenden Prozentwertes R, bezogen auf den anfangs eingegebenen Maximalwert M. In der inneren Schleife wird dann jeweils ein Datenlistenelement D, beginnend mit dem ersten, gelesen und mit R vergli-

chen. Dieser Vorgang wiederholt sich, solange R kleiner als D ist. Bei jedem Schleifendurchlauf wird N um eins erhöht. Wenn die Schleife verlassen wird, hat N den Wert der Note, die der Punktzahl P entspricht. Darauf folgt die Ausgabe von N und das Rück-

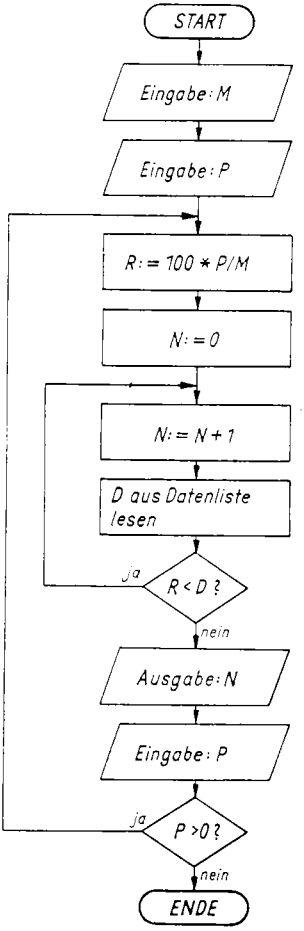


Bild 8.1. Berechnung von Prüfungsnoten

D	Vergleichswert aus Liste	P	Punktzahl
M	maximale Punktzahl	N	Note

setzen des Zeigers, der den Zugriff auf die Datenliste steuert. Die Wiederholung der beschriebenen Aktionen mit einem neuen Wert P ist somit vorbereitet. Es schließt sich die Eingabe von P an, auf die abhängig vom eingegebenen Zahlenwert die erneute Ausführung des Algorithmus oder die Verzweigung zum Ende folgt. Als Schlußkennzeichen dient eine beliebige Punktzahl gleich oder kleiner als Null. Im entsprechenden Programm 8.1 wird die Datenliste mit Hilfe einer DATA-Anweisung definiert und das Lesen der Listenelemente durch eine READ-Anweisung realisiert. Das Programm macht sichtbar, daß die Anweisungen READ, DATA und RESTORE außer für Mehrfachzuweisungen besonders für Wertzuweisungen in Programmschleifen vorteilhaft einsetzbar sind.

```
10  REM PRUEFUNGSNOTEN BESTIMMEN
20  PRINT "PUNKTZAHL ";
30  INPUT P
40  LET N=0
50  LET N=N+1
60  READ D
70  IF P<D THEN 50
80  PRINT "NOTE :";N
90  RESTORE
100 PRINT
110 PRINT "PUNKTZAHL ";
120 INPUT P
130 IF P>0 THEN 40
140 DATA 96,80,60,40,0
150 END
RUN
PUNKTZAHL ? 91
NOTE : 2

PUNKTZAHL ? 47
NOTE : 4

PUNKTZAHL ? 0
OK
```

Programm 8.1. Berechnung von Prüfungsnoten

Obiges Programm unterscheidet sich vom Algorithmus nach Bild 8.1 durch die Vorgabe M = 100 = konstant. Der Leser möge die Modifikation für M ≠ 100 selbst vornehmen.

Beispiel: Vokabel-Trainer Englisch-Deutsch

Von zwei Feldern soll das eine englische Vokabeln (Feld E○) und das andere die entsprechenden deutschen Worte (Feld D○)

enthalten. Zunächst werden diese Felder aus Elementen einer Datenliste so aufgebaut, daß Elemente gleicher Position in beiden Feldern den gleichen Begriff darstellen (Bild 8.2 und Programm 8.2). Anschließend folgt die Auswahl eines Feldelementes E□(K) mit Hilfe einer Zufallszahl K und die Anzeige des betreffenden englischen Wortes auf dem Bildschirm. Danach ist das entsprechende deutsche Wort einzugeben, das mit der im Feldelement D□(K) gespeicherten Übersetzung verglichen wird. Besteht Übereinstimmung, so schließen sich Auswahl und Ausgabe des nächsten englischen Wortes an. Anderenfalls erscheint die Information "FALSCH" und die richtige Übersetzung wird angezeigt.

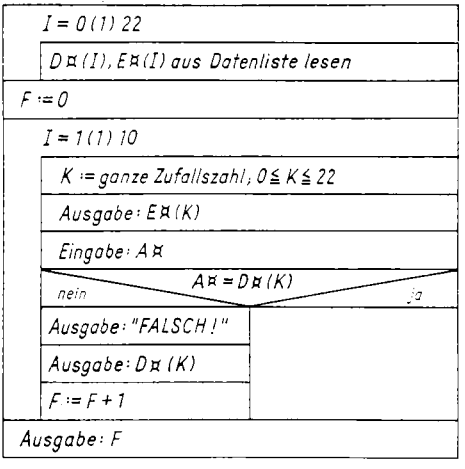


Bild 8.2. Vokabel-Trainer
D□ Feld der deutschen Worte
E□ Feld der englischen Worte
F Anzahl der Fehler

Nach Ausgabe von 10 Vokabeln wird der Verarbeitungsvorgang abgeschlossen und die Anzahl F der festgestellten Fehler angezeigt. Im BASIC-Programm sichert die Anweisung RANDOMIZE, daß bei mehrmaliger Programmausführung unterschiedliche Zufallszahlenfolgen und damit auch unterschiedliche Vokabelfolgen entstehen.

```
10  DIM D¤(22),E¤(22)
20  RANDOMIZE
30  PRINT "VOKABEL-TRAINER ENGLISCH-DEUTSCH"
40  PRINT "* * * * * * * * * * * * * * * * * *"
50  PRINT
60  FOR I=0 TO 22
70  READ D¤(I),E¤(I)
80  NEXT I
90  LET F=0
100 FOR I=1 TO 10
110 LET K=INT(23*RND)
120 PRINT E¤(K),
130 INPUT A¤
140 IF A¤=D¤(K) THEN 170
150 PRINT "FALSCH !      RICHTIG: ";D¤(K)
160 LET F=F+1
170 PRINT
180 NEXT I
190 PRINT F;" FEHLER BEI 10 VOKABELN"
200 DATA GEHEN,TO GO,DRUCKEN,TO PRINT,WENN,IF,DANN,THEN
210 DATA UNTERPROGRAMM,SUBROUTINE,FELD,ARRAY,FUER,FOR
220 DATA BEMERKUNG,REMARK,ZU,TO,SONST,ELSE,SCHRITT,STEP
230 DATA ZEILE,LINE,ZEICHENKETTE,STRING,DATEN,DATA
240 DATA DATEI,FILE,LESEN,TO READ,NULL,ZERO,FEHLER,ERROR
250 DATA UEBERLAUF,OVERFLOW,SPEICHER,MEMORY,FALSCH,BAD
260 DATA EINGABE,INPUT,AUSGABE,OUTPUT
270 END
RUN
VOKABEL-TRAINER ENGLISCH-DEUTSCH
* * * * * * * * * * * * * * * * * *

FOR            ? FUER

LINE           ? ZEILE

OUTPUT         ? EINGABE
FALSCH !      RICHTIG: AUSGABE

REMARK         ?
```

Programm 8.2. Vokabel-Trainer

9. Unterprogramme

9.1. Unterprogrammtechnik

Ein Unterprogramm (engl.: subroutine) ist eine Anweisungsgruppe zur Lösung eines Teilproblems, die von einem übergeordneten Programm aufgerufen werden kann. Der Aufruf bewirkt einen Sprung an den Anfang des Unterprogramms. Nachdem die-

ses ausgeführt wurde, veranlaßt eine spezielle Rückkehranweisung (letzte auszuführende Anweisung des Unterprogramms) die Fortsetzung des übergeordneten Programms. Ein Programm kann mehrere Aufrufe des gleichen Unterprogramms enthalten und ein Unterprogramm kann wiederum andere Unterprogramme aufrufen. Ist das übergeordnete Programm keinem anderen Programm untergeordnet, wird es als Hauptprogramm bezeichnet. Bild 9.1 stellt schematisiert den zweimaligen Aufruf des gleichen Unterprogramms durch ein Hauptprogramm dar. Dabei geben Pfeile und Zahlen die Reihenfolge der Ausführung der einzelnen Programmteile wieder. Das Hauptprogramm wird nach einem Aufruf des Unterprogramms mit der Anweisung fortgesetzt, die auf den betreffenden Unterprogrammaufruf folgt (Anweisung r bzw. s im Bild 9.1).

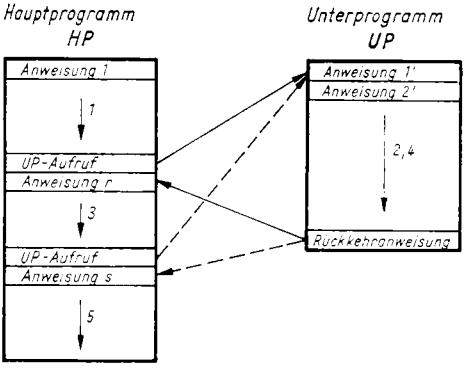


Bild 9.1. Zusammenwirken von Haupt- und Unterprogramm

Übergeordnetes Programm und Unterprogramm können sich gegenseitig Variablenwerte übergeben. Man bezeichnet diese Werte als Parameter und den Vorgang als Parameterübergabe. Anweisungsgruppen werden vor allem dann als Unterprogramm notiert, wenn die gleiche Anweisungsfolge in einem Programm mehrfach auftritt. Auf diese Weise wird der Speicherplatzbedarf des Gesamtprogramms verringert, weil das Unterprogramm nur einmal abzuspeichern ist. Außerdem erhält man dadurch eine übersichtlichere Programmstruktur. Die Unterprogrammtechnik

bietet auch die Möglichkeit, den Aufwand für die Erarbeitung neuer Programme zu reduzieren, indem für häufig und in unterschiedlichen Programmen zu lösende Teilaufgaben Unterprogramme erstellt werden. Allerdings sind dabei in BASIC gewisse Einschränkungen zu beachten (s. 9.2.).

9.2. Unterprogrammanweisungen [GOSUB, ON-GOSUB, RETURN]

Die Anweisung für den Aufruf eines Unterprogramms hat die Form

GOSUB zielzeilennummer

GOSUB bewirkt einen Sprung zur angegebenen Zielzeile im Unterprogramm. Dieses kann auf alle im übergeordneten Programm benutzten Variablen direkt zugreifen (Variablen, die im aufrufenden und im aufgerufenen Programm die gleichen Namen haben, belegen auch die gleichen Speicherplätze.).
Die Rückkehr aus dem Unterprogramm in das übergeordnete Programm bewirkt die Anweisung

RETURN

Die Programmausführung wird dann im übergeordneten Programm mit der auf den Unterprogrammaufruf folgenden Anweisung fortgesetzt. Ein Unterprogramm darf mehrere RETURN-Anweisungen enthalten, d.h., von verschiedenen Stellen des Unterprogramms kann zum aufrufenden Programm zurückgesprungen werden.

Beispiel: Hauptprogramm/Unterprogramm

Programm 9.1 demonstriert das Zusammenwirken eines Haupt- und eines Unterprogramms. Das auf den Zeilen 10 bis 60 notierte Hauptprogramm ruft zweimal das Unterprogramm auf, das bei jeder Ausführung einen entsprechenden Text ausgibt. Die im Hauptprogramm mit einem Wert belegte Variable Z ist Eingangsparameter des Unterprogramms. Das Hauptprogramm wird

```
10  REM  HAUPTPROGRAMM
20  LET Z=1
30  GOSUB 100
40  LET Z=2
50  GOSUB 100
60  STOP
100 REM  UNTERPROGRAMM
110 PRINT "*UP-AUFRUF";Z
120 RETURN
130 END
RUN
*UP-AUFRUF 1
*UP-AUFRUF 2
Break in 60
OK
```

Programm 9.1. Hauptprogramm/Unterprogramm

durch die STOP-Anweisung abgeschlossen. Diese bewirkt beim Programmhalt die Ausgabe der Systemnachricht "BREAK IN 60". Systemnachrichten sind Mitteilungen des BASIC-Interpreters an den Nutzer. Bezüglich ihrer Textform unterscheiden sie sich bei den einzelnen BASIC-Systemen.
In vielen BASIC-Systemen steht auch die ON-GOSUB-Anweisung für den Unterprogrammaufruf zur Verfügung.

ON numausdruck **GOSUB** zielznr,zielznr,...,zielznr

In dieser Anweisung sind hinter dem Schlüsselwort GOSUB mehrere Zielzeilennummern (zielznr) notiert. Vom Wert des numerischen Ausdruckes (numausdruck) ist, wie bei der ON-GOTO-Anweisung, abhängig, zu welcher Programmzeile gesprungen wird. Damit ist es möglich, unter verschiedenen Unterprogrammen bzw. verschiedenen Startpunkten eines Unterprogramms eine Auswahl zu treffen.

Parameterübergabe

Für die Unterprogramme, die über GOSUB bzw. ON-GOSUB aufgerufen werden, ist die im Programm 9.1 verwendete Form der Parameterübergabe die einzig mögliche. Demnach müssen die Namen, die im Unterprogramm und im übergeordneten Programm für die Parameter verwendet werden, in jedem Fall übereinstimmen. Außerdem müssen die in beiden Programmen verwendeten Variablennamen so abgestimmt sein, daß beim Aufruf

des Unterprogramms nicht unbeabsichtigt Werte von Variablen des aufrufenden Programms verändert werden. Diese Bedingungen schränken in BASIC die Nutzung der Unterprogrammtechnik bzw. den Austausch von Unterprogrammen stark ein.

Beispiel: Würfelspiel

Bei diesem Spiel werden zwei Würfel benutzt. Der Spieler hat gewonnen, wenn er

a) beim ersten Wurf 7 oder 11 Punkte erreicht oder
b) beim ersten Wurf eine 4, 5, 6, 8, 9 oder 10 erzielt und in einem folgenden nochmals die gleiche Punktzahl erreicht, ohne daß dazwischen eine "7" gewürfelt wurde.

In allen anderen Fällen hat der Spieler verloren [8]. Das Programm 9.2 simuliert die Würfe und wertet die Ergebnisse aus. Der Simulation eines Wurfes dient die Anweisungsfolge des Unterprogramms (ab Zeile 300). Dieses fordert den Spieler durch Ausgabe des Textes "WURF ?" zum Auslösen eines Wurfes auf und wartet auf die Eingabe eines Kennzeichens. Wird "W" eingegeben, so simuliert das Unterprogramm einen Wurf mit zwei Würfeln und übermittelt das Ergebnis dem Hauptprogramm durch die Variable K. Dagegen wird bei Eingabe von "H" das Würfelspiel beendet. Diese Eingabe bewirkt, daß der Variablen K keine Zufallszahl (ganze Zahl aus dem Bereich 2,...,12) zugeordnet wird. Die Variable K behält deshalb den Wert 1, der ihr am Anfang des Unterprogramms zugewiesen wurde. Bei der nachfolgenden Auswertung des Wurfresultats im Hauptprogramm wird dann zum Programmende verzweigt. Nach dem ersten Aufruf des Unterprogramms durch das Hauptprogramm (Simulation des 1. Wurfes) führt der Rücksprung zur ON-GOTO-Anweisung. Diese wertet den Wurf entsprechend den Spielregeln aus, indem sie ein Sprungziel auswählt und den Sprung realisiert. Im Programmablaufplan (Bild 9.2) wird die Auswertung des Wurfergebnisses mit Hilfe einer Fallunterscheidung beschrieben. In Abhängigkeit vom Wert K verzweigt diese zu einem der Ausgänge (K=1,2, ...,12). In den Fällen "VERLOREN" oder "GEWONNEN" wird nach Ausgabe des entsprechenden Textes ein neues Spiel begonnen. Ist das Würfeln zu wiederholen, wird die Punktzahl des ersten Wurfes gespeichert (Variable V) und dann das Unterprogramm "Würfeln" nochmals aufgerufen (Simulation eines weiteren Wurfs). Der Rücksprung aus dem Unterprogramm führt in

```
10  PRINT "WUERFELSPIEL"
20  PRINT "* * * * * * *"
30  RANDOMIZE
40  PRINT "WUERFELN: 'W' EINGEBEN"
50  PRINT "SPIEL BEENDEN: 'H' EINGEBEN"
60  PRINT
70  PRINT "**"
80  REM SIMULATION DES 1. WURFES
90  GOSUB 300
100 ON K GOTO 390,110,110,160,160,160,130,160,160,160,130,110
110 PRINT TAB(21);"VERLOREN !"
120 GOTO 60
130 PRINT TAB(21);"GEWONNEN !"
140 GOTO 60
150 REM BEIM 1. WURF 4,5,6,8,9 ODER 10
160 LET V=K
170 PRINT TAB(21);"NOCHMALS WUERFELN !"
180 REM SIMULATION EINES WEITEREN WURFES
190 GOSUB 300
200 IF K=1 THEN 240
210 IF K=7 THEN 110
220 IF K<>V THEN 170
230 GOTO 130
240 PRINT "DURCH ABBRUCH VERLOREN !"
250 GOTO 390
300 REM UP WUERFELN
310 LET K=1
320 PRINT "WURF";
330 INPUT X¤
340 IF X¤="H" THEN 380
350 IF X¤<>"W" THEN 320
360 LET K=INT(6*RND+1)+INT(6*RND+1)
370 PRINT "      RESULTAT:";K;
380 RETURN
390 END
RUN
WUERFELSPIEL
* * * * * * *
WUERFELN: 'W' EINGEBEN
SPIEL BEENDEN: 'H' EINGEBEN


**
WURF? W
     RESULTAT: 5     NOCHMALS WUERFELN !
WURF? W
     RESULTAT: 12    NOCHMALS WUERFELN !
WURF? W
     RESULTAT: 7     VERLOREN !

**
WURF? W
     RESULTAT: 7     GEWONNEN !

**
WURF? W
     RESULTAT: 4     NOCHMALS WUERFELN !
WURF? H
DURCH ABBRUCH VERLOREN !
OK
```

Programm 9.2. Würfelspiel

diesem Fall zu den IF-THEN-Anweisungen, die auf die GOSUB-Anweisung folgen. Diese übernehmen die Auswertung des zweiten Wurfes und aller folgenden bis zum Abschluß eines Spiels ("GEWONNEN", "VERLOREN" oder "ABBRUCH").

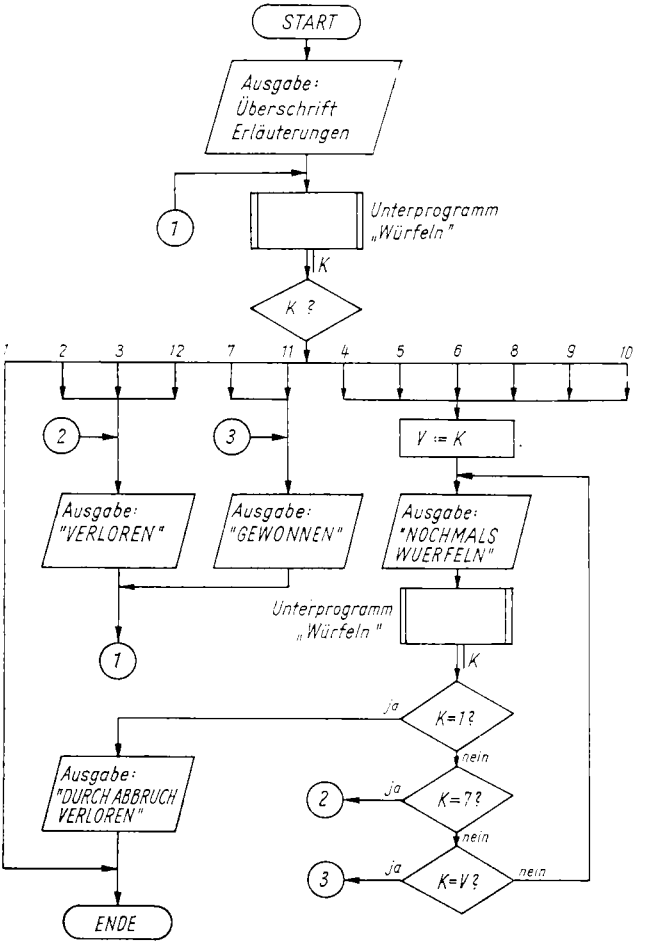


Bild 9.2. Würfelspiel

10. Anwenderdefinierte Funktionen

10.1. Funktionsdefinition [DEF]

Der Anwender kann zusätzlich zu den BASIC-Standardfunktionen weitere Funktionen definieren. Unter dem Definieren einer Funktion versteht man das Festlegen des Funktionsnamens und der Vorschrift für die Ermittlung des Funktionswertes mit Hilfe der DEF-Anweisung. Sie hat die allgemeine Form

DEF FNm=ausdruck
oder
DEF FNm(parameter)=ausdruck

Auf das Schlüsselwort DEF folgt der Funktionsname, der aus drei Buchstaben besteht. Von diesen sind zwei vorgegeben (FN) und nur der letzte (m) ist frei wählbar. Hinter dem Funktionsnamen kann in runden Klammern ein Parameter angeführt sein. Rechts vom Gleichheitszeichen schließt sich dann der Ausdruck zur Ermittlung des Funktionswertes an. Die DEF-Anweisung muß im Programm vor dem ersten Aufruf der Funktion stehen und darf die Länge einer Anweisungszeile nicht überschreiten.

Beispiel:

```
10 DEF FNS(X)=X^3+2*X^2+15.5
```

Diese Anweisung definiert die Funktion $y=f(x)=x^3+2x^2+15{,}5$. Minimal-BASIC sieht nur die Definition numerischer Funktionen mit maximal einem Parameter vor. Viele BASIC-Versionen gestatten jedoch die Angabe mehrerer Parameter, die dann durch Komma zu trennen sind (Parameterliste).
Die als Parameter aufgeführten Variablen sind nicht mit eventuell gleichnamigen Variablen außerhalb der DEF-Anweisung identisch. Sie dienen lediglich der Funktionsbeschreibung und werden deshalb formale Parameter genannt. Der Ausdruck in der Funktionsdefinition kann auch Variablen enthalten, die nicht in der Parameterliste stehen. Sie sind dann mit den namensgleichen Variablen außerhalb der Funktionsdefinition identisch.

10.2. Funktionsaufruf

Anwenderdefinierte Funktionen werden wie Standardfunktionen über ihren Namen aufgerufen, und ebenso wird der Funktionswert über den Namen dem aufrufenden Programm übermittelt. Beim Aufruf muß die Zahl der hinter dem Funktionsnamen notierten Argumente mit der Zahl der Parameter in der Funktionsdefinition übereinstimmen. Die Werte der Argumente gehen anstelle der formalen Parameter in die Berechnung des Funktionswertes ein. Dabei wird die Zuordnung durch die Reihenfolge der Parameter und Argumente festgelegt. Als Argumente sind, wie bei den Standardfunktionen, nicht nur Konstanten, sondern auch Variablen und Ausdrücke zugelassen.

Beispiel:

```
10  DEF FNS(X)=X^3+2*X^2+15.5
20  INPUT A,B
30  PRINT FNS(A),FNS(B)
:
.
```

Auf Zeile 30 wird die definierte Funktion FNS zweimal aufgerufen. Mit den eingegebenen Werten der Variablen A und B berechnet der Computer die Werte der Ausdrücke A^3+2*A^2+15.5 und B^3+2*B^2+15.5 und gibt sie aus.
Das folgende Beispiel soll die Parameterübergabe bei Funktionen mit mehreren Parametern erläutern.

Beispiel:
Eine Funktion zur Berechnung des Volumens eines Kreiskegels mit dem Radius *r* der Grundfläche und der Höhe *h* wird definiert und aufgerufen.

```
10  DEF FNK(R,H)=3.14159*R*R*H/3
20  PRINT FNK(7,11)
30  END
```

Beim Aufruf der Funktion FNK auf Zeile 20 nimmt die Konstante 7 (1. Argument) die Stelle von R (1. formaler Parameter) und die Konstante 11 die Stelle des 2. formalen Parameters H ein.

Durch die PRINT-Anweisung wird deshalb der Wert des Ausdrucks 3.14159 *7 *7 *11/3 ausgegeben.
Das nächste Beispiel zeigt besonders die programmtechnischen Vorteile, die sich aus der Nutzung anwenderdefinierter Funktionen bei bestimmten Aufgaben ergeben.

Beispiel: Numerische Integration

Der Inhalt einer Fläche F', die eine oberhalb der x-Achse liegende Kurve $y=f(x)$ im Intervall $[a,b]$ mit der x-Achse einschließt, ist zu berechnen (Bild 10.1; Fläche schraffiert). Dazu wird eine Näherungsformel (Newton-Cotes) verwendet. Diese geht von der Einteilung des Intervalls $[a,b]$ in 4 gleichgroße Teilintervalle aus und berechnet die Fläche F aus den Funktionswerten an den Grenzen der Teilintervalle. Die Formel lautet

$$F = \frac{b-a}{90}\,[7\cdot f(a)+32\cdot f(a+\Delta x)+12\cdot f(a+2\cdot\Delta x) \\ +32\cdot f(a+3\cdot\Delta x)+7\cdot f(b)],\ \text{mit}\ \Delta x = (b-a)/4.$$

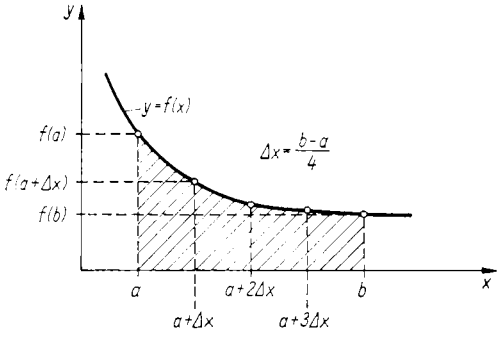


Bild 10.1. Festlegung der Stützstellen im Beispiel "Numerische Integration"

Diese Aufgabe wird durch das Programm 10.1 gelöst. Die DEF-Anweisung auf Zeile 30 definiert die Funktion $f(x)$ unter dem Namen FNN. Dabei wird $f(x) = 2/x+1$ angenommen. Die Intervallgrenzen werden in der DATA-Anweisung festgelegt und auf Zeile 50 den Variablen A und B zugewiesen. Die Anweisungen zur Berechnung des Flächeninhaltes F benutzen die definierte

Funktion FNN. Dadurch ergibt sich ein relativ einfacher Ausdruck. Außerdem, und das ist ein wesentlicher Vorteil, kann das Programm in dieser Form nach Änderung nur einer Zeile auch bei anderen Funktionen $y = f(x)$ verwendet werden. Ebenso sind durch Auswechslung der DATA-Anweisung beliebige Intervallgrenzen vorgebbar. Dem vom Programm 10.1 ermittelten Wert F des Flächeninhaltes sei vergleichsweise der genaue Wert F' gegenübergestellt. Für das angegebene Beispiel ist $F' = 4{,}27258$. Die Differenz zwischen F' und F ist vor allem darauf zurückzuführen, daß beim vorgegebenen Algorithmus zur Flächenberechnung nur fünf Ordinatenwerte des Intervalls $[a,b]$ genutzt werden.

```
10  REM  NUMERISCHE INTEGRATION
20  REM  NACH NEWTON-COTES
30  DEF  FNN(X)=2/X+1
40  DATA 0.5,2
50  READ A,B
60  LET D=(B-A)/4
70  LET F=7*(FNN(A)+FNN(B))
80  LET F=F+32*(FNN(A+D)+FNN(A+3*D))
90  LET F=F+12*FNN(A+2*D)
100 PRINT "INTEGRALWERT=";F*(B-A)/90
110 END
RUN
INTEGRALWERT= 4.27879
OK
```

Programm 10.1. Numerische Integration

11. Programmtest und Fehleranalyse

Programme sind nicht immer sofort fehlerfrei. Der Programmtest hat die Aufgabe, die Richtigkeit eines Programms nachzuweisen bzw. vorhandene Fehler zu analysieren. In der Regel erfolgt er so, daß die Programme mit Testdaten, für die die Lösungen bekannt oder errechenbar sind, ausgeführt werden. Dabei ist jeder Zweig des Programms mindestens einmal zu durchlaufen. Es sind demnach im allgemeinen mehrere Testläufe mit unterschiedlichen Daten erforderlich. Oft ist jedoch für die Feststellung der Richtigkeit noch nicht hinreichend, daß beim Test jeder Programmzweig einmal durchlaufen wurde. Manche Fehler zeigen sich nur bei be-

stimmten Werten der Testdaten. Aus diesem Grunde ist die Prüfung von Programmen unter Umständen sehr kompliziert und arbeitsaufwendig.
Im allgemeinen unterscheidet man drei Fehlerarten: Syntaxfehler, Laufzeitfehler und logische Fehler.

Syntaxfehler

Verstöße gegen die formalen Regeln der Programmiersprache sind Syntaxfehler. Sie werden vom Rechner bei der Eingabe oder der Übersetzung der betreffenden Anweisung erkannt und angezeigt. Oft wird als Fehlerhinweis eine Fehlernummer ausgegeben, deren Bedeutung aus der Beschreibung der jeweiligen Sprachversion ersichtlich ist. Beispiele für Syntaxfehler sind:

```
10 INPUT A
```

Fehler: Leerzeichen in der Zeilennummer

```
80 PRIN A*A
```

Fehler: Schlüsselwort ist falsch

```
300 LET B=A:5
```

Fehler: Unzulässiges Operationszeichen

```
410 LET S="SUMME"
```

Fehler: Variablen- und Konstantentyp passen nicht zueinander

Laufzeitfehler

Fehler, die erst während der Ausführung (der Laufzeit) der Anweisungen, d.h. nach der Übersetzungsphase, auftreten bzw. erkannt werden, nennt man Laufzeitfehler. Hierzu zählen beispielsweise Zahlenbereichsüberschreitungen, Überschreitungen von Feldgrenzen, Division durch Null, negative Argumentwerte der Standardfunktion SQR und Sprunganweisungen mit Zeilennummern, die im Programm nicht vorhanden sind.
Auch Laufzeitfehler werden vom BASIC-Interpreter analysiert und in ähnlicher Form wie Syntaxfehler angezeigt.

Logische Fehler

Ein Programm enthält logische Fehler, wenn der zugrunde liegende Algorithmus falsch ist bzw. bei der Programmierung verfälscht wurde. Ein logischer Fehler liegt beispielsweise vor, wenn in einem Ausdruck eine Variable verwendet wird, der noch kein Wert zugewiesen wurde oder wenn man statt des Operationszeichens "+" irrtümlich das Zeichen "−" notiert. Ob ein Programm logische Fehler enthält, kann nur mit Hilfe von Testbeispielen festgestellt werden. Sofern aus Differenzen zwischen Ergebnissen der Testläufe und bekannten oder durch Handrechnung ermittelten Resultaten auf das Vorhandensein eines logischen Fehlers zu schließen ist, muß dieser analysiert, d.h. gefunden werden. Dafür stehen folgende Mittel und Wege zur Verfügung:

- genaue Durchsicht des Programms und nochmaliger Vergleich mit dem zugrunde liegenden Algorithmus
- Ausgeben von Zwischenergebnissen durch Einfügen zusätzlicher PRINT-Anweisungen in das Programm (Diese werden nach Abschluß des Programmtests wieder gelöscht.)
- Bildung von Programmblöcken durch Einfügen von STOP-Anweisungen. Wird bei der Programmausführung ein solcher Haltepunkt erreicht, kann im Direktbetrieb der aktuelle Wert von Variablen ausgegeben und so die Arbeitsweise des Programms abschnittsweise geprüft werden. Durch Eingabe eines speziellen Kommandos (CONTINUE, CONT) kann man danach die Programmausführung bei der auf die STOP-Anweisung folgenden Anweisung fortsetzen.
- Verfolgung des Programmlaufs durch Einfügen von STOP-Anweisungen in Schleifen oder Verzweigungen.

Einige BASIC-Systeme gestatten außerdem noch die Verwendung spezieller Testkommandos für das Verfolgen des Programmlaufs, der Ausführung von Sprüngen und der Wertzuweisungen an ausgewählte Variablen.

Im allgemeinen ist das Finden und Beheben logischer Fehler wesentlich aufwendiger als die Korrektur von Syntax- oder Laufzeitfehlern. Die nachträgliche Beseitigung von Fehlern im Algorithmus erfordert oft wesentliche Veränderungen des erstellten Programms und birgt so auch die Gefahr des Einschleppens neuer Fehler in sich. Deshalb sollten Algorithmen vor ihrer Umsetzung in ein Programm sorgfältig geprüft werden. Dazu kann ein "Trockentest" dienen. Testdaten werden dabei nach den Vorgaben

des zu prüfenden Algorithmus Schritt für Schritt "von Hand" (mit Papier und Bleistift) verarbeitet. Die Ergebnisse jedes Verarbeitungsschrittes notiert man zweckmäßigerweise in Tabellenform.

12. Zusammenfassung

Abschließend sei noch einmal in Bild 12.1 zusammengestellt, welche Phasen bei der Entwicklung eines BASIC-Programms zu durchlaufen sind [4], [5]. Ausgangspunkt ist eine gegebene Problemstellung.

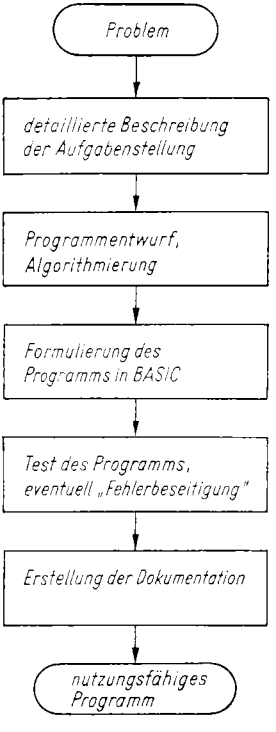


Bild 12.1. Phasen der Programmentwicklung

Zu jedem Programm gehört eine entsprechende Dokumentation. Dazu sind die in den einzelnen Phasen erarbeiteten Teildokumentationen (Aufgabenstellung, Beschreibung des Algorithmus, Programmliste, Darstellung der Ein- und Ausgabedaten u.a.) zusammenzufassen und das Programm in Grundaufbau, Funktionsweise und Leistungsumfang zu beschreiben. Außerdem sollten auch Nutzungshinweise enthalten sein, damit andere Anwender ein vorliegendes Programm problemlos benutzen können.

Anhang A

Übersicht der Sprachelemente von Minimal-BASIC (entsprechend ISO-Standard)

A1. Zeichensatz

In Minimal-BASIC ist der in Tabelle A1 aufgeführte international vereinbarte Zeichensatz definiert. Die Darstellung entspricht einer Codetabelle, in der jedem Zeichen eine Zahl von 0 bis 127 in Dezimaldarstellung (unter jedem Zeichen eingetragen) bzw. ein entsprechendes Binärwort (seitlich und über der Tabelle angegeben) zugeordnet ist. Die Dezimalzahlen sind der Wertigkeit der Zeichen äquivalent, mit der bei STRING-Vergleichen gearbeitet wird. Die Binärworte entsprechen einer rechnerinternen Darstellung der Zeichen. Die zwei linken Spalten der Tabelle (Position 1 bis 31) enthalten Steuerzeichen. Bei vielen Systemen können die Steuerzeichen über die Tastatur in den Computer eingegeben werden, indem man die sogenannte CTRL-Taste und eine entsprechende Buchstabentaste (in der Tabelle mit "c buchstabe" gekennzeichnet) gleichzeitig oder nacheinander betätigt. Im rechten Teil der Tabelle (Position 32 bis 126) sind alle druckbaren Zeichen enthalten. Die Zeichen auf Position 0 (NUL) und Position 127 (DEL) sind für spezielle Funktionen vorgesehen.
Der gesamte Zeichensatz wird in Minimal-BASIC in zwei Bereiche eingeteilt: den Mindestzeichensatz (helle Felder in Tabelle A1) und den für Erweiterungen vorgesehenen Zeichensatz (grau hinterlegt). Die Bedeutungen der einzelnen druckbaren Zeichen enthält Tabelle A2.

Tabelle A1 International vereinbarter Zeichensatz für Minimal BASIC (d_1 bis d_7 Binärcodierung der Zeichen, *c buchstabe* Symbolik für Erzeugung der Steuerzeichen mit Hilfe des CTRL-Zeichens; NUL, DEL spezielle Zeichen, SP Leerzeichen)

Binärcodierung								
d_7	0	0	0	0	1	1	1	1
d_6	0	0	1	1	0	0	1	1
d_5	0	1	0	1	0	1	0	1

d_4	d_3	d_2	d_1								
0	0	0	0	NUL 0	cP 16	SP 32	O 48	@ 64	P 80	\ 96	p 112
0	0	0	1	cA 1	cQ 17	! 33	1 49	A 65	Q 81	a 97	q 113
0	0	1	0	cB 2	cR 18	" 34	2 50	B 66	R 82	b 98	r 114
0	0	1	1	cC 3	cS 19	# 35	3 51	C 67	S 83	c 99	s 115
0	1	0	0	cD 4	cT 20	¤ 36	4 52	D 68	T 84	d 100	t 116
0	1	0	1	cE 5	cU 21	% 37	5 53	E 69	U 85	e 101	u 117
0	1	1	0	cF 6	cV 22	& 38	6 54	F 70	V 86	f 102	v 118
0	1	1	1	cG 7	cW 23	' 39	7 55	G 71	W 87	g 103	w 119
1	0	0	0	cH 8	cX 24	(40	8 56	H 72	X 88	h 104	x 120
1	0	0	1	cI 9	cY 25	) 41	9 57	I 73	Y 89	i 105	y 121
1	0	1	0	cJ 10	cZ 26	* 42	: 58	J 74	Z 90	j 106	z 122
1	0	1	1	cK 11	27	+ 43	; 59	K 75	[91	k 107	{ 123
1	1	0	0	cL 12	28	, 44	< 60	L 76	\ 92	l 108	| 124
1	1	0	1	cM 13	29	– 45	= 61	M 77	] 93	m 109	} 125
1	1	1	0	cN 14	30	. 46	> 62	N 78	^ 94	n 110	~ 126
1	1	1	1	cO 15	31	/ 47	? 63	O 79	_ 95	o 111	DEL 127

Minimal-BASIC-Zeichensatz

Erweiterter Zeichensatz

A2. Wortsymbole (Schlüsselworte)

Schlüsselworte sind Folgen von Buchstaben, die als Anweisung oder als Teilkomponente einer Anweisung fungieren. Nachstehende Schlüsselworte sind festgelegt:
BASE, DATA, DEF, DIM, END, FOR, GO, GOSUB, GOTO, IF, INPUT, LET, NEXT, ON, OPTION, PRINT, RANDOMIZE, READ, REM, RESTORE, RETURN, STEP, STOP, SUB, THEN und TO

Tabelle A2 Bedeutung der Zeichen

Name	Zeichen
Leerzeichen	
Ausrufezeichen	!
Anführungszeichen	"
Nummernzeichen	#
Währungszeichen	◯ ($)
Prozentzeichen	%
Ampersand	&
Apostroph	'
linke Klammer	(
rechte Klammer	)
Stern	*
Pluszeichen	+
Komma	,
Minuszeichen	−
Punkt	.
Schrägstrich	/
Ziffer	0 bis 9
Doppelpunkt	:
Semikolon	;
Kleiner-Zeichen	<
Größer-Zeichen	>
Gleichheitszeichen	=
Fragezeichen	?
Buchstabe	A bis Z
Akzentzeichen	^
Unterstreichungszeichen	_

A3. Syntax

A3.1. Beschreibungsform

Die Verknüpfung von Sprachelementen zu Sprachkonstruktionen erfolgt auf der Basis von Vorschriften, sogenannten syntaktischen Regeln. Diese werden mit Hilfe einer Metasprache beschrieben. Die Sprachelemente der Metasprache notiert man mit Kleinbuchstaben, um sie von Elementen der Programmiersprache unterscheiden zu können. Eine syntaktische Regel hat in metasprachlicher Notierung die Form:

```
sprachelement ::= syntaktische-konstruktion
```

Für "sprachelement" werden verbale Bezeichnungen (mit Kleinbuchstaben und mit Hilfe des Bindestrichs als ein Wort geschrieben) eingesetzt. Das Symbol ::= ist das Definitionssymbol (Bedeutung: "ist definiert durch"). Die "syntaktische-konstruktion" gibt die Vorschriften an, nach denen das BASIC-Sprachelement aus noch elementareren Sprachelementen aufgebaut werden kann. Für deren Darstellung werden außerdem folgende Symbole benutzt [1]:

- Alternativsymbol: /

Es dient der Aufzählung von Möglichkeiten, von denen genau eine ausgewählt werden muß.

- Kann-Symbol: ?

Ist der syntaktischen Einheit ein Fragezeichen nachgestellt, dann kann diese je nach Bedarf geschrieben werden oder nicht.

- Iterationssymbol: *

Das Symbol wird einer syntaktischen Konstruktion nachgestellt und gibt an, daß die so gekennzeichnete Konstruktion mehrmals hintereinander geschrieben werden darf, aber auch entfallen kann.

- Klammersymbole: ( )

Die syntaktischen Klammern kennzeichnen den Wirkungsbereich von metasprachlichen Symbolen in einer aus mehreren Sprachelementen zusammengesetzten syntaktischen Konstruktion. BASIC-Schlüsselworte werden in der syntaktischen Darstellung mit Großbuchstaben wiedergegeben.

A3.2. Zeichen, Folgen von Zeichen

Alle in der Sprache BASIC verwendeten Zeichen werden im folgenden verbal beschrieben, um eine Unterscheidung zu den syntaktischen Darstellungssymbolen zu erreichen:

```
buchstabe ::= A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/
              W/X/Y/Z
ziffer ::= 0/1/2/3/4/5/6/7/8/9
zeichen ::= anführungszeichen / string-zeichen
```

string-zeichen ::= ausrufezeichen / nummernzeichen /
währungszeichen / prozentzeichen /
ampersand / apostroph / linke-klammer /
rechte-klammer / stern / komma / schrägstrich /
doppelpunkt / semikolon / kleiner-zeichen /
größer-zeichen / gleichheitszeichen / fragezeichen /
akzentzeichen / unterstreichungszeichen /
gewöhnliches-string-zeichen
gewöhnliches-string-zeichen ::= leerzeichen / gewöhnliches-zeichen
gewöhnliches-zeichen ::= pluszeichen / minuszeichen / punkt /
ziffer / buchstabe
angeführter-string ::= anführungszeichen string-zeichen*
anführungszeichen
nicht-angeführter-string ::= gewöhnliches-zeichen / gewöhnliches-
zeichen gewöhnliches-string-zeichen*
gewöhnliches-zeichen
kommentar-string ::= zeichen*

A3.3. Programmaufbau

programm ::= block* endezeile
block ::= (zeile / for-block)*
zeile ::= zeilennummer anweisung zeilenendezeichen
zeilennummer ::= ziffer ziffer? ziffer? ziffer?
endezeile ::= zeilennummer endeanweisung zeilenendezeichen
endeanweisung ::= END
anweisung ::= data-anweisung / def-anweisung / dimension-anweisung / gosub-anweisung / goto-anweisung / if-then-anweisung / input-anweisung / let-anweisung / on-goto-anweisung / option-anweisung / print-anweisung / randomize-anweisung / read-anweisung / remark-anweisung / restore-anweisung / return-anweisung / stop-anweisung

A3.4. Konstanten, Variablen

konstante ::= numerische-konstante / string-konstante
numerische-konstante ::= vorzeichen? zeichenfolge
vorzeichen ::= pluszeichen / minuszeichen
zeichenfolge ::= signifikante-folge exponent?
signifikante-folge ::= ganzer-teil punkt? / ganzer-teil? gebrochener-teil
ganzer-teil ::= ziffer ziffer*
gebrochener-teil ::= punkt ziffer ziffer*
exponent ::= E vorzeichen? ganzer-teil
string-konstante ::= angeführter-string
variable ::= numerische-variable / string-variable

numerische-variable ::= einfache-numerische-variable / numerisches-feldelement
einfache-numerische-variable ::= buchstabe ziffer?
numerisches-feldelement ::= numerischer-feld-name indexangabe
numerischer-feldname ::= buchstabe
indexangabe ::= linke-klammer numerischer-ausdruck (komma numerischer-ausdruck)? rechte-klammer
string-variable ::= buchstabe währungszeichen

A3.5. Ausdrücke

ausdruck ::= numerischer-ausdruck / string-ausdruck
numerischer-ausdruck ::= vorzeichen? term (addoperator term)*
term ::= faktor (muloperator faktor)*
faktor ::= basisausdruck (potenzoperator basisausdruck)*
basisausdruck ::= zeichenfolge / numerische-variable / numerische-funktion / linke-klammer numerischer-ausdruck rechte-klammer
addoperator ::= pluszeichen / minuszeichen
muloperator ::= stern / schrägstrich
potenzoperator ::= akzentzeichen
string-ausdruck ::= string-variable / string-konstante

A3.6. Funktionen

numerische-funktion ::= numerischer-funktionsname argumentliste?
numerischer-funktionsname ::= numerische-funktion-standard / numerische-funktion-nutzerdefiniert
argument-liste ::= linke-klammer argument rechte-klammer
argument ::= numerischer-ausdruck
numerische-funktion-standard ::= ABS / ATN / COS / EXP / INT / LOG / RND / SGN / SIN / SQR / TAN
numerische-funktion-nutzerdefiniert ::= FN buchstabe
def-anweisung ::= DEF numerische-funktion-nutzerdefiniert parameterliste? gleichhheitszeichen numerischer-ausdruck
parameter-liste ::= klammer-auf-zeichen parameter klammer-zu-zeichen
parameter ::= einfache-numerische-variable

A3.7. Anweisungen

let-anweisung ::= numerische-let-anweisung / string-let-anweisung
numerische-let-anweisung ::= LET numerische-variable gleichheits-zeichen numerischer-ausdruck
string-let-anweisung ::= LET string-variable gleichheitszeichen string-ausdruck

goto-anweisung ::= GO leerzeichen* TO zeilennummer
if-then-anweisung ::= IF vergleichsausdruck THEN zeilennummer
vergleichsausdruck ::= numerischer-ausdruck numerischer-
vergleichsoperator numerischer-ausdruck /
string-ausdruck string-vergleichsoperator
string-ausdruck
numerischer-vergleichsoperator ::= gleichheitszeichen /
ungleichheitsoperator /
kleiner-zeichen /
größer-zeichen /
kleiner-gleich-operator /
größer-gleich-operator
string-vergleichsoperator ::= gleichheitszeichen /
ungleichheitsoperator
kleiner-gleich-operator ::= kleiner-zeichen gleichheitszeichen
größer-gleich-operator ::= größer-zeichen gleichheitszeichen
ungleichheitsoperator ::= kleiner-zeichen größer-zeichen
gosub-anweisung ::= GO leerzeichen* SUB zeilennummer
return-anweisung ::= RETURN
on-goto-anweisung ::= ON numerischer-ausdruck GO leerzeichen*
TO zeilennummer (komma zeilennummer)*
stop-anweisung ::= STOP
for-block ::= for-zeile block next-zeile
for-zeile ::= zeilennummer for-anweisung zeilenendezeichen
next-zeile ::= zeilennummer next-anweisung zeilenendezeichen
for-anweisung ::= FOR laufvariable gleichheitszeichen
anfangswert TO endwert (STEP schrittweite)?
laufvariable ::= einfache-numerische-variable
anfangswert ::= numerischer-ausdruck
endwert ::= numerischer-ausdruck
schrittweite ::= numerischer-ausdruck
next-anweisung ::= NEXT laufvariable
print-anweisung ::= PRINT print-liste?
print-liste ::= (print-ausdruck? print-trennzeichen)*
print-ausdruck?
print-ausdruck ::= ausdruck / tab-aufruf
tab-aufruf ::= TAB linke-klammer numerischer-ausdruck
rechte-klammer
print-trennzeichen ::= komma / semikolon
input-anweisung ::= INPUT variable (komma variable)*
read-anweisung ::= READ variable (komma variable)*
restore-anweisung ::= RESTORE
data-anweisung ::= DATA datum (komma datum)*
datum ::= angeführter-string / nicht-angeführter-string

dimension-anweisung ::= DIM feld-deklaration (komma feld-deklaration)*
feld-deklaration ::= numerischer feldname linke-klammer ziffer ziffer* (komma ziffer ziffer*)? rechte klammer
option-anweisung ::= OPTION BASE indexanfangswert
indexanfangswert ::= 0 / 1
remark-anweisung ::= REM kommentar-string
randomize-anweisung ::= RANDOMIZE

Anhang B

Abgeleitete numerische Funktionen

mathematische Bezeichnung	Erläuterung	Berechnung in BASIC
cot(x)	Cotangens	1/TAN(X)
arcsin(x)	Arcussinus	ATN(X/SQR(1−X*X))
arccos(x)	Arcuscosinus	P/2−ATN(X/SQR(1−X*X))
arccot(x)	Arcuscotangens	P/2−ATN(X) [1]
sinh(x)	Hyperbelsinus	(EXP(X)−EXP(−X))/2
cosh(x)	Hyperbelcosinus	(EXP(X)+EXP(−X))/2
tanh(x)	Hyperbeltangens	(EXP(X)−EXP(−X))/(EXP(X)+EXP(−X))
coth(x)	Hyperbelcotangens	(EXP(X)+EXP(−X))/(EXP(X)−EXP(−X))
arsinh(x)	Areasinus	LOG(X+SQR(X*X+1))
arcosh(x)	Areacosinus	±LOG(X+SQR(X*X−1))
artanh(x)	Areatangens	LOG((1+X)/(1−X))/2
arcoth(x)	Areacotangens	LOG((X+1)/(X−1))/2

[1] P Konstante π (3.141592654)

Literatur- und Quellenverzeichnis

[1] International Standard ISO 6373, Data processing-Programming languages, Minimal-BASIC. – 1984
[2] Standard ECMA-55, Minimal BASIC. – 1978
[3] ANSI X3J2 /84-10: Mach, Draft Proposed American National Standard for BASIC. – 1984
[4] Herrlich, D.; Lindner, U.: Strukturierte Programmierung. – Leipzig: BSB B.G. Teubner Verlagsgesellschaft, 1981
[5] Kimm, R. u.a.: Einführung in Software Engineering. – New York: Walter de Gruyter, 1979
[6] TGL 22 451, DDR-Standard Informationsverarbeitung, Datenfluß- und Programmablaufpläne, Sinnbilder. – 1975. – 30 S.
[7] Standard DIN 66 001, Informationsverarbeitung, Sinnbilder und ihre Anwendung. – 1983
[8] Gottfried, Byran S.: Programmieren in BASIC. – New York [u.a.]: McGraw-Hill Book Company, 1975
[9] Klein, R.-D.: BASIC-Interpreter. – München: Franzis-Verlag, 1982

Sachwortverzeichnis

BASIC-Standardfunktionen

ABS(X)	Absoluter Betrag
ATN(X)	Arcustangensfunktion
COS(X)	Cosinusfunktion
EXP(X)	Exponentialfunktion
INT(X)	Integerfunktion
LOG(X)	Natürlicher Logarithmus
RND	Zufallsfunktion
SGN(X)	Signumfunktion
SIN(X)	Sinusfunktion
SQR(X)	Quadratwurzel
TAB(X)	Tabellierungsfunktion
TAN(X)	Tangensfunktion

Die Programmierung von Computern ist heute nicht mehr nur eine Sache für Spezialisten. Moderne Programmiersprachen ermöglichen es praktisch jedem, diese Tätigkeit zu erlernen und Computer zur Lösung vielfältiger Aufgaben einzusetzen. Unter den vielen Programmiersprachen, die entwickelt wurden, zeichnet sich BASIC durch besondere Einfachheit bei großer Leistungsfähigkeit aus. Diese Sprache hat deshalb international weite Verbreitung gefunden und wird vor allem auch zur Programmierung von Klein- und Kleinstrechnern verwendet.

Das vorliegende Buch führt in die Computerprogrammierung ein und vermittelt die Grundlagen der Programmiersprache BASIC. Es beschreibt detailliert die wichtigsten BASIC-Sprachelemente, verdeutlicht ihre Anwendung anhand ausführlich erläuterter Programmbeispiele und erklärt das Erarbeiten und Testen von BASIC-Programmen im Dialog mit dem Computer. Die Darstellung des Stoffes ist so gewählt, daß auch Leser, die über keine Vorkenntnisse auf dem Gebiet der Informationsverarbeitung verfügen, sich die Programmiersprache BASIC im Selbststudium aneignen können.

ISBN 3-343-00126-0