E. Unger

DAS STANDARD BASIC-BUCH zum

Ein BASIC-Lehrgang im Dialog

Eine Edition aus dem Heim-Verlag

E. Unger

DAS STANDARD BASIC-BUCH zum

Ein BASIC-Lehrgang im Dialog

Eine Edition aus dem Heim-Verlag

Das Standard Basic-Buch zum Schneider CPC 464
Erhard Unger
– 1. Auflage – Darmstadt: Heim, 1984
ISBN 3 – 923250 – 12 – 6

(C)opyright 1984
beim Heim-Verlag · Organisation + Datentechnik
Heidelberger Landstr. 194 · 6100 Darmstadt
Telefon 0 61 51 - 5 53 75

Alle Rechte vorbehalten. Kein Teil dieses Buches darf ohne schriftliche Genehmigung des Heim-Verlages in irgendeiner Form reproduziert oder in eine von Maschinen, insbesondere auch von Datenverarbeitungsmaschinen, verwendete Sprache oder Aufzeichnungs bzw. Wiedergabeart übertragen oder übersetzt werden.

Die Wiedergabe von Warenbezeichnungen, Handelsnamen oder sonstigen Kennzeichen in dem Buch berechtigt nicht zu der Annahme, daß diese von jedermann frei benutzt werden dürfen. Es kann sich auch dann um eingetragene Warenzeichen oder sonstige gesetzlich geschützte Kennzeichen handeln, wenn sie nicht als solche besonders gekennzeichnet sind.

Druck: Druckerei der Heim OHG, 6100 Darmstadt.

VORWORT

Lieber Leser,

das vorliegende Buch stammt nicht aus der Feder eines einzelnen Autors. Es ist vielmehr das Werk eines Autorenteams.

Als Herausgeber danke ich dem Team ganz herzlich. Insbesondere gilt der Dank dem "Chef" des Teams und zugleich pädagogischen Leiter Herrn Raimund Grum, Darmstadt, sowie dem Informatiker und Spezialisten für knifflige, fachliche Fragen, Herrn Reinhard Hund, ebenfalls Darmstadt. Außerdem gilt ein ganz spezieller Dank den verständnisvollen Angehörigen aller Beteiligten, die während der Entstehungszeit des Buches auf das gesellschaftliche Beisammensein mit "ihren" Autoren fast völlig verzichten mußten.

Das Ergebnis dieser harten Arbeit liegt vor Ihnen, lieber Leser! Denn Sie halten jetzt ein umfassendes und außerordentlich reichhaltiges Fachbuch in den Händen, das Ihnen das Tor zu Ihrem neuen Computer, dem Schneider >>CPC464<<, öffnet.

Wir haben versucht, Ihnen den Einstieg in die neue Materie so leicht wie möglich zu machen. So finden sie zum Beispiel die vielen Programme des Buches alle auf einer Cassette, die Sie beim Heim-VERLAG beziehen können. Sie ersparen sich die riesige Mühe des Eintippens und können gleich zur Sache kommen.

Doch noch ein weiterer Punkt ist von Bedeutung:

Wenn Sie einmal Experte am Schneider-Computer geworden sind, werden Sie nach weiterer Literatur suchen. Auch diese finden Sie im Heim-VERLAG

Wir haben mit der Erfahrung vieler Programmiererjahre für Sie ein Nachschlagwerk geschaffen, in dem Sie den gesamten Befehlssatz des Schneider-Computers >>CPC464<< finden. Es kommt aber noch vielmehr hinzu:

Jeder Befehl ist nach einem klaren, leicht verständlichen Schema dargestellt und, was das Allerwichtigste ist, durch ausführliche Beispiele erläutert.

Dieses Nachschlagewerk hat also neben seinem Charakter als Lexikon noch den unschätzbaren Vorteil, auch eine wertvolle Programmsammlung zu sein mit einem hohen Nutzwert für Sie als Programmierer. Und wie Sie richtig vermutet haben: Alle diese Programme finden Sie wieder auf einer Cassette, die Sie ebenfalls beim Heim-VERLAG beziehen können.

Über den Inhalt des Buches und über die Art, wie Sie mit dem Buch arbeiten sollen, brauche ich im Vorwort nichts weiter zu sagen, denn das hat Herr Grum ausführlich und sehr verständlich in seiner Einleitung getan.

Wenn Sie fachliche Fragen haben, so rufen Sie bitte nicht im Heim-VERLAG sondern in der Redaktion des Verlages an: Tel.: 06151-33348. Noch lieber aber wäre es mir, wenn sie uns an die Redaktionsanschrift schreiben:

****

Redaktion des Heim-VERLAGS

Binger Str. 10

6100 Darmstadt

****

Mit freundlichen Grüßen

Erhard Unger

Darmstadt, den 23. August 1984

Inhaltsübersicht

Seite

1. Einleitung

Der >> CPC 464 << ist sicherlich viel mehr als nur ein Gerät für Telespiele. Er besitzt viele gute Eigenschaften, die Sie schnell erkennen werden, wenn Sie durch dieses Standard-BASIC-Buch zum Ausprobieren und BASIC-Kennenlernen aufgefordert werden.

Das Buch bietet Ihnen methodisch strukturiert die Grundlagen, die Sie benötigen, damit Sie selbst die ganze Palette des Programmierens anhand von attraktiven Programmen erfahren.

Lernen Sie durch dieses Standard-BASIC-Buch in BASIC denken, analysieren und programmieren. Ihr eigener Lernerfolg wird Sie auf jeder Seite des Buches anhalten, tiefer in die problemorientierte Arbeitsweise mit BASIC einzusteigen. Sie werden lernen, gezielt Problemstellungen in der für den Computer verständlichen Sprache BASIC zu formulieren und in ein Programm umzusetzen.

Sie werden in die Materie des Programmierens einsteigen und zielstrebig zum selbständigen Programmieren geführt. Sie werden erstaunt feststellen, wie leicht es ist, die Vielseitigkeit von BASIC zu nutzen.

Eine Programmiersprache zu beherrschen heißt, sie durch praktisches Anwenden am Computer zu erproben und begreifen zu lernen. Das sind die Ziele, die mit diesem Standard-BASIC-Buch angestrebt werden.

Die Programmbeispiele sind auf die Zielrichtung abgestellt, so daß nicht allein fertige Programmlösungen geboten werden, sondern Programme, die die Kreativität fördern und die Platz für eigene Ideen lassen. Freiräume also für Erweiterungen jeglicher Art, weil nur dadurch die notwendigen Kenntnisse und Erkenntnisse möglich werden, die nicht allein durch "Abtippen" oder "Laden" irgendwelcher Übungsprogramme erlangt werden.

Trotzdem bietet das Buch Lösungen für bestimmte Problemstellungen an, wie z. B. eines der Projekt-Programme "Zeichendefinition". Hier werden einerseits die Möglichkeiten des >> CPC 464 << aufzeigt, andererseits dem Lernenden exemplarisch Programmierpraxis vermittelt, die er für sich selbst nutzen kann.

Auch die reine Information, das Nachschlagenkönnen, ist in dem Buch verwirklicht. Alle Schneider BASIC-Befehle sind schlagwortartig in einer ausführlichen Form dargestellt. Das Stichwort alleine reicht für den Suchenden nicht aus. Selbstverständlich wurden das Format und der Anwendungszweck der Anweisungen alphabetisch aufgeführt.

Zudem wurde ein weiteres, nützliches Kapitel den Fehlermeldungen gewidmet. Hier findet man die deutsche Beschreibung aller Fehlermeldungen. Einige interessante Programme, die die Fehlermeldungen erzeugen, runden das Kapitel ab.

2. Das Standard-BASIC-Buch als Dialogpartner

Die Entstehung von Programmen

Sie können fertige Programme - Software - kaufen, von einem Programmierer zugeschnitten auf Ihren Anwendungsfall schreiben lassen oder aber selber ein Programm schreiben.
Allen drei Möglichkeiten haben eines gemeinsam, die Programme sollen einer Problemlösung dienen und immer gleichbleibende Tätigkeiten übernehmen.

Entscheidend ist daher die Betrachtung des Problems oder korrekter die Problemanalyse, die die Basis für die Konzeption des Programms bietet.
Gibt es für ein gestelltes Problem kein geeignetes, fertiges Programm, oder wollen Sie ein eigenes Programm schreiben, so läßt sich hier eine Empfehlung für die Vorgehensweise aussprechen:

Von dem eigentlichen Problem bis zur eigentlichen Problemlösung lassen sich einzelne Stufen der Problemlösung bestimmen, die für das Schreiben des eigentlichen Programms sehr nützlich und bestimmend sind.

Stufen der Programmentwicklung:

1 Stufe: Analyse des Problems
Fakten und Teilprobleme werden betrachtet und der Umfang des zu lösenden Problems exakt bestimmt.

2. Stufe: Analyse der Datenströme
Bestimmung der Datenströme, die zu vorgegebenen Zeitpunkten koordiniert ablaufen bzw. vorliegen müssen, damit ein sinnvoller Ablauf gewährleistet ist.

3. Stufe: Analyse der Programmlogik
Bestimmung der Tätigkeit, die zu einem bestimmten vorgegebenen Zeitpunkt vom Programm ausgeführt werden müssen, damit die analysierten Datenströme in ihrer logischen Reihenfolge fließen können.

4. Stufe: Deduzierung der Programmiersprache
Die Analysenwerte ermöglichen nun die Festlegung auf eine bestimmte Programmiersprache, z.B. BASIC, FORTRAN oder PASCAL. Alle besitzen für bestimmte Problemlösungen sowohl Vor- als auch Nachteile.

5. Stufe: Generierung des Programms
Hier werden nun alle analysierten Größen in ein Programm umgesetzt.

6. Stufe: Verifikation bzw. Falsifikation der Programmlösung
Es muß nun in allen nur denkbaren Programmteilen eine Prüfung auf Richtigkeit in Bezug auf die Problemlösung durchgeführt werden. Erweist sich das Programm in der Programmlogik und in Steuerung der vorgegebenen Datenströme für richtig, so besitzen Sie ein sogenanntes "lauffähiges Programm".

Das Lernen im Medienverbund

Die schrittweise Heranführung an die eigene Entwicklung von BASIC-Programmen ist eine der Zielrichtungen in diesem Buch.
Die Arbeitsstufen bilden in den einzelnen Kapiteln das Grundgerüst des Lernens. Die Stufen der Programmentwicklung werden indirekt in jedem der einzelnen Kapitel an nicht zu komplexen Problemen vorgestellt und mit Ihnen erarbeitet.

Problem- und Programmanalysen sind ein wichtiges Element der vorgeschlagenen Übungen.

Unterstützt werden Sie hierbei von den angebotenen "Hilfen zur Programmanalyse" und den "Vorüberlegungen", die Bestandteile in den einzelnen Kapiteln sind. Es soll hiermit die Struktur des Programmierens deutlich gemacht und die Kenntnisse vermittelt werden, die es Ihnen erlauben, auch andere, nicht in diesem Standard-BASIC-Buch thematisierte Anwendungsfälle mit BASIC zu lösen.

2. Das Standard-BASIC-Buch als Dialogpartner

Noch ein Hinweis für Ihre praktische Arbeit mit dem Buch als Ihrem Dialogpartner!
Folgen Sie dem Rat und arbeiten Sie stets den Text des Buches in der Weise durch, daß Sie sofort und gleich alles am >> CPC 464 << ausprobieren.

Nur in der Arbeit im Medienverbund:

"Buch, Programmkassette und >> CPC 464 <<"

erreichen Sie den gewünschten Erfolg!

Also bitte keinen "Trockenkurs"!

BASIC ist ein Kunstwort, das aus den Anfangsbuchstaben der nachstehenden englischen Worte zusammengesetzt ist:

Beginner's All-purpose Symbolic Instruction Code

zu deutsch "symbolischer Allzweckbefehlscode für Anfänger". BASIC gehört zu dem Kreis der problemorientierten Programmiersprachen wie etwa COBOL, FORTRAN und ALGOL, wobei BASIC eine gewisse Ähnlichkeit mit FORTRAN besitzt.

BASIC wurde Anfang der sechziger Jahre im Dartmouth College in den USA von den Professoren John. G. Kemeny und Thomas E. Kurtz geschaffen. Das angestrebte Ziel der beiden Sprachschöpfer war es, eine

- leicht zu erlernende
- interaktive und
- leicht anzuwendende

Programmiersprache zu gestalten, die der allgemeinen Programmierung dienen sollte.

BASIC hat seinen Siegeszug aber erst durch seine Verwendbarkeit im Bereich der Mikrocomputer angetreten. Mikrocomputer werden oft auch als HOME-Computer bezeichnet, obwohl sie sich mehr und mehr als ernstzunehmende Partner im beruflichen Alltag durchgesetzt haben.

BASIC läßt sich für mathematisch-technische und kaufmännische Programme, für einfache und komplizierte Unterhaltungsspiele, für Textbe- und -verarbeitung, prinzipiell für jede nur denkbare Art von Programmen verwenden.

BASIC ist eine problemorientierte, aber eine einfach zu erlernende Programmiersprache. Kenntnisse von irgendwelchen Maschinensprachen sind nicht erforderlich. Kommandos bzw. Anweisungen an den Computer werden in knappen, englischen Worten eingegeben. Die Programme sind dabei "transportierbar", d.h. sie sind auf den unterschiedlichsten Computermodellen und -fabrikaten lauffähig. Es kommt also nicht darauf an, welchen Computer man einsetzt. Allerdings gilt diese Aussage mit gewissen Einschränkungen.

3. Grundsätzliches über BASIC

BASIC hat im Verlaufe von 20 Jahren über 200 Dialekte entwickelt, so daß man folgendes feststellen kann:

In der einen BASIC-Version gibt es Kommandos, die in der anderen Version fehlen und umgekehrt.

Oftmals werden unterschiedliche Worte für ein- und dieselbe Funktion benutzt.

Die Umwandlung von einem BASIC-Dialekt in einen anderen ist oftmals nicht besonders schwierig, jedoch recht lästig.

Ihr >> CPC 464 << besitzt aber einen sehr umfangreichen BASIC-Wortschatz, der aus einem weitverbreiteten BASIC-Dialekt stammt und zusätzlich über einige sehr nützliche und interessante Graphik- und Sound-Anweisungen verfügt.
Wenn Sie jetzt fragen, wo der Schneider >> CPC 464 << einzuordnen ist, so kann man sagen, daß er ebenso als HOME-Computer dienen kann, wie auch nach entsprechender Aufrüstung als Büro-Computer.

Der BASIC-Interpreter

Die meisten BASIC-Sprachprozessoren sind Interpreter. Unter einem Interpreter versteht man ein Programm, das BASIC - Kommandos in die Maschinensprache des Mikrocomputers übersetzt. Jeder Mikrocomputertyp hat seinen eigenen BASIC-Interpreter; der Benutzer merkt davon jedoch nichts, er sieht nur ein- und dieselbe BASIC-Sprache!

So hat auch der Schneider >> CPC 464 << einen sehr umfangreichen BASIC - Interpreter, der ihnen sofort nach dem Einschalten zur Verfügung steht.

Zur Arbeitsweise des BASIC-Interpreters:

Jedesmal, wenn in einem Programm eine BASIC-Anweisung durchlaufen wird, muß der Interpreter sie in die Maschinensprache übersetzen. Wird eine Anweisung 1000 mal durchlaufen, wird sie auch 1000 mal übersetzt, d.h. interpretiert.

Der BASIC-Compiler

Anders geht ein BASIC-Compiler vor:

Ein Compiler übersetzt das gesamte Programm in die Maschinensprache, bevor es ausgeführt wird. Auf diese Weise wird jede BASIC-Anweisung nur einmal übersetzt, und zwar ohne Rücksicht darauf, wie oft die Anweisung im Programm vorkommt. Es ist deshalb leicht einzusehen, daß ein compiliertes Programm schneller läuft als ein interpretiertes.

Ein weiterer Vorteil des Compilers ist es, daß er beim Umwandeln das gesamte Programm auf die Einhaltung der sprachlichen Regeln überprüft und Syntax-Fehler, d.h. fehlende Kommandos, fehlerhaft geschriebene Befehlswörter meldet. Die erkannten Fehler werden ausgegeben, so daß der Programmierer sie in Ruhe beseitigen kann.

Der Interpreter dagegen meldet nur dann einen Fehler, wenn er bei der Programmausführung einen Fehler in einer Programmzeile findet. Ein scheinbar fehlerfreies Programm kann einige Male ohne Fehlermeldung gelaufen sein, bis es dann plötzlich zu einer Fehlermeldung kommt, die von einer vorher nicht benutzten Programmzeile ausgeht. Wird ein Fehler angetroffen, so wird das Programm vom Interpreter abgebrochen, d.h. es ist dann "abgestürzt".

Allerdings hat das Arbeiten mit dem Compiler auch seine Nachteile:

Zum einen benötigt der Compiler mehr Platz im Arbeitsspeicher des Computers, da im Speicher sowohl Platz für den Quellcode, das BASIC-Programm, als auch für den Objekt-Code, das Maschinenprogramm, vorhanden sein muß. Zum anderen macht jede kleine und kleinste Programmänderung in einem compilierten Programm eine erneute und zeitaufwendige Compilierung erforderlich. Viel leichter und schneller lassen sich Änderungen durchführen, wenn man mit dem BASIC-Interpreter arbeitet. Hier schreibt man eine fehlerhafte Anweisung neu und startet danach das Programm erneut. Bequemer geht es nicht!

Interpreter und Compiler im Wechsel

Es gibt aber auch einen Weg, die Vorteile von Interpreter und Compiler gemeinsam zu nutzen:

* Zuerst wird das Programm mit dem Interpreter erstellt und ausgetestet.

* Anschließend wird es mit dem Compiler bearbeitet, d.h. es wird compiliert.

* Programmänderungen werden mit dem Interpreter ausgeführt und wiederum ausgetestet.

* Danach wird das Programm wieder compiliert.

Auf diese Weise kann man in der ersten Phase des Programmierens das Programm sehr schnell ändern, unter dem Interpreter testen und besitzt nach der Compilierung ein schneller laufendes Programm, als bei einer Interpreterversion.

In welcher Sprache wird programmiert?

Um Mißverständnisse zu vermeiden:

Programmiert wird immer in BASIC.

Lediglich die Ausführung des Programms im Computer unterscheidet sich in der Ausführungsart als

* Interpreterfassung oder

* compilierte Fassung.

Mit dem Compilieren von BASIC-Programmen sollten Sie sich erst beschäftigen, wenn Ihnen BASIC geläufig ist.

Eine umfangreiche Tastatur und einen fest eingebauten Datacorder bietet der >> CPC 464 << in der Grundausstattung. Die vorhandenen Sondertasten und der Datacorder ermöglichen ein schnelles und bequemes Arbeiten mit dem Computer, wenn sie optimal genutzt werden.
Die Funktion der Sondertasten und des Datacorders sollte Ihnen vertraut sein, bevor Sie Ihr erstes BASIC-Programm schreiben.

Die folgenden knappen, grundsätzlichen Bedienungshinweise bieten Ihnen Hilfen für den Umgang mit dem Computer an.

Die Tastatur des >> CPC 464 <<

Sie finden auf der Tastatur die Sondertasten:

* ENTER
* SHIFT
* CAPS LOCK
* CTRL
* ESC
* DEL
* CLR
* Cursorsteuertasten
* COPY

Funktion der Sondertasten

ENTER-Taste

Die ENTER-Taste wird Eingabetaste genannt. Auf der Tastatur sind zwei vorhanden, die in ihrer Funktion gleich sind. Benutzt werden muß die ENTER-Taste immer dann, wenn Eingaben beendet werden sollen.
Arbeiten Sie im Direktmodus, dann werden die Kommandos direkt nach Drücken der ENTER-Taste ausgeführt. Schreiben Sie Programme, so werden hiermit die Programmzeilen in den Arbeitsspeicher des >>CPC 464<< übernommen.

SHIFT-Taste

Je eine SHIFT-Taste ist rechts und links auf der Tastatur angeordnet. Wird eine der SHIFT-Tasten in Kombination mit einer weiteren Taste gedrückt, so wird auf Großbuchstaben oder auf die zweite Tastenbelegung umgeschaltet.

CAPS LOCK-Taste

Die CAPS LOCK-Taste schaltet permanent auf Großbuchstaben-Modus um. Durch erneutes Drücken wird der Modus ausgeschaltet.
Die zweite Tastenbelegung auf den Zifferntasten der Haupttastatur wird weiterhin nur über die SHIFT-Taste erreicht.

CTRL-Taste

Die CTRL-Taste ist nur in Kombination mit anderen Tasten wirksam und dient dem Aufruf von Sonderfunktionen. Wird die CTRL- und die CAPS-LOCK-Taste gedrückt, so wird der Modus SHIFT LOCK ein- bzw. wieder ausgeschaltet. Die Wirkung entspricht der Umschaltung bei einer Schreibmaschine, d.h. es wird die zweite Tastenbelegung ausgegeben. In Verbindung mit den Cursorsteuertasten wird in einer Eingabezeile ein Springen des Textcursors an das Ende bzw. an den Anfang der Zeile bewirkt.

ESC-Taste

Die ESC-Taste bewirkt einen Abbruch des laufenden Programms unter BASIC und es erscheint die Meldung "*Break*".
In Kombination mit der SHIFT- und CTRL-Taste wird ein Reset des Systems ausgelöst, was einem Aus- und Einschalten des Geräts entspricht.

DEL-Taste

Mit der Taste kann das links neben dem Cursor stehende Zeichen gelöscht werden. Findet sich hier kein Zeichen, so wird ein Signalton ausgegeben.

CLR-Taste

Mit der Taste wird das vom Cursor angezeigte Zeichen gelöscht.

Cursorsteuertasten

Im Kommandomodus kann damit der Textcursor gesteuert werden. Er kann innerhalb einer Eingabezeile frei bewegt werden. Ist kein Zeichen eingegeben worden, so bleibt er über das gesamte Textfenster frei bewegbar.
Gedrückte SHIFT-Taste plus Cursorsteuertasten leiten die COPY-Funktion ein. Es erscheint der COPY-Cursor, der über das gesamte Bildschirmfenster frei bewegt werden kann.

COPY-Taste

Kopieren von Zeichen und Zeilen

Mit der COPY-Taste werden Zeichen oder ganze Programmzeilen kopiert.

Vorgehensweise:

1. Der normale Textcursor wird mit den Cursorsteuertasten auf die Einfügeposition gesetzt.

2. SHIFT- plus Cursor-Steuertaste drücken und hiermit COPY-Cursor auf die Anfangsposition der zu kopierenden Zeichenfolge setzen.

3. COPY-Taste drücken und so lange gedrückt halten, bis die gewünschte Zeichenfolge nach rechts gehend übernommen ist.

4. Die mit dem COPY-Cursor übernommenen Zeichen werden an der Textcursorposition immer rechts angehängt und so dupliziert.

5. Befindet sich der Textcursor innerhalb einer Eingabezeile, so wird die kopierte Zeichenfolge links vom Textcursor eingefügt und die ab der Textcursorposition vorhandenen Zeichen nach rechts verschoben.

6. Das Ende des Kopiervorganges wird mit Drücken der ENTER-Taste bestätigt. Sollen die schon kopierten Zeichen nicht übernommen werden, kann der Kopiervorgang durch Drücken der ESC-Taste aufgehoben werden.

Bildschirmeditor mit der COPY-Taste

Mit der COPY-Taste kann aber auch ein Bildschirmeditor realisiert werden:

Beispiel:

Soll eine Programmzeile auf dem Bildschirm editiert werden, so ist die Vorgehensweise wie folgt:

1. Der gewünschte Programmabschnitt ist mit der LIST-Anweisung zu listen.

2. Die zu ändernde Zeile ist mit dem Text-Cursor anzusteuern.

3. Es darf außer den vier Cursor-Steuertasten keine weitere Taste betätigt werden.

4. Mit der COPY-Taste muß die gesamte zu editierende Programmzeile in sich selbst übernommen werden, d.h. die Zeile wird aus dem Bildschirmspeicher kopiert.

5. Die zu ändernde Zeile ist nun als Eingabezeile in den Zeilenpuffer übernommen, so daß Sie geändert werden kann.

Die Vorgehensweise besitzt Vorteile!

* Der Überblick über den Programmblock bleibt erhalten.

* Beliebige Programmzeilen können auf dem Bildschirm zu neuen Programmzeilen zusammengefügt werden.

* Es sind Änderungen durchführbar, die normalerweise nur mit Texteditoren möglich sind.

Der Datacorder des >> CPC 464 <<

Das Magnetband als externer Datenträger

Im Gegensatz zu anderen tragbaren, batteriegepufferten Computern ist Ihr >> CPC 464 << mit einem sogenannten "volatilen", flüchtigen Speicher ausgestattet. Diese Art Speicher sichert nur bedingt die Daten und Programme. Wird die Stromzuführung unterbrochen, z.B. durch Betätigung des Netzschalters oder aber durch einen generellen Stromausfall, so sind die im Arbeitsspeicher befindlichen Daten und Programme unwiderruflich verloren bzw. gelöscht. Erst eine vorherige Sicherung der Daten und der Programme auf einem externen Datenträger garantiert die Datensicherheit. Synonym hierzu wird auch häufig der Begriff "peripherer Speicher" benutzt. Das Thema wird in einem weiteren Buch zum >> CPC 464 << aufgearbeitet, in dem Sie die Diskette als peripheren Speicher kennenlernen. In diesem Band werden Sie ausführlich in die Arbeitsweise der Magnetbandabspeicherung eingeführt.

Die Magnetbandaufzeichnung

Alle mit Magnetband arbeitenden Rekorder zeichnen die Daten seriell auf. Die Daten werden bei dieser Art der Abspeicherung hintereinander abgespeichert. Einen direkten oder auch wahlfreien Zugriff, "random access" genannt, gibt es nicht. Der wahlfreie

Zugriff ist erst beim Einsatz einer Diskettenstation oder einer Festplatte möglich. Die Zugriffszeit auf bestimmte Datensätze ist somit bei dem Einsatz eines Magnetbandes länger. Andererseits ist es die wohl kostengünstigste Abspeicherungsart und zusätzlich für viele Datenarchivierungsfälle die bessere Form der Langzeit-Speicherung.

Die Bedienung des Datacorders

Der in Ihrem >> CPC 464 << eingebaute Datacorder ist abgestimmt auf die speziellen Anforderungen, die bei dem Sichern und Abspeichern von Computerdaten gestellt werden.
Er zeigt auf den ersten Blick viel Ähnlichkeit mit einem handelsüblichen Rekorder. Dennoch sollten Sie das Kapitel nicht überblättern, damit Sie den Datacorder optimal ausnutzen können.

Die Funktionstasten des Datacorders

Im unteren rechten Bereich des >> CPC 464 << liegen die beschrifteten, für die Bedienung des Datacorders verfügbaren 6 Funktionstasten. Sie werden nun in ihrer Funktion kurz beschrieben, um Ihnen eine problemlose Handhabung der Übungs- und Programm-Kassetten zu ermöglichen.

REC

REC ist die Abkürzung für RECORD und bedeutet Aufnahme. Die Taste ist nur dann zu drücken, wenn Sie Daten oder Programme auf einer Compact-Kassette abspeichern wollen. Die REC - Taste ist ohne eingelegte Compact-Kassette und geschlossenem Datacorderschacht blockiert.
Sollen aus dem Arbeitsspeicher des >> CPC 464 << Daten auf eine Compact-Kassette abgelegt werden, so drücken Sie bitte zunächst die REC - Taste, bis sie einrastet. Halten Sie die Taste gedrückt, bis Sie zusätzlich die PLAY - Taste betätigt haben. Beide Tasten sind zum Abspeichern einzurasten, erst dann können Sie Ihre Daten abspeichern.

PLAY

Die gedrückte PLAY - Taste aktiviert den Datacorder, ohne daß hierbei ein Bandlauf durchgeführt wird. Ist die Taste eingerastet, dann können vom Datacorder computer- bzw. programmgesteuert Daten oder Programme in den Arbeitsspeicher des >> CPC 464 << eingelesen werden.
Erreicht Ihre Compact-Kassette das Bandende, so wird die PLAY - Taste durch einen mechanischen Auslöser in die Ruheposition zurückgesetzt.

REW

Die Taste ermöglicht Ihnen ein schnelles Zurückspulen des Kassettenbandes von der rechten auf die linke Spule. Beachten Sie hierbei aber, daß keine automatische Endabschaltung vorgenommen wird. Sie sollten daher diese Funktion nicht unbeaufsichtigt nutzen, um eine Motorüberlastung nach Erreichen des Bandendes zu vermeiden.

F.F.

Die Abkürzung F.F. steht für "fast forward", gleichbedeutend mit schnell vorwärts. Hiermit können Sie Ihr Kassettenband von rechts nach links spulen, d.h. vorwärts spulen.
Die F.F. - Funktion entspricht sonst der REW - Funktion, so daß auch hier der Hinweis auf die automatische Endabschaltung gilt.

STOP/EJECT

Die Taste ist mit zwei Funktionen belegt. Drücken Sie die STOP - Taste, so werden alle Datacorder-Funktionen augenblicklich abgebrochen und die vorher gedrückte Taste springt in die Ausgangsposition.

Die zweite Tastenfunktion dient dem Einlegen einer neuen Compact-Kassette. Wird die STOP - Taste wiederholt gedrückt, so öffnet sich das Kassettenfach.

PAUSE

Die PAUSE-Taste unterbricht die momentan eingestellte Funktion des Datacorders. Die vorher eingestellte Funktion kann durch erneutes Drücken der PAUSE-Taste wieder eingeschaltet werden. Sie wirkt nur in Verbindung mit der REC- und der PLAY-Taste. Beachten Sie, daß sie während eines Speicher- oder Ladevorgangs nicht genutzt wird, da hierdurch Schreib- und Lesefehler entstehen! Verfügen Sie über die zum Buch gehörenden Compact-Kassetten, werden softwaremäßig alle Pausen automatisch gesteuert, so daß Sie die Funktionstaste nicht benutzen müssen.

Die Auswahl der Compact-Kassette

Direkten Einfluß auf die Leistung des >> CPC 464 << Datacorders hat die Qualität und der Zustand der benutzten Compact-Kassetten.

* Verwenden Sie nur Qualitäts-Compact-Kassetten.

* C90 (2 x 45 Minuten) und C120 (2 x 60 Minuten) Compact-Kassetten sind für den Computereinsatz nicht empfehlenswert.

* C10 (2 x 5 Min.) bis C15 (2 x 7,5 Min.) Computer-Compact-Kassetten sind besonders geeignet für die Speicherung von Computerdaten.

* Die Bandsorte sollte zum Computereinsatz geeignet sein.

* Mit Musik bespielte Compact-Kassetten gehören in den HIFI-Kassettenrekorder.

* Neue, ausschließlich für den Computer bestimmte Compact-Kassetten sichern auf lange Zeit ihren Datenbestand.

Der Umgang mit der Compact-Kassette

Handelsübliche Compact-Kassetten sind auf ihrer Rückseite mit Schutzlaschen versehen. Wollen Sie ein unbeabsichtigtes Löschen oder ein Überspielen von Aufzeichnungen verhindern, so können die Schutzlaschen mit einem geeigneten Werkzeug ausgebrochen werden. Sind beide Schutzlaschen an der Compact-Kassette entfernt, so sind beide Seiten geschützt. Wollen Sie nur die A-Seite der Compact-Kassette schützen, so muß bei Draufsicht, das Magnetband zeigt zu Ihnen, die linke Lasche entfernt werden.
Wird eine so präparierte Compact-Cassette von Ihnen in den Datacorder eingelegt, so ist eine unbeabsichtigte Löschung nicht mehr möglich, da die Aufnahmetaste (REC) des Datacorders durch die fehlende Schutzlasche blockiert ist.
Wollen Sie auf einer so gesicherten Compact-Kassette Daten oder Programme löschen oder überspielen, so muß die Compact-Kassette wieder aufnahmetüchtig gemacht werden. Die durch die ausgebrochenen Schutzlaschen entstandenen Öffnungen sind dann mit einem Selbstklebeband abzudecken.

Die Speicher- und Lesegeschwindigkeit

Der >> CPC 464 << bietet zwei Abspeicherungsgeschwindigkeiten:

- Supersafe 1000 Baud
- Speedload 2000 Baud

Die Einheit "Baud" ist hierbei die Geschwindigkeit der seriellen Datenübertragung. Sie wird in Bits pro Sekunde (bps) bzw. Baud angegeben. Die Speedload - Geschwindigkeit liest Daten oder Programme demnach zweimal so schnell.

Der Vorteil der zwei verfügbaren Speicher- und Lesegeschwindigkeiten wird sofort deutlich, wenn Sie Ihre eigenen Programme mit Speedload abspeichern und wieder einlesen. Hier können Sie mit Speedload Zeit für das eigentliche Programmieren gewinnen, und Sie haben zugleich die Gewißheit, daß Ihre Programme und Daten sicher abgespeichert und geladen werden.

Auch die für den >> CPC 464 << angebotenen Basic Übungs- und Programm-Compact-Kassetten können Sie mit Speedload in den Arbeitsspeicher einlesen.

Die Supersafe - Geschwindigkeit wählen Sie immer dann, wenn Sie Ihr Programm nicht ausschließlich auf Ihrem >> CPC 464 << benutzen wollen. Das kann z. B. der Fall sein, wenn Sie Ihre Programme mit anderen >> CPC 464 << Besitzern tauschen wollen.

Geringere Geschwindigkeit bedeutet hier noch größere Einlese- und Abspeichersicherheit, trotz der vielleicht auftretenden Toleranzen bei der automatischen Aussteuerung unterschiedlicher Computer.

Ihr >> CPC 464 << erkennt außerdem automatisch die Einlesegeschwindigkeit, so daß Sie zu Ihrem Buch erworbene Übungs- und Programm-Compact-Kassetten ohne weitere Befehle einlesen lassen können.

Die BASIC - Datacorderbefehle

Der BASIC-Interpreter des >> CPC 464 << ist auch in dem Bereich der Datacorderbefehle ausgereift und umfangreich. Er garantiert mit den vorhandenen BASIC-Anweisungen ein komfortables Arbeiten. Für Ihren ersten Überblick sind die nutzbaren Anweisungen alphabetisch aufgelistet:

CAT	CHAIN	CHAIN MERGE	CLOSEIN
CLOSOUT	EOF	INPUT #9	LINE INPUT #9
LOAD	MERGE	OPENIN	OPENOUT
PRINT #9	RUN	SAVE	SPEED WRITE
WRITE #9			

Fünf der Anweisungen sollten Sie vorab kennenlernen, da sie für Ihre Arbeit mit dem Buch und der Programmkassette unbedingt benötigt werden.

Das Inhaltsverzeichnis erstellen

Die CAT-Anweisung

Anwendung: Die CAT-Anweisung liest die Kassette und erstellt ein Inhaltsverzeichnis, d.h. es werden die Namen der abgespeicherten Programme, die Programmblöcke und das Aufzeichnungsformat der Programme und Dateien auf dem Monitor ausgegeben.
Es werden mit der Anweisung keine Programme von Kassette eingelesen!
Lesen Sie möglichst jedes abgespeicherte Programm vor einer NEW-Anweisung oder Abschalten des >> CPC 464 <<.
Ist Ihr Programm in einer lesbaren Qualität abgespeichert worden, so erhalten Sie durch die CAT-Anweisung ein OK.
Das im Arbeitsspeicher vorhandene Programm wird hierbei nicht beeinflußt, so daß Sie, falls kein OK erscheint, das Programm wiederholt abspeichern können.

Auf dem Monitor wird ausgegeben:

Dateiname/Programmname Blocknummer Kennzeichen

Das Kennzeichen beschreibt das Aufzeichnungsformat nach folgendem Schlüssel:

$ = BASIC-Programme
% = geschütztes BASIC-Programm
* = ASCII-Datei
+ = binärer Datenblock

Wurde mit SAVE "" ein Programm abgespeichert, so erscheint der Hinweis:

"Unnamed file block >X<"

auf dem Monitor.

Das Laden von Programmen

Die LOAD-Anweisung

Format: LOAD "<Programmname>"

Anwendung: Die Anweisung lädt Programme bzw. Dateien von Kassette in den Arbeitsspeicher ein. Der <Programmname> darf maximal 16 Buchstaben aufweisen. Sie dürfen hier Groß- und Kleinbuchstaben vermischt benutzen.

Beispiel 1: LOAD ""

lädt das nächstliegende Programm von Kassette in den Arbeitsspeicher ein.

Beispiel 2: LOAD "FARBKREISE"

Es wird das Programm "Farbkreise" von der Kassette eingelesen.

Beispiel 3: LOAD "!Kartei"

Es wird das Programm "Kartei" in den Arbeitsspeicher des >> CPC 464 << geladen, ohne die englischen Bedienungshinweise auf dem Monitor auszugeben.

Beispiel 4: LOAD"<Programmname>",Anfangsadresse

Zum Laden von Binärdateien kann zusätzlich eine Anfangsadresse im Speicher angegeben werden, ab der die Binärdatei abgelegt wird.

Das Laden und Starten von Programmen

Die RUN-Anweisung

Format: RUN "<Programmname>"

Anwendung: Die Anweisung lädt ein Programm mit dem angegebenen <Programmnamen> von der Programmkassette in den Arbeitsspeicher des >> CPC 464 << ein und startet danach unmittelbar das Programm ab der ersten Programmanweisung.
Der <Programmname> darf maximal 16 Tastaturzeichen aufweisen. Wird hierbei als erstes Zeichen des Programmnamens ein Ausrufezeichen " ! " geschrieben, so werden die Anweisungen der Rekorderbedienung und der gefundenen Programmnamen auf dem Monitor nicht ausgegeben.
Beachten Sie bei der Anwendung der Anweisung, daß ein im Speicher vorhandenes Programm und alle Variablen gelöscht werden!

Beispiel 1: RUN "Adressenprogramm"

Gibt Bedienungshinweise auf dem Monitor aus, lädt das Programm "Adressenprogramm" in den Arbeitsspeicher, löscht ein vorhandenes Programm im Speicher und startet das Programm unmittelbar.

Beispiel 2: RUN "!Telefon"

Gibt keine Hinweise auf dem Monitor aus und entspricht sonst dem Beispiel 1.

Beispiel 3: RUN ""

Lädt das nächste auf der Kassette gefundene Programm in den Arbeitsspeicher und entspricht sonst Beisp. 1 .

Beispiel 4: RUN "!"

Gibt keine Hinweise auf dem Monitor aus und entspricht sonst dem Beispiel 3.

Hinweis: RUN "" entspricht auch der Tastenkombination CTRL plus kleine ENTER-Taste!

Die Tastenkombination CTRL plus "kleine ENTER-Taste" wird verwendet, um das nächste Programm auf der eingelegten Programmkassette in den Arbeitsspeicher einzulesen und automatisch zu starten.

Die Vorgehensweise ist: Die CTRL-Taste wird gedrückt und festgehalten und dann zusätzlich die kleine ENTER-Taste gedrückt. Es erscheint auf dem Bildschirm die Meldung:

RUN"
Press PLAY then any key:

Drücken Sie nun die Taste PLAY auf Ihrem Datacorder und danach eine beliebige Taste außer CAPS LOCK, SHIFT und CTRL.

Abgebrochen wird die Funktion mit der ESC-Taste.

Das Speichern von Programmen

Die SAVE-Anweisung

Format: SAVE "<Dateiname>"[,<Dateityp>][,<binäre Parameter>]

Anwendung: SAVE speichert ein Programm mit dem angegebenen Programmnamen bzw. <Dateinamen> aus dem Arbeitsspeicher auf Kassette. Der <Dateiname> kann maximal 16 Tastaturzeichen aufweisen. Die Angabe des <Dateinamens>, des Dateityps und der binären Parameter kann zur Programmabspeicherung entfallen.

Der Dateityp wird bestimmt durch:

A = ASCII - Abspeicherung des Programms
P = geschütztes Programm "LIST - Schutz"
B = Abspeicherung eines Speicherbereiches

Beispiel 1: SAVE ""

BASIC speichert das Programm als unbenannte Datei ab und es erscheint der Hinweis

"Saving Unnamed file block 1"

auf dem Monitor.

Beispiel 2: SAVE "!Textpro",P

BASIC speichert hierbei das Programm "Textpro" LIST-geschützt auf Kassette und unterdrückt Bedienungshinweise auf dem Monitor.

Vorsicht!
Sie können Ihre eigenen Programme mit der vorangegangenen Anweisung nicht mehr "listen".

Das Einstellen der Abspeichergeschwindigkeit

Die SPEED WRITE-Anweisung

Anwendung: Die SPEED WRITE-Anweisung gestattet es, Daten und Programme in zwei unterschiedlichen Geschwindigkeiten auf Kassette abzulegen.

SPEED WRITE 1 legt mit 2000 baud und

SPEED WRITE 0 mit 1000 baud auf Kassette ab.

Werden abgespeicherte Programme oder Daten wieder von Kassette eingelesen, so erkennt der >> CPC 464 << automatisch die vorgegebene Lesegeschwindigkeit. Die nach dem Einschaltvorgang vorgegebene Geschwindigkeit ist SPEED WRITE 0, d.h. die Abspeichergeschwindigkeit beträgt hier 1000 Datenbits pro Sekunde.
Sollte irrtümlich ein negativer oder ein Wert größer "1" eingegeben worden sein, so erscheint die Fehlermeldung:

"Improper argument".

Hinweis:

Aufnahmen sollten prinzipiell nicht direkt am Bandanfang beginnen. Der Vorspann am Bandanfang und -ende ist für eine Aufzeichnung ungeeignet. Er dient dazu, Verschmutzungen an dem eigentlichen Magnetband zu verhindern. Häufig sind aber trotz aller Umsicht die ersten Bandabschnitte für eine Speicherung von Daten nicht geeignet. Es ist daher ratsam, Daten erst nach einem Vorlauf von 5 Sekunden auf die Datenkassette abzuspeichern, um eventuellen Störungen, d.h. im schlimmsten Falle Datenverlusten, zu begegnen.

Begriffserläuterungen

Bevor mit dem eigentlichen Programmieren begonnen wird, sollten Sie sich mit einigen Begriffen vertraut machen, die im weiteren Verlauf des Buches immer wieder benutzt werden.

Dieses sind die Begriffe:

- Laden
- Kommando
- Anweisung und Anweisungsnummer
- Konstante
- Variable
- Alphanumerische Zeichen
- Numerische Zeichen
- Zeichenketten
- Zeichenkettenvariable
- Zeichenkettenliteral
- Arithmetische Anweisung
- Reservierte BASIC-Worte

Laden

Der BASIC-Interpreter befindet sich in nichtflüchtigen Speicherbausteinen, ROM genannt. Sie unterscheiden sich zu den RAM Speicherbausteinen dadurch, daß ihr Inhalt auch nach dem Abschalten des Computers erhalten bleibt.
Der Interpreter muß nicht mit einem besonderen Vorgang geladen werden, sondern steht Ihnen sofort nach dem Einschalten automatisch zur Verfügung. Sie erkennen es daran, daß sich der BASIC-Interpreter nach dem Einschalten mit einem "Ready" meldet.

Wenn Sie sich später mit der Diskettenarbeit beschäftigen, werden Sie auch ein BASIC kennenlernen, das Sie von einer Diskette in den Speicher des >> CPC 464 << laden müssen.

5. Begriffe aus der Datenverarbeitung

Kommando

Kommandos sind Systembefehle, die im Regelfall außerhalb des BASIC - Programms verwendet werden.

Beispiel:

Sie haben ein BASIC-Programm in den Mikrocomputer eingetippt und wollen es sich auf dem Bildschirm anschauen. Sie geben den Systembefehl, das Kommando:

```
LIST
```

über die Tastatur ein und erhalten auf dem Bildschirm das eingegebene Programm angezeigt. Ist Ihrer Meinung nach alles in Ordnung, dann geben Sie den Systembefehl:

```
RUN
```

ein und starten die Programmausführung.
Sind BASIC-Anweisungen in der Art anzuwenden, so wird in diesem Buch vom Kommando- bzw. Direktmodus gesprochen.

Anweisung, Anweisungsnummer

Eine Anweisung tritt im Regelfall innerhalb eines Programms auf. Der Anweisung geht eine Anweisungsnummer voraus, auch Zeilennummer genannt.

Einige Beispiele:

```
10 INPUT B,C
20 A = B * C
30 PRINT A,B,C
```

Für das allgemeine Verständnis hier die deutsche Übersetzung:

```
10 GIB EIN (die Werte für) B,C
20 (Berechne) A (als Produkt von)B * C
30 GIB AUS (auf dem Bildschirm die Werte von) A,B,C
```

Selbstverständlich dürfen Sie nur in der erstgenannten englischen Fassung programmieren. Die deutsche Fassung dient hier nur dem Verständnis.

Konstante

Eine Konstante ist ein Festwert, der während der Programmausführung nicht verändert werden kann.

Ein Beispiel:

```
10 G = 9.80665
```

Die Konstante ist hier die Zahl 9.80665.

Variable

Unter einer Variablen versteht man beim Programmieren einen symbolischen Namen, unter dem auf einem Speicherplatz ein Wert abgelegt wird. Der Wert kann eine Konstante sein, kann aber auch im Verlauf des Programms errechnet werden.

Beispiel:

```
10 A = 9.80665
20 B = C * A
```

In der Anweisung mit der Anweisungsnummer 10 wird während der Programmausführung der Variablen A der Wert 9.80665 zugewiesen. In der Anweisung mit der Anweisungsnummer 20 wird während der Programmausführung der Variablen B der Wert aus dem Produkt C * A zugewiesen, wobei die Variable C vorher bereits einen Wert erhalten haben muß. Ist dies nicht geschehen, so hat sie den Wert "NULL". Bei der Programmausführung ergibt sich dann natürlich für die Variable B auch der Wert NULL, da bekanntlich jeder mit NULL multiplizierte Wert nur den Wert NULL ergeben kann.

5. Begriffe aus der Datenverarbeitung

Alphanumerische Zeichen

Hierunter versteht man jedes druckbare Zeichen, das für den jeweiligen Computer in einem Zeichensatz (Tabelle aller verfügbaren Zeichen) festgelegt ist. Zwar gibt es international anerkannte, standardisierte Zeichensätze, dennoch gibt es in bestimmten Punkten wichtige Unterschiede, so etwa verschiedene Ländervarianten. Ihr Schneider >> CPC 464 << besitzt die amerikanische, die internationale Fassung.

Ein Auszug aus dem Zeichenvorrat:

```
  ! " # $ % & ' ( ) * + , - . / 0 1 2 3
4 5 6 7 8 9 : ; < = > ? @ 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 [
\ ] ^ _ ` 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 { | } ~
```

Der >> CPC 464 << besitzt zudem noch Sonderzeichen und einen Zeichensatz mit Graphik-Symbolen und die besondere Fähigkeit, sämtliche denkbaren Symbole oder Zeichen vom Benutzer selbst definieren zu lassen, also auch jeden denkbaren Länderzeichensatz, wie z.B. den griechischen oder spanischen.
Mit Ziffern, die als alphanumerische Zeichen, so etwa in Texten, verwendet werden, kann man keine Rechnungen ausführen! Dazu bedarf es der numerischen Zeichen.

Numerische Zeichen

Hierunter versteht man Zahlen der unterschiedlichsten Art. Auf wichtige Unterteilungen wird im weiteren Verlauf noch eingegangen; hier in "Kurzfassung" die grundsätzlichen Erläuterungen:

* Ganze Zahlen (INTEGER-Zahlen), wie z.B. die Zahlen:

  1 2 36 -60 123456 und -2643

* Gleitpunktzahlen, die der Einfachheit halber hier erst einmal durch Dezimalzahlen dargestellt werden.

  Beachten Sie bitte, daß in Amerika und England an die Stelle des Dezimalkommas der Dezimalpunkt tritt. Sie schreiben also in BASIC:

  9.80665 1.567 0.23 1234.56 usw.

Zeichenketten

Unter Zeichenketten versteht man alle in doppelten Anführungszeichen eingeschlossenen, alphanumerischen Zeichen.

Beispiel:

"Dies ist eine Zeichenkette mit insgesamt 51 Zeichen"

Zeichenkettenvariablen

Zeichenkettenvariablen sind symbolische Namen für Speicherplätze bzw. Speicherbereiche, auf denen Zeichenketten untergebracht sind. Die Zuordnung geschieht durch entsprechende Anweisungen.

Beispiele:

```
10 K$ = "Krankenkassenbeitrag"
20 GG$ = "Grundlohn"
```

Beachten Sie bitte, daß die Namen der Zeichenkettenvariablen am Ende stets durch ein Dollarzeichen ($) gekennzeichnet sein müssen (K$, GG$).

Zeichenkettenliteral

Steht eine in doppelte Anführungszeichen gesetzte Zeichenkette in einer Anweisung, ohne dabei einer Zeichenkettenvariablen zugeordnet zu sein, handelt es sich um ein Zeichenkettenliteral.

Beispiel:

```
10 PRINT"Menge Einheitspreis Gesamtpreis"
20 PRINT"------------------------------"
```

Hier werden die Zeichenkettenliterale dazu benutzt, um Überschriften auf dem Bildschirm darzustellen.

Arithmetische Anweisung, arithmetischer Ausdruck

Ein arithmetischer Ausdruck steht in einer arithmetischen Anweisung stets rechts vom "Gleichheitszeichen".

Beispiel:

```
10 A = B * C
```

In dieser arithmetischen Anweisung ist B * C der arithmetische Ausdruck.

Gleichheitszeichen in arithmetischen Anweisungen

Beispiel:

```
10 A = B * C
```

Bei dieser arithmetischen Anweisung handelt es sich nicht um eine Gleichung im herkömmlichen, mathematischen Sinn, sondern um eine Anweisung an den Computer, der Variablen A den Wert des Produktes B * C zuzuweisen. Die Anweisung mit der Anweisungsnummer 10 wird folgendermaßen gelesen:

> Multipliziere B mit C und schreibe das Ergebnis auf den Speicherplatz mit dem symbolischen Namen A.
>
> oder:
>
> Ersetze den Wert A durch den Wert des Produktes B * C.

Das Gleichheitszeichen hat also die Bedeutung:

"Ersetze... durch..."

Liste der Variablen

In einer Eingabe- bzw. Ausgabeanweisung müssen die ein- oder ausgehenden Variablen aufgeführt sein.

Beispiel:

```
PRINT A,B,C,D,X
```

Die Anweisung veranlaßt den Computer, die aktuellen Werte von A, B, C, D und X auf dem Bildschirm anzuzeigen. Die Liste der Variablen lautet hier "A, B, C, D, X".

Reservierte BASIC-Worte

Reservierte BASIC-Wörter wie etwa LIST, PRINT, RUN usw. dürfen nicht als Namen von Variablen benutzt werden. Sie dienen als Instruktionen (Befehle) für den Computer. Die vollständige Liste der reservierten BASIC-Worte finden Sie im Anhang.

Allgemeiner Hinweis

Auf die Angabe weiterer Begriffserläuterungen in diesem einführenden Abschnitt wird verzichtet. Sie werden an geeigneter Stelle in den jeweiligen Kapiteln erfolgen.

> Beachten Sie bitte: Die vorstehenden Erläuterungen dienen der ersten Information und sind in diesem Sinne kein auswendig zu lernender Lehrstoff.

6. Erste Anwendungsversuche in BASIC

Einführung

Die Einleitung zum Thema BASIC sollte dazu dienen, Sie in die problemorientierte Programmiersprache BASIC einzuführen. Grundsätzliches über BASIC wurde bereits angesprochen. Lassen Sie uns nun kurz einige Dinge wiederholen und andere neu hinzufügen.

* BASIC ist eine Mikrocomputer-Sprache
* Mikrocomputer wurden aus Amerika eingeführt
* Ihre Zahl wächst in Deutschland ständig
* Moderne Technologie läßt die Mikrocomputer immer besser, handlicher und bedienerfreundlicher werden.
* Der >> CPC 464 <<, mit dem Sie jetzt BASIC lernen, zählt zu diesen Mikrocomputern.

Verständlicherweise wächst durch derartig handliche Computer die Zahl der Hobby- und Profi-Programmierer, die sich des neuen Mediums aus den unterschiedlichsten Gründen bedienen. Versuche, BASIC auf Großrechenanlagen umfassend einzusetzen, hatten bei weitem nicht den durchschlagenden und erwarteten Erfolg.

Die Situation ist bei den Mikrocomputern umgekehrt. In allen größeren Orten findet man inzwischen neben den Computern selbst ein sehr reichhaltiges Literaturangebot über BASIC und eine Fülle von BASIC-Programmen, wie die z.B. aus dem HEIM-Verlag.

Aufgabenstellung: Die Addition von Zahlen

Alle Welt lernt heute BASIC, versuchen Sie es auch. Begonnen wird jetzt mit einigen sehr einfachen Beispielen. Eine kleine Aufgabe soll gelöst werden, die ebensogut mit einem Taschenrechner gelöst werden könnte.

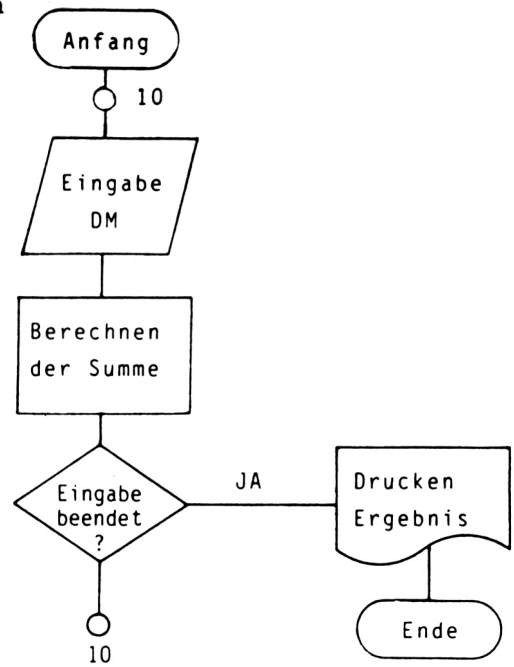
Es soll die Summe aus einer beliebigen Anzahl von Geldbeträgen gebildet werden:

123.05 DM
99.50 DM
256.70 DM
1.75 DM

Führen Sie die Rechnung mit dem Taschenrechner aus, so erhalten Sie den Betrag von 481.00 DM.

Diese Zahlen dienen nur als Beispiel, d. h. an ihre Stelle könnten ebenso andere Zahlen treten.
Es handelt sich hierbei um eine Addition von Variablen, so daß sich der folgende Programmablaufplan ergibt.

Programmablaufplan



Zu dem Ablaufplan gehört das recht einfache BASIC-Programm:

```
10 INPUT DM
20 SUM = SUM + DM
30 IF DM<>0 THEN 10
40 PRINT "--------------"
50 PRINT SUM
```

Im vorigen Kapitel wurde schon ein Beispiel für Anweisungen und Anweisungsnummern vorgestellt. Betrachten Sie jetzt schrittweise diese Programmzeilen!

Die Anweisung mit der Anweisungsnummer 10:

```
10 INPUT DM
```

Die INPUT-Anweisung

Das Wort INPUT ist ein reserviertes BASIC-Wort, mit dem der Bediener aufgefordert wird, über die Tastatur des >> CPC 464 << eine Eingabe zu machen. Die Aufforderung zur Eingabe wird auf dem Bildschirm durch ein Fragezeichen wiedergegeben. Es handelt sich um ein Anforderungszeichen oder auch Aufforderungszeichen mit der Bedeutung:

```
"Gib jetzt etwas ein!"
```

Beachten Sie, daß die Aufforderung zur Eingabe durch das BASIC-Wort INPUT ausgelöst wird. Das Fragezeichen ist hierbei die bildhafte Darstellung der Aufforderung zur Eingabe.

Fehleingaben und Fehlermeldung

Was aber dürfen Sie eingeben?

Im vorhergehenden Kapitel wurde von numerischen und alphanumerischen Zeichen sowie Zeichenketten gesprochen. Nach dem Wort INPUT dürfen Sie in diesem Programmbeispiel nur Zahlen, den Dezimalpunkt und die Vorzeichen Plus (+) und Minus (-) eingeben.

Geben Sie ein anderes Zeichen wie etwa einen Buchstaben oder ein

Sonderzeichen ein, wird das Anforderungszeichen (?) erneut auf dem Bildschirm ausgegeben. Dahinter erscheint als Fehlermeldung die Nachricht:

```
?Redo from start
```

Dieses kann frei übersetzt werden mit:

"Geben Sie erneut, und zwar von Anfang an ein!"

Der häufigste Eingabefehler ist hierbei, daß anstelle des Dezimalpunktes ein Dezimalkomma eingegeben wird. Wird z.B. 123,05 und nicht 123.05 eingegeben , so erscheint wieder die Fehlermeldung:

```
?Redo from start
```

Die Einschränkung "in diesem Programm" bedeutet, daß das Wort INPUT nicht für sich allein gesehen werden darf. Betrachten Sie noch einmal die Anweisung 10:

```
10 INPUT DM
```

Hinter INPUT steht der Name einer Variablen, nämlich DM. Im Abschnitt "Begriffserläuterungen" wurde unter dem Stichwort "Variable" der Begriff erläutert und unter dem Stichwort "Zeichenkettenvariable" dieser Begriff dagegen abgegrenzt. Lesen Sie bitte dort noch einmal nach!

Regeln zur INPUT-Anweisung:

* Die INPUT-Anweisung erlaubt, Eingaben über die Tastatur in ein Programm einzulesen.

* Zeichenketten und Zahlenwerte können mit der INPUT-Anweisung eingelesen werden.

* Die Kommastelle von Dezimalzahlen muß mit einem Punkt angegeben werden.

* Zeichenkettenvariablen müssen mit dem $ Zeichen gekennzeichnet werden.

* Numerische Variablen dürfen nicht mit dem $ Zeichen gekennzeichnet werden.

* Eine fehlerhafte Eingabe führt zu der Fehlermeldung "?Redo from start"

BASIC-Sprachregeln zur INPUT-Anweisung

Die Variable DM enthält daher kein Dollarzeichen am Namensende. Es handelt sich hierbei um eine numerische Variable.

Es bleibt zu fragen, wie vorgegangen werden muß, wenn nicht nur ein einfaches Fragezeichen auf dem Bildschirm als Aufforderung zur Eingabe erscheinen soll, sondern eine wörtliche Abfrage, wie etwa: "Betrag"?

Dies kann an einem Beispiel verdeutlicht werden. Hierzu wird die Anweisung Nr. 10 wie folgt geändert:

```
10 INPUT "Betrag:";DM
```

Wie Sie erkennen können, wurde eine Zeichenkette hinter der INPUT-Anweisung eingefügt, nämlich das Wort "Betrag:" und anschließend ein Semikolon (;) gesetzt.
Das Semikolon ist durch die BASIC-Sprachregeln in einer Eingabeanweisung (INPUT) dann vorgeschrieben, wenn eine Zeichenkette hinter INPUT folgt.
Erst hinter dem Semikolon darf die Variable, in dem Beispiel DM, stehen.

Das Fragezeichen wird nach dem Zeichenkettenliteral auf dem Bildschirm angezeigt. Beachten Sie hierbei die BASIC-Sprachregeln und das Format der Anweisung.

Es sind Fälle denkbar, bei denen nach einer Zeichenkette das Fragezeichen absolut überflüssig ist und nicht ausgegeben werden soll.

Anstelle des Semikolons muß dann ein Komma gesetzt werden, das Fragezeichen erscheint dann nicht!

Beispiel:

```
10 INPUT "Betrag:";DM
```

Dieses Format der INPUT-Anweisung liefert auf dem Bildschirm die Abfrage:

```
Betrag:?
```

Beispiel:

```
10 INPUT "Betrag:",DM
```

Das Format der INPUT-Anweisung liefert auf dem Bildschirm die Abfrage:

```
Betrag:
```

Das Fragezeichen hinter "Betrag:" wurde hier nicht ausgegeben!

Zusammenfassung:

* Eine Zeichenkette hinter der INPUT-Anweisung ist mit einem Semikolon oder Komma von der (den) Variablen zu trennen.

* Wird das Semikolon gewählt, so wird ein Fragezeichen angezeigt.

* Wird ein Komma nach der Abfrage gesetzt, so erscheint kein Fragezeichen.

Einer Zeichenkettenvariablen kann ein bestimmter Wert zugeordnet werden. Was geschieht aber, wenn im Zeichenkettenliteral auch ein Komma auftaucht?

Das folgende Beispiel macht die Lösung deutlich:

```
100 A$ = "Abzuege, ohne Beitraege zu Ver
baenden"
```

Das Zeichenkettenliteral muß in Hochkommata eingeschlossen werden, erst dann darf innerhalb der Zeichenkette selbst auch ein Komma vorkommen.

Ein ergänzender Hinweis zur INPUT-Anweisung:

```
10 INPUT A,A$
```

In dieser Eingabe-Anweisung kommen zwei Variablen vor, eine numerische (A) und eine Zeichenkettenvariable (A$), die durch ein Komma voneinander getrennt sind. Dies hat zur Folge, daß Sie nach der Eingabe der Variablen A ein Komma eingeben müssen, das vom BASIC-Interpreter auch tatsächlich erwartet wird. Danach dürfen Sie dann die Zeichenkettenvariable eingeben.
Hier gilt die Regel, daß die Zeichenkettenvariable ohne Hochkommata eingegeben werden kann, wenn sie selbst kein Komma enthält. Andernfalls muß sie in Hochkommata eingeschlossen werden.

Es ist sicher ganz nützlich, wenn nun die verschiedenen Möglichkeiten zusammengestellt werden:

```
10 INPUT A,A$
20 PRINT"---------------------------------
---------"
30 PRINT A,A$
```

Sie starten das vorstehende Programm mit RUN und geben ein:

1. Beispiel:

Auf dem Bildschirm erscheint:

```
? 15,Stueck
-----------------------------------------
 15            Stueck
```

2. Beispiel:

Auf dem Bildschirm erscheint:

```
? ,Stueck
-----------------------------------------
 0              Stueck
```

3. Beispiel:

Auf dem Bildschirm erscheint:

```
? 15,
-----------------------------------------
 15
```

4. Beispiel:

Auf dem Bildschirm erscheint:

```
? 15,"DM, ohne Skonto"
-----------------------------------------
 15             DM, ohne Skonto
```

Diese Beispiele werden Ihnen sicher die Möglicheiten direkt am Computer aufgezeigt haben. Kehren Sie nun wieder zu dem ursprünglichen Programmbeispiel zurück:

```
10 INPUT DM
20 SUM = SUM + DM
30 IF DM<>0 THEN 10
40 PRINT"--------------"
50 PRINT SUM
```

Die Eingabe-Anweisung wurde schon ausführlich besprochen, so daß Sie zur Anweisung 20 bzw. zu der Zeile 20 Übergehen können!

Arithmetische Anweisungen in BASIC-Programmen

```
20 SUM = SUM + DM
```

Es handelt sich hier um eine arithmetische Anweisung, vgl. Sie bitte auch im vorhergehenden Kapitel unter Punkt "Arithmetische Anweisung".
In dem Abschnitt "Gleichheitszeichen in arithmetischen Anweisungen" wird angegeben, daß es sich bei einer arithmetischen Anweisung nicht um eine Gleichung im herkömmlichen, mathematischen Sinn handelt. Hier steht rechts vom Gleichheitszeichen der arithmetische Ausdruck:

```
SUM = SUM + DM
```

Die Variable SUM steht aber bereits links vom Gleichheitszeichen. Da das Gleichheitszeichen auch als "Ersetze... durch ..." gelesen werden kann, darf man also wie folgt lesen:

Ersetze den Inhalt von SUM durch den Inhalt von SUM + DM.

Dieses ist so zu verstehen:

Steht eine Variable links vom Gleichheitszeichen und ebenso rechts davon, so handelt es sich bei der Variablen gewissermaßen um ein Rechenfeld. Sie müssen eine derartige Anweisung dann so lesen:

> "Ersetze den bisherigen Inhalt der Variablen SUM durch den Wert, der sich aus der Addition des bisherigen Inhalts von SUM und dem Inhalt der Variablen DM ergibt!"

oder kürzer:

> "Addiere den Inhalt von DM zum Inhalt von SUM hinzu."

Dieser Vorgang läuft im >> CPC 464 << wie folgt ab:

Im Mikrocomputer ist das Rechenwerk mit seiner arithmetischen Einheit für den Ablauf verantwortlich.

Dieser läßt sich ohne technische Einzelheiten so erläutern:

1. Schritt: Bringe den Inhalt des Speicherplatzes SUM in den Akkumulator.

2. Schritt: Addiere den Inhalt der Variablen DM zu dem Inhalt des Akkumulators hinzu.

3. Schritt: Übertrage den Inhalt des Akkumulators auf den Speicherplatz der Variablen SUM und lösche bei der Übertragung den alten Inhalt von SUM durch Überschreiben.

Die Bezeichnung "Rechenfeld" wird sowohl für die rechts als auch links von dem "Gleichheitszeichen" auftretende Variable angewendet. Eine andere Bezeichnung, die sich für die Variable links vom Gleichheitszeichen eingebürgert hat, ist die Bezeichung: "Summenfeld".

Die IF THEN-Anweisung

Die nächste Anweisung in dem Programm lautet:

```
30 IF DM<>0 THEN 10 !!
```

Es handelt sich hierbei um eine

WENN-DANN-Anweisung,

umgangssprachlich formuliert:

WENN DM UNGLEICH NULL, DANN (gehe nach Zeile) 10

Die Worte "gehe nach Zeile" wurden in Klammern gesetzt, weil sie in der englischen Fassung bzw. in der BASIC-Anweisung nicht vorkommen.

Diese Anweisung ermöglicht, das Ende der Eingabe abzufragen und bei Ende der Eingabe in die Schlußroutine zu springen.

Wenn also DM nicht gleich Null ist, wird vom Programm automatisch zur erneuten Eingabe nach Zeile 10 verzweigt.

Die gleiche Anweisung könnte aber auch lauten:

```
IF DM<>0 THEN GOTO 10
```

Hieraus werden die notwendigen Grundbestandteile der WENN-DANN - Anweisung in BASIC erkennbar:

"IF Bedingung THEN Sprungziel"

Nun zur nächsten Anweisung im Programmbeispiel!

Die PRINT-Anweisung

Sie lautet:

```
40 PRINT"--------------"
```

Hier wird, wie bei den Begriffserläuterungen unter dem Stichwort "Zeichenkettenliteral" an einem Beispiel gezeigt, das Zeichenkettenliteral

"--------------"

auf dem Bildschirm ausgegeben.

Entsprechend ist auch die letzte Anweisung

```
50 PRINT SUM
```

zu deuten. Sie gibt den Inhalt von SUM auf dem Bildschirm aus.

Die Ausgabe-Anweisung PRINT wirkt hier auf den Bildschirm und stellt die in der Liste angegebenen Variablen und Literale dar. Allerdings wird mit PRINT ohne jeden Format-Zusatz ein Standardformat genutzt, das in den Sprachregeln enthalten ist.

Zusammenfassung:

* Ein Zeichenkettenliteral muß in Hochkommata eingeschlossen sein, wenn innerhalb der Zeichenkette ein Komma vorkommt.

* Variable können auch als "Rechenfeld" bezeichnet werden, wenn rechts und links vom Gleichheitszeichen eine Variable steht.

* Variable links vom Gleichheitszeichen werden auch "Summenfeld" genannt.

* Mit der WENN-DANN-Anweisung werden Abfragen zur Programmverzweigung programmiert.

* Die PRINT-Anweisung gibt Variable und Literale auf dem Monitor oder Drucker aus.

* PRINT ohne Format-Zusatz nutzt ein Standardformat auf dem Bildschirm.

Übungen

1 Schreiben Sie eine Anweisung, die auf dem Bildschirm folgende Überschrift liefert:

Auswertungsergebnisse des Versuchs

Antwort:

.....

2 Schreiben Sie eine Anweisung, in der der Seitenzähler "SEITE" um 1 hochgezählt wird.

Antwort:

.....

3 Schreiben Sie eine Anweisung, in der das Produkt aus Menge mal Einheitspreis unter der Variablen Betrag abgelegt wird.

Antwort:

.....

4 Definieren Sie die Begriffe "arithmetische Anweisung" und "arithmetischer Ausdruck" an dem folgenden Beispiel:

```
10 GESAMTBETRAG=BETRAG+MEHRWERTSTEUER
```

Antwort:.....

.....

5 Wodurch wird eine Aufeinanderfolge von Zeichen zu einem Zeichenkettenliteral?

Antwort:.....

Lösungen

Zu 1:

```
PRINT "Auswertungsergebnisse des Versuchs"
```

Zu 2:

```
10 SEITE = SEITE + 1
```

Zu 3:

```
BETRAG = MENGE * EINHEITSPREIS
```

Zu 4:

Die arithmetische Anweisung besteht aus einer Variablen links vom Gleichheitszeichen und einem arithmetischen Ausdruck rechts vom Gleichheitszeichen. In unserer Übung ist die Variable links vom Gleichheitszeichen "GESAMT-BETRAG" und der arithmetische Ausdruck rechts vom Gleichheitszeichen "BETRAG+MEHRWERTSTEUER".

Zu 5:

Eine Aufeinanderfolge von Zeichen wird dadurch zu einem Zeichenkettenliteral, daß man am Anfang und am Ende der Zeichenfolge jeweils ein doppeltes Hochkomma setzt.

Beispiel:

Aufeinanderfolge von Zeichen: Dies ist der Beweis!
Zeichenkettenliteral: "Dies ist der Beweis!"

7. Kommentare und Kommentarzeilen

Kurzvortrag zur REM-Anweisung

In den Beispielprogrammen der nächsten Kapitel werden Sie Anweisungsnummern finden, hinter denen das Wort REM oder ein Apostroph und ein erläuternder Text zum Programm bzw. Programmablauf angefügt ist.

Hier einige Beispiele:

```
 10 REM Adressenprogramm

120 REM Unterprogramm Skonto

 10 ' Programm berechnet die MwSt

 65 ' Programmschleife

 90 PRINT B$," ":REM Ausgabe plus Leerzeichen

 10 C=1 'Setzt den Zaehler auf 1
```

Die REM-Anweisung und gleichwertig auch das Apostroph-Zeichen leiten Programm-Kommentare ein. Der Kommentar ist nur für den Programierer oder Anwender gedacht. Er soll einen Sachverhalt erläutern und das beim Programmieren Gedachte fixieren, um z.B. Programmerweiterungen ohne große Schwierigkeiten vornehmen zu können, oder um in einem umfangreichen Programm schnell einen bestimmten Programmteil ausfindig zu machen.
Die Verwendung des Apostrophs ist nicht bei allen Interpretern zulässig. Das BASIC des >> CPC 464 << gestattet die Benutzung des Apostrophs als eine Abkürzung für die früher notwendige REM-Anweisung.

REM steht für das englische Wort REMARK, übersetzt bedeutet dies "Bemerkung".

Regeln zur REM-Anweisung:

* Kommentarzeilen werden mit der REM-Anweisung eingeleitet.

* REM kann durch einen Apostroph als Kurzanweisung ersetzt werden.

* REM-Zeilen beeinflußen den Programmablauf nicht.

* REM-Zeilen dürfen an jeder Stelle des Programms eingefügt werden.

* REM-Anweisungen sind auch am Ende einer Anweisungskette getrennt mit einem Doppelpunkt von der letzten Ausführungsanweisung erlaubt.

* Wird die REM-Anweisung mit einem Apostroph gekennzeichnet, so darf der sonst geforderte Doppelpunkt in einer Anweisungskette fehlen.

* Die REM-Anweisung und der nachfolgende Text benötigen Speicherplatz.

* Kommentierung und Fixierung eines Programms bzw. einer Programmstruktur sind immer notwendig.

Kommentare werden Ihnen in den Programmbeispielen und auch in den "Projektprogrammen" immer wieder begegnen.

Dort heißt es dann z.B.:

```
10'Projekt: *** ZEICHEN DEFINIEREN ***
10'Projekt: *** KARTEI-PROGRAMM ***
100'Zweite Schleife
150'Ende der zweiten Schleife
```

Diese Kommentare dienen der Erläuterung und sollen für Sie eine Hilfe bei der Programmbetrachtung darstellen.

8. Die FOR NEXT-Schleife

Wiederholtes Abarbeiten von BASIC-Programmteilen

Die besondere Stärke eines Computers ist es, eine Fülle von Operationen immer wieder auszuführen, ohne hierbei zu "ermüden". Die wiederholte Ausführung ("Iteration") von aufeinanderfolgenden Anweisungen ist für das Programmieren von so großer Bedeutung, daß man in BASIC besondere Sprachelemente geschaffen hat, um dem Programmierer die Arbeit zu erleichtern.

Um die Zahl der Wiederholung von aufeinanderfolgenden Operationen kontrollieren zu können, müssen diese gezählt werden.

Beispiel:

```
10 C=1'Setzt den Zaehler auf 1
20 .....
30 .....
40 .....
50 C=C+1'Erhoeht den Zaehler um 1
60 IF C<=N THEN 20'Pruefung, ob Zaehlvor
gang beendet
70 .....
```

In diesem Beispiel werden die Operationen bereits kontrolliert.

Erläuterung:

* In Zeile 50 wird der Zähler C bei jedem Durchlauf der Anweisung um 1 hochgezählt.

* In Zeile 60 wird in einer WENN-DANN-Anweisung abgefragt, ob C kleiner oder gleich N ist.

* Ist dies der Fall, wird nach der Anweisung 20 verzweigt, andernfalls wird mit der auf die Anweisung 60 folgenden Anweisung 70 fortgefahren.

Die Anzahl der Wiederholungen ist daher einfach zu bestimmen!

Programmschleifen

Das wiederholte Durchlaufen von Anweisungen wird als "Programm-Schleife" bezeichnet. In diesem Sinne enthält auch das obige Beispiel bereits eine Programmschleife, die mit einer IF THEN-Anweisung programmiert wurde.

Die FOR NEXT-Anweisung

In BASIC verwirklicht man eine Programm-Schleife noch auf eine zweite Art: Mit der FOR NEXT-Anweisung!

Beispiel für die Struktur der FOR NEXT - Schleife:

```
10 FOR I=A TO N STEP 1
20 .....
30 .....
40 .....
50 NEXT I
```

Vergleichen Sie jetzt die beiden Arten von Schleifenbildungen!

Die Anweisung

```
FOR I=A TO N STEP 1
```

läßt sich in drei wesentliche Elemente aufgliedern:

```
Start:          FOR I=A
Ende:           TO N
Schrittweite:   STEP 1
```

Die Variable I wird häufig als Schleifenzähler, die Variable A als Anfangswert und die Variable N als Endwert bezeichnet.

Durch die Anweisung

```
NEXT I
```

geht die Kontrolle zur Anweisung

```
FOR I=A TO N STEP 1
```

zurück, ohne daß die Zeilennummer der FOR-Anweisung angeben werden muß.
Diese Anweisung, die oft auch Laufanweisung genannt wird, fordert das Programm auf, alle Anweisungen, die zwischen der FOR-Anweisung und der NEXT-Anweisung stehen, solange erneut zu durchlaufen, bis I den Wert N erreicht hat.

Intern wird die FOR NEXT-Anweisung so abgearbeitet, daß immer zuerst geprüft wird, ob der Schleifenzähler den Endwert erreicht hat.
Ist dies nicht der Fall, wird der Schleifenzähler um den Wert der Schrittweite, in dem Beispiel um 1, erhöht und zu der auf die FOR-Anweisung nachfolgenden Anweisung verzweigt.

Die FOR NEXT-Schleife ist somit nur eine versteckte WENN-DANN-Anweisung, die der zu Beginn des Kapitels gezeigten Lösung mit der IF THEN-Anweisung entspricht.

Intern ist der Programmablauf gleich, für den Programmierer ist die FOR NEXT-Schleife aber bequemer.
Die FOR NEXT-Schleife gehört zu den besonderen Stärken von BASIC. Diese Aussage werden wird im Zusammenhang mit der Tabellenarbeit näher erläutern!

Zusammenfassung:

* Die FOR NEXT-Anweisung dient der Schleifenprogrammierung.
* FOR kennzeichnet den Startpunkt der Schleife.
* TO N bestimmt das Ende der Schleifendurchläufe.
* STEP 1 bestimmt die Schrittweite in der gezählt wird.

* Die Anzahl der Schleifendurchläufe beträgt:

  Durchläufe=(Endwert+1-Anfangswert):Schrittweite

* Eine FOR NEXT-Anweisung kann in eine IF THEN-Anweisung aufgelöst werden.

Anwendungsbeispiele

Beispiele für die Anwendung der FOR NEXT-Anweisung:

Beispiel:

```
10 FOR J=5 TO 20 STEP 2
20 PRINT J
30 NEXT J
40 END
```

Ergebnis:

```
RUN
 5
 7
 9
 11
 13
 15
 17
 19
Ready
```

Die Laufanweisung in Zeile 10 besagt, daß alle folgenden Anweisungen bis zur NEXT-Anweisung so oft durchlaufen werden müssen, wie die Laufvariable J Werte annehmen kann. Diese Werte sind durch bestimmte Schranken festgelegt.

Für die Laufvariable J gilt somit:

Anfangswert: 5
Endwert: 20
Schrittweite: 2

Programmablauf:

* Im ersten Durchlauf wird die Laufvariable J auf 5 gesetzt. Beim nächsten Durchlauf wird sie von 5 um 2 auf 7 erhöht, dann auf 9, 11, 13, 15, 17 und 19.

* Den als Endwert angegebenen Wert 20 kann J nicht annehmen, da nach dem Erreichen von 19 eine Erhöhung um 2 bereits den Wert 21 ergibt, der außerhalb der Endwertschranke von 20 liegt.

Grundsätzlich kann gesagt werden, daß "alle" Anweisungen, die zwischen der FOR-Anweisung und der NEXT-Anweisung liegen, wiederholt durchlaufen werden.
In diesem Beispiel liegt nur eine einzige Anweisung, nämlich eine PRINT-Anweisung dazwischen, mit der der jeweilige Wert von J auf dem Bildschirm ausgegeben wird.

Hier nun das zweite Beispiel:

```
10 FOR L=15 TO 2 STEP -3
20 PRINT L
30 NEXT L
```

Programm-Ausführung:

```
RUN
 15
 12
 9
 6
 3
Ready
```

Programmablauf:

* Bei dieser Aufgabe wird die Laufvariable L heruntergezählt, und zwar in Schritten von -3.

* Den Endwert 2 kann die Laufvariable nicht annehmen, weil sie bei 3 - 3 den Wert 0 annehmen würde, der außerhalb der Endwertschranke liegt.

Im übrigen ist der Ablauf der gleiche wie im vorigen Beispiel!

Ein neues Beispiel mit einer veränderten Laufanweisung:

```
10 FOR M=-10 TO 15 STEP 5
20 PRINT M
30 NEXT M
40 END
```

Programm-Ausführung:

```
RUN
-10
-5
 0
 5
 10
 15
Ready
```

Programmablauf:

* Der Anfangswert beginnt im negativen Bereich (-10) und wird bis zu einem positiven Endwert in Schritten um 5 hochgezählt.

Sonst bleibt alles so wie Sie es bisher kennengerlernt haben.

Ein letztes Beispiel mit einer wiederum veränderten Laufanweisung:

```
10 FOR I=14 TO -12 STEP -3
20 PRINT I
30 NEXT I
40 END
```

Programm-Ausführung:

```
RUN
 14
 11
 8
 5
 2
-1
-4
-7
-10
Ready
```

Hier sind in der Laufanweisung der Endwert und die Schrittweite negativ. Nachzutragen bleibt die Regel:

* Ist die Schrittweite gleich 1, so braucht sie nicht angegeben zu werden.

Die Laufanweisung hat dann folgendes Aussehen:

```
FOR I=A TO N
```

Hier wird also bei jedem Schleifendurchlauf die Laufvariable, vom Anfangswert A ausgehend, bis zum Endwert N um jeweils 1 erhöht.

Die Anweisung

```
FOR I=A TO N
```

stellt eine allgemein gültige Form dar.

Der "allgemein gültigen Form" sollte nun noch die Angabe der Schrittweite hinzugefügt werden:

```
FOR I=A TO N STEP J
```

Erläuterung:

* Die Laufvariable I wird bei jedem Durchlauf verändert. In welcher Form dies geschieht, wird durch die Schrittweite angegeben.

* Die Anfangswert-Variable muß vor dem Erreichen der FOR-Anweisung mit einem entsprechenden Wert besetzt werden. Geschieht dies nicht, so hat sie den Wert 0.

* Die Endwert-Variable muß ebenfalls vorher besetzt werden. Geschieht dies nicht, so hat sie den Wert 0.

* Haben sowohl die Anfangswert-Variable als auch die Endwert-Variable den Wert Null, so wird die FOR NEXT-Anweisung nur ein einziges Mal durchlaufen.

Im übrigen müssen Sie immer darauf achten, daß die FOR-Anweisung in einer vernünftigen Form programmiert wird. Hat z.B. die Variable A den Wert 2, die Variable N den Wert 10 und die Variable für die Schrittweite den Wert -3, hat die FOR-Anweisung also folgendes Aussehen:

```
FOR I=2 TO 10 STEP -3
```

so ergibt die Anweisung keinen Sinn, denn beim normalen Ablauf kann man nicht von 2 nach 10 in Schritten von -3 gelangen.
Enthält ein Programm eine solche falsch programmierte Zeile und wird mit der RUN-Anweisung gestartet, so läuft das Programm nach dem Start auf einen Programmstop und wird sofort beendet.

Auf dem Bildschirm ist folgendes zu sehen:

```
RUN
Ready
```

Ein weiteres Beispiel, das sowohl für die Ausgabe auf dem Bildschirm, als auch für die Ausgabe auf einem Drucker von Bedeutung ist, wenn nach der PRINT-Anweisung das Nummernzeichen # und die Codezahl 8 angegeben wird.
Wenn z.B. auf dem Bildschirm Leerzeilen erzeugt werden sollen, so können Sie programmieren:

```
10 FOR I=1 TO 10
20 PRINT
30 NEXT I
40 END
```

Die "Liste" in der PRINT-Anweisung ist leer, es steht weder eine Variable noch ein Literal hinter PRINT. Das Programm gibt daher bei jedem Durchlauf auf dem Bildschirm eine Leerzeile aus.

Es kann aber auch ein Literal, das in der "Liste" der PRINT-Anweisung steht, auf dem Bildschirm ausgegeben werden. Aus Gründen der "Schönheit" soll eine Linie gezogen werden, die aus lauter Sternen besteht.

Programmierversuch:

```
10 FOR I=1 TO 40
20 PRINT "*"
30 NEXT I
40 END
```

Prüfen Sie dieses Programm zunächst daraufhin ab, ob es tatsächlich eine Sternenlinie in einer Linie ausgibt!

Hilfen für Ihre Programmanalyse:

Das Programm würde 40 Zeilen mit nur einem Stern (*) je Zeile ausgeben, aber nicht 40 Sterne nebeneinander! Um das Problem zu lösen, muß die PRINT-Anweisung folgendermaßen heißen:

```
20 PRINT "*";
```

Das Semikolon (der Strichpunkt) am Schluß der PRINT-Anweisung unterdrückt den normalen Zeilenvorschub.

Im nächsten Durchlauf wird der Stern auf der nächstfolgenden Stelle in derselben Zeile abgedruckt.

Das Ergebnis der FOR-NEXT-Schleife sieht dann auf dem Bildschirm so aus:

```
**********************************************
```

Auf diese Weise können bestimmte Bildschirmausgaben "eingerahmt" werden, was manchmal optisch vorteilhaft wirkt.

Zusammenfassung:

* Anfangs- und Endwert der Laufanweisung kann negative Werte annehmen.

* Die Schrittweite darf negative Schritte vorgeben.

* Ist die Schrittweite gleich "1", so kann diese Angabe entfallen.

* Die Angaben zu einer FOR NEXT-Schleife müssen mathematisch sinnvoll sein.

9. Die geschachtelte FOR NEXT-Schleife

Ausgangspunkt der nächsten Aufgabenstellung ist das korrigierte, letzte Beispielprogramm, das geringfügig erweitert wird.

```
10 N=N+1
20 FOR I=1 TO N
30 PRINT "*";
40 NEXT I
50 PRINT
60 IF N<20 GOTO 10
70 END
```

Überprüfen Sie auch hier erst wieder anhand des Programmlistings den Programmablauf!

Hilfen für Ihre Programmanalyse:

* Die Anweisungen Nr. 20, 30 und 40 bilden eine reine FOR NEXT-Schleife.

* Diese Schleife ist von einer "Hochzähl-Anweisung" (Nr. 10) und einer WENN DANN-Anweisung (Nr. 60) "eingeschlossen".

* Die Anweisungen Nr. 10 und 60 stellen aber ihrerseits auch eine Programmschleife dar!

Innere und äußere FOR NEXT-Schleife

Schreiben Sie nun anstelle der Anweisungen Nr. 10 und 60 eine FOR NEXT-Anweisung.

Programmbeispiel:

```
10 FOR N=1 TO 20
20 FOR I=1 TO N
30 PRINT "*";
40 NEXT I
50 PRINT
60 NEXT N
70 END
```

Sie haben nun eine innere und eine äußere FOR NEXT-Schleife programmiert!

Starten Sie das Programm mit RUN, so erhalten Sie folgendes Bild auf dem Monitor:

```
*
**
***
****
*****
******
*******
********
*********
**********
***********
************
*************
**************
***************
****************
*****************
******************
*******************
********************
```

Aus diesem Ergebnis können Sie eine Regel für die geschachtelten FOR NEXT-Schleifen ableiten:

9. Die geschachtelte FOR NEXT-Schleife

Regel:

* Die äußere Schleife wird gestartet und danach die innere vollständig abgearbeitet. Erst dann geht die Kontrolle auf die äußere Schleife über.

Programmbeispiele

Überlegen Sie bitte einmal was geschieht, wenn Sie die Programmzeile 10, wie angegeben, ändern:

```
FOR N = 20 TO 1 STEP -1
```

und Sie den Rest des Programms unverändert lassen?

Hilfen für Ihre Programmanalyse:

* Das obige Bild wird auf dem Kopf stehen, da die erste auszugebende Zeile N den Wert 20 besitzt, d.h. es stehen 20 Sterne (*) in der ersten Zeile.

* Der Schleifenzähler N wird in der äußeren Programmschleife in Schritten um -1 vermindert, also heruntergezählt.

* Dadurch verringert sich in der folgenden Zeile die Zahl der Sterne um 1.

Die von Ihnen abgeleitete, vorstehende Regel soll noch einmal anhand eines Beispiels näher erläutert werden.

```
10 I=1
20 FOR M=1 TO 5
30 I=2*I
40 FOR N=1 TO I
50 PRINT "*";
60 NEXT N
70 PRINT
80 NEXT M
90 END
```

Prüfen Sie die Anweisungen und beschreiben Sie zunächst die Ausgabe, die auf dem Monitor erscheinen könnte.

Wenn Sie das Programm mit RUN gestartet haben, so erscheint bei der Ausführung des Programms Zeile für Zeile folgendes Bild auf dem Monitor:

```
**
****
********
****************
********************************
```

Sie erkennen, daß die äußere Schleife

```
20 FOR M=1 TO 5
30 ...
80 NEXT M
```

genau fünfmal durchlaufen "FOR M=1 TO 5" wird. Sie können dies auch an den genau fünf Stern-Zeilen, die beim Durchlaufen des Programms ausgegeben wurden, überprüfen.

Zusammenfassung:

* Die Anzahl der Zeilen wird von der äußeren FOR NEXT-Schleife bestimmt.

* Die Anzahl der Sterne in jeder einzelnen Zeile wird durch die innere FOR NEXT-Schleife bestimmt.

* Die Laufvariable N nimmt dabei die Werte von 1 bis I an.

* Die Variable I ihrerseits wird innerhalb der äußeren FOR NEXT-Schleife verändert.

* Der jeweilige Wert von I wird durch den doppelten Wert von I ersetzt (I = 2 * I).

9. Die geschachtelte FOR NEXT-Schleife

Zur Verdeutlichung des Sachverhalts folgende Skizze:

```
M = 1,I =  2: **
M = 2,I =  4: ****
M = 3,I =  8: ********
M = 4,I = 16: ****************
M = 5,I = 32: ********************************
```

Starten Sie nun mit der RUN-Anweisung nachstehendes Programm, analysieren Sie den Programmablauf und auch das Ergebnis.

```
10 FOR K=5 TO 8
20 FOR I=1 TO 10
30 M=I*K
40 PRINT I " * " K " = " M
50 NEXT I
60 NEXT K
70 END
```

Hilfen zur Programmanalyse:

* Es ist wieder eine innere und eine äußere FOR NEXT-Schleife vorhanden. Zur Vereinfachung werden nur die beiden FOR NEXT-Anweisungen dargestellt:

```
10 FOR K=5 TO 8
20 FOR I=1 TO 10
...
...
50 NEXT I
60 NEXT K
```

* Die innere FOR NEXT-Schleife wird zehnmal, mit den Werten von 1 bis 10, durchlaufen. Die äußere FOR NEXT-Schleife wird viermal, mit den Werten von 5 bis 8, durchlaufen. Entscheidend sind die beiden Anweisungen Nr. 30 und 40:

```
30 M=I*K
40 PRINT I " * " K " = " M
```

* In der Anweisung kommen Hochkommata vor, die Zeichenkettenliterale einschließen:

```
PRINT I " * " K " = " M
        └─┬─┘   └─┬─┘
          1       2
```

* (1) Die beiden Hochkommata schließen ein Zeichenkettenliteral ein, das aus einem Leerzeichen, einem Stern und einem weiteren Leerzeichen besteht.

* (2) Diese beiden Hochkommata schließen ein weiteres Zeichenkettenliteral ein, das aus einem Leerzeichen, einem Gleichheitszeichen und einem weiteren Leerzeichen besteht.

* Die Buchstaben I, K und M bezeichnen Variable, die im Verlauf des Programms verwendet werden.

* I und K werden als Laufvariable benötigt. M dagegen ist eine Variable, in der das Produkt aus I mal K abgelegt wird.

* Beim ersten Programmdurchlauf hat K den Wert 5 und I den Wert 1, so daß M nach Ausführung der Anweisung Nr.

```
30 M=I*K
```

den WERT (1 * 5 =)5 besitzt.

* Die Anweisung Nr.40 gibt auf dem Bildschirm aus:

```
1  *  5  =  5
```

* Da die innere Schleife zuerst abgearbeitet wird, behält K seinen Wert 1 und I wird um 1 erhöht. Es ergibt sich folgender Rechengang:

M = 2 * 5, d. h.: M = 10

* Die Beschreibung der weiteren Schritte ergibt die Tabelle:

```
1  *  5  =  5
2  *  5  =  10
3  *  5  =  15
4  *  5  =  20

...
...

10  *  5  =  50
```

Das Ergebnis des Programmablaufs:

Das Programm liefert, wie Sie richtig analysiert haben, das Einmaleins mit 5, das Einmaleins mit 6 usw. und zuletzt das Einmaleins mit 8.

Regeln zu den geschachtelten FOR NEXT-Schleifen:

* Schleifen dürfen ineinandergeschachtelt sein, sie dürfen sich aber nicht kreuzen.

* Von außen her darf in eine Schleife nicht hineingesprungen werden.

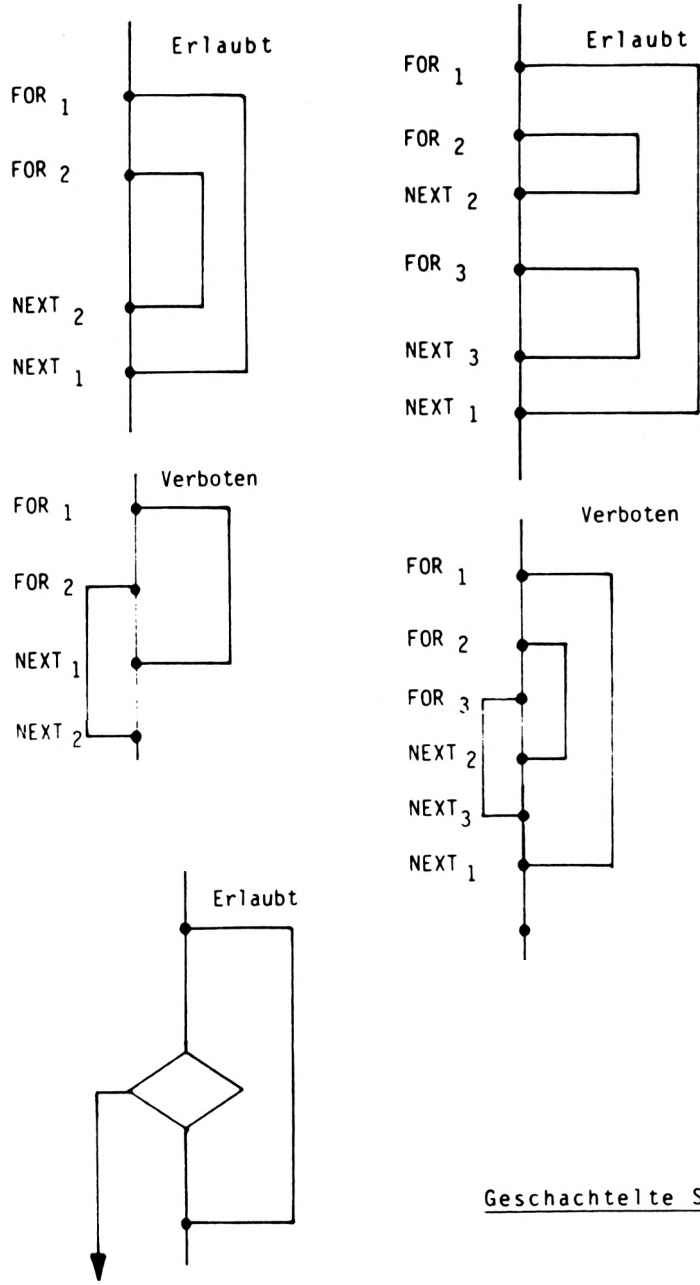
* Im Zusammenhang mit einer IF-Anweisung kann aus einer Schleife herausgesprungen werden.

* Die Laufvariablen sollten eindeutig vergeben werden.

* Die Anfangswerte, die Endwerte und die Schrittweitenwerte dürfen Zahlen, arithmetische Ausdrücke und Variablen sein.

Vergleichen Sie hierzu die graphische Darstellung von erlaubten und verbotenen, geschachtelten Schleifen.

Graphik zu erlaubten und verbotenen FOR NEXT-Schleifen



Geschachtelte Schleifen

9. Die geschachtelte FOR NEXT-Schleife

Die nachstehenden Programmbeispiele sollen Ihnen bei der praktischen Anwendung der FOR NEXT-Schleife helfen und Ihnen einen häufig vorkommenden Programmierfehler bei geschachtelten Schleifen aufzeigen.

Ein häufiger Programmierfehler

Beispiel mit Programmierfehler:

```
10 'Erste Schleife
20 FOR I=1 TO 10
30 IF I=5 THEN 70
40 PRINT I
50 NEXT I
60 'Ende der ersten Schleife
70 PRINT"Es geht!"
80 N=25
90 IF N=25 THEN PRINT"N = "N:GOTO 120
100 'Zweite Schleife
110 FOR K=1 TO 20
120 PRINT"Dies ist ein Einsprung"
130 PRINT"Wir sind in der zweiten Schleife"
140 NEXT K
150 'Ende der zweiten Schleife
160 END
```

Lesen Sie bitte das Programmlisting zuerst gründlich durch und versuchen Sie, sich Klarheit über den Programmablauf zu verschaffen.

Beachten Sie hierzu auch die genannten Regeln!

Die folgenden Zeilen bringen Ihnen die Darstellung so, wie sie auf dem Bildschirm ausgegeben wird.

```
RUN
 1
 2
 3
 4
Es geht!
N =  25
Dies ist ein Einsprung
Wir sind in der zweiten Schleife
Unexpected NEXT in 140
Ready
```

Sie haben bemerkt, daß die Fehlermeldung "Unexpected NEXT in 140" erscheint.

Die TRON- und TROFF-Anweisung

Es gibt aber eine Möglichkeit, den Ablauf des Programms mit einer Hilfsroutine zu protokollieren. Sie wird vom BASIC-Interpreter zur Verfügung gestellt, um so die Fehlerquelle zu bestimmen. Geben Sie dazu unmittelbar über die Tastatur des Schneider >> CPC 464 << das Kommando TRON ein. TR steht für das englische Wort TRACING und bedeutet soviel wie "Verfolgung ", hier die Verfolgung des Programmablaufs. Das englische ON steht für "ein" und bedeutet hier "einschalten". Starten Sie nun erneut mit RUN!

Ergebnis:

```
TRON
Ready
RUN
[10][20][30][40] 1
[50][30][40] 2
[50][30][40] 3
[50][30][40] 4
[50][30][70]Es geht!
[80][90]N =  25
[120]Dies ist ein Einsprung
[130]Wir sind in der zweiten Schleife
[140]
Unexpected NEXT in 140
Ready
```

9. Die geschachtelte FOR NEXT-Schleife

Die in eckige Klammern gesetzten Zahlen sind die Nummern der jeweils durchlaufenen Anweisungen, die Zeilennummern. Sie können damit den Programmablauf genau verfolgen.

Nach dem Starten des Programms mit RUN werden zwar in der ersten Zeile des Protokolls die Anweisungen 10, 20, 30 und 40 als Durchlaufen gekennzeichnet, aber von der zweiten Zeile an heißt es dann nur noch 50, 30 und 40; eigentlich müßte es doch wohl 50, 20, 30 und 40 heißen?

Sie müssen bei Ihrer Programmanalyse davon ausgehen, daß der Interpreter die durchlaufenen Anweisungen stets neu in eine maschinennahe Sprache übersetzt (interpretiert). Dabei werden die FOR NEXT-Anweisungen in einzelne Programmschritte aufgelöst. Das TRACING-Programm gibt in diesem Zusammenhang stets die Nummer der NEXT-Anweisung und die Nummer der auf die FOR-Anweisung folgenden Zeilennummer aus. Hinter den in Klammern gesetzten Zahlen steht je Zeile das Ergebnis der Programmausführung.

Danach durchläuft das Programm die Schleife nicht zehnmal, wie in der FOR-Anweisung gefordert, sondern auf Grund der WENN DANN-Bedingung in der Anweisung Nr. 30 nur viermal. Erreicht der Schleifenzähler I den Wert 5, so verzweigt das Programm aus der Schleife hinaus zur Anweisung mit der Nr. 70.

Es sollte mit diesem Programm zunächst gezeigt werden, daß mit einer IF-Anweisung aus einer Schleife hinausgesprungen werden kann. Im zweiten Teil der Programmanalyse wird nun die Fehlermeldung betrachtet.

Die Variable N wird in Zeile 80 auf 25 gesetzt. In der Anweisung 90 ist die WENN DANN-Bedingung somit erfüllt. Auf dem Bildschirm wird

N = 25

ausgegeben. Das Programm verzweigt in die zweite Schleife hinein, und zwar zu der Anweisung mit der Nr. 120.

Auf dem Bildschirm werden zwei Zeilen ausgegeben:

```
Dies ist ein Einsprung
Wir sind in der zweiten Schleife
```

und es wird die Anweisung 140 protokolliert.

Anschließend erscheint als Fehlermeldung:

```
Unexpected NEXT in 140
```

auf dem Bildschirm, weil der Einsprung in eine Schleife nicht gestattet ist. Die Bedeutung der Fehlermeldung ist: NEXT wurde ohne dazugehöriges FOR gefunden. Vergleichen Sie hierzu das Kapitel "Fehlermeldungen und Fehlercodes".

Hieran erkennen Sie eindeutig, daß ein Hineinspringen in eine Schleife vom Computer mit einer Fehlermeldung beantwortet und das Programm abgebrochen wird. Das Abbrechen des Programms wird auf dem Bildschirm durch "Ready" angezeigt.

Dieses "Ready" heißt hier aber lediglich:

> Der Interpreter ist in den Kommandozustand bzw. in den Direktmodus zurückgekehrt, von dem aus er wieder angesprochen werden kann. Das Programm ist - wie erwähnt - "ausgestiegen".

Zusammenfassung zum TRACE-Modus:

* Der TRACE-Modus dokumentiert den Programmablauf.
* TRON schaltet den TRACE-Modus ein.
* Im TRACE-Modus werden die durchlaufenen Anweisungsnummern in eckigen Klammern ausgegeben und das Ergebnis nachgestellt.
* TROFF schaltet den TRACE-Modus wieder aus.
* Auch eine NEW-Anweisung schaltet den TRACE-Modus aus.

Übungen zu der FOR NEXT-Anweisung

Aufgabe 1:

Welche Zahlen werden durch folgende Programmschleifen auf dem Bildschirm ausgegeben?

1.1
```
10 'Aufgabe 1.1
20 FOR I = 1 TO 9 STEP 2
30 PRINT I
40 NEXT I
50 END
```

1.2
```
10 'Aufgabe 1.2
20 FOR G = -35 TO -19 STEP 3
30 PRINT G
40 NEXT G
50 END
```

1.3
```
10 'Aufgabe 1.3
20 FOR J = -13 TO -21 STEP -3
30 PRINT J
40 NEXT J
50 END
```

Aufgabe 2:

Schreiben Sie ein Programm, das Ihnen das Einmaleins der 7, 8 und 9 auflistet!

Aufgabe 3:

Welche Variablennamen sind gültige Namen für eine Zeichenkettenvariable?

a) AA2 b) A$A c) H7$ d) 8PQ$ e) 3$ f) B$

Aufgabe 4:

Welche der folgenden Anweisungen sind gültige BASIC-Anweisungen?

a) A = 87
b) B$ = 9854
c) M$ = "BASIC"
d) L17 = 23

Lösungen

Um Ihnen die Arbeit zu erleichtern, sind die Programme noch einmal abgedruckt:

Zur Aufgabe 1.1:

```
10 'Aufgabe 1.1
20 FOR I = 1 TO 9 STEP 2
30 PRINT I
40 NEXT I
50 END
```

```
RUN
 1
 3
 5
 7
 9
Ready
```

Zur Aufgabe 1.2:

```
10 'Aufgabe 1.2
20 FOR G = -35 TO -19 STEP 3
30 PRINT G
40 NEXT G
50 END

RUN
-35
-32
-29
-26
-23
-20
Ready
```

Zur Aufgabe 1.3:

```
10 'Aufgabe 1.3
20 FOR J = -13 TO -21 STEP -3
30 PRINT J
40 NEXT J
50 END


RUN
-13
-16
-19
Ready
```

Zur Aufgabe 2:

```
10 'Aufgabe 2
20 FOR I = 7 TO 9
30 FOR J = 1 TO 10
40 M = J * I
50 PRINT J;" * ";I;" = ";M
60 NEXT J
70 PRINT
80 NEXT I
90 END
```

```
RUN
 1   *   7   =   7
 2   *   7   =   14
 3   *   7   =   21
 4   *   7   =   28
 5   *   7   =   35
 6   *   7   =   42
 7   *   7   =   49
 8   *   7   =   56
 9   *   7   =   63
 10   *   7   =   70

 1   *   8   =   8
 2   *   8   =   16
 3   *   8   =   24
 4   *   8   =   32
 5   *   8   =   40
 6   *   8   =   48
 7   *   8   =   56
 8   *   8   =   64
 9   *   8   =   72
 10   *   8   =   80

 1   *   9   =   9
 2   *   9   =   18
 3   *   9   =   27
 4   *   9   =   36
 5   *   9   =   45
 6   *   9   =   54
 7   *   9   =   63
 8   *   9   =   72
 9   *   9   =   81
 10   *   9   =   90

Ready
```

Zur Aufgabe 3:

Gültig sind: c) H7$ und f) B$

Der Name der Zeichenkettenvariablen muß mit einem Buchstaben beginnen und mit einem Dollarzeichen ($) enden!

Zur Aufgabe 4:

Gültige BASIC-Anweisungen sind: a), c) und d)!

Die Anweisung b) ist deshalb ungültig, weil links von dem Gleichheitszeichen eine Zeichenkettenvariable, rechts vom Gleichheitszeichen aber eine ganzzahlige Konstante steht. Der Computer kann solch eine Operation nicht ausführen. Er meldet einen Fehler "Type mismatch", in freier Übersetzung etwa: "Nicht zueinander passende Variablenarten".

Daten als Konstanten und als Variablen

Die Kapitel, die Sie bisher bearbeitet haben, zeigten die eigentliche Stärke des Computers auf. Daten wurden hier immer in unterschiedlicher Form be- bzw. verarbeitet.
Es wird daher verständlich, daß auch sonst in Verbindung mit dem Computer meist gleichzeitig von Datenverarbeitung gesprochen wird.

Die Verarbeitung von Daten wird dabei immer in mindestens drei Stufen verwirklicht:

1. Einlesen von Daten

   Daten werden vom Benutzer eingegeben --> variable Daten
   Daten werden vom Programmierer vorgegeben --> konstante Daten

2. Verarbeitung von Daten

   Vergleichen und Berechnen
   Vermischen und Tauschen
   Selektieren und Sortieren

3. Ausgeben und/oder Speichern von Daten

Alle drei Stufen der Datenverarbeitung sind Ihnen mittlerweile schon vertraut. In diesem Kapitel werden Sie mit der zweiten Art, dem Programm Daten zu übergeben, bekannt gemacht.
Benutzt haben Sie bislang die INPUT-Anweisung, um Größen unterschiedlichster Art und Typs einzulesen, die zudem noch von Fall zu Fall verschieden waren.

Die READ- und DATA-Anweisung

Sind aber Konstanten, dieses sind Werte von gleichbleibender Größe und Typ, in einem BASIC-Programm für den Ablauf notwendig, so ist es sehr unbequem, diese Daten immer wieder aufs Neue während des Programmablaufs eingeben zu müssen.
Hierzu wird die Kombination: READ- mit DATA-Anweisung benötigt.

Mit dieser Kombination können konstante Werte, die bei jedem Programmablauf stets in gleicher Weise benötigt werden, an Variablen übergeben werden. Die Daten können dann bereits im Programm in sogenannte DATA-Zeilen in genau der Reihenfolge abgelegt werden, in der sie während des Programmablaufs abgerufen werden.

Beispiel:

```
10 DATA 5,7,9,11
20 DATA 13
30 READ P1,P2,P3
40 READ P4,P5
50 PRINT P1;P2;P3;P4;P5

RUN
 5  7  9  11  13
Ready
```

Hilfen für Ihre Programmanalyse:

* In Zeile 10 und 20 befindet sich je eine DATA-Anweisung, in denen numerische Konstanten, in einer Konstantenliste durch Kommata getrennt, bereitgehalten werden.

* Die erste READ-Anweisung finden Sie in Zeile 30. Hier werden zunächst drei Elemente aus der ersten DATA-Anweisung eingelesen.
  Eingelesen wird durch die erste READ-Anweisung:

  P1 = 5; P2 = 7 und P3 = 9

* In Zeile 40 finden Sie die zweite READ-Anweisung, die wiederum zuerst auf die DATA-Anweisung in Zeile 10 zugreift und das letzte Element einliest, dann aber auch auf das in der DATA-Zeile 20 stehende Element zugreift.
  An die Variablen wird hierbei der entsprechende Wert übergeben:

  P4 = 11 und P5 = 13

In DATA-Anweisungen können aber nicht nur numerische, sondern auch nichtnumerische Konstanten, also auch Zeichenketten, im Programm fest vorgegeben werden.
Diese können mit der READ-Anweisung gelesen und an Variable während des Programmablaufs übergeben werden. DATA-Anweisungen werden ohne zugehörige READ-Anweisung vom laufenden Programm übergangen, da sie selbst keine Funktion besitzen. Die im Programm vorhandenen DATA-Zeilen dürfen an jeder für die Programmübersicht sinnvollen Stelle eingefügt werden, d. h. die Programmausführung wird hierdurch nicht beeinträchtigt.
In einer DATA-Zeile können Konstanten nacheinander angegeben werden, indem sie durch Kommata getrennt werden. Sollen Zeichenketten auch Kommata oder Doppelpunkte enthalten oder aber Leerzeichen zu Beginn bzw. am Ende aufweisen, so müssen die angegebenen Zeichenketten als Zeichkettenliterale geschrieben werden.

In DATA-Zeilen dürfen dagegen keine numerischen Ausdrücke stehen, da hier keine Berechnungen durchgeführt werden können.

Beispiel:

```
10 DATA >>  CPC,464,<<
20 READ A$,Z,B$
30 PRINT A$;Z,B$

RUN
>>  CPC 464  <<
Ready
```

Übung zur READ- und DATA-Anweisung

Übung:

Prüfen Sie den Programmverlauf, wenn Sie die Zeile

```
40 GOTO 20
```

in das letzte Programmbeispiel 40 einfügen!

Hilfen für Ihre Programmanalyse:

* In Zeile 10 sind numerische und nichtnumerische Konstanten vorgegeben in einer DATA-Anweisung.

* Die READ-Anweisung liest zunächst die Zeichenkette ">> CPC" in die Variable A$, dann den numerischen Wert "464" in Z und als letzte Konstante wieder eine Zeichenkette in B$ ein.

* Die PRINT-Anweisung gibt dann die Inhalte der Variablen auf dem Bildschirm aus.

Die Fehlermeldung, die nach dem Einfügen der Zeile 40 ausgegeben wurde, haben Sie richtig interpretiert:

* Konstanten können nicht ohne besondere Anweisung zweimal gelesen werden.

Die RESTORE-Anweisung

Die zugehörige READ-Anweisung liest diese numerischen Werte, Zeichenketten oder Zeichenkettenliterale linear. Dies bedeutet, daß sie nacheinander eingelesen werden und sich innerhalb des Programms "verbrauchen", wenn nicht eine zusätzliche RESTORE-Anweisung dieses unterbindet, so daß mehrmals in einem Programmablauf auf dieselben DATA-Anweisungen zugegriffen werden kann.

Übung:

Fügen Sie eine Zeile 35 ein, sie lautet:

```
35 RESTORE
```

und belassen Sie das übrige Programm. Prüfen Sie, ob nun wieder die gleiche Fehlermeldung erscheint.

Hinweis:

> Es kommt während der Testphase von Programmen immer wieder zu den unterschiedlichsten Fehlermeldungen. Sie sollten in einem solchen Fall das Kapitel "Fehlermeldungen und Fehlercodes" zu Rate ziehen!

Zu beachten ist, daß die READ-Anweisung ohne eine RESTORE-Anweisung immer zuerst auf die niedrigste Zeilennummer und dann auf die nächst höhere Zeilennummer mit einer DATA-Anweisung zugreift, wenn die vorgegebenen Konstanten in der ersten DATA-Zeile schon durch eine READ-Anweisung vollständig gelesen wurden.

Im nächsten Programmbeispiel sind die häufigsten Anwendungsfälle von READ-,DATA- und RESTORE zusammengefaßt. Schauen Sie sich bitte erst das Programmlisting an und prüfen Sie dann den Programmverlauf!

Programmbeispiel

```
10 MODE 1
20 DATA Januar,Februar,Maerz,April,Mai,Juni,Juli
30 DATA August,September,Oktober,November,Dezember
40 CLS
50 PRINT"Bitte Zahlen eingeben fuer:"
60 INPUT "Tag  : ",T
70 INPUT "Monat: ",M
80 IF T=24 AND M=12 THEN RESTORE 180:GOTO 190
90 IF M<1 OR M>12 THEN 40
100 FOR I=1 TO M
110 READ MONAT$
120 NEXT I
130 PRINT
140 PRINT T"."M".  =  "T". "MONAT$
150 RESTORE
160 FOR I=1 TO 2000:NEXT
170 GOTO 40
180 DATA "  frohe Weihnachten !    "
190 READ TEXT$:PRINT:PRINT TEXT$
200 GOTO 150
```

Anhand des Programms lassen sich leicht allgemeingültige Regeln für die hier besprochenen Anweisungen ableiten.

Regeln zur READ-,DATA und RESTORE-Anweisung

* READ liest genausoviele konstante Werte aus DATA-Anweisungen, wie Variablen in der Liste aufgeführt sind und ordnet die Werte, entsprechend ihrer Reihenfolge, den Variablen zu.

* Wird bei der READ-Anweisung eine numerische Variable angegeben, so muß die entsprechende Konstante in der Datazeile ein numerischer Wert sein.

* Wird bei der READ-Anweisung eine Zeichenkettenvariable angegeben, so kann jede Konstante einer DATA-Anweisung gelesen werden. Numerische Konstanten können somit als Zeichenketten eingelesen werden.

Hinweis:

Diese eigentlichen numerischen Werte liegen dann als Zeichenkette vor, können aber durch die VAL-Anweisung wieder in einen numerischen Wert umgewandelt werden.

* DATA-Anweisungen dürfen an jeder sinnvollen Stelle des Programms stehen.

* DATA-Zeilen besitzen keinen direkten Einfluß auf den Programmablauf.

* Sind Doppelpunkte, Kommata oder Leerzeichen am Anfang oder Ende der Zeichenkette gewünscht, so muß diese als Zeichenkettenliteral geschrieben werden.

* RESTORE setzt den sogenannten DATA-Zeiger auf den Anfang zurück, das heißt, die nächste READ-Anweisung greift wieder auf die erste DATA-Zeile zu.

* Wird der RESTORE-Anweisung eine Zeilennummer nachgestellt, so wird der DATA-Zeiger auf eben diese Zeile zurückgesetzt, so daß die nächste READ-Anweisung auf diese Zeile zugreift.

Die PRINT-Anweisung haben Sie bereits zur Ausgabe von Ergebnissen auf dem Bildschirm eingesetzt. Diese Anweisung gibt Texte, Graphiksymbole und Zahlen auf dem Bildschirm, einem Bildschirmfenster oder auf dem Drucker aus. Bei der Eingabe von Programmen oder bei den Anweisungen im Kommandomodus kann das "PRINT" durch die Kurzform "?" gleichwertig ersetzt werden.
Das Fragezeichen ist hierbei die kürzere und schneller eingebbare Anweisungsform und wird von BASIC automatisch in ein "PRINT" umgewandelt.
Sicher werden Sie schon festgestellt haben, daß manche Bildschirmausgabe unübersichtlich war. Die Forderung ist also eine aufbereitete bzw. formatierte Ausgabe auf dem Bildschirm, die auch Druckaufbereitung genannt wird.
Der BASIC-Interpreter stellt für eine formatierte Bildschirmausgabe spezielle PRINT-Anweisungen und weitere zusätzliche BASIC-Anweisungen zur Verfügung.

Es sind zwei Arten der Druckaufbereitung zu unterscheiden:

1) Die nichtformatierte Ausgabe im BASIC-Standardformat

2) Die formatierte Ausgabe im PRINT USING-Format

Das Standardformat

Die Druckaufbereitung im Standardformat ist die am häufigsten benutzte Ausgabeform. Sie benutzt innerhalb der PRINT-Anweisung die Zeichen: Komma (,) und Semikolon (;). Diese Zeichen werden hier vom BASIC-Interpreter als Steuerzeichen ausgewertet. Die Wirkung dieser Steuerzeichen sollen Sie in diesem Kapitel selbst erproben.

Geben Sie nicht direkt das Programm ein, da der >> CPC 464 << noch über frei belegbare Funktionstasten verfügt. Dieses Ausstattungsmerkmal sollten Sie häufig nutzen, da es eine enorme Arbeitserleichterung darstellt.

Die Funktionstasten des >> CPC 464 <<

Kurzinformation: Funktionstastenbelegung

* Funktionstasten sind alle Tasten des "Zehnerblocks", rechts neben der Haupttastatur.

* Auch die kleine ENTER-Taste und der danebenliegende Punkt können als Funktionstasten belegt werden.

* Jede dieser Tasten besitzt eine eigene Tastennummer bzw. einen eigenen Tastencode, unter der die einzelne Taste angesprochen werden kann.

* Die Tasten lassen sich mit der KEY-Anweisung mit jeglichen Zeichen belegen. Diese Zeichen können auch BASIC-Anweisungen sein.

* Die Funktionstasten lassen sich jederzeit wieder mit der KEY-Anweisung umbelegen.

Die KEY-Anweisung

Die Anwendung der KEY-Anweisung lernen Sie direkt an diesem Beispiel kennen.

Sie geben nun im Direktmodus

```
KEY 129,"PRINT "

KEY 130,"INPUT "
```

ein und drücken nach jeder der beiden Eingabezeilen die ENTER-Taste. Die Belegung der zwei Funktionstasten ist damit abgeschlossen. Überprüfen Sie dieses durch Drücken der "1" und der "2" im Zehnerblock. Es werden PRINT und danach INPUT auf dem Monitor ausgegeben.

Die AUTO-Anweisung

Eine weitere Arbeitserleichterung bei der Eingabe von Programmen stellt die BASIC-Anweisung AUTO dar. Sie schreibt automatisch Zeilennummern, so daß Sie diese nicht eingeben müssen.

Beispiele für die Anwendung der AUTO-Anweisung:

AUTO	erzeugt die Zeilennummern 10,20,30,...
AUTO 230	erzeugt die Zeilennummern 230,240,250,...
AUTO ,5	erzeugt die Zeilennummern 10,15,20,25,...
AUTO 183,4	erzeugt die Zeilennummern 183,187,191,...

Soll der AUTO-Modus verlassen werden, so drücken Sie einmal die ESC-Taste.

Eingabeanleitung für das BASIC-Programm:

1. AUTO im Direktmodus eingeben.
2. Taste "2" im Zehnerblock anstelle von INPUT drücken
3. Taste "1" im Zehnerblock anstelle von PRINT drücken
4. Abschließend ESC-Taste drücken.

Beispielprogramm:

```
10 INPUT A
20 PRINT A
30 INPUT B
40 PRINT A B A
50 PRINT A;B;A
60 PRINT A,B,A
70 PRINT "..............................."
80 END
```

Ist das Programm vollständig eingegeben, dann starten Sie es mit der RUN-Anweisung und geben Sie die untenstehenden Beispielwerte bei den Abfragen ein.

Echo-Check

Ihre Eingabe wird unmittelbar hinter dem Aufforderungszeichen auf dem Bildschirm protokolliert. Diese Art der Protokoll-Ausgabe heißt "Echo-Check", da sie wie ein "Echo" wirkt und Ihnen die sofortige Prüfung auf richtige Eingabe gestattet.

Die Anweisung 20 gibt eine Bildschirmzeile tiefer den eingegebenen und bereits auf dem Bildschirm protokollierten Wert von A im BASIC-Standardformat aus.
Nach Ihrer Eingabe von B wird diese wiederum unmittelbar hinter dem Fragezeichen auf dem Bildschirm protokolliert.

```
RUN
? 789
 789
? 456
```

Dann werden in den Folgezeilen durch mehrere unterschiedliche PRINT-Anweisungen die Variablen A und B nebeneinander ausgegeben.

Die Wirkung der Steuerzeichen bei der PRINT-Anweisung

Überprüfen Sie nun die Wirkweise der Steuerzeichen!

Beachten Sie, daß

* in der PRINT-Anweisung 40 A,B und A nebeneinander stehen und durch ein Leerzeichen getrennt sind.

* in der PRINT-Anweisung 50 die Variablen A, B und A durch Semikola (;) getrennt sind.

* in der PRINT-Anweisung 60 die Variable A, B und A durch Kommata getrennt stehen.

* die gepunktete Linie Ihnen als Zählhilfe bei der Bestimmung der Positionen der ausgegebenen Werte dient.

Die Bildschirmausgabe hat folgendes Aussehen:

```
RUN
? 789
 789
? 456
 789   456   789
 789   456   789
 789             456             789
.................................
Ready
```

Sie erkennen, daß

* die Ergebnisse der Anweisungen in Zeile 40 und 50 identisch,

* die in Zeile 60 ausgegebenen Werte an den Anfang bestimmter Abschnitte geschrieben sind.

Ableitung allgemeingültiger Regeln

1. Ein Leerzeichen oder ein Semikolon zwischen den Variablen oder Zeichenketten einer Ausgabe-Liste bewirkt eine Ausgabe unmittelbar an das bereits ausgegebene, vorherige Zeichen. Diese Regel gilt in gleicher Weise für den Bildschirm wie für den Drucker (Nadeldrucker, Schönschreibdrucker).

2. Stehen Variablen oder Zeichenketten einer Ausgabeliste durch ein Komma getrennt, so wird der Wert oder die Zeichenkette nach dem Komma am Anfang des nächsten Bildschirmabschnitts ausgegeben. Diese Bildschirmabschnitte werden Bildschirmzonen genannt. Die Abstände zwischen den Zonen sind auf 14 Zeichen bzw. 14 Spalten voreingestellt.

3. Wird eine numerische Variable ausgegeben, so folgt auf die letzte Ziffer stets ein Leerzeichen.

4. Positive Werte erhalten anstelle des positiven Vorzeichens "+" ein Leerzeichen (blank). Negative Werte werden mit ihrem Vorzeichen ausgegeben.

5. Ist am Ende einer Ausgabeliste kein Steuerzeichen (Semikolon oder Komma) angegeben, so wird bei der nächsten PRINT-Anweisung auf der nächsten Zeile die Ausgabe fortgesetzt.

6. Wird ein Steuerzeichen als letztes Zeichen einer Ausgabeliste angefügt, so erfolgt die Datenausgabe der nächsten PRINT-Anweisung in der gleichen Art, als ob diese hier vorhandene Datenliste Bestandteil der vorherigen wäre, d. h. es wird kein Zeilenvorschub erzeugt.

Überprüfen Sie die Regeln mit dem nächsten Beispiel!

```
10 INPUT A
20 PRINT A
30 INPUT B
35 Z$="ZEICHENKETTE"
40 PRINT A B Z$
50 PRINT A;B;Z$
60 PRINT A,B,Z$
70 PRINT "..............................
........."
80 END

RUN
? 456789
 456789
? -456789
 456789 -456789 ZEICHENKETTE
 456789 -456789 ZEICHENKETTE
 456789       -456789       ZEICHENKETTE
.....................................
Ready
```

Die Bildschirmzonen und die ZONE-Anweisung

Sie haben den Begriff Bildschirmabschnitt bzw. Bildschirm-Zone kennengelernt. Der >> CPC 464 << erlaubt es, im Gegensatz zu anderen Computertypen, diese Zonen mit der ZONE-Anweisung zu variieren.

Kurzinformation zur ZONE-Anweisung:

* ZONE(X) setzt die vertikalen Bildschirmzonen, die die PRINT-Anweisung mit dem Komma als Formatierungsanweisung nutzt.

* Die Angabe von X bestimmt die Größe des Zwischenraums von zwei Ansprungpositionen und muß eine Zahl zwischen 1 und 255 sein.

Beispiel:

```
10 MODE 1
20 PRINT "A","B","C"
30 ZONE 10
40 PRINT "ABC","DEF","GHI"
50 END
```

Die LOCATE-Anweisung

Eine weitere Möglichkeit, die Datenausgabe auf dem Bildschirm spalten- und zeilenweise zu steuern, bietet die LOCATE-Anweisung.

Kurzinformation zur LOCATE-Anweisung:

* Die LOCATE-Anweisung setzt den Textcursor an eine neue Bildschirmfensterposition.

Beispiel:

```
10 MODE 2
20 X=5:Y=6
30 LOCATE X,Y
40 PRINT "Dieser Text beginnt in Spalte"X"und steht in Zeile"Y
50 END
```

* Die erste Angabe bestimmt die Spaltenposition.

* Die zweite Angabe bestimmt die Zeilenposition.

* Die Angaben sollten ganzzahlige, numerische Ausdrücke sein, die in Abhängigkeit von der Größe des Bildschirmfensters zu wählen sind.

* Die linke obere Ecke des Monitorfensters besitzt hierbei die Koordinaten Spalte=1 und Zeile=1.

* Beachten Sie, daß keine negativen Werte angegeben werden.

* Die LOCATE-Anweisung erlaubt die Angabe des Datenstroms.

Die formatierte Ausgabe

Diese Art der Ausgabe ist überall dort gefordert, wo zwingend auf einem exakt vordefinierten Platz oder Stelle eine Ausgabe erfolgen muß. Diese Genauigkeit ist meist in kaufmännischen oder statistischen Tabellen die Voraussetzung für die Stimmigkeit der Ergebnisse. Hier müssen z.B. mathematisch vorgegebene Dezimalstellen, die sich am Dezimalpunkt orientieren, entsprechend ihres Stellenwertes formatiert ausgegeben werden. Dieses hat so zu geschehen, daß auch die Einheit des Zahlenmaterials mit berücksichtigt wird.
Sie werden diese Notwendigkeit der Ausgabe direkt an dem untenstehenden Beispiel ablesen können.

Beispiel:	unformatiert	formatiert
	1286.2655	1286.2655
	22335.5	22335.5000
	8.633	8.6330

Die unformatierte Ausgabe der Zahlenkolonne ist unübersichtlich und, z.B. für eine Addition, kaum geeignet.

Die vorstehende, formatierte Ausgabe weist neben der einheitlichen Dezimalpunktposition noch eine gleiche Anzahl von Nachkommastellen auf. Die fehlenden Nachkommastellen wurden hier durch eine entsprechende Anzahl Nullen ausgeglichen. Die Dimensionen der Zahlen sind in dieser Form über den optischen Eindruck abschätzbar und entsprechen außerdem der bei einer Addition üblichen Form.

Die PRINT USING-Anweisung

Eine formatierte Ausgabe zu programmieren, ist für Sie einfach zu verwirklichen, da hierzu die PRINT USING-Anweisung zur Verfügung steht.

Die Form der Ausgabe wird hierbei durch "Steuerzeichen" bzw. "Platzhalter" angegeben.

Programmbeispiele

Beispiel für die PRINT USING-Anweisung:

```
10 PRINT USING "##.#####";69.4
```

In Zeile 10 erkennen Sie die Platzhalter, die in Hochkommata eingeschlossenen Doppelkreuze (#), die auch Numerus-Zeichen genannt werden.
Jedes Numerus-Zeichen entspricht hierbei einer Ziffer. Der angegebene Punkt kennzeichnet die Position des Dezimalpunktes bei der Ausgabe und ist beliebig setzbar. Der Dezimalpunkt benötigt keinen Platzhalter.

Überprüfen Sie nun die Ausgabeformen anhand des Programms und leiten Sie aus den Ergebnissen und dem Programmablauf allgemeingültige Regeln ab!

```
10 MODE 1
20 PRINT " Nr Zahl      Format  Ausgabe"
30 PRINT STRING$(32,154):PRINT
40 ANZAHL1=5
50 ZAHL=123.425
60 FOR ANZAHL2=0 TO 5
70 FORMAT$=STRING$(ANZAHL1,"#")+"."+STRING$(ANZAHL2,"#")
80 PRINT STR$(ANZAHL2+1);".";ZAHL;" ";FORMAT$;"  ";
90 PRINT USING FORMAT$;ZAHL
100 ANZAHL1=ANZAHL1-1
110 IF ANZAHL1=2 THEN PRINT
120 NEXT ANZAHL2
130 PRINT STRING$(32,154):PRINT
```

Ergebnis:

Nr	Zahl	Format	Ausgabe
1.	123.425	#####.	123.
2.	123.425	####.#	123.4
3.	123.425	###.##	123.43
4.	123.425	##.###	%123.425
5.	123.425	#.####	%123.4250
6.	123.425	.#####	%123.42500

Hilfen für Ihre Programmanalyse:

* ANZAHL1 bestimmt die Anzahl der Platzhalter vor dem Dezimalpunkt und wird in den Anweisungen der Zeilen 40 und 100 bestimmt.

* ANZAHL2 wird durch die FOR NEXT-Schleife bestimmt und gibt die Anzahl der Platzhalter nach dem Dezimalpunkt an.

* In Zeile 70 wird die eigentliche FORMAT$-Angabe zusammengesetzt aus: Platzhalter vor dem Dezimalpunkt plus Dezimalpunkt plus Platzhalter nach dem Dezimalpunkt.

* Die Anweisung in Zeile 80 gibt die laufende Nummer, die auszugebende Zahl und das Format aus.

* Die Anweisung in Zeile 90 gibt die Zahl in ihrem angegebenen Format aus.

Sie werden sicher den Zusammenhang zwischen Formatangabe und dazugehöriger Zahlenausgabe erkannt haben. Damit Sie Ihre gefundenen Regeln überprüfen können, sind die ableitbaren Regeln nachstehend aufgeführt.

Regeln zur PRINT USING-ANweisung

* Ist die Zahl stellenmäßig kleiner als die der Platzhalter, so wird die Zahl rechtsbündig mit führenden Leerzeichen ausgegeben.

* Besitzt die angegebene Zahl rechts vom Dezimalpunkt weniger Stellen als Platzhalter rechts vom Dezimalpunkt vorhanden sind, so werden diese Stellen mit Nullen aufgefüllt.

* Sind mehr Platzhalter als Stellen der Zahl links vom Dezimalpunkt, so werden diese fehlenden Stellen mit Leerzeichen aufgefüllt.

* Weist eine Zahl nur Nachkommastellen auf und sind links vom Dezimalpunkt Platzhalter angegeben, so wird bei der Ausgabe eine Null vor dem Dezimalpunkt vorgesetzt.

* Ist die Anzahl der Platzhalter vor dem Dezimalpunkt kleiner als die Anzahl der Ziffern der auszugebenden Zahl vor dem Dezimalpunkt, so wird die gesamte Zahl geschrieben und zusätzlich ein Prozentzeichen (%) als Warnung für das nicht eingehaltene Format vorgestellt.

* Wird die Anzahl der Platzhalter rechts vom Dezimalpunkt von der auszugebenden Zahl überschritten, so wird diese um die entsprechenden Stellen gerundet ausgegeben, d. h. ab- bzw. aufgerundet.

Die Formatierungszeichen der PRINT USING-Anweisung

Formatangaben

Symbol	Formatausführung
#	Gibt eine Ziffer einer Zahl aus. Das Ausgabeformat entspricht der Anzahl der Symbole. Wird diese kleiner als die Zahl, so richtet sich die Ausgabe nach der Zahl und setzt ein Prozentzeichen "%" als Warnung vor die Zahl.
.	Bestimmt die Position des Dezimalpunktes bei einer Zahl.
+	Gibt das Vorzeichen Plus "+" bei positiven Zahlen aus, bei negativen das entsprechende, negative Vorzeichen.
**	Füllt bei einer kleineren Zahl die führenden Leerstellen mit Sternen "*" auf und entspricht in der Formatausführung dem Symbol "#".
$$	Es wird ein Dollarsymbol vor der ersten Ziffer ausgegeben und entspricht in der Formatausführung dem Symbol "#".
**$	Füllt die führenden Leerstellen mit Sternen auf und gibt ein Dollarsymbol vor der ersten Ziffer aus.
↑↑↑↑	Gibt eine Zahl im Exponentialformat aus.
,	Fügt alle drei Stellen links vom Dezimalpunkt ein Komma ein.
!	Gibt bei einer Zeichenkette nur das erste Zeichen aus.
&	Gibt ein variables Format an.
<space>	Gibt entsprechend viele Leerzeichen aus.

Die Druckaufbereitung mit Tabulatoren

Die PRINT TAB-Anweisung

Kenntnisse über die Anwendung der Anweisung PRINT TAB(n) sind bei der Arbeit mit Tabellen, beim Drucken bzw. Ausgeben auf dem Bildschirm von Ergebnissen und Bedieneranweisungen erforderlich.

Beispiel:

```
10 'Programmbeispiel fuer TAB
20 '
30 'Tabulatoren setzen
40 '------------------------
50 MODE 2
60 FOR I=1 TO 48 STEP 6
70 PRINT TAB(I) "Hallo!"
80 NEXT
90 END
```

```
RUN
Hallo!
      Hallo!
            Hallo!
                  Hallo!
                        Hallo!
                              Hallo!
                                    Hallo!
                                          Hallo!
Ready
```

Bevor Sie die kurze Programmbeschreibung lesen, sollten Sie die Variable I betrachten und die Wirkung von PRINT TAB(I) erkannt haben.

Hilfen zur Programmanalyse:

* Sie finden die Anweisung PRINT TAB(I) in der Anweisungszeile Nr. 70.

* Sie steht innerhalb einer FOR NEXT-Schleife.

* Die Schrittweite, in der die Laufvariable I hochgezählt wird, ist sechs.

* Nach jedem Schleifendurchlauf gilt I=I+6.

* Die Ausgabe mit der PRINT TAB(I) -Anweisung wird daher um sechs Spalten nach rechts versetzt vorgenommen.

* Der neue Zeilenanfang erklärt sich aus dem fehlenden bzw. nicht nachgestellten Komma bzw. Semikolon in der Zeile 70.

* Das Bildschirmformat wird in Zeile 50 auf 80 Spalten gesetzt.

TAB wird stets in Verbindung mit einem Klammerpaar verwendet. Innerhalb der Klammern steht entweder eine Variable, oder eine Konstante, oder auch ein arithmetischer Ausdruck. Die in dem Klammerpaar stehende Größe bestimmt die Spaltenposition.

Beispiele für die TAB Anwendung

```
TAB(I)        (I = Variable)
TAB(10)       (10 = Konstante)
TAB(I+N*2)    (I+N*2 = arithmetischer Ausdruck)
```

Programmbeispiel für die PRINT TAB-Anweisung

In dem folgenden Programmbeispiel werden Mengen und Preise eingegeben und die jeweiligen Beträge berechnet. Insgesamt sollen neun Wertepaare nacheinander eingegeben werden. Auf dem Bildschirm erscheint also von 1 bis 9 nacheinander:

```
1. Menge:?
   Preis:?
```

Das Fragezeichen als Anforderungszeichen fordert Sie zur Dateneingabe auf. In Abhängigkeit von der FOR NEXT-Schleife geschieht dies insgesamt 9 mal. Nachdem die letzte Dateneingabe erfolgt ist, wird das Ergebnis so in Tabellenform ausgegeben, wie dies nachstehend beispielhaft gezeigt wird.

Beispielprogramm:

```
10 'Zweites Programmbeispiel fuer TAB
20 MODE 2
30 'Arbeiten mit Tabulatoren
40 '--------------------------------
50 'Eingabe von Mengen und Preisen
60 FOR I=1 TO 9
70 PRINT
80 PRINT I".";:INPUT "Menge: ";M(I)
90 INPUT "    Preis: ";P(I)
100 B(I)=M(I)*P(I)
110 NEXT I
120 '--------------------------------
130 'Ausgabe der Daten in Tabellenform
140 '--------------------------------
150 PRINT:PRINT:PRINT
160 PRINT TAB(20)"Menge  x"TAB(30) "Preis = "TAB(39) "Betrag"
170 PRINT TAB(20)"------------------------"
180 FOR I=1 TO 9
190 PRINT TAB(20) M(I) TAB(30) P(I) TAB(40) B(I)
200 NEXT I
```

Ergebnis:

```
!!! IMenge  x  Preis =  Betrag !!!
    ------------------------
      2        3        6
      2        5        10
      1        5        5
      3        8        24
      4        6        24
      1        9        9
      4        6        24
      12       5        60
      11       14       154
```

Sie sollten nun anhand der ausgegebenen Tabelle die Wirkung der TAB-Anweisung analysieren und dann die untenstehenden, allgemeinen Erläuterungen zur PRINT TAB-Anweisung zu Rate ziehen.

Hinweis:

Die Daten für Menge, Preis und Betrag werden in diesem Programmbeispiel in drei "eindimensionalen Tabellen" eingelesen. Dieses Thema wird ausführlich auf den nächsten Seiten abgehandelt. Beschränken Sie sich hier nur auf die Wirkung der TAB-Anweisung.

Die Wirkweise der PRINT TAB(I)-Anweisung

Allgemein läßt sich die Wirkungsweise von PRINT TAB(I) so beschreiben:

Durch TAB(I) wird in einer PRINT-Anweisung der Cursor bis zu derjenigen Position nach rechts bewegt, die dem in Klammern stehenden Wert von I entspricht.

Für den Fall, daß der Cursor bereits rechts von der Position I steht, wird der Zeilenrest mit Leerzeichen aufgefüllt und eine neue Zeile begonnen. Sie wird dann bis zur Position I mit Leerzeichen angefüllt.

Im übrigen läßt sich TAB(I) annähernd mit der Tabulation auf der Schreibmaschine vergleichen. Auch dort dienen Tabulatoren dazu, das Schreiben von Tabellen zu erleichtern.

Zusammenfassung:

* TAB(I) kann nur in Verbindung mit einer PRINT-Anweisung genutzt werden.

* Die PRINT TAB(I)-Anweisung positioniert den Cursor auf die I-te Spalte einer Zeile.

12. Das Arbeiten mit Tabellen

Tabellenarten

Die Programmierung von Tabellen, in denen ganze Folgen von Namen oder Folgen von Zahlen abgelegt und von BASIC bearbeitet werden können, setzt Kenntnisse von der Funktion und dem Aufbau von Tabellen voraus. Tabellen dienen immer der Entflechtung von komplexen Zusammenhängen, die in ihnen übersichtlich und schnell überschaubar dargestellt werden können.

Es sind bei der Betrachtung verschiedene Arten von Tabellen zu unterscheiden:

* Eindimensionale Tabellen

  und

* Mehrdimensionale Tabellen

Die Unterscheidung in der Dimension zeigt die Richtungen an, in der sich die Tabelle ausdehnt, d.h. die Tabellen-Struktur wird hiermit angegeben. Die Dimension einer Tabelle bestimmt indirekt auch ihre spätere An- und Verwendung.

Die eindimensionale Tabelle:

Eine eindimensionale Tabelle besteht immer aus einer einzigen Zeile, weist aber eine oder beliebig viele Spalten aus.

Beispiel:

Es wird die Spannung an einem bestimmten Meßpunkt abgegriffen und in einer eindimensionalen Tabelle notiert.

Meßwertreihe: 10.2, 10.5, 10.8, 10.9, 11.0, 10.5

Die mehrdimensionale Tabelle:

Eine mehrdimensionale Tabelle besteht aus mehreren Spalten und besitzt gleichzeitig auch zwei oder mehrere Zeilen.

Beispiel:

Diese zweidimensionale Tabelle gibt eine laufende Nummer, den Artikel, den eingelesenen Einzelpreis, die Menge und den Betrag wieder. Die Summe der Beträge und der Durchschnittsbetrag werden in der Tabelle erfaßt.

Nr.	Artikel	Einzelpreis [DM]	Menge	Betrag [DM]
1.	Butter	2.49	1.00	2.49
2.	Zucker	1.49	3.00	4.47
3.	Brot	3.70	1.00	3.70
4.	Schokolade	1.39	5.00	6.95
5.	Marmelade	1.98	4.00	7.92
6.	Milch	1.09	2.00	2.18
7.	Joghurt	0.89	6.00	5.34
		Gesamt:	22.00	33.05

Durchschnittsbetrag ..... = 4.72 DM

Das Beispiel zeigt eine mehrdimensionale Tabelle mit der Dimension zwei, d.h. sie dehnt sich in zwei Richtungen aus. Die waagrechte (horizontale) Richtung wird bei einer zweidimensionalen Tabelle meist mit X, die senkrechte (vertikale) Richtung meist mit Y bezeichnet und entsprechend beschriftet.
Vergleichen Sie hierzu das Projektprogramm: "Zeichen-Definition".

Eindimensionale Tabellen

Vektoren

Eine eindimensionale Tabelle wird in der Datenverarbeitung auch oftmals Vektor genannt. Ein derartiger Vektor kann wie in dem obigen Beispiel waagerecht oder senkrecht angeordnet sein. Er besteht aus einem oder mehreren Elementen, die unter einem Oberbegriff zusammengefaßt werden können.

12. Das Arbeiten mit Tabellen

Dem Oberbegriff kann unter BASIC eine Variable zugeordnet werden. In dem zweiten Beispiel kann der Vektor mit Anschrift bezeichnet werden.

Beispiel:

Anschrift: Titel, Name, Vorname, Straße, Ort, Branche

Sowohl die Elemente als auch der Vektorname sind hierbei vom Typ Zeichenkette. Es können also nicht nur Zahlen, sondern auch Zeichenketten in einer eindimensionalen Tabelle erfaßt werden. Für die Kennzeichnung der Variablen gilt weiterhin die Regel, daß die Variable das Dollarzeichen ($) aufweisen muß, d.h. es kann

N$=Anschrift

gesetzt werden.

Indizes

Der Vektor ist in dem Beispiel waagerecht angeordnet und besteht aus insgesamt 6 Elementen:

Vektorelemente:
1. Element ist der "Titel"
2. Element ist der "Name"
3. Element ist der "Vorname"
4. Element ist die "Straße"
5. Element ist der "Ort"
6. Element ist die "Branche"

Die Variable N$ kennzeichnet den Vektornamen, so daß

N$(1),N$(2),N$(3),N$(4),N$(5),N$(6)

die einzelnen Vektorelemente bezeichnen. Die in Klammern gesetzte Zahl wird als Index bezeichnet und bestimmt die Position eines Elements im Vektor.
In der mathematischen Schreibweise ist die Indexangabe tiefgestellt. In der Computer-Schreibweise müssen die Indizes (Plural von Index) in Klammern hinter den Variablennamen angegeben werden, entsprechend dem oben im Beispiel angegebenen Format.

Die DEFSTR-Anweisung

Sie erkennen, daß die Variable N sechsmal auftritt und zusätzlich ein Dollarzeichen angefügt werden muß, da es sich bei den Vektorelementen um Zeichenkettenvariablen handelt. Der BASIC-Interpreter des >> CPC 464 << erlaubt hier eine einfachere Vorgehensweise. Die Zeichenkettenvariablen müssen dann nicht mit dem Dollarzeichen gekennzeichnet werden, wenn Sie am Anfang des Programms vor dem ersten Aufruf im selben Programm als Zeichenkettenvariable definiert werden. Das kann mit der

DEFSTR

-Anweisung erfolgen.

Sie ersparen sich Schreibarbeit und vermeiden durch die DEFSTR-Anweisung Fehler, da der Computer automatisch die Variable als eine Zeichenkettenvariable verarbeitet.
Wird N als Zeichenkettenvariable mit der DEFSTR-Anweisung am Programmanfang definiert, so darf für die Vektorelemente:

Titel, Name, Vorname, Straße, Ort, Branche

N(1), N(2), N(3), N(4), N(5), N(6)

ohne Dollarzeichen geschrieben werden.

Es besteht nun folgender Zusammenhang:

In N(1) steht der Titel
In N(2) steht der Name
In N(3) steht der Vorname
In N(4) steht die Straße
In N(5) steht der Ort
In N(6) steht die Branche

Unter BASIC kann nun auf ein bestimmtes Element des Anschrift-Vektors N, z.B. auf den "Vornamen", durch einfachen Aufruf der indizierten Variable N(3) zugegriffen werden.

Zusammenfassung:

* Man unterscheidet eindimensionale und mehrdimensionale Tabellen.

* Die eindimensionale Tabelle besitzt immer nur eine Zeile, kann aber mehrere Spalten aufweisen.

* Die mehrdimensionale Tabelle besteht aus mindestens zwei Zeilen und mehreren Spalten.

* Die eindimensionale Tabelle wird in der Datentechnik Vektor genannt.

* Ein Vektor besteht aus mehreren Vektorelementen.

* Die Vektorelemente können Zeichenketten oder numerische Werte sein.

* Der Index einer Variablen kennzeichnet die Position im Vektor und muß in runden Klammern angegeben werden.

* Auf die indizierte Variable kann unter BASIC direkt zugegriffen werden.

* Die DEFSTR-Anweisung erklärt einen Anfangsbuchstaben zum Typ Zeichenkette.

Die Dimensionierung von Feldern

Die DIM-Anweisung

Der BASIC-Interpreter kann ohne besondere Anweisung auf elf Elemente eines Vektors zugreifen, dessen Indizes im Bereich von 0 bis 10 liegen müssen. Werden bei einem Programmablauf Indizes größer als zehn benötigt, so muß das dem BASIC-Interpreter mitgeteilt werden.
Die Anweisung hierfür lautet DIM, wobei DIM für DIMension steht.

Die BASIC-Anweisung DIM dimensioniert und stellt den erforderlichen Platz im Arbeitsspeicher des >> CPC 464 << für die Elemente des Vektors zur Verfügung. Hinter DIM wird die zu indizierende Variable gesetzt und in Klammern der höchste, benötigte Index angegeben.

Die Anweisung

```
DIM N(13)
```

teilt dem Computer mit, daß eine indizierte Variable mit dem Namen N existieren soll. Die größtmögliche Anzahl von Elementen liegt um eins größer als der angegebene höchste Index, da der BASIC-Interpreter mit der Indizierung beim Index Null beginnt:

N(0), N(1), N(2), N(3), N(4), N(5), N(6), N(7), N(8),
N(9), N(10), N(11), N(12), N(13)

So ergeben sich bei der Dimensionierungsanweisung DIM N(13) insgesamt 14 Tabellenelemente!

Zusammenfassung:

* In einer DIM-Anweisung steht zuerst die Variable, die als Index-Variable deklariert wird.

* Der Index, der das höchste Element bestimmt, wird in runden Klammern nach der Variablen gesetzt.

* Der BASIC-Interpreter stellt immer zusätzlich das Element Null zur Verfügung.

* Die Anzahl der maximal möglichen Elemente ist daher immer um eins größer als der angegebene höchste Index.

* Elemente bis zum Index 10 werden vom BASIC-Interpreter automatisch zur Verfügung gestellt.

Sie haben sich nun die Voraussetzungen für das folgende kleine Programmbeispiel erarbeitet.

Programmbeispiele mit Hilfen zur Analyse:

```
10 DIM N(6)
20 FOR I=1 TO 6
30 N(I)=I*2
40 NEXT I
50 FOR I=1 TO 6
60 PRINT N(I)
70 NEXT I
80 PRINT :PRINT
90 FOR I=6 TO 1 STEP -1
100 PRINT N(I)
110 NEXT I
120 END
```

Bevor die einzelnen Programmzeilen erläutert werden, starten Sie das Programm bitte mit der RUN-Anweisung.

Sie erhalten dann das Ergebnis:

```
RUN
 2
 4
 6
 8
 10
 12


 12
 10
 8
 6
 4
 2
Ready
```

Anhand des Ergebnisses sollten Sie nun in der Lage sein, das Programm zu analysieren.

Hilfen zur Programmanalyse:

* In der ersten Programmzeile finden Sie die Dimensionierungsanweisung DIM N(6).

* Diese Dimensionierung ist nicht notwendig, da der Interpreter ohnehin bei einer indizierten Variablen 11 Elemente (von 0 bis 10) zur Verfügung stellt.

* Es werden aber durch die Dimensionierung N(6) nicht 11, sondern nur 7 Elemente (von 0 bis 6) zur Verfügung gestellt.

* Durch eine Dimensionierung kleiner 10 wird im Arbeitsspeicher des >> CPC 464 << Platz gespart, den man anderweitig verwenden kann!

* Das Programm enthält drei FOR NEXT-Schleifen mit folgendem Aufbau:

1. Schleife:
```
20 FOR I = 1 TO 6
30 ...
40 NEXT I
```

2. Schleife
```
50 FOR I = 1 TO 6
60 ...
70 NEXT I
```

und die

3. Schleife:
```
90 FOR I = 6 TO 1 STEP -1
100 ...
110 NEXT I
```

Mit anderen Worten: In den ersten beiden Schleifen wird die Laufvariable I von 1 auf 6 hochgezählt. In der 3. Schleife wird sie dagegen von 6 auf 1 heruntergezählt.

* In den ersten beiden Schleifen ist der Anfangswert 1 und der Endwert 6, in der dritten Schleife ist der Anfangswert 6 und der Endwert 1.

* In keiner der Schleifen gibt es also einen Anfangs- oder Endwert "NULL".

Ergebnis:

Es werden für den Programmablauf nur 6 Elemente benötigt. Daher ist es sinnvoll, eine andere Programmiertechnik anzuwenden, um eine indizierte Variable mit nur sechs Elementen vorzusehen. Betrachten Sie hierzu bitte das folgende Programm:

```
10 DIM A(5)
20 FOR I=0 TO 5
30 A(I)=(I+1)*2
40 NEXT I
50 FOR I=0 TO 5
60 PRINT A(I)
70 NEXT I
80 PRINT:PRINT
90 FOR I=5 TO 0 STEP -1
100 PRINT A(I)
110 NEXT I
120 END
```

In diesem Programm hat die Laufvariable einen Anfangswert von 0 und einen Endwert von 5, insgesamt also 6 Elemente.
Es wird zusätzlich in der dritten Programmzeile

```
30 A(I)=(I+1)*2
```

dafür gesorgt, daß keine Multiplikation mit Null stattfindet, indem der Multiplikand I um 1 erhöht wird (I+1).

In beiden Programmen wird deutlich, daß

* in der ersten Schleife sechs Elemente besetzt werden.

* in der zweiten Schleife die Inhalte der Elemente auf dem Bildschirm ausgegeben werden, wobei die Indizes in aufsteigender Folge angesprochen werden und

* in der dritten Schleife die Inhalte der Elemente A(I) bzw. N(I) mit absteigender Index-Folge ausgegeben werden.

Dieses geschieht mit der Laufanweisung

```
FOR I=5 TO 0 STEP -1 !
```

im zweiten Programm bzw. durch die entsprechende Anweisung

```
90 FOR I=6 TO 1 STEP -1
```

im ersten Programm.

Sie sollten an dieser Stelle nochmals den Programmablaufplan zu dem Beispiel durcharbeiten, der auch dem vorstehenden Programm entspricht, wobei Sie die kleinen Unterschiede soeben erarbeitet haben.

Mehrdimensionale Tabellen

Im Alltag sind mehrdimensionale, in der Regel zweidimensionale Tabellen am häufigsten anzutreffen. Tabellen also, die sich in der waagrechten (horizontalen) und in der senkrechten (vertikalen) Richtung hin ausdehnen. Diese zweidimensionalen Tabellen besitzen demnach eine Matrixstruktur, bestehend aus Zeilen und Spalten.

Sie haben bereits eindimensionale Tabellen kennengelernt, deren Elementvariablen indiziert wurden, d. h. die Variable war mit einem Index versehen.

Beispiel:

N(1),N(2),N(3),N(4)

Indizierung von mehrdimensionalen Tabellenelementen

Ähnlich der eindimensionalen kann auch eine mehrdimensionale Tabelle ganz einfach mit indizierten Variablen programmiert werden. Die Elementvariablen weisen in diesem Fall entsprechend viele Indizes auf.

Beispiel für ein zweidimensionales Tabellenelement:

A(3,4)

Feldelemente eines zweidimensionalen Feldes

Hiermit wird in einer zweidimensionalen Tabelle eindeutig ein Element bzw. Feldelement angesprochen. Dieses Element findet sich in der dritten Spalte und der vierten Zeile der Tabelle.

A(1,1) A(2,1) A(3,1) A(4,1)

A(1,2) A(2,2) A(3,2) A(4,2)

A(1,3) A(2,3) A(3,3) A(4,3)

A(1,4) A(2,4) A(3,4) A(4,4)

A(1,5) A(2,5) A(3,5) A(4,5)

Die vorstehende Tabelle enthält 4 mal 5 Feldelemente. Eine DIM-Anweisung ist in diesem Beispiel nicht unbedingt notwendig, da der BASIC-Interpreter automatisch Elemente bis zum Index 10 auch bei mehrdimensionalen, indizierten Variablen zur Verfügung stellt, sobald eines dieser Elemente aufgerufen wird.
Es ist aber ratsam, dennoch eine DIM-Anweisung zu programmieren, da sonst unnötig Speicherplatz verschenkt wird.

Dimensionierung des zweidimensionalen Feldes

Sollte nun eine DIM-Anweisung gewünscht sein, so muß sie für das Beispiel wie folgt aussehen:

DIM A(4,5)

Hierdurch werden (4 plus 1) mal (5 plus 1)= 30 Elemente für die Variable A als indizierte Variable reserviert. Die Gesamtzahl der möglichen Elemente liegt bei jedem Index um eins größer, da der BASIC-Interpreter mit der Indizierung beim Index Null beginnt. An dem nächsten Beispielprogramm können Sie Ihre bisher gewonnenen Kenntnisse Überprüfen, indem Sie den Aufbau und die Zählweise des Programms analysieren und mit dem Programmergebnis vergleichen.

```
10 MODE 1
20 DIM A(3,5)
30 FOR I=1 TO 3
40 FOR J=1 TO 5
50 A(I,J)=I*J
60 PRINT"I ="I;"  J ="J;";   A("I","J") =";A(I,J)
70 NEXT J
80 NEXT I
```

Sie erhalten:

```
RUN
I =  1   J =  1 ;   A(  1 ,  1 ) =  1
I =  1   J =  2 ;   A(  1 ,  2 ) =  2
I =  1   J =  3 ;   A(  1 ,  3 ) =  3
I =  1   J =  4 ;   A(  1 ,  4 ) =  4
I =  1   J =  5 ;   A(  1 ,  5 ) =  5
I =  2   J =  1 ;   A(  2 ,  1 ) =  2
I =  2   J =  2 ;   A(  2 ,  2 ) =  4
I =  2   J =  3 ;   A(  2 ,  3 ) =  6
I =  2   J =  4 ;   A(  2 ,  4 ) =  8
I =  2   J =  5 ;   A(  2 ,  5 ) =  10
I =  3   J =  1 ;   A(  3 ,  1 ) =  3
I =  3   J =  2 ;   A(  3 ,  2 ) =  6
I =  3   J =  3 ;   A(  3 ,  3 ) =  9
I =  3   J =  4 ;   A(  3 ,  4 ) =  12
I =  3   J =  5 ;   A(  3 ,  5 ) =  15
Ready
```

Hilfen für Ihre Programmanalyse:

* In Zeile 20 wird ein Feld mit 24 Elementen dimensioniert.

* Das Programm enthält zwei FOR NEXT-Schleifen, die geschachtelt sind.

* Die äußere FOR NEXT-Schleife wird 3 mal durchlaufen.

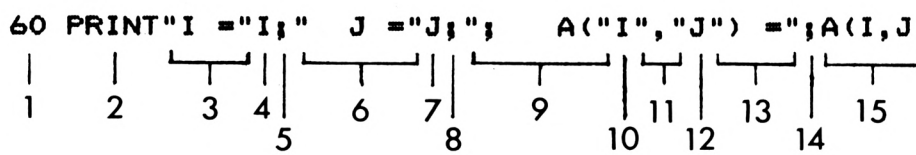
* Die innere FOR NEXT-Schleife mit der Laufvariablen J wird durch die äußere Schleife insgesamt dreimal aufgerufen und bei jedem Aufruf fünfmal durchlaufen.

* Die innere Schleife wird demnach insgesamt fünfzehn mal durchlaufen, so daß 15 Feldelementen das Produkt der Laufvariablen zugewiesen wird.

* In Zeile 60 werden die Laufvariablen, ihre Werte, die Bezeichnung des aufgerufenen Feldelements und sein Inhalt ausgegeben.

Die PRINT-Anweisung mit Zeichenkettenliteralen

Einige zusätzliche Erläuterungen zur Ausgabezeile 60:



1) Laufende Zeilennummer = Anweisungsnummer

2) Geschütztes BASIC-Wort PRINT = Drucke (hier: gib auf dem Bildschirm aus!

3) Zeichenkette zur Ausgabe der Zeichenfolge I = auf dem Bildschirm; die Zeichenkette muß in Anführungszeichen gesetzt werden: "I = ".

4) Laufvariable I aus der äußeren Schleife, deren jeweiliger Inhalt auf dem Bildschirm ausgegeben werden soll.

5) Trennzeichen; wir sprechen darüber im Kapitel "Druckaufbereitung".

6) Zeichenkette; vgl. unter 3).

7) Laufvariable J aus der inneren Schleife, deren jeweiliger Inhalt auf dem Bildschirm ausgegeben werden soll.

8) Trennzeichen; vgl. unter 5).

9) Zeichenkette; vgl. unter 3).

10) Hier wird der Inhalt der Laufvariablen I auf dem Bildschirm ausgegeben.

11) Zeichenkette; vgl. 3).

12) Hier wird der Inhalt der Laufvariablen J auf dem Bildschirm ausgegeben.

13) Zeichenkette; vgl. unter 3).

14) Trennzeichen; vgl. unter 5).

15) Indizierte Variable A, hinter der in Klammern die beiden Indizes I und J angegeben sind.

13. Zeichenkettenverarbeitung in Tabellen

Einführung

In dem Einführungskapitel "Arbeiten mit Tabellen" wurden zwei eindimensionale Tabellen vorgestellt. Die erste erfaßte die abgegriffene Spannung an einem bestimmten Meßpunkt:

Meßwertreihe: 10.2, 10.5, 10.8, 10.9, 11.0, 10.5

Die zweite eindimensionale Tabelle enthielt die Tabellenelemente:

Titel, Name, Vorname, Straße, Ort, Branche

Der Oberbegriff bzw. Vektornamen lautete "Anschrift" und für die Vektorelemente wurde die indizierte Variable N$(I) eingeführt:

Vektorelemente:

N$(1)	<--	Titel	N$(4)	<--	Straße
N$(2)	<--	Namen	N$(5)	<--	Ort
N$(3)	<--	Vornamen	N$(6)	<--	Branche

Durch Aufruf einer der indizierten Variablen kann nun auf ein bestimmtes Element unter BASIC zugegriffen werden.

Wenn Sie die beiden hier angesprochenen, eindimensionalen Tabellen miteinander vergleichen, werden Sie feststellen, daß sie sich in einem Punkt voneinander unterscheiden.

Prüfen Sie den Unterschied!

Hilfen:

* Im Beispiel der Meßwertreihe handelt es sich um numerische Werte.

* Die zweite vorgestellte Tabelle enthält als Vektorelemente ausschließlich Zeichenketten.

Im Einführungskapitel "Arbeiten mit Tabellen" wurde als Ergebnis abgeleitet, daß die Vektorelemente numerische Werte oder aber Zeichenketten sein können.

Nun handelt es sich aber bei einer normalen Anschrift nicht nur um Zeichenketten. Die wichtigen Daten, Postleitzahl und Telefonnummer, sind numerische Werte, so daß beide Arten nebeneinanderstehend verarbeitet werden müssen.
Weiterhin ist es nützlich, auch Sonderzeichen für bestimmte Kennungen innerhalb der numerischen Teile der Anschrift verwenden zu können, wie z. B. bei der Telefonnummernangabe eine Klammer oder aber einen Binde- oder Schrägstrich zur Trennung von Vorwahl- und Teilnehmernummer.
Das kann nur erreicht werden, indem auch die numerischen Teile als Zeichenketten behandelt werden. BASIC ermöglicht die Verarbeitung von numerischen Werten als Zeichenketten, wenn sie schon beim Einlesen mit der LINE INPUT-Anweisung als solche behandelt werden.

Die LINE INPUT-Anweisung

Kurzinformation zur LINE INPUT-Anweisung:

* Sie ermöglicht, Daten zeilenweise in das laufende Programm im Dialog einzugeben und an eine Variable zu übergeben.

* Es werden alle eingegebenen Zeichen bis zum nächsten ENTER der <Variable$> übergeben.

* Die Eingabe darf maximal 255 Zeichen umfassen und muß durch Drücken der ENTER- Taste abgeschlossen werden.

* Wird das Semikolon ";" direkt nach der INPUT-Anweisung gesetzt, so wird der automatische Zeilenvorschub von der LINE INPUT-Anweisung unterdrückt.

* Das zwischen <Abfrage> und <Variable> befindliche Semikolon ";" kann durch ein Komma "," ersetzt werden. Es hat hier nicht die Bedeutung, wie Sie sie von der INPUT-Anweisung kennen.

* Das Fragezeichen wird nicht ausgegeben und muß, falls gewünscht, an die <Abfrage> mit angehängt werden.

* Der Vorteil dieser Anweisung gegenüber der INPUT-Anweisung ist der, daß der unkundige Programmbenutzer mehrere Daten als Einheit eingeben kann, ohne hierbei auf die Trennung achten zu müssen.

* Es müssen bei vorangestellten Leerzeichen bzw. enthaltenen Kommata keine Anführungszeichen gesetzt werden.

Die STR$-Anweisung

Eine weitere Möglichkeit bietet die STR$-Anweisung, die Zahlen in Zeichenketten umwandelt.

Kurzinformation zur STR$-Anweisung:

* STR$(<Zahl>) verwandelt den numerischen Wert einer <Zahl> in eine Zeichenkette

* Die jeweiligen Vorzeichen "+" und "-" gehören zur Zeichenkette.

* Das positive Vorzeichen wird als "blank" (Leerzeichen) ausgegeben.

Beispiel:

```
10 MODE 1
20 A$=STR$(PI)
30 B$=STR$(-PI)
40 PRINT PI;-PI
50 PRINT A$;B$
60 PRINT LEN(A$),LEN(B$)

RUN
 3.14159265 -3.14159265
 3.14159265-3.14159265
 11            11
Ready
```

Anwendungsfall: "Telefonnummernverzeichnis"

Es soll nun ein BASIC-Programm erstellt werden, das den Namen, den Vornamen und die Telefonnummer mit der Vorwahlnummer, getrennt mit einem Schrägstrich, in eine zweidimensionale Tabelle einliest. Das Programm kann für Ihre Telefonnummern-Verwaltung dienen.
Die Zahl der Teilnehmer soll hierbei auf 15 Stück begrenzt bleiben. Der Wert ist beispielhaft und änderbar!

Vorüberlegungen zum Programmaufbau

1. Die Elemente des Telefon-Verzeichnisses können alle als Zeichenkette aufgefaßt werden, so daß die indizierte Variable das Dollarzeichen als Kennung einer Stringvariable aufweisen muß. Eine Vereinfachung der Schreibweise bietet in diesem Fall die DEFSTR-Anweisung. Sie erlaubt die Definition aller Variablennamen mit dem angegebenen Anfangsbuchstaben als Typ Zeichenkette, so daß das Dollarzeichen entfallen kann. Hieraus leitet sich die Programmanweisung DEFSTR N ab, mit der N als Typ Zeichenkette bestimmt wird.

2. Soll nur eine Telefonnummer mit den entsprechenden anderen Daten eingelesen und erfaßt werden, so ist eine eindimensionale Tabelle mit drei Vektorelementen ausreichend.

```
Teilnehmer <-- Vorname  Name  Vorwahl/Rufnummer
Teilnehmer <--  N(1)    N(2)        N(3)
```

   Dagegen muß eine zweidimensionale Tabelle programmiert werden, wenn mehrere Teilnehmer erfaßt werden sollen.

```
1. Teilnehmer: N(1,1) N(1,2) N(1,3)
2. Teilnehmer: N(2,1) N(2,2) N(2,3)
3. Teilnehmer: N(3,1) N(3,2) N(3,3)
4. Teilnehmer: N(4,1) N(4,2) N(4,3)
5. usw.
```

   Sie erkennen, daß der erste Index immer der Nummer des Teilnehmers entspricht, so daß die Elemente des dritten Teilnehmers immer den Index N(3,X) besitzen.

Wird der zweite Index, hier mit dem Platzhalter X gekennzeichnet, mit 2 angegeben, so wird eindeutig das zweite Element, hier der Name des dritten Teilnehmers, angesprochen. Für die Dimensionierungsanweisung gilt weiterhin, daß 15 Teilnehmer erfaßt werden sollen. Die entsprechende Anweisung lautet dann:

```
DIM N(15,3)
```

d.h. es können maximal 15 Teilnehmer mit je drei Angaben eingelesen und ausgegeben werden.

3. Der zweite Index besitzt bei jedem Teilnehmer die gleichen Werte, so daß sie fest vorgegeben werden können. Der erste Index läuft von 1 und erhöht sich um 1, wenn ein neuer Teilnehmer eingelesen wird, bis maximal zum Wert 15. Es kann somit

```
N(I,1) N(I,2) N(I,3)
```

programmiert werden, wobei der aktuelle Wert der Variable I über eine FOR NEXT-Schleife bestimmt wird.

4. Ein vorzeitiger Eingabeschluß soll ermöglicht werden, indem lediglich die ENTER-Taste bei der ersten Abfrage eines Teilnehmers gedrückt wird. Das kann mit der Anweisung

```
IF N(I,1) = "" GOTO <Zeilennummer>
```

abgefragt werden, d.h. ist die Eingabe ein Leerstring, so wird aus der Eingabe-Schleife zur Ausgabe gesprungen.

5. Die Teilnehmernummer und die Nummer des gerade angesprochenen Vektorelements sollen in der Dateneingabephase als Hinweise ausgegeben werden, damit die Zahl der schon eingegebenen Teilnehmer ablesbar wird.

6. Die Teilnehmer sollen in dem Programm ohne alphabethische Sortierung entsprechend der Reihenfolge der Eingabe, mit PRINT-Anweisungen ausgegeben werden.

7. Name und Vorname sollen mit einem Komma getrennt und vor der Telefonnummer das Kürzel "Tel.:" ausgegeben werden.

Das Programm: "Telefonnummernverzeichnis"

Hiermit sind die Vorüberlegungen zum Programmaufbau abgeschlossen. Sie sollten nun zuerst das Programmlisting auf die besprochenen Programmanweisungen hin prüfen. Achten Sie dabei wieder auf die Kommentarzeilen. Sie erhalten damit den notwendigen Überblick über die Programmblöcke.

Programmlisting

```
10 '**********************************
20 '*   Zeichenketten in Tabellen    *
30 '**********************************
40 '---- Ueberschrift
50 MODE 1
60 LOCATE 5,2
70 PRINT"Zeichenketten in Tabellen
80 LOCATE 1,4
90 '--- Variablendefinition
100 DEFSTR N
110 '-- Dimensionierung
120 DIM N(15,3)
130 '-- Eingabeschleife
140 FOR I=1 TO 15
150 PRINT I;1;:LINE INPUT"Name:     ";N(I,1)
160 IF N(I,1)="" GOTO 210
170 PRINT I;2;:LINE INPUT"Vorname: ";N(I,2)
180 PRINT i;3;:LINE INPUT"Telefon: ";N(I,3)
190 PRINT
200 NEXT I
210 PRINT:PRINT
220 '-- Ausgabeschleife
230 FOR I=1 TO 15
240 IF N(I,1)="" GOTO 290
250 PRINT N(I,1)", ";
260 PRINT N(I,2)";Tel.:";
270 PRINT N(I,3)
280 NEXT
290 PRINT:PRINT"Ende der Ausgabe
300 END
```

Ergebnis einer Beispieleingabe:

```
          Zeichenketten in Tabellen

 1   1 Name:     Schneider
 1   2 Vorname:  Herbert
 1   3 Telefon:  061/76541

 2   1 Name:     Kraft
 2   2 Vorname:  Doris
 2   3 Telefon:  06151/99287

 3   1 Name:     Heinze
 3   2 Vorname:  Sylvia
 3   3 Telefon:  07531/101395

 4   1 Name:

Schneider, Herbert;Tel.:061/76541
Kraft, Doris;Tel.:06151/99287
Heinze, Sylvia;Tel.:07531/101395

Ende der Ausgabe
Ready
```

In dem Programmbeispiel können Sie fünfzehn Eintragungen vornehmen. Im praktischen Betrieb, d.h. bei größeren Datenmengen, müssen "externe Datenspeicher", z.B. Disketten oder Magnetbandkassetten, verwendet werden, damit Daten nach dem Ausschalten Ihres >> CPC 464 << dauerhaft abgespeichert werden können. In den Abschnitten "Projekt Text" und "Projekt Kartei" werden Sie jene Kenntnisse erwerben.

Bemerkungen zum Thema "Bedienerführung"

Eine Eingabe-Erleichterung auf dem Bildschirm ergibt sich durch einen geeignet formulierten Klartext, d.h. was gerade eingegeben werden soll, ist im Klartext anzugeben.

Der Bediener eines Programms muß wissen, ob nun gerade der Name, der Vorname oder die Telefonnummer einzugeben ist!
Ein zusätzlicher Punkt ist hierbei die Gestaltung des Bildschirmaufbaus, der sogenannten "Eingabemasken".

Es ist eine der Stärken der modernen Mikrocomputer, über den Bildschirm mit dem Bediener (Benutzer) in einen Dialog treten zu können. Dadurch wird die Arbeit erleichtert, zum anderen werden auch Eingabe- und Bedienungsfehler vermieden.
Wird geschickt programmiert, so wird der Bediener während des gesamten Programmablaufs geführt. Er erhält dann alle Informationen, die er benötigt.

14. Programmbeispiel zur Tabellenarbeit

Themenübersicht

Am Anfang des vorigen Kapitels wurde eine zweidimensionale Tabelle vorgestellt. Sie soll nun für die weiteren Betrachtungen zum Thema Tabellenarbeit als Grundlage dienen. Ein nicht so umfangreiches BASIC-Programm, das diese Tabelle erstellt, führt Sie in das projektartige Programmieren ein. Aus diesem Programm werden sich Themenbereiche ergeben, die auf Ihren bereits erworbenen Kenntnissen aufbauen und diese erweitern. Die teils neu zu erarbeitenden,teils kurz zu wiederholenden Themen sind:

* Kommentarzeilen
* Programmablaufplan
* Formatvariable
* IF THEN - Anweisung
* Datenbearbeitung
* PRINT TAB(I) Anweisung
* Programmblöcke
* Feld - Dimensionierung
* FOR NEXT-Schleifen
* Dateneingaben
* Formatierte Datenausgabe
* LOCATE - Anweisung

Anwendungsfall für die Tabellenarbeit

Vorgestellt wird ein BASIC-Programm, das Tastatureingaben auswertet,in die Tabelle einfügt, für Berechnungen nutzt und tabellarisch, wie abgebildet, ausgibt:

Nr.	Artikel	Einzelpreis [DM]	Menge	Betrag [DM]
1.	Butter	2.49	1.00	2.49
2.	Zucker	1.49	3.00	4.47
3.	Brot	3.70	1.00	3.70
4.	Schokolade	1.39	5.00	6.95
5.	Marmelade	1.98	4.00	7.92
6.	Milch	1.09	2.00	2.18
7.	Joghurt	0.89	6.00	5.34
		Gesamt:	22.00	33.05

Durchschnittsbetrag ..... = 4.72 DM

Durchschnittlicher Einzelpreis = 1.50 DM

Die zweidimensionale Tabelle gibt eine laufende Nummer, den Artikel, den eingegebenen Einzelpreis, die Menge und den Betrag wieder. Die Summe der Beträge, der Durchschnittsbetrag und zusätzlich der durchschnittliche Einzelpreis werden berechnet und ausgegeben.

Es gelten hierbei die mathematischen Beziehungen:

* Der Betrag eines Einkaufs berechnet sich aus Menge mal Einzelpreis.

* Der Gesamtbetrag ist die Summe der einzelnen Beträge.

* Der Durchschnittsbetrag ergibt sich aus Gesamtbetrag, geteilt durch die Anzahl der Einkäufe.

* Der durchschnittliche Einzelpreis berechnet sich aus der Summe der Einzelpreise, geteilt durch die Gesamtmenge der Artikel.

Das dazu gehörende Programm wurde besonders umfangreich mit Kommentaren versehen, damit Sie in der ersten Übung die Programmstruktur, d.h. die Programmblöcke in ihrer auftretenden Reihenfolge herausarbeiten können.

Sehen Sie sich hierzu das Listing auf der folgenden Seite an und notieren Sie sich die gefundenen Programmblöcke!

Programmlisting

```
10 '*****************************************************************
20 '*                  Programm TABELLE                             *
30 '*****************************************************************
40 '
50 REM Bildschirmeinstellung: Format und Farbe
60 '-----------------------------------------------------------
70 MODE 2
80 BORDER 11
90 LOCATE 20,2:PRINT "Erstellung einer Tabelle"
100 LOCATE 1,4'Cursor positionieren
110 '
120 'Dimensionierung von 4 * 100 Feldern
130 '-----------------------------------------------------------
140 DIM MENGE(100),EINZELPREIS(100),BETRAG(100),ARTIKEL$(100)
150 '
160 'Vorbesetzung der Formatvariablen
170 '-----------------------------------------------------------
180 U$="##. ":R$="######.##  ":RR$="    ######.##"
190 '
200 '*****************************************************************
210 'Schleife zur Eingabe von Artikel, Einzelpreis und Menge
220 '-----------------------------------------------------------
230 '
240 FOR I=1 TO 100:'Schleifenanfang
250 '
260 PRINT USING U$;I;'Ausgabe der laufenden Nr.
270 '
280 'Eingabe des Artikels
290 '-----------------------------------------------------------
300 LINE INPUT "Artikel : ";ARTIKEL$(I)'Eingabe des Artikels
310 IF ARTIKEL$(I)="" THEN 580'kein Artikel --> Eingabeende
320 ARTIKEL$(I)=LEFT$(ARTIKEL$(I),15)'15 Zeichen uebertragen
330 '
340 'Eingabe des Einzelpreises
350 '-----------------------------------------------------------
360 PRINT "    ";'Leerzeichen vor Einzelpreis setzen
370 INPUT "Einzelpreis = ",EINZELPREIS(I)'Eingabe Einzelpreis
380 '
```

```
390 'Eingabe der Menge
400 '-------------------------------------------------------
410 PRINT"    ";'Leerzeichen vor Einzelmenge setzen
420 INPUT "Menge = ",MENGE(I)'Eingabe der Einzelmenge
430 IF MENGE(I)=0 GOTO 580'Menge Null --> Eingabeende
440 '
450 '
460 'Berechnungen
470 '-------------------------------------------------------
480 BETRAG(I) = MENGE(I)*EINZELPREIS(I)'Mengenpreis
490 SUMBETRAG=SUMBETRAG+BETRAG(I)'Zwischen- Endsumme
500 SUMMENGE=SUMMENGE+MENGE(I)'Gesamtmenge
510 '
520 NEXT I'Schleifenende
530 '
540 '*********************************************************
550 'Aufbereiten der Ausgabe
560 '-------------------------------------------------------
570 '
580 CLS:LOCATE 1,2'Bildschirm loeschen und Cursor positionieren
590 N=I-1'Ermitteln der Anzahl der Eingaben
600 '
610 'Berechnungen der Durchschnitte
620 DURCHSCHNITT = SUMBETRAG/N'Durchschnittsbetrag
630 DPREIS=SUMBETRAG/SUMMENGE'Durchschnittspreis
640 '
650 'Ueberschriftszeile
660 PRINT #8,"   Nr.  !"TAB(12)"Artikel";
670 PRINT #8,TAB(27)"! Einzelpreis [DM]  !";
680 PRINT #8,TAB(52)"Menge  !"TAB(61)"Betrag [DM]"
690 PRINT #8,STRING$(72,"-")'Tabellenlinie
700 '
710 '
720 'Zweite Schleife zur Ausgabe der Tabellenelemente
730 '-------------------------------------------------------
740 FOR I=1 TO N'Schleifenanfang
750 '
760 IF MENGE(I)=0 THEN 900'Abfrage des Tabellenendes
```

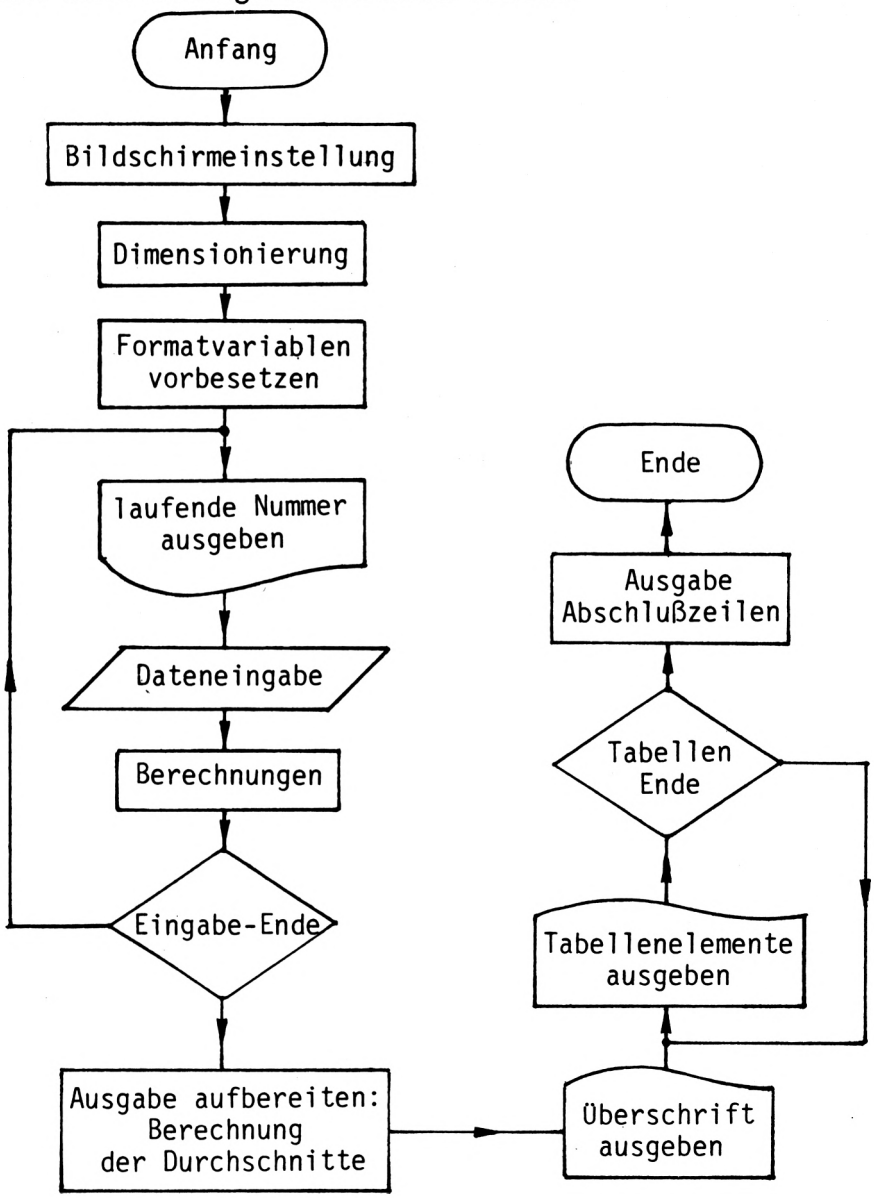
```
770 '
780 'Ausgabe der Tabellenelemente
790 PRINT #8,"   ";USING U$;I;
800 PRINT #8,TAB(9)"¦  "ARTIKEL$(I);
810 PRINT #8,TAB(27)"¦  " USING R$;EINZELPREIS(I);
820 PRINT #8,TAB(47)"¦" USING R$;MENGE(I);
830 PRINT #8,TAB(59)"¦ "USING R$;BETRAG(I)
840 '
850 NEXT I'Schleifenende
860 '
870 'Ausgabe der Abschlusszeilen
880 '-----------------------------------------------------------
890 '
900 PRINT #8,TAB(47)STRING$(26,"-")'Summenstrich
910 PRINT #8,TAB(37)"Gesamt:"USING RR$;SUMMENGE;SUMBETRAG
920 PRINT #8,CHR$(10)'zwei Leerzeilen
930 PRINT #8," Durchschnittsbetrag .........  =";
940 PRINT #8,TAB(35) USING RR$;DURCHSCHNITT;
950 PRINT #8," DM"CHR$(10)'Einheit  und eine Leerzeile ausgeben
960 PRINT #8," Durchschnittlicher Einzelpreis = ";
970 PRINT #8,TAB(35) USING RR$;DPREIS;
980 PRINT #8," DM"
990 END
1000 '**************  Programmende  ********************************
```

Reihenfolge der Programmblöcke

* Bildschirmeinstellung
* Dimensionierung von 4 mal 100 Feldern
* Vorbesetzung der Formatvariablen
* Schleife zur Eingabe von Artikel, Einzelpreis und Menge
* Aufbereiten der Ausgabe
* Schleife zur Ausgabe der Tabellenelemente
* Ausgabe der Abschlußzeilen

Flußdiagramm

Die Struktur des Programmes ist ihnen nun deutlicher, so daß Sie sich mit dem Flußdiagramm befassen können.



14. Programmbeispiel zur Tabellenarbeit

Nun soll mit Ihnen das Programm und gleichzeitig die oben genannten Themenbereiche schrittweise erarbeitet werden.

Feld-Dimensionierung

In dem Programmbeispiel werden vier Variablen bzw. "Felder" dimensioniert und für jede der Variablen 101 "Elemente" oder "Feldelemente" reserviert. Da die Dimensionierung mit Null beginnt und bei 100 endet, ergeben sich 101 Feldelemente.

Die Anzahl von 101 Feldelementen ist nur willkürlich gewählt worden. Selbstverständlich hätte man auch eine größere Anzahl von Feldvariablen vorsehen können. Letztlich bestimmt die Problemstellung selbst den Umfang der zu reservierenden Variablen in einem Programm. Sie könnten bei Bedarf andere Größen angeben, so daß eine größere Datenmenge als 100 Daten eingegeben werden könnte. Hierbei ist die Kapazität des Arbeitsspeichers des >> CPC 464 << zu berücksichtigen.

Die FRE(I)-Anweisung

Hierzu eine Übung, die Sie am >> CPC 464 << selbst nachvollziehen sollen. Sie gehen dabei folgendermaßen vor:

Lassen Sie in der Dimensionierungsanweisung

```
DIM M(10), E(10), B(10)
```

die Anzahl der Felder von Mal zu Mal wachsen. Sie schreiben also zuerst 10, dann 100, dann 2500. Danach fragen Sie jedesmal den noch frei verfügbaren Platz für Programmzeilen im Arbeitsspeicher ab. Die freie Kapazität ermitteln Sie durch die Anweisung:

```
PRINT FRE(I)
```

und erhalten die jeweils noch freie Anzahl von Bytes auf dem Bildschirm angezeigt.

Nun das Übungsbeispiel:

```
10 DIM M(10),E(10),B(10)
20 PRINT "Es sind noch "FRE(I)"Bytes frei!"
RUN
Es sind noch  43265 Bytes frei!
Ready


10 DIM M(100),E(100),B(100)
20 PRINT "Es sind noch "FRE(I)"Bytes frei!"
RUN
Es sind noch  41915 Bytes frei!
Ready


10 DIM M(2500),E(2500),B(2500)
20 PRINT "Es sind noch "FRE(I)"Bytes frei!"
RUN
Es sind noch  5912 Bytes frei!
Ready
```

Sie erkennen aus dem Ergebnis, daß die Dimensionierung von Variablen immer Arbeitsspeicherplatz benötigt!

Die Folgerung für Ihre weitere Programmierarbeit ist, daß Sie bereits bei der Planung von Programmlösungen sorgfältig überdenken sollten, in welchem Umfang Reservierungen von Feldern vorgenommen werden müssen, da der Computer zwar über einen sehr großen, aber trotzdem begrenzten Arbeitsspeicherplatz verfügt.

Hinweis:

Dimensionierung zu großer Felder belegt unnötig Speicherplatz!

Vorbesetzung der Formatvariablen

Nachstehend wird von "Steuerzeichen" gesprochen, damit sind die Doppelkreuze "## ## ## ##" gemeint. In der Programmzeile 180 des Programms "TABELLE" werden die Zeichenkettenvariablen U$, R$ und RR$ mit Steuerzeichen für die Druckaufbereitung besetzt, die für die PRINT-Anweisungen in den Zeilen 260, 790, 810, 820, 830, 910, 940 und 970 benötigt werden. Dies geschieht in Verbindung mit dem geschützten BASIC-Wort USING.

PRINT USING bedeutet "Drucke unter Verwendung von ...". Die Formatvariable, die die entsprechenden Steuerzeichen für die Druckaufbereitung enthält, muß nachgestellt angegeben werden. Anschließend ist (sind) die Variable(n) aufzuführen, die ausgegeben werden soll(en).

Beispiel: PRINT USING U$;I

Eingabe-Schleife

Schleife zur Eingabe Artikel, Einzelpreis und Menge

Überdenken Sie bitte nun, wie oft Sie maximal einen Datensatz eingeben können. Betrachten Sie hierzu die Laufanweisung in der Zeile 240!

```
240 FOR I=1 TO 100:'Schleifenanfang
```

Sie erkennen, daß die Laufvariable I den Wert von 1 bis 100 annehmen kann, d.h. Sie können maximal 100 mal die Größen Artikel, Einzelpreis und Menge eingeben.

Die entsprechenden Eingabe-Anweisungen stehen in den Zeilen 300,370 und 420, wie auch den Kommentaren in den genannten Zeilen zu entnehmen ist. Die Zeile 300 unterscheidet sich von den anderen beiden dadurch, daß mit der Anweisung LINE INPUT eine Zeichenkette eingelesen wird.

```
300 LINE INPUT "Artikel : ";ARTIKEL$(I)'Eingabe des Artikels
```

An dieser Stelle eine wichtige Anmerkung: Betrachten Sie die Programmzeile 370:

```
370 INPUT "Einzelpreis = ",EINZELPREIS(I)'Eingabe Einzelpreis
```

Zeile 370 enthält nicht etwa nur eine, sondern zwei Anweisungen:

1. Eine Eingabe-Anweisung (INPUT)
2. Eine Kommentar-Anweisung

Stehen mehrere Anweisungen in einer Zeile, so ist es eine Anweisungskette oder auch eine Mehrfachanweisung.

Regeln zu Anweisungsketten

* Die Anweisungen in einer Anweisungskette sind durch Doppelpunkte voneinander zu trennen.

* Ist die letzte Anweisung in einer Zeile eine Kommentaranweisung, so kann der davorstehende Doppelpunkt entfallen.

* Nach einer Kommentaranweisung darf in der Zeile jedoch keine weitere Anweisung folgen! BASIC behandelt sie sonst als zum Kommentar dazugehörig!

Prüfen Sie nun anhand des Programmlistings, auf welche Weise Sie die Dateneingabe bereits früher beenden können.

Damit der Benutzer des Programms selbst bestimmen kann, wann die Dateneingabe beendet wird, wurde eine IF THEN-Anweisung in den Zeilen 310 und 430 programmiert.

Es kann aus der Eingabeschleife herausgesprungen werden, wenn entweder die Abfrage des Artikels mit ENTER beantwortet wird

oder

die Abfrage der Menge mit Eingabe von Null quittiert wird.

Die Schreibweisen der IF THEN GOTO-Anweisung

Beachten Sie, daß in der Zeile 310 die Anweisung GOTO und in Zeile 430 das THEN fehlen. Hier ist nur ein GOTO zu finden. Es hat aber seine Richtigkeit!

Der >> CPC 464 << BASIC-Interpreter erlaubt folgende BASIC-Schreibweisen:

```
IF ... THEN GOTO ...
IF ... GOTO ...
IF ... THEN ...
```

Auf THEN und GOTO muß die Angabe eines durch eine Anweisungsnummer bezeichneten Sprungziels folgen.

Regeln zur IF THEN GOTO-Anweisung

* Trifft eine Abfrage zu, dann wird im Programm zur angegebenen Zeile gesprungen.

* Folgt auf die IF-Abfrage kein Sprungziel bzw. keine Zeilennummer, sondern eine sonstige Anweisung, dann muß THEN stehen.

Betrachten Sie nun die Anweisungen in Zeile 480, 490 und 500!

In der Anweisung 480 wird der Betrag(I) je Zeile aus den Elementen MENGE(I) mal EINZELPREI'(I) ermittelt. In der Zeile 490 wird der BETRAG(I), also der Betrag jeder Zeile, dem Summenfeld SUMBETRAG hinzuaddiert und so der Gesamtbetrag errechnet. In Zeile 500 wird die MENGE(I) der Gesamtmenge SUMMENGE hinzuaddiert und auf diese Weise die Gesamtmenge der Artikel berechnet.

Mit der nächsten ausführbaren Anweisung gelangen Sie jetzt an das Schleifenende. Wenn die Laufvariable I den Endwert 100 noch nicht erreicht hat, kehrt das Programm an den Schleifenanfang zurück und erhöht I um 1.
Es gilt daher nach jedem Schleifendurchlauf: I=I+1

Wird festgestellt, daß I bereits den Wert 101 erreicht hat, also größer als der zulässige Endwert 100 ist, so wird sofort zu derjenigen Anweisung verzweigt, die nach der NEXT-Anweisung steht.

Das ist die Anweisung Nr. 530. Bei den Zeilennummern 530 bis 570 handelt es sich aber nur um Kommentarzeilen, die vom Interpreter übergangen werden. Die nächste ausführbare Anweisung ist somit in Zeile 580.

Aufbereiten der Ausgabe

Die CLS-Anweisung in Zeile 580 löscht den Bildschirm und es wird mit der LOCATE-Anweisung der Textcursor an den Anfang der zweiten Bildschirmzeile positioniert.

Die Anweisung 590 veringert den Wert von I nach Eingabeende um 1.

```
590 N=I-1'Ermitteln der Anzahl der Eingaben
```

Analysieren Sie anhand des Programmablaufs, welchen Wert die Variable I bei einer bestimmten Anzahl von Eingaben annimmt. Benutzen Sie bitte bei der Bestimmung des Variablenwertes den nachstehenden Fall!

Fallbeispiel:

Sie haben zwei vollständige, abgeschlossene Eingaben vorgenommen. Steht nun in der Variablen I der Wert "2", "3" oder "4"?

Prüfen Sie dieses bitte "praktisch" nach, indem Sie zwei vollständige Eingaben jeweils mit dem Artikel, dem Einzelpreis und der Menge vornehmen.

Die nächste Abfrage des Artikels beantworten Sie mit der ESC-Taste, d.h. Sie brechen das Programm ab.

Nun fragen Sie im Direktmodus die Variable I mit nachstehender Anweisung ab:

```
?I
```

Sie erhalten als Ergebnis den Wert "3".

Hilfen für die Analyse:

* Der Wert von I entspricht dem Wert , der erhalten wird, wenn bei Abfrage des dritten Artikels nur ENTER gedrückt würde.

* Die Bedingung in der IF-Anweisung in der Zeile 310 ist dann erfüllt und die Schleife wird verlassen, bevor die Laufvariable I den Endwert 100 erreicht hat.

* Die Laufvariable I wurde bereits um eins erhöht, da es sich um den dritten Schleifendurchlauf handelt.

Die Schlußfolgerung ist, daß I um eins größer ist als die Anzahl der vollständigen Eingaben und daher N=I-1 gesetzt werden muß.

Schöpfen Sie die maximale Datenmenge von 100 Eingaben aus, so wird die Schleife vollständig durchlaufen und die Laufvariable I ist um eins größer als der Endwert der FOR NEXT-Schleife.

Daher muß auch in diesem Fall "N = I - 1" gesetzt werden.

Berechnung der Durchschnitte

Die Berechnungen der Durchschnitte dienen nur statistischen Zwecken, die Sie z.B. für Ihren geschäftlichen Bedarf nutzen können.
Sie ermitteln mit dem Programmblock den Durchschnittsbetrag und den Durchschnittspreis Ihrer Ein- und Verkäufe. Die Berechnung erfolgt in den Zeilen

```
620 DURCHSCHNITT = SUMBETRAG/N'Durchschnittsbetrag
630 DPREIS=SUMBETRAG/SUMMENGE'Durchschnittspreis
```

In der Variablen SUMBETRAG befindet sich gemäß Anweisung Nr. 490

```
490 SUMBETRAG=SUMBETRAG+BETRAG(I)'Zwischen- Endsumme
```

die Gesamtsumme der eingegebenen Beträge.

In der Variablen SUMMENGE befindet sich aufgrund der Anweisung Nr. 500

```
500 SUMMENGE=SUMMENGE+MENGE(I)'Gesamtmenge
```

die Gesamtmenge der gekauften bzw. verkauften Artikel.

Ausgabe der Überschriftszeile

Anschließend wird mit den Anweisungen 660 bis 690 die Überschriftszeile und die Tabellenlinie auf dem Drucker ausgegeben. Wünschen Sie die Ausgabe der Tabelle auf dem Monitor, so ersetzen Sie bei allen PRINT-Anweisungen die Datenstrom-Ziffer "8" durch die Ziffer "0", oder Sie lassen ersatzlos den gesamten Ausdruck "#8," entfallen.

Zweite Schleife zur Ausgabe der Tabellenelemente

In Zeile 740 befindet sich die Laufanweisung mit der Laufvariablen I, die einen Anfangswert von 1 und einen Endwert von N annehmen kann. In Zeile 760 findet sich eine IF-Anweisung:

```
760 IF MENGE(I)=0 THEN 900'Abfrage des Tabellenendes
```

WENN das gerade bearbeitete Element MENGE(I) gleich Null ist, DANN soll die Schleife verlassen werden. Das bedeutet einen unbedingten Sprung zu der Anweisung 900, die nach der NEXT-Anweisung steht. Ist der Inhalt von MENGE(I) ungleich Null, so wird auf dem Drucker eine formatierte Zeile mit dem Inhalt ausgegeben, der in den Überschriftszeilen 660 bis 680 beschrieben wird.

14. Programmbeispiel zur Tabellenarbeit

Es wird auf der laufenden Tabellenposition

1. Laufende Nummer
2. Artikel
3. Einzelpreis
4. Menge
5. Betrag

mit der PRINT USING-Anweisung geschrieben.

Ein Rücksprung zum Schleifenanfang erfolgt in der Zeile 850, wenn die Tabelle noch nicht vollständig ausgegeben ist.

Ausgabe der Abschlußzeilen

Betrachten Sie bitte die Anweisungen in den Zeilen von 870 bis 980 und notieren Sie, was die einzelnen Anweisungen bewirken. Für Ihre Korrektur dienen die untenstehenden Angaben.

Hilfen für die Programmanalyse:

* Die Anweisungen 870 bis 890 sind Kommentarzeilen.

* Die Anweisung 900 kennzeichnet das Additionsende, indem mit STRING$ 26 mal das Zeichen "-" ab Position 47 ausgegeben wird.

* Die Anweisung 910 gibt die Gesamtmenge und die Gesamtsumme aus.

* In Zeile 920 werden zwei Leerzeilen durch die Anweisung: PRINT #8,CHR$(10) erzeugt. Die Anweisung bewirkt zwei Zeilenvorschübe, wobei der erste durch das Steuerzeichen CHR$(10) und der zweite durch den BASIC-Interpreter des >> CPC 464 << ausgegeben wird, da nach CHR$(10) kein Semikolon gesetzt wurde.

* Die Anweisungen 930 und 960 geben die Worte "Durchschnittsbetrag" und "Durchschnittlicher Einzelpreis" aus.

* Die Durchschnittswerte werden in den Anweisungen 940 und 970 ausgegeben.

* Die Anweisungen 950 und 980 setzen die Einheit "DM".

Das geschützte BASIC-Wort END in Zeile 990 beendet das Programm und bewirkt eine Rückkehr in den Kommandomodus. Die END-Anweisung darf entfallen, wenn sie die letzte ausführbare Programmanweisung ist.

Sie haben damit das Programmende erreicht und zudem wichtige Kenntnisse erworben, die es Ihnen nun erlauben werden, das Programm "TABELLE" auf Ihre ganz speziellen Bedürfnisse anzupassen. Denkbar sind z.B. Änderungen an den Benutzerhinweisen, die das Programm ausgibt. Auch Erweiterungen lassen sich einbinden, so z.B. ein Programmblock "Speicherung", der die Daten auf dem Datacorder ablegt.

15. Sortieren von Zahlen und Zeichenketten

Einführung und Aufgabenstellung

In Technik und Verwaltung ist eine der am häufigsten zu lösenden Aufgaben das Sortieren von umfangreichem Datenmaterial. Das Sortieren von Hand, z.B. das Sortieren von Karteikarten nach unterschiedlichen Kriterien, ist dabei ein recht mühsames Geschäft! Hier erleichtert uns die elektronische Datenverarbeitung die Aufgabe außerordentlich. Die Arbeit von Stunden und Tagen läßt sich auf Minuten und Sekunden schrumpfen!

Auch Sie werden mit solchen Programmen Zeit gewinnen für andere, kreativere Dinge!

Wegen der außerordentlichen Bedeutung, die das Sortieren in der Datenverarbeitung hat, wurden die verschiedenartigsten Verfahren entwickelt, um möglichst schnell nach möglichst vielen Gesichtspunkten sortieren zu können.

Aus der Fülle der vorhandenen Sortierverfahren kann hier natürlich nur ein einfaches Sortierprogramm vorgestellt werden. Das Programmbeispiel vermittelt Ihnen Grundkenntnisse über die Sortierung von Daten, die Sie befähigen sollen, Sortierprogramme zu analysieren und zu beurteilen.

Zunächst wird in diesem Kapitel die Aufgabenstellung besprochen, um Ihnen vorab einen Einblick in das Thema "Sortieren" zu bieten und den notwendigen Programmaufbau zu erarbeiten.

Vorgaben zum Sortierprogramm:

* Maximal 100 alphanumerische Variable sollen in aufsteigender Reihenfolge sortiert werden.

* Die Daten sollen über die Tastatur eingegeben und dem Sortierprogramm übergeben werden.

* Sortiert werden sollen Nachnamen ohne weitere Zusätze.

Vorüberlegungen zum Programmaufbau

1. Die anfallenden Datenelemente können mit dem Oberbegriff "Nachname" zusammengefaßt werden.

2. Die Datenelemente können in eine eindimensionale Tabelle geschrieben werden.

3. Die eindimensionale Tabelle muß mindestens 100 Elemente aufnehmen können.

4. Die Daten sind vom Typ Zeichenkette, d.h. es sind "alphanumerische" Variable zu benutzen.

5. Der Name der zu dimensionierenden Tabelle muß am Schluß ein Dollarzeichen als Kennung erhalten.

6. Es soll die dimensionierte Variable A$(I) benutzt werden.

Dimensionierung

Aufgrund dieser Überlegungen ist eine eindimensionale Tabelle zu programmieren und mit der DIM-Anweisung:

```
DIM A$(99)
```

Platz für 100 Elemente zu schaffen.

Um in den Vorüberlegungen fortzufahren, soll der eigentliche Ablauf des Sortierprogramms dargestellt werden.

Die zu sortierenden Daten bzw. Namen sollen sein:

```
A$(1) = Merlemann
A$(2) = Seuring
A$(3) = Berghoff
A$(4) = Zergel
```

Das Ergebnis des Sortiervorgangs soll nachstehende Liste ergeben:

A$(3) = Berghoff
A$(1) = Merlemann
A$(2) = Seuring
A$(4) = Zergel

Vergleichen Sie die beiden Listen. Sie stellen fest, daß ein "Platz-Tausch" vorgenommen wurde. Die Namen haben in der Tabelle durch Vertauschen ihre Plätze gewechselt.

Sind die Daten sortiert, so sollen sie auf dem Bildschirm ausgegeben werden.

Der Programmablauf soll sich dann wiederholen:

1. Eingabe über die Tastatur

2. Verarbeitung der Daten mit dem Ziel der Sortierung

3. Ausgabe der sortierten Daten

Übersicht der Programmblöcke

Sie erkennen, daß drei Programmblöcke für den gewünschten Programmablauf notwendig sind. Im ersten und letzten Programmblock finden Sie bekannten Lehrstoff, den Sie schon an ähnlichen Teilproblemstellungen erarbeitet haben.

Erster Programmblock: Daten-Eingabe

Die Eingabe über die Tastatur wird beim Programmablauf über eine "LINE-INPUT"-Anweisung innerhalb einer FOR NEXT-Schleife erreicht.

Zweiter Programmblock: Sortierung

Der mittlere Block ist in seiner Programmierung für Sie neu.

Dritter Programmblock: Daten-Ausgabe.

Die Ausgabe der sortierten Daten geschieht mit einer PRINT-Anweisung, und zwar ebenfalls innerhalb einer FOR NEXT-Schleife.

Programmblock Sortieren

Vorüberlegungen zum zweiten Programmblock

1. Kernstück des Blocks muß ein Programmteil sein, in dem die Variablen in einer gewünschten Folge "ihre Plätze tauschen".

2. Es muß dabei geprüft werden, wieviele Variablen zu sortieren sind und in welcher Weise dies zu geschehen hat.

3. Bei der gegebenen Aufgabenstellung soll in aufsteigender Folge sortiert werden, d.h. in alphabethischer Folge.

4. Sortieren setzt eine vorgegebene Rang-Ordnung und somit eine Sortierfolge voraus.

5. Die zu sortierenden Daten müssen mit der vorgegebenen Ordnung verglichen werden.

Sortieren heißt also mit anderen Worten:

"Ordnen gegebener Elemente durch Vergleichen mit einer vorgegebenen Rangfolge".

Hinweis:

Eine andere Art der Sortierung, die überwiegend beim Sortieren von Zahlen vorkommt, ist die Sortierung in absteigender Folge, bei der die höchste Zahl zuerst und die kleinste Zahl zuletzt steht.

ASCII-Wert	Zeichen	ASCII-Wert	Zeichen
32		80	P
33	!	81	Q
34	"	82	R
35	#	83	S
36	$	84	T
37	%	85	U
38	&	86	V
39	'	87	W
40	(	88	X
41	)	89	Y
42	*	90	Z
43	+	91	[
44	,	92	\
45	-	93	]
46	.	94	↑
47	/	95	←
48	0	96	`
49	1	97	a
50	2	98	b
51	3	99	c
52	4	100	d
53	5	101	e
54	6	102	f
55	7	103	g
56	8	104	h
57	9	105	i
58	:	106	j
59	;	107	k
60	<	108	l
61	=	109	m
62	>	110	n
63	?	111	o
64	@	112	p
65	A	113	q
66	B	114	r
67	C	115	s
68	D	116	t
69	E	117	u
70	F	118	v
71	G	119	w
72	H	120	x
73	I	121	y
74	J	122	z
75	K	123	{
76	L	124	|
77	M	125	}
78	N	126	~
79	O		

Erkenntnisse für die Programmierung des Sortiervorgangs

In dem zweiten Programmblock wird es also zumindest eine Abfrage geben müssen, die einen Vergleich enthält. Auf der vorigen Seite ist der in aufsteigender Reihen- bzw. Rangfolge geordnete, standardisierte Zeichensatz in einer Tabelle wiedergegeben, wie er in dem Mikrocomputer >> CPC 464 << anzutreffen ist. Der Tabelle können Sie die "Wertigkeit" eines Zeichens entnehmen. So ist z.B. das Zeichen mit der laufenden Nummer 49, also die Ziffer 1, in der Sortierfolge niedriger als das Zeichen mit der laufenden Nummer 57, das der Ziffer 9 entspricht. Eine weitere Vorüberlegung ist dann dem Platz-Tausch zu widmen.

Vorüberlegungen zum Platztausch

1. Durch Übertragen wird der Inhalt der empfangenden Speicherstelle überschrieben, der Inhalt der sendenden Speicherstelle dagegen bleibt erhalten.

2. Die Variable B, die mit der Variablen A zu tauschen ist, muß "gerettet" werden.

3. Damit die Variable B nicht überschrieben wird, muß sie vorübergehend an einer anderen Stelle untergebracht werden, bis sie den endgültigen Platz einnehmen kann, der ihr nach der Sortierordnung zusteht.
   Den Vorgang nennt man Zwischenspeichern!

4. Ist die Variable B zwischengespeichert, kann die Variable A auf dem bisherigen Platz der Variablen B durch Übertragen untergebracht werden. Durch das Übertragen wird also der bisherige Inhalt des empfangenden Speicherplatzes B ersetzt, was jetzt ja bedenkenlos geschehen kann.

5. Die "gerettete" Variable B wird auf den vorhergehenden Platz der inzwischen umgesetzten Variablen A übertragen.
   Auch bei dieser Übertragung wird der bisherige Inhalt der Speicherstelle A überschrieben und durch den Inhalt der Variablen B ersetzt.

```
HA  <---------< B     Zwischenspeichern

 B  <------< A        Platztausch 1. Teil

 A  <---------< HA    Platztausch 2. Teil
```

6. Das Überschreiben kann jetzt ohne Bedenken geschehen, da die Variable A bereits auf ihrem neuen Platz untergebracht und somit ausgetauscht worden ist.

Es bleibt nur noch abzuklären, auf welche Weise die Variable B zwischengespeichert wird. Es wird ein Speicherplatz als Hilfsspeicher für die Variable A mit dem Namen HA benannt.

Der Platztausch läuft dann folgendermaßen ab:

```
10 HA = B
20 B  = A
30 A  = HA
```

1. In Zeile 10 wird die Variable B zwischengespeichert.

2. In Zeile 20 wird der Inhalt der Variablen A auf den Platz der Variablen B übertragen.

3. In Zeile 30 wird der Inhalt der auf dem Platz der Hilfsvariablen HA zwischengespeicherten Variablen B auf den Platz von A übertragen.

Für den Platztausch werden also insgesamt drei Anweisungen benötigt.

Das Austauschen von Tabellenelementen ist, wie Sie inzwischen gesehen haben, ein wesentliches Element des Sortierens. Die Frage, ob es nicht auch ein Sortieren ohne jegliches Austauschen gibt, kann man mit "Nein" beantworten.

Ob allerdings die Tabellenelemente selbst ausgetauscht werden müssen, oder ob es andere Wege gibt, mit dem Sortieren zum Erfolg zu kommen, läßt sich nicht einfach mit "Nein" beantworten.
In diesem Buch kann das Thema jedoch nicht untersucht werden. Es bleibt einem weiteren Buch zum >> CPC 464 << aus dem Heim-Verlag vorbehalten. Für diejenigen Leser, die sich eingehender mit dem Gebiet des Sortierens beschäftigen wollen, sei nur das Stichwort "Stellvertretersortierung" genannt.

Stellvertretersortierung

Begriffsklärung:

> Bei der Stellvertretersortierung werden nicht die Inhalte der Tabellenelemente selbst ausgetauscht, sondern die Platznummern, auf denen die Elemente innerhalb der Tabelle stehen. Anhand einer Liste der in Abhängigkeit vom Tabelleninhalt sortierten Platznummern werden dann die Elemente ausgegeben und erscheinen somit als sortiert.

Ein Programmbeispiel für das Sortieren

Nach den Vorüberlegungen sollten Sie sich jetzt dem BASIC-Programm zuwenden, das das Sortierproblem lösen soll.

Suchen Sie bitte zuerst die Programmzeilen, die Kommentare enthalten. Die Zeilen sind nur für Sie zur Erläuterung bestimmt und haben nichts mit dem Programmablauf selbst zu tun!

Programmlisting:

```
10 '**************************************
20 '*   SORTIEREN VON DATENSAETZEN     *
30 '**************************************
40 '
50 '--- Dimensionierung --------------
60 DIM A$(99)
70 '
80 '--- Bildschirmaufbau -------------
90 MODE 1
100 PAPER 0:INK 0,0
110 PEN 1:INK 1,26
120 CLS
130 PRINT "    Sortieren von Datensaetzen"
140 PRINT:PRINT " -> Eingabe von Zeichenketten <-"
150 PRINT :PRINT "Fuer Eingabeende bitte '*' eingeben."
160 WINDOW #1,1,40,10,25
170 WINDOW SWAP 0,1
180 '
190 '-- Dateneingabe -----------------
200 FOR N=0 TO 99
210 LINE INPUT "neue Zeichenkette:",A$(N)
220 A$(N)=UPPER$(A$(N))
230 IF A$(N)="*" THEN 270
240 NEXT N
250 '
260 '-- Sortieren der Tabellenelemente
270 Z=0
280 FOR I=0 TO N-2
290 IF A$(I)<=A$(I+1) THEN 320
300 HA$=A$(I):A$(I)=A$(I+1):A$(I+1)=HA$
310 Z=1
320 NEXT I
330 IF Z=1 THEN 270
340 CLS
350 '
```

```
360 '-- Ausgabe der sortierten Elemente
370 LOCATE #1,5,3:PRINT #1,"Aus
380 LOCATE #1,1,5
390 PRINT #1," Weitere Ausgabe --> ENTER druecken "
400 FOR I=0 TO N-1
410 PRINT USING "##";I;
420 PRINT" : "A$(I)
430 IF I MOD 10 = 0 AND I THEN LINE INPUT ENTER$
440 NEXT I
450 '-- Ausgabe beenden ---------------
460 PRINT
470 PRINT "Laenge der Liste: "N" Elemente
480 PRINT
490 LOCATE #1,1,5:PRINT #1,STRING$(49," ")
500 END
```

Es sind die Programmzeilen:

```
 50 'Dimensionierung
 80 'Bildschirmaufbau
190 'Dateneingabe
260 'Sortieren der Tabellen-Elemente
360 'Ausgabe der sortierten Elemente
450 'Ausgabe beenden
```

Sie sehen, wie leicht es ist, Programmblöcke zu erkennen, wenn sie durch Kommentarzeilen entsprechend gekennzeichnet sind. Auch in Ihren eigenen Programmen sollten Sie dieses berücksichtigen, um längere BASIC-Programme lesbar zu gestalten.

Programmblöcke

Die Vorüberlegungen zu dem Sortierprogramm ergaben, daß zum Sortieren von 100 Nachnamen neben einer Dimensionierung eines Feldes mindestens drei Programmblöcke zu programmieren sind.

```
Programmblock  ---> Dateneingabe
Programmblock  ---> Sortierung
Programmblock  ---> Ausgabe der sortierten Daten
```

15. Sortieren von Zahlen und Zeichenketten

Die weiteren Überlegungen zur "Sortierung von Daten" führten zu der Erkenntnis, daß hier für den Programmablauf bestimmte Anweisungen notwendig sind.

Anweisungen im Programmblock: "Sortierung"

Notwendige Anweisungen:

* Anzahl der zu sortierenden Daten feststellen
* ASCII-Werte vergleichen
* Daten vertauschen über Zwischenspeichern in HA$

Ihre Aufgabe ist es nun, diese Anweisungen in dem Programmlisting zu finden und deren Wirkung zu bestimmen.

Hilfen für Ihre Analyse:

1. Anweisung:
```
200 FOR N=0 TO 99
240 NEXT N
```

* Die Anzahl der zu sortierenden Daten wird durch die erste FOR NEXT-Schleife bestimmt und dem Programmblock "Sortieren der Tabellenelemente" mit der Variablen N übergeben.

2. Anweisung:
```
290 IF A$(I)<=A$(I+1) THEN 320
```

* In der Anweisung werden zwei Feldelemente bzw. zwei Zeichenketten verglichen und daraus eine Programmverzweigung abgeleitet. Für den Vergleich zweier Zeichenketten nutzt der BASIC-Interpreter den ASCII-Wert, indem zuerst die beiden Anfangsbuchstaben beider Zeichenketten anhand ihres ASCII-Werts verglichen werden. Sind beide Werte gleich, d.h. es sind beide Anfangsbuchstaben gleich, so werden die zweiten Buchstaben der Zeichenketten miteinander verglichen. Sind sie wieder identisch, so wiederholt sich der Vorgang.

Beachten Sie die folgenden Beispiele für Zeichenkettenvergleiche.

Wort	ASCII-Werte der Buchstaben					
Hase	72	97	115	101		
Hasen	72	97	115	101	110	
Haus	72	97	117	115		
Garten	71	97	114	116	101	110

Die Beziehung zwischen den drei Zeichenketten ist demnach:

Garten < Hase < Hasen < Haus

so daß sich auch diese endgültige, sortierte Reihenfolge für die Zeichenketten ergeben soll, d.h. sie sind so zu sortieren bzw. zu tauschen.

3. Anweisung: `300 HA$=A$(I):A$(I)=A$(I+1):A$(I+1)=HA$`

* Sie erinnern sich an die Vorüberlegungen, daß die Daten vor dem Vertauschen zwischengespeichert werden müssen, da sie sonst überschrieben werden.

HA$	<--------< A$(I)	Zwischenspeichern
A$(I)	<--------< A$(I+1)	Platztausch 1. Teil
A$(I+1)	<--------< HA$	Platztausch 2. Teil

Die Anweisungskette in Zeile 300 enthält genau jene Anweisungen.

Ein Platztausch zum Zwecke des Sortierens findet dann statt, wenn der Inhalt des Feldelementes A$(I) größer als der Inhalt des Feldelementes A$(I+1) ist.

Allgemein gilt also:

> Ist das gegebene Element größer als das unmittelbar folgende Element, so müssen beide ihre Plätze tauschen.

Sie haben jetzt die Wirkung der drei Anweisungen bestimmt.

Schauen Sie bitte nun auf die Anweisungen 270 und 310:

```
270 Z=0
310 Z=1
```

Bei Ihren Überlegungen sollten Sie insbesondere die Anweisung 290 näher betrachten.

Die Variable Z wirkt wie ein Schalter.

Schalterstellung "0":

> Z bleibt auf Null gestellt, wenn der Inhalt sämtlicher Feldelemente A$(I) kleiner oder gleich dem Inhalt der nachfolgenden Feldelemente A$(I+1) ist, d.h. sämtliche Feldelemente sind fertig sortiert.
>
> Ist sortiert, so trifft die Bedingung in der Anweisung 290 immer zu und es wird jedesmal in die Zeile 320 gesprungen, d.h. Z wird nicht auf 1 gestellt. In diesem Fall wird in Zeile 330 nicht erneut zur Zeile 270 zurückgesprungen und der Programmblock "Ausgabe der sortierten Elemente" bearbeitet.

Schalterstellung "1":

> Sind die Feldelemente noch nicht vollständig sortiert, so wird ein Sortiervorgang ausgelöst, d.h. die Zeilen 300 und 310 durchlaufen. In Zeile 310 wird dann Z auf 1 gestellt. Z bewirkt in der Schalterstellung 1 in Zeile 330 eine Rückverzweigung zur Anweisung 270.
> Danach wird die FOR NEXT-Schleife in dem Programmblock "Sortieren der Tabellenelemente" erneut durchlaufen.

Mit anderen Worten:

Im ungünstigsten Fall muß jedes Element getauscht werden, wodurch das Sortieren sehr lange dauern kann.

Kurzinformation zu den weiteren Programmblöcken

Dimensionierung des Feldes A$(I)

Mit der Anweisung DIM A$(99) wird ein Feld dimensioniert und Platz für 100 Feldelemente geschaffen.

Bildschirmaufbau

Die MODE 1 -Anweisung setzt das Bildschirmformat auf 40 Spalten pro Zeile.
Mit den Anweisungen in den Zeilen 100 und 110 wird die Hintergrundfarbe auf die Farbe Schwarz und die Zeichenfarbe auf Hellweiß gesetzt.
Die CLS-Anweisung löscht und färbt entsprechend der PAPER- und der INK-Anweisung in Zeile 100 den Bildschirm ein.
Mit WINDOW #1,1,40,10,25 wird das Bildschirmfenster mit dem Datenstrom "1" definiert. WINDOW SWAP 0,1 setzt den Textcursor in dieses Bildschirmfenster, so daß die Eingaben hier erfolgen können, ohne die vorhandenen Bedienungshinweise bei größeren Datenmengen zu beeinträchtigen.

Dateneingabe

Die Dateneingabe erfolgt mit einer LINE INPUT-Anweisung, so daß auch Kommata eingegeben werden können. Abgefragt wird in Zeile 230 das "*" -Zeichen, für die Kennung des vorzeitigen Eingabeendes. Wird es eingegeben, so wird zum Programmblock "Sortieren der Tabellenelemente" verzweigt.
Vorgestellt wird Ihnen die Wirkung der UPPER$-Anweisung in Zeile 220. Sie bewirkt eine automatische Umwandlung in Großbuchstaben.

Ausgabe der sortierten Elemente

LOCATE positioniert den Textcursor auf das angesprochene Bildschirmfenster. Über eine FOR NEXT-Schleife werden die sortierten Tabellenelemente ausgegeben. Die blockartige Ausgabe von je 10 Elementen wird in Zeile 430 durch eine MODulo-Division erreicht. Zeile 430 wurde programmiert, damit Sie als Benutzer die Möglichkeit besitzen, die Ausgabegeschwindigkeit der Tabellenelemente selbst zu bestimmen. Erreicht wird es durch eine IF THEN-Anweisung, die abfragt, ob der Schleifenzähler durch 10 teilbar ist. Ist dies gegeben, so kann über Eingabe von ENTER der nächste sortierte Block angefordert werden. Beachten Sie hierbei, daß bei dem ersten Element der Schleifenzähler gleich Null ist und eine Division mit 10 auch in diesem Fall die Bedingung der IF THEN-Anweisung erfüllt, so daß mit AND der Fall ausgeschlossen wurde.

Ausgabe beenden

Zusätzlich wird zu Ihrer Information die Anzahl der Elemente ausgegeben. Zeile 490 löscht einen Teil der Bedienerhinweise, um das Ausgabeende zu signalisieren.

Mit dem Programm haben Sie die Grundlagen des Sortierens kennengelernt, so daß Sie nun zu komplexeren Sortierlösungen übergehen können, die auch in der Edition HEIM erschienen sind.

Sie ausführlich in die Arbeitsweise mit den mathematischen Funktionen einzuführen, ist nicht vorgesehen, da es nicht das Thema in dem vorliegenden Band sein soll. An dem Thema interessierte Leser finden in der Edition HEIM entsprechende Literatur. Hier sollen nur das Format und die mathematische Wirkung in einer Übersicht aufgezeigt werden.

Eine mathematische Aufgabe

Zu Beginn eine kleine mathematische Problemstellung und ihre Lösung:

Problemstellung:

> Um aus der Zahl 25 die Quadratwurzel zu ziehen, schreiben Sie folgendes "Programm":
>
> ```
> PRINT SQR(25)
> ```
>
> und erhalten auf dem Bildschirm die Anzeige
>
> ```
> 5
> Ready
> ```

In der PRINT-Anweisung finden Sie die mathematische Funktion SQR(25). Der Name der Funktion ist SQR. Hinter dem Funktionsnamen muß immer ein Klammerausdruck stehen. Es kann sich dabei um eine Zahl, wie in dem Beispiel oder aber um einen allgemeinen numerischen Ausdruck handeln, für den der Computer den Wert errechnen soll.

Der in Klammern stehende Ausdruck wird als Argument bezeichnet. Allgemein wird das Argument mit X angegeben, also:

```
SQR(X).
```

Da aus einer negativen Zahl keine Wurzel gezogen werden kann, meldet der Computer auch einen Fehler, wenn Sie programmieren:

```
PRINT SQR(-25)
```

Er meldet auf dem Bildschirm:

```
Improper argument
```

zu deutsch: falsches Argument. Vergleichen Sie hierzu das Kapitel "Fehlermeldungen und Fehlercodes".

Die Funktion SQR(X) gehört zu den Standardfunktionen, die man auch als "eingebaute" Funktionen bezeichnen kann, da sie der BASIC-Interpreter dem Benutzer von sich aus zur Verfügung stellt, so daß sie nicht programmiert werden müssen.

Überblick der mathematischen Standardfunktionen

Es wird der ROM-BASIC-Interpreter des >> CPC 464 << besprochen, so daß in der nachstehenden, tabellarischen Übersicht nur auf die vorhandenen Standardfunktionen eingegangen wird.
Derjenige Leser, der nicht das mathematische Problemlösen mit Standardfunktionen liebt, kann auch zum nächsten Abschnitt "Zeichenkettenfunktionen" übergehen.

Arithmetische Funktionen

ABS(X) - Absolutwert

X kann jeder beliebige numerische Wert sein. Das Ergebnis hat denselben Typ wie das Argument.

```
PRINT ABS(-9.80665)
 9.80665
```

SGN(X) - Signumfunktion (Vorzeichen)

Liefert -1 für negative und 1 für positive X, sowie 0 wenn X gleich 0 ist.

Beispiel:

```
PRINT SGN(-345.1),SGN(0),SGN(99)
-1             0             1
```

SQR(X) - Quadratwurzel

Berechnet die Quadratwurzel einer positiven Zahl.

```
PRINT SQR(9),SQR(10)
 3             3.16227766
```

LOG(X) - natürlicher Logarithmus

LOG berechnet den natürlichen Logarithmus einer positiven Zahl.

```
PRINT LOG(1.234)
 0.210260926
```

EXP(X) - Exponentialfunktion

Die Funktion EXP berechnet die natürliche Exponentialfunktion zur Basis e (=2.7182818), wobei X kleiner sein muß als 88.

```
I1=EXP(LOG(1.234))
?I1
 1.234
```

Trigonometrische Funktionen

SIN(X), COS(X), TAN(X)

Die o.a. Funktionen liefern Sinus, Cosinus und Tangens eines Winkels im Bogenmaß. X ist irgendein numerischer Ausdruck.

Soll ein Winkel im Gradmaß bearbeitet werden, so muß zuvor die Anweisung DEG gegeben werden.

```
?SIN(1)
 0.841470985
Ready
DEG
Ready
?SIN(30)
  0.5
```

ATN(X)

Berechnet denjenigen Winkel im Bogenmaß, dessen Tangens X ist. Um den Winkel im Gradmaß zu erhalten, ist zuvor die Anweisung DEG einzugeben.

```
PRINT ATN(TAN(1.2345))
 1.2345
```

Konvertierungsfunktionen

INT(X), FIX(X), CINT(X) - Ganzzahlkonvertierung

INT liefert die nächste ganze Zahl, die kleiner ist als X, FIX schneidet die Stellen nach dem Dezimalpunkt ab, CINT rundet X auf die nächste ganze Zahl.

```
10 PRINT"    X      INT     FIX    CINT"
20 PRINT STRING$(27,"-")
30 FOR Z=-1 TO 2.1 STEP 0.3
40 PRINT USING "###.#   ";Z;INT(Z);FIX(Z);CINT(Z)
50 NEXT Z
```

Ergebnis:

X	INT	FIX	CINT
-1.0	-1.0	-1.0	-1.0
-0.7	-1.0	0.0	-1.0
-0.4	-1.0	0.0	0.0
-0.1	-1.0	0.0	0.0
0.2	0.0	0.0	0.0
0.5	0.0	0.0	1.0
0.8	0.0	0.0	1.0
1.1	1.0	1.0	1.0
1.4	1.0	1.0	1.0
1.7	1.0	1.0	2.0
2.0	2.0	2.0	2.0

CREAL(X) - Gleitkommakonvertierung

Wandelt den Wert des Ausdrucks X in eine Gleitkommazahl um.

HEX$(X), BIN$ (X) - Hexadezimal-/Binär-Werte

Wandelt den Wert X in eine Zeichenkette um, welche die Hexadezimal-/Binär-Darstellung des Wertes enthält.

```
PRINT HEX$(23456),BIN$(23456)
5BA0          1011011100100000
```

ASC(S$),CHR$(N) - Zeichen/Ganzzahlkonvertierung

ASC wandelt das erste Zeichen von S$ in eine ganze Zahl um, während CHR$ die ganze Zahl N in ein Zeichen umwandelt. Die Umwandlung erfolgt nach dem ASCII-Code.

```
PRINT ASC("Zeichen"),ASC("Z")
 90             90
PRINT CHR$(12);:'Loescht den Bildschirm
PRINT CHR$(243);:'Pfeil nach rechts
```

VAL(S$) - Wert einer Zeichenkette

Die Funktion liefert den numerischen Wert der Zeichenkette. VAL betrachtet die Zeichenkette nur bis zum ersten Zeichen, das nicht +, -, &, E oder 0-9 ist.

Beispiele:

```
? VAL("123"),VAL("100 kg"),VAL("DM 500.-")
 123            100             0
Ready
? VAL("10000"),VAL("&10"),VAL("-9.81")
 10000          16             -9.81
Ready
```

STR$(X) - Ziffernumwandlung in Zeichenkette

Wandelt den Wert des Ausdrucks X in eine Zeichenkette um. Sie hat die umgekehrte Aufgabe wie die VAL-Funktion. Die erste Stelle der Zeichenkette enthält das Vorzeichen.

Beispiel:

```
10 A=15
20 A$=STR$(A)
30 PRINT "[";A;"] [";A$;"]

RUN
[ 15 ] [ 15]
Ready
```

Beachten Sie, daß die erste 15 eine Zahl ist, mit der gerechnet werden kann, die zweite 15 durch die STR-Funktion dagegen eine Zeichenkette geworden ist. Es ist

in obigem Beispiel daran zu erkennen, daß bei der Ausgabe der Zeichenkette A$ kein Leerzeichen angefügt wurde, wie dies bei der Ausgabe von Zahlenwerten geschieht, denn der zweiten 15 folgt sofort die "eckige Klammer zu".

Zeichenkettenfunktionen

LEN(S$) - Zeichenkettenlänge

Liefert die Länge einer Zeichenkette, wobei alle Leerstellen und Kontrollzeichen mitgezählt werden.

```
10 PRINT LEN("eins");" + ";LEN("zwei");" = ";
20 PRINT LEN("eins"+"zwei")

RUN
 4  +  4  =  8
Ready
```

LEFT (S$,n), RIGHT (S$,n)

Liefert die linken bzw. rechten Zeichen der Zeichenkette S$.

```
10 A$="RECHTS ist das Gegenteil von LINKS"
20 X$=LEFT$(A$,6)
30 Y$=RIGHT$(A$,5)
40 Z$=" ist das Gegenteil von "
50 PRINT A$
60 PRINT Y$;Z$;X$
70 END

RUN
RECHTS ist das Gegenteil von LINKS
LINKS ist das Gegenteil von RECHTS
Ready
```

MID$(S$,N (,M))

und

MID$(S$,N (,M)) = T$

Die erste Form dient dazu, eine Teilzeichenkette aus der Zeichenkette S$ zu entnehmen. Mit der zweiten Form ist es möglich, Teile einer Zeichenkette zu ersetzen. Dabei gibt N die Startposition der Teilzeichenkette und deren Länge an. Wird M nicht angegeben, werden alle rechts von N stehenden Zeichen kopiert.

Für die zweite Form gilt außerdem:

Ist M nicht angegeben, werden alle Zeichen von T$ kopiert. Es werden aber auf keinen Fall mehr als LEN(S$)-N + 1 Zeichen übertragen, die Länge von S$ ändert sich also nicht.

```
10 A$="Der Fach-Verlag ist in Darmstadt"
20 B$="Heim-"
30 PRINT A$
40 PRINT MID$(A$,5,5)
50 PRINT B$
60 MID$(A$,5)=B$
70 PRINT A$
RUN
Der Fach-Verlag ist in Darmstadt
Fach-
Heim-
Der Heim-Verlag ist in Darmstadt
Ready
```

STRING$(N,CODE%)

und

STRING$(N,C$) - Wiederholung von Zeichen

Liefert eine Zeichenkette, die das Zeichen mit dem ASCII-Code CODE% oder das erste Zeichen von C$ N-mal enthält.

```
10 PRINT " "STRING$(37,"=") '37 Gleichheitszeichen
20 a$=STRING$(13,"*")
30 PRINT " "a$;" Einladung ";a$

RUN
 =====================================
 ************ Einladung ************
Ready
```

INSTR((N,)X$,S$) - Suchen von Teilketten

Die Funktion liefert die Position der Zeichenkette S$ in S$, oder 0, wenn S$ nicht in X$ vorkommt. Gesucht wird ab Position N oder von Anfang an, falls N nicht angegeben wird.

```
5 k$=CHR$(34):lf$=CHR$(10)
6 CLS
10 A$="wird"
20 B$=" In diesem String wird gesucht "
30 N=INSTR(B$,A$)
40 IF N=0 THEN PRINT"NICHT GEFUNDEN":END
50 PRINT "Das Wort "lf$
60 PRINT k$;A$;k$;lf$
70 PRINT "wurde gefunden im String "lf$
80 PRINT k$;B$;k$;lf$
90 PRINT" an Position ";N;".";lf$
```

```
Ergebnis:     Das Wort

              "wird"

              wurde gefunden im String

              " In diesem String wird gesucht "

               an Position  19 .

              Ready
```

SPACE$(X) - Leerzeichenkette

Liefert eine Zeichenkette mit X Leerzeichen.

```
10 DATA B,A,S,I,C
20 FOR I=1 TO 5
30 READ A$(I)
40 PRINT SPACE$(I) A$(I)
50 NEXT
60 END

RUN
 B
  A
   S
    I
     C
Ready
```

Einführung

Alle diese Standardfunktionen, die Sie nun kennengelernt haben, besitzen zwei Gemeinsamkeiten:

1. Standardfunktionen stehen unter BASIC permanent zur Verfügung, d.h. sie sind fester Bestandteil des BASIC-Wortschatzes.

2. Standardfunktionen liefern als Ergebnis einer Berechnung einen einzigen Wert.

Trotz des umfangreichen BASIC-Wortschatzes des >> CPC 464 << gibt es Berechnungen, die nicht direkt über den Aufruf einer solchen Standardfunktion ausführbar sind. Hierzu ein konkretes Beispiel:

Ein Anwendungsfall

Häufig besteht der Wunsch, mehrere Bedienerhinweise in einem Programmablauf mittig bzw. zentriert auf dem Bildschirm auszugeben. Dieses ist über den Aufruf von nur einer der Standardfunktionen nicht direkt erreichbar. Das bedeutet:

Es müßte mehrmals und zwar an jeder Stelle des Programms, an denen Bedienerhinweise zentriert ausgegeben werden sollen, gleiche Funktionen programmiert werden. Die "Zentrierfunktion" hat dann folgendes Aussehen:

```
CINT((40-LEN(A$))/2)
```

und bestimmt die Anzahl der Leerzeichen vor der auszugebenden Zeichenkette. Die Variable A$ ist dabei die Zeichenkettenvariable, in der die zu zentrierende Zeichenkette steht. Der Wert 40 ist die Bildschirmbreite, d.h. für eine Ausgabe im MODE 2 ist hier eine 80 anzugeben. Die Zentrierfunktion setzt sich dabei aus mehreren Standardfunktionen zusammen.
Damit nicht bei jedem Aufruf die gesamte Zentrierfunktion geschrieben werden muß, besteht unter BASIC die Möglichkeit, dieser Funktion einen Namen zu geben und sie mit diesem "Kurznamen" zu schreiben und aufzurufen.

Diese selbst zu definierenden Funktionen werden "Anwender-" oder "Benutzerfunktionen" genannt.

Definition einer Benutzerfunktion

Definiert wird die Zentrierfunktion als Benutzer-Funktion durch die Anweisung:

```
DEF FNZ(A$)=CINT((40-LEN(A$))/2)
```

Erklärung:

Dem BASIC-Interpreter ist zuerst mitzuteilen, daß eine Benutzer-Funktion definiert werden soll. Dieses wird durch das Schlüsselwort

DEF

kenntlich gemacht.

Funktionsname

Die zweite Angabe ist der Name der Funktion. Zu beachten ist, daß der Name der Benutzer-Funktion grundsätzlich mit

FN

beginnen muß. Für den Rest des Namens gelten die Regeln für die Vergabe von Variablen-Namen. In diesem Fall soll die Funktion "Z" heißen, d.h. der Name entspricht der einer numerischen Variablen. Die numerische Variable enthält die Anzahl der Leerzeichen, die vor der Zeichenkette ausgegeben werden sollen. Die "Teil"-Definition lautet dann:

DEF FNZ oder DEF FN Z

Die zweite, deutlichere Schreibweise ist hierbei auch erlaubt.

Formales Argument und aktuelles Argument

Weiter muß ein Platzhalter hier "A$", der beim Aufruf der Funktion ersetzt werden soll, für das Argument angegeben werden. Der Platzhalter "A$" wird auch "formales Argument" genannt. Das formale Argument ist hierbei quasi eine "interne Variable" der Benutzer-Funktion.

Benannt wird das formale Argument in dem Beispiel mit A$, da eine Zeichenkette der Funktion zur Berechnung übergeben werden soll. Sie erhalten dann:

```
DEF FNZ(A$)
```

Die Formel, in der das formale Argument auftritt, ist nun für die Berechnung anzugeben ; Sie erhalten:

```
DEF FN Z(A$) = CINT((40-LEN(A$))/2)
```

Die Benutzer-Funktion "FN Z" ist hiermit eindeutig definiert. Sie kann im gesamten Programm mit "FNZ" aufgerufen werden.
An das formale Argument A$ wird beim Aufruf der Funktion der aktuelle Wert von WORT$ übergeben. In diesem Beispiel wurde die Variable WORT$ bezeichnet, d.h. aktuelles und formales Argument müssen vom gleichen Typ sein, aber nicht den gleichen Namen besitzen.
Außerdem bleibt A$ im Programm als Variable verfügbar, d.h. die formalen Argumente sind ebenso wie die Namen der Benutzer-Funktionen unabhängig von gleichlautenden Variablennamen.

Aufruf der Benutzerfunktion

Beispiel für einen Aufruf der definierten Zentrierfunktion:

```
10 DEF FN Z(A$)=CINT((40-LEN(A$))/2)
20 WORT$="Zeichen-Definition"
30 PRINT FN Z(WORT$)
40 END
```

Ergebnis: 11

* Das Programm gibt die Anzahl der Leerzeichen aus, die für eine Zentrierung der Zeichenkette "Zeichen-Definition" notwendig sind, damit diese zentriert auf dem Bildschirm erscheint.

Hinweis:

Diese definierte Funktion wird durch BASIC nicht "gesichert" und läßt sich nicht in anderen Programmen, sondern nur in dem Programm aufrufen, in dem sie definiert ist.

Hilfen für die Programmanalyse:

* Der Name dieser Funktion ist nur Z. Der Zusatz FN muß dem Namen der Funktion vorangestellt sein, damit BASIC erkennt, daß es sich nicht um eine Variable, sondern um eine Benutzer-Funktion handelt.

* In Zeile 20 wird der Variablen WORT$ die Zeichenkette "Zeichen-Definition" zugewiesen.

* Der Funktion Z wird der Wert von WORT$, der zur Ausführung der Berechnung nötig ist, als Argument übergeben. Als Ergebnis wird nur ein Wert zurückgegeben.

* Der Wert wird durch die PRINT-Anweisung in Zeile 30 ausgegeben. Er entspricht den Leerzeichen, die vor der Zeichenkette ausgegeben werden müssen, um diese zu zentrieren.

Nach diesen Vorüberlegungen finden Sie nun das vollständige Programm mit Ausgabe der zentrierten Zeichenkette:

Beispiel:

```
10 DEF FNZ(A$)=CINT((40-LEN(A$))/2)
20 TEXT$="Karteiprogramm"
30 PRINT " "STRING$(38,"-");
40 PRINT SPACE$(FNZ(TEXT$))TEXT$

RUN
 ----------------------------------------
                Karteiprogramm
Ready
```

Weitere Benutzerfunktionen

Weitere für Ihre Anwendungsfälle vordefinierte, mathematische Benutzer-Funktionen finden Sie nachstehend.

Für die Berechnung des Arcussinus:

```
DEF FN AS(X) = ATN(X/SQR(1-X*X))
```

Für die Berechnung des Arcustangens:

```
DEF FN AT(X) = -ATN(X/SQR(1-X*X))+PI
```

Für Berechnungen nach dem Satz des Pythagoras:

```
DEF FN P(X,Y) = SQR(X*X + Y*Y)
```

Für die Berechnungen nach dem Satz des Pythagoras sind zwei Angaben notwendig, so daß auch zwei Parameter angegeben werden müssen. Diese werden sowohl bei der Definition, als auch beim Aufruf der Funktion durch Kommata getrennt angegeben.

Regeln zur DEF FN-Anweisung

* DEF leitet eine Definitionsanweisung ein.

* FN kennzeichnet eine Benutzer-Funktion.

* DEF und FN müssen durch ein Leerzeichen getrennt sein.

* Der Name darf von der Funktionskennung FN mit einem Leerzeichen getrennt angegeben werden.

* Der Name der Benutzer-Funktion ist unabhängig von den Variablen in dem aufrufenden Programm.

* Die Definition muß vor dem ersten Aufruf der Benutzer-Funktion erfolgt sein.

* Der Typ des Funktionsnamens muß mit dem Typ des zurückgegebenen Werts übereinstimmen.

* Der Funktionsname muß ein zulässiger Variablenname sein.

* Eine definierte Benutzer-Funktion wird durch die NEW- und CLEAR-Anweisung und durch ein Ausschalten des Computers gelöscht.

* Eine Benutzer-Funktion besitzt nur in dem Programm Gültigkeit, in dem sie definiert wird.

* Eine Benutzer-Funktion darf nur aus Standard-Funktionen zusammengesetzt sein.

* Sollen mehrere Parameter an die Benutzer-Funktion übergeben werden, so müssen diese durch Kommata getrennt werden.

Hinweis:

Ruft man eine Funktion auf, bevor sie definiert worden ist, meldet der Computer: "Unknown user function".

Einführung

Die Definition von benutzereigenen Funktionen haben Sie anhand einiger nützlicher Programmbeispiele kennengelernt. Diese erworbenen Kenntnisse sollen Ihnen in diesem Abschnitt als Grundlage für die Neuerarbeitung der Unterprogrammtechnik dienen. Sie werden feststellen, daß die Unterprogrammtechnik sehr viele Gemeinsamkeiten mit den Benutzerfunktionen aufweist und Sie sehr schnell in der Lage sein werden, diese interessante Programmiertechnik selbst anzuwenden.
Damit Sie direkt in diese Materie einsteigen können, werden die Ergebnisse stichpunktartig wiederholt:

Wiederholung zur DEF FN-Anweisung:

* Die Funktionen, die im Programm an den unterschiedlichsten Stellen vorkommen, sollten mit einer DEF FN-Anweisung programmiert werden.

* Diese Funktion muß nur an einer einzigen Stelle im Programm definiert werden.

* Die Definition der Funktion muß vor dem ersten Aufrufen der Funktion erfolgen, d.h. im Programmaufbau ist darauf zu achten, daß diese am Programmanfang steht.

* Im Programmablauf kann die benutzereigene Funktion unter ihrem Namen mit Angabe der aktuellen Argumente aufgerufen werden.

Weiterhin wurden die Vorteile dieser Programmiertechnik vermittelt, die auch in der Unterprogrammtechnik zum Tragen kommen.

Die Vorteile der DEF FN-Anweisung

* Sie vermeidet Wiederholungen beim Programmieren, das heißt:

  1. Das Programm wird übersichtlicher.

  2. BASIC-Syntaxfehler treten nicht so häufig auf.

3. Die Zeit für das Eingeben des Programms verringert sich.

4. Es wird Platz im Arbeitsspeicher entsprechend der Länge der vorab definierten Funktion eingespart.

Nun kommt es in der Programmierpraxis häufig vor, daß bestimmte Programmteile im Programmablauf mehrfach benötigt werden. Es ist demnach eine ähnliche BASIC-Anweisung wie die DEF FN-Anweisung gefordert, nur daß es sich hierbei nicht um einen mehrmaligen Aufruf einer gleichen Funktion, sondern um den mehrmaligen und gezielten Aufruf eines ganzen Programmteils handelt.
Diese immer wieder neu an den entsprechenden Stellen gleichlautend einzufügen, ist sicher müßig. Zweitens nicht die beste Lösung, da es zu der Umkehrung der bei der DEF FN-Anweisung genannten Vorteile führen würde.

Anwendungsfälle

Denken Sie bitte einmal an folgende, vorgegebene Anwendungsfälle, die Sie mit einem BASIC-Programm bewältigen wollen:

Fall 1:

Es sollen mehrfach gleichgroße, farbige Kreise auf dem Monitor ausgegeben, wobei die Farbe und die Position der neuen Kreise gewechselt werden.

Fall 2:

Es soll innerhalb eines Programmablaufs mehrfach, d.h. in unterschiedlichen Programmblöcken, Anweisungen stehen, die einen gleichlautenden Text zur Bedienerführung ausgeben sollen und danach eine vom jeweiligen Programmblock abhängige Programmfortsetzung erfolgen.

Die Problematik ist, wie Sie auch festgestellt haben werden, daß

1. ein bestimmter Programmblock mit einer Folge von Anweisungen in einem Programm in der gleichen Form mehrmals benötigt wird.

2. der Programmblock an unterschiedlichen Programmstellen zwischen zwei Anweisungen verfügbar sein muß bzw. eingefügt werden kann.

3. der mehrmals benötigte Programmblock nur einmal im Programm vorhanden sein soll.

Die Sprachschöpfer von BASIC haben zur Lösung des Problems eine besondere Aufruftechnik für ganze Programmblöcke geschaffen.

Betrachten Sie hierzu die Programmskizze!

Programmskizze

```
10 '**************** Hauptprogramm ********************
20 Anweisung(a1):Anweisung(a2):GOSUB 1020:Anweisung(a3)
..
70 ..
80 Anweisung(b1):GOSUB 1020
90 Anweisung(c1)
100 END '---------- Hauptprogramm-Ende  ------
1000 '
1010 '*************** UNTERPROGRAMM ********************
1020 Anweisung(u1):Anweisung(u2)
1030 Anweisung(u3)
1040 RETURN '------ Unterprogramm-Ende --------
```

Die GOSUB- und RETURN-Anweisung

In der Programmskizze befinden sich zwei neue BASIC-Worte:

GOSUB und RETURN

GOSUB bedeutet:

Verzweige (GO) in das Unterprogramm (SUBroutine).

RETURN bedeutet:

Kehre zurück zu der unmittelbar auf die der GOSUB-Anweisung folgenden Anweisung.

Den in der Programskizze dargestellten Programmablauf sollten Sie nun mitverfolgen!

Programmablauf der Skizze

1. In Zeile 20 werden die Anweisungen (a1) und (a2) durchlaufen.

2. GOSUB 1020 verzweigt dann nach der Zeile mit der Nummer 1020.

3. Die Anweisungen (u1),(u2) und in Zeile 1030 die Anweisung (u3) des Unterprogramms werden nun abgearbeitet.

4. In Zeile 1040 befindet sich nun die Anweisung RETURN, die Auslöser für den Rücksprung in das Hauptprogramm ist.

5. Der Rücksprung erfolgt zu Anweisung(a3), die jetzt durchlaufen wird.

6. Das Hauptprogramm wird nun bis zur Anweisung(b1) abgearbeitet und durch die weitere GOSUB-Anweisung ein zweitesmal in das Unterprogramm verzweigt.

7. Anweisung (u1), (u2) und (u3) werden durchlaufen und in Zeile 1040, ausgelöst durch die RETURN-Anweisung, wieder in das Hauptprogramm zurückgesprungen.

8. Anweisung(c1) ist hierbei die nächste auszuführende Anweisung.

9. Nachdem Anweisung(c1) abgearbeitet ist, wird in Zeile 100 das Programm durch die END-Anweisung beendet.

Übungen

Prüfen Sie anhand der Programmskizze den Programmverlauf, wenn in Zeile 100 nicht die END-Anweisung programmiert worden wäre!

Überlegen Sie, warum die Anordnung von Unterprogrammen am Schluß des Programms sinnvoll ist.
Überdenken Sie, ob man in dieser Programmskizze eine oder auch mehrere GOTO-Anweisungen ersatzweise für die GOSUB-Anweisungen verwenden kann.

Ein Beispielprogramm beschreibt nun die Programmierung:

```
10 '--- Hauptprogramm -----------------
20 MODE 0
30 DATA 1,2,6,7,9,10
40 INK 0,0:CLS
50 I=2:Y=260
60 FOR X=170 TO 470 STEP 100
70 READ FARBE
80 INK I,FARBE
90 GOSUB 220
100 I=I+1
110 NEXT X
120 Y=140
130 FOR X=270 TO 370 STEP 100
140 READ FARBE
150 INK I,FARBE
160 GOSUB 220
170 I=I+1
180 NEXT X
190 END
200 '
210 '--- Unterprogramm ----------------
220 FOR A=1 TO 44
230 PLOT X,Y
240 DRAW X+80*COS(A),Y+80*SIN(A),I
250 NEXT
260 RETURN
```

Lesen Sie bitte zuerst die Kurzinformationen zu den neuen, im Programm angewendeten BASIC-Anweisungen und versuchen Sie, danach den Programmablauf zu bestimmen.

Die CLS-, PLOT-, DRAW- und INK-Anweisung

Kurzinformationen zu den neuen BASIC-Anweisungen:

* CLS steht für "CLEAR SCREEN" und löscht alle Zeichen auf dem Monitor. Wird die Angabe CLS#<Datenstrom> vorgegeben, so färbt CLS das entsprechende Bildschirmfenster ein.

* PLOT positioniert den nicht auf dem Monitor sichtbaren Graphik-Cursor und setzt einen Pixel (Graphikpunkt). Die Werte für die X- und Y-Koordinaten sind bezogen auf die Pixel-Koordinaten.

* DRAW zeichnet eine Linie von der aktuellen Graphik-Cursorposition zu dem durch X- und Y-Koordinate bestimmten Punkt.
  Die Koordinatenangaben beziehen sich auf die Pixel-Koordinaten. Die untere linke Ecke des Bildschirmfensters besitzt die Koordinaten 0,0, die rechte obere Ecke die Koordinaten 639,399.

* INK wird benutzt, um die Farbe von PAPER bzw. PEN zu wechseln. Die erste Codezahl bezeichnet den bei PAPER bzw. PEN verwendeten Code. Der an zweiter Stelle anzugebende Farbcode bestimmt die neue Farbe, die PEN bzw. PAPER annimmt.

Die Bildschirmausgabe wird Ihnen aufzeigen, daß Sie mit Ihren Überlegungen zum Programmablauf recht haben und der vorab genannte erste Beispielfall: "mehrfach gleichgroße, farbige Kreise" programmiert wurde. Das Hauptprogramm und das mit der RETURN-Anweisung abgeschlossene Unterprogramm sind recht einfach bestimmbar. Sie erkennen zudem, daß die Anordnung der beiden GOSUB-Anweisungen in diesem Programm ähnlich der in der Programmskizze ist.

Hilfen für Ihre Programmanalyse:

* In Zeile 30 werden Konstanten in einer DATA-Anweisung für die Farbe der Kreise bereitgehalten.

* In Zeile 40 wird die Hintergrundfarbe des Bildschirms mit der INK-Anweisung und der CLS-Anweisung auf Schwarz eingefärbt.

* In Zeile 50 wird die Y-Koordinate der oberen vier Graphiken festgelegt und der Farbcode wird mit I=2 auf zwei voreingestellt.

* Die FOR NEXT-Schleife in Zeile 60 bestimmt die X-Koordinaten der Mittelpunkte der Graphiken.

* In Zeile 70 wird mit der READ-Anweisung die FARBE der ersten Graphik eingelesen.

* Die Ink-Anweisung in Zeile 80 ordnet dem Farbcode I die eingelesene FARBE zu.

* Mit der GOSUB-Anweisung in Zeile 90 wird zum Unterprogramm, das in Zeile 220 beginnt, gesprungen.

* In Zeile 100 wird der Farbcode I für die nächste Graphik um eins erhöht, es gilt I=I+1.

* Das Schleifenende befindet sich in Zeile 110.

* In Zeile 120 wird die Y-Koordinate für die unteren beiden Graphiken mit Y=140 festgelegt.

* Die Zeilen von 130 bis 180 ähneln den Zeilen 60 bis 110, es werden aber für die Graphiken andere X-Werte verwendet.

* Zeile 160 enthält die zweite GOSUB-Anweisung, d.h. das Unterprogramm wird zum zweitenmal aufgerufen.

* Zeile 190 beendet das Hauptprogramm mit der END-Anweisung.

* Die FOR NEXT-Schleife des Unterprogramms gibt den Winkel im Bogenmaß vor, den die DRAW-Anweisung nutzt.

* Durch PLOT X,Y wird der Ursprung für die DRAW-Anweisung auf die Koordinate X,Y festgelegt. Dieses entspricht dem Mittelpunkt der Graphiken.

* Die DRAW-Anweisung zieht Linien auf der Basis der Berechnungen in der Farbe, die dem Farbcode I entspricht.

* Die FOR NEXT-Schleife des Unterprogramms wird mit der NEXT-Anweisung in Zeile 250 abgeschlossen.

* Die RETURN-Anweisung in Zeile 260 verursacht den Rücksprung zum Hauptprogramm.

Zusammenfassung:

* Mit der GOSUB-Anweisung und einer zusätzlichen Angabe der Startzeile des Unterprogramms kann in Unterprogramme gesprungen werden.

* Die angegebene Startzeile muß vorhanden sein und sollte keine REM-Zeile sein.

* Unterprogramme müssen mit der Anweisung RETURN abgeschlossen sein.

* Die RETURN-Anweisung löst den Rücksprung aus einem Unterprogramm zum Hauptprogramm aus.

* Zurückgesprungen wird zu derjenigen Anweisung, die nach der GOSUB-Anweisung steht, von der aus in das Unterprogramm verzweigt wurde.

* Auch bei mehreren GOSUB-Anweisungen in einem Hauptprogramm "merkt" sich BASIC die Absprungstelle.

* Das Hauptprogramm muß mit einer END-Anweisung abgeschlossen werden, wenn ein Unterprogramm ohne vorherige GOSUB-Anweisung durchlaufen werden kann.

* Unterprogramme sollten am Ende des Hauptprogramms angeordnet werden, da hierdurch die Übersichtlichkeit gewahrt bleibt und das Unterprogramm nicht über verkomplizierende Programmiertechniken umgangen werden muß.

Die Schachtelung von Unterprogrammen

Wenn man von einem Unterprogramm in ein weiteres Unterprogramm springt, "schachelt" man gewissermaßen die Unterprogramme. Den Ausdruck "Schachtelung" kennen Sie bereits von den "Geschachtelten FOR NEXT-Schleifen". Bei der Unterprogrammtechnik wird im Mikrocomputer die Schachtelungstiefe durch einen Stapelspeicher bestimmt.

Der >> CPC 464 << besitzt eine Schachtelungstiefe von 84, d.h. es können maximal 84 geschachtelte Unterprogrammaufrufe ausgeführt werden. Danach erscheint dann die Fehlermeldung

"Memory full in <Zeilennummer>".

19. Berechnete Sprünge mit der ON GOTO-Anweisung

Einführung

Das Treffen von Entscheidungen gehört zu den wichtigsten Forderungen an einen Computer. Deshalb hat man in BASIC einige Anweisungsformen entwickelt, die den Computer in den Stand versetzen, Entscheidungen zu treffen.

BASIC verfügt über Anweisungen, die vergleichend Entscheidungen im Programmablauf und hierdurch vorprogrammierte Sachentscheidungen zulassen. Sie kennen bislang die IF THEN-Anweisung, mit der Sie Programmentscheidungen abgefragt haben. Sie basierte auf dem Erfülltsein oder Nichterfülltsein bestimmter, vorgegebener Bedingungen.

Bei dem Anweisungstyp, den Sie jetzt erarbeiten, ist es im wesentlichen nicht anders. Auch hier wird verglichen und in Abhängigkeit vom Vergleichsergebnis reagiert.
Nur ist die Reaktion auf den Vergleich eine Verzweigung von mehreren möglichen Verzweigungen.

Die ON GOTO-Anweisung

Der letzte Teil der Anweisung ist Ihnen schon bekannt. Sie wissen, daß GOTO <Zeilennummer> unbedingt in die hinter GOTO stehende Zeile mit der angegebenen Zeilennummer hin verzweigt. Es wird von daher bei der GOTO-Anweisung auch von einem "unbedingten Sprung" gesprochen.

Die ON GOTO-Anweisung hat, allgemeingültig betrachtet - man nennt es in BASIC-Handbüchern das "FORMAT" der Anweisung - folgendes Aussehen:

ON <Ausdruck> GOTO <Liste von Zeilennummern>

Sie erkennen, daß hier nicht nur eine Zeilennummer erlaubt ist, sondern eine von Ihnen zu bestimmende Anzahl. Diese Zeilennummern stehen, durch Kommata getrennt, in einer Liste. Die GOTO-Anweisung kannte nur eine Zeilennummer, zu der BASIC hinverzweigte.

19. Berechnete Sprünge mit der ON GOTO-Anweisung

Die ON GOTO-Anweisung unterscheidet hier mehrere Verzweigungsmöglichkeiten, so daß über ein vorher berechnetes Ergebnis ein gezielter Sprung in eine entsprechende Zeile bzw. Anweisung erfolgt.

Hier wird dann von "berechneten Sprüngen" gesprochen. Die Sprungzeile ist demnach abhängig von einem Ergebnis oder einem bestimmten Wert des <Ausdrucks>.

Das BASIC-Schlüsselwort der ON GOTO-Anweisung ist "ON". Wird es eingelesen, so interpretiert BASIC es so:

1. Sofortige Berechnung des <Ausdrucks>, der auf ON folgt.

2. Das Ergebnis ist hierbei immer eine Integerzahl (ganze Zahl).

3. Diese Integerzahl betimmt eindeutig die Position in der Liste, die dem zweiten Schlüsselwort GOTO folgt.

4. Eine Programmverzweigung erfolgt zu derjenigen Anweisungsnummer, die in der Liste der Zeilennummern auf der betreffenden Position steht.

Beispiel:

Die Anweisung lautet: ON I GOTO 100,250,160,80,600,1560,1900

Erklärung:

Der Variablen "I" werden nachfolgend in einer Tabelle Werte zugewiesen, anschließend die Position in der <Liste der Zeilennummern> angegeben und die Sprungzeile hierzu in Beziehung gesetzt:

I	Position	Verzweigung nach
6	6	Zeilennummer 1560
1	1	Zeilennummer 100
4	4	Zeilennummer 80
3	3	Zeilennummer 160
5	5	Zeilennummer 600
7	7	Zeilennummer 1900

Die Entscheidung der Programmverzweigung ist demnach hier durch den gesetzten Wert der Variable I bestimmt. Sie erkennen, daß hier nur ganze positive Zahlen für I angegeben wurden. Positive Zahlen müssen es sein, dagegen sind aber ganze Zahlen (Integerzahlen) nicht zwingend vorgegeben, es erfolgt ansonsten eine Auf- bzw. Abrundung der Zahl.

Zusammenfassung zur ON GOTO-Anweisung:

* Ausführung eines berechneten Sprungs zu einer Programmzeile.

* Der Sprung erfolgt in Abhängigkeit zum Wert des numerischen Ausdrucks.

* Die Angabe des Wertes erfolgt nach dem Schlüsselwort "ON".

* Der numerische Ausdruck darf nur positive Werte zwischen 0 und 255 annehmen.

* Wird der numerische Ausdruck Null oder größer als die Anzahl der Zeilennummern in der Liste, so wird zur nächsten Anweisung gesprungen.

* Die Angabe der Zeilennummern erfolgt getrennt mit Kommata in einer nach GOTO angelegten Liste.

* Die angegebenen Zeilennummern müssen existieren.

Programmbeispiel

```
10 'Programmbeispiel zu ON GOTO
20 '-----------------------------------
30 '
40 I=I+1
50 '-----------------------------------
60 ON I GOTO 100,160,120,180,80,140,200
70 '-----------------------------------
80 PRINT"eigene"
90 GOTO 40
100 PRINT"Auch ";
110 GOTO 40
120 PRINT"lieben ";
130 GOTO 40
140 PRINT"Schoenheit!";
150 GOTO 40
160 PRINT"Stachelschweine ";
170 GOTO 40
180 PRINT"ihre ";
190 GOTO 40
200 END
```

Das ausgegebene Ergebnis vermittelt eine "Lebensweisheit", die Sie sicher noch nicht kannten.

Das Beispielprogramm ist aber ganz bewußt auf die Erläuterung der berechneten Sprunganweisung ausgelegt, ohne unnötig zusätzliche, mathematische Formeln. Betrachten Sie bitte allein die Wirkungsweise der ON GOTO-Anweisung.

Analysieren Sie nun das Programm und berücksichtigen Sie dabei insbesondere die Anweisung Nr. 60, die "Berechnete-Sprung-Anweisung"!

Hilfen für Ihre Analyse:

* Die Anweisungen Nr. 10, 20, 30, 50 und 70 enthalten nur Kommentare und haben auf den Programmablauf keinen Einfluß.

* In der Anweisung Nr. 40 wird der Zähler I um 1 hochgezählt. Die Anweisung wird im Verlaufe der Programmausführung von sechs verschiedenen Anweisungen angesprungen, und zwar von den Anweisungen 90, 110, 130, 150, 170 und 190.

* Der Zähler I hat bei Beginn des Programms den Wert Null. Wird die Anweisung 40 zum ersten Mal durchlaufen, erhält I den Wert 1, da 0 + 1 = 1 ist.

* Mit dem Anfangswert 1 wird die Anweisung Nr. 60 erreicht.

* Im ersten Durchlauf besitzt I den Wert 1.

* In der ersten Position der Liste der Zeilennummern steht die Zeilennummer 100, demnach wird zuerst zur Zeile 100 verzweigt und ausgeführt.

* In Anweisung 110 wird zur Anweisung Nr. 40 verzweigt; dort wird der Zähler um 1 erhöht.

* Wird die Anweisung Nr. 60 erreicht, so hat I den Wert 2. Es wird zu der auf der 2. Position in der Liste stehenden Anweisung mit der Nr. 160 verzweigt. Auf dem Bildschirm wird das Wort < Stachelschweine > ausgegeben, und zwar hinter dem dort schon stehenden Wort "Auch".

* Diesmal hat I den Wert 3, deshalb wird die Anweisung angesprungen, die auf der 3. Position in der Liste steht: Es ist die Anweisung 120. Hinzugefügt wird das Wort < lieben >.

In der beschriebenen Weise wird fortgefahren, bis auf dem Bildschirm der Satz

```
Auch Stachelschweine lieben ihre eigene Schoenheit!
```

ausgegeben wurde.

Die Variable besitzt dann den Wert 6. Von der Anweisung Nr. 150 aus wird zur Anweisung Nr. 40 verzweigt und I um 1 auf 7 erhöht.

* In der Anweisung Nr. 60 wird damit in der Liste der 7. Platz erreicht.

* Die Verzweigung zur Anweisung Nr. 200 wird ausgeführt. Der Programmdurchlauf ist damit abgeschlossen und das Programmende erreicht.

Sie haben den reinen Programmablauf überprüft und dabei sicherlich erkannt, daß die Zielanweisungen, zu denen hinverzweigt werden soll, in der Liste in beliebiger, also unsortierter Reihenfolge stehen können.

Für den ordnungsgemäßen Programmablauf ist allein der Wert der Variablen I und die Plazierung der anzuspringenden Zeilennummer innerhalb der Liste der Zeilennummern ausschlaggebend.

Anwendungsfall: "Währungsumrechnung"

Die ON GOTO-Anweisung wird in der Regel nicht für eine Ausgabe von "Lebensweisheiten", sondern für Auswahlmenüs in Programmen genutzt. Ein solcher Anwendungsfall wird im folgenden Beispielprogramm aufgegriffen.

Programmerkmale

Für das Programmbeispiel wird ein Devisenumrechnungsprogramm projektiert.

19. Berechnete Sprünge mit der ON GOTO-Anweisung

Der Leistungsumfang wird vorgegeben mit:

1. Verarbeitung von 9 Landeswährungen
2. Umrechnung von DM in Fremdwährung
3. Umrechnung von Fremdwährung in DM
4. Umrechnung zwischen Fremdwährungen
5. Fest vorgegebene Wechselkurse
6. Menü- und Dialog-Technik bei Abfragen
7. Routinen für die Plausibilitätsprüfung
8. Ausgabe aller Devisensorten über ein Menü
9. Auswahl der Devisensorte im Dialog
10. Ausgabe des Wechselkurses bezogen auf die Einheit 100
11. Formatierte Ausgabe der berechneten Beträge mit der Währungseinheit
12. Ausgang zum Kommandomodus (Betriebssystem)

Das Programm: "Währungsumrechnung"

Prüfen Sie bitte zuerst den Programmablauf und vergleichen Sie anhand des Programmlistings die Programmblöcke mit den unterschiedlichen Zeilen, die mit ON GOTO angesprungen werden.

Überlegen Sie weiter, wann das Unterprogramm "Sortenauswahl" aufgerufen wird und welche GOSUB-Anweisung in welchen Zeilen dieses bewirkt.

19. Berechnete Sprünge mit der ON GOTO-Anweisung

```
10 'Programm zur Waehrungsumrechnung
20 '-----------------------------------------------------
30 MODE 1
40 DATA US-Dollar,35.3,$,engl Pfund,24.510,"£"
50 DATA schw. Franken,82.474,sfr.,ital. Lire,59880.2,Lire
60 DATA oesterr. Schillinge,697.350,OeS
70 DATA fr. Francs,298.507,FF,span. Pesetas,5263.1,Pta
80 DATA holl. Gulden,111.111,hfl.,belg. Francs,2020.2,bfrs
90 FOR I=1 TO 9
100 READ WAEHRUNG$(I),KURS(I),EINHEIT$(I)
110 NEXT I
120 A$="########.##"
130 '
200 '------- MENUE 1 mit ON GOTO-Anweisung ----------------
210 CLS
220 PRINT "  *** Programm: Waehrungsumrechnung ***":PRINT
230 PRINT "M E N U E":PRINT
240 PRINT STRING$(40,154)
250 PRINT "Umrechnen von ":PRINT
260 PRINT "DM in Fremdwaehrung ..... (1)":PRINT
270 PRINT "Fremdwaehrung in DM ..... (2)":PRINT
280 PRINT "Fremdwaehrung in Fremdwaehrung (3)":PRINT
290 PRINT "Programmende ............. (4)":PRINT
300 PRINT:INPUT"Bitte Kennziffer eingeben  -->  ",KENNZ
310 IF KENNZ<1 OR KENNZ>4 THEN 210
320 ON KENNZ GOTO 410,610,810,1120
330 '
400 '------------- DM in Fremdwaehrung --------------------
410 CLS:GOSUB 1010
420 CLS:LOCATE 1,3
430 PRINT"Programmteil: Umrechnen in Fremdwaehrung"
440 PRINT STRING$(40,154)
450 PRINT"DM in "WAEHRUNG$(KENNZ):PRINT
460 PRINT"Wechselkurs: 100.00 DM =";
470 PRINT USING A$;KURS(KENNZ);:PRINT"  "EINHEIT$(KENNZ);
480 LOCATE 2,12:INPUT "Wieviel DM: ",MENGE
490 FREMD=MENGE*KURS(KENNZ)/100
500 LOCATE 1,15:PRINT"Sie erhalten fuer :"
```

```
510 PRINT:PRINT"DM";USING A$;MENGE;
520 PRINT"  --> ";USING A$;FREMD;:PRINT" "EINHEIT$(KENNZ)
530 PRINT STRING$(40,154):LOCATE 5,24
540 INPUT "Weiter --> ENTER druecken!",ENTER$
550 GOTO 210
560 '
600 '-------- Fremdwaehrung in DM ------------------------
610 CLS:GOSUB 1010
620 CLS:LOCATE 1,3
630 PRINT"Programmteil: Umrechnen in DM"
640 PRINT STRING$(40,154)
650 PRINT WAEHRUNG$(KENNZ);" in DM":PRINT
660 PRINT"Wechselkurs: 100.00 ";EINHEIT$(KENNZ);" = ";
670 PRINT USING a$;10000/KURS(KENNZ);:PRINT" DM";
680 LOCATE 2,12
690 PRINT"Wieviel "EINHEIT$(KENNZ);:INPUT MENGE
700 DM=MENGE*100/KURS(KENNZ)
710 LOCATE 1,15:PRINT"Sie erhalten fuer :"
720 PRINT:PRINT EINHEIT$(KENNZ);USING A$;MENGE;
730 PRINT" --> ";USING A$;DM;:PRINT"  DM"
740 GOTO 530
750 '
800 '------ Umrechnen zwischen Fremdwaehrungen -----------
810 CLS:LOCATE 5,18:PRINT"umrechnen von:"
820 GOSUB 1010:KENNZ1=KENNZ:CLS
830 LOCATE 5,18:PRINT"umrechnen in":GOSUB 1010
840 KENNZ2=KENNZ:CLS
850 PRINT"Programmteil: Fremdwaehrungswechsel";
860 PRINT STRING$(40,154)
870 PRINT WAEHRUNG$(KENNZ1);" in "WAEHRUNG$(KENNZ2)
880 PRINT:PRINT"Wechselkurs: 100.00 ";EINHEIT$(KENNZ1);
890 KURS=KURS(KENNZ2)*100/KURS(KENNZ1)
900 PRINT " ="USING A$;KURS;:PRINT" "EINHEIT$(KENNZ2);
910 LOCATE 1,15
920 PRINT"Wieviel "EINHEIT$(KENNZ1);:INPUT MENGE
930 ENDWERT=KURS*MENGE/100
940 LOCATE 1,15:PRINT"Sie erhalten fuer :"
950 LOCATE 1,17
```

```
960 PRINT EINHEIT$(KENNZ1);USING A$;MENGE;
970 PRINT" --> ";USING A$;ENDWERT;:PRINT" ";EINHEIT$(KENNZ2)
980 GOTO 530
990 '
1000 '------- Unterprogramm Sortenauswahl ------------------
1010 LOCATE 5,3:PRINT"Programmteil: Sortenauswahl":PRINT
1020 PRINT STRING$(40,154)
1030 FOR I=1 TO 9
1040 X=21-LEN(WAEHRUNG$(I))
1050 PRINT TAB(5);WAEHRUNG$(I)+" "+STRING$(X,".")+"  =";I
1060 NEXT I
1070 LOCATE 1,22:PRINT SPACE$(39):LOCATE 1,22
1080 INPUT"Bitte Kennziffer eingeben --> ",KENNZ
1090 IF KENNZ<1 OR KENNZ>9 THEN 1070
1100 RETURN
1110 '------- Programmende --------------------------------
1120 CLS:PRINT"Programmende":END
```

Hilfen für Ihre Programmanalyse:

* In den Zeilen 200 bis 320 befinden sich die Anweisungen für die Ausgabe des Hauptmenüs. Mit der ON GOTO-Anweisung wird in Abhängigkeit zur eingegebenen Kennziffer der einzelne Programmblock angesprungen:

* Programmblöcke:
  - DM in Fremdwährung
  - Fremdwährung in DM
  - Umrechnen zwischen Fremdwährungen

Zusätzlich kann der Ausgang zum BASIC-Kommandomodus über die Kennziffer 4 ausgewählt werden.

* Das Unterprogramm "Sortenauswahl" wird mit den GOSUB-Anweisungen in den Zeilen 410, 610, 820 und 830 aufgerufen.

* Im Programmblock "Umrechnen zwischen Fremdwährungen" wird das Unterprogramm "Sortenauswahl" zweimal aufgerufen, da zwei Devisensorten einzulesen sind.

20. Berechnete Sprünge mit der ON GOSUB-Anweisung

Der Begriff "berechneter Sprung" wurde im letzten Kapitel anhand der ON GOTO-Anweisung vorgestellt, die eine berechnete Programmverzweigung in eine entsprechende Programmzeile aus der Liste der Programmzeilen ausführt. Fast gleich in der Wirkweise ist die Anweisung

ON GOSUB

mit dem Format:

ON <numerischer Ausdruck> GOSUB <Liste mit Zeilennummern>

Sie erkennen, daß beide Formate fast identisch sind. Einziger sichtbarer Unterschied ist das BASIC-Wort GOSUB, das hier anstelle von GOTO in der ON GOSUB-Anweisung steht.
Die BASIC-Anweisung GOSUB findet sich im letzten Programmbeispiel für einen Unterprogrammaufruf. Sie liefert den Wirkunterschied zu der ON GOTO-Anweisung. Sie erinnern sich:

* Mit der GOSUB-Anweisung und einer zusätzlichen Angabe der Startzeile des Unterprogramms kann in Unterprogramme gesprungen werden.

* Unterprogramme müssen mit der RETURN-Anweisung abgeschlossen sein.

* Die RETURN-Anweisung löst den Rücksprung aus einem Unterprogramm zum Hauptprogramm aus.

* Zurückgesprungen wird zu derjenigen Anweisung, die nach der GOSUB-Anweisung steht, von der aus in das Unterprogramm verzweigt wurde.

* Das Hauptprogramm muß mit einer END-Anweisung abgeschlossen werden, wenn ein Unterprogramm ohne vorherige GOSUB-Anweisung durchlaufen werden kann.

* Unterprogramme sollten am Ende des Hauptprogramms angeordnet werden, da hierdurch die Übersichtlichkeit gewahrt bleibt und das Unterprogramm nicht über verkomplizierende Programmiertechniken umgangen werden muß.

20. Berechnete Sprünge mit der ON GOSUB-Anweisung

Aus der Kombination beider Anweisungen ist die Wirkung der ON GOSUB-Anweisung ableitbar. Sie bewirkt das gleiche, mit der Ausnahme, daß nicht berechnet in eine Programmzeile, sondern berechnet in ein Unterprogramm gesprungen wird. Die Startzeilen der aufzurufenden Unterprogramme sind wie bei der ON GOTO-Anweisung in der Liste der Zeilennummern durch Kommata getrennt anzugegeben. Das Unterprogramm muß außerdem wie bei der GOSUB-Anweisung mit RETURN abgeschlossen werden.

21. Benutzereigene Zeichen definieren

Einführung

Ein feature des >> CPC 464 << ist es, alle nur denkbaren Zeichen zu kreieren, zu definieren und mit der SYMBOL-Anweisung auf einen bestimmten ASCII-Wert zu legen.

Konkret:

> Allen 255 vorhandenen ASCII-Werten können neue Zeichen zugeordnet werden. Das bedeutet, daß Sie ohne große Probleme jeden Länderzeichensatz selbst definieren und auf dem >> CPC 464 << nutzen können.

Das Kapitel wurde bewußt "Projekt" genannt, da es Ihnen ein fertiges, lauffähiges Dienstprogramm "Zeichen definieren" anbietet. Sie können mit dem Programm im Dialog z. B. einen deutschen Zeichensatz erstellen und die Tastatur so anpassen, daß eine deutsche DIN-Tastatur, wie Sie sie von der Schreibmaschine her kennen, entsteht. Das Dienstprogramm bietet die Voraussetzung für die optimale Nutzung der Projekte "Text" und "Kartei".

Zeichen zu definieren heißt, mit Bit und Byte umgehen. Die Einführung in "gesetzte" und "nichtgesetzte Bit" -Folgen wird direkt auf das Projekt bezogen, um Sie nicht mit der "grauen Theorie" zu belasten.

Bit und Byte

Dezimalzahlen- und Binärzahlen-System

Das "Dezimalzahlen-System" ist für Sie eine bekannte Größe. Sie rechnen im Alltag meist in und mit diesem Zehner-System.
Ein anderes Zahlen-System ist das "Binär-System", es kennt nur zwei Ziffern 0 und 1. Ihr >> CPC 464 << arbeitet intern mit diesem System.

Die Einheit heißt Bit und ist die Abkürzung für "BInary digiT".

Rechnen mit Dezimalzahlen und Binärzahlen

Betrachten Sie zunächst einmal die Zahl 6846 im Dezimal-System. Die angegebene Zahl kann nach den mathematischen Regeln von links nach rechts zerlegt werden in:

```
Tausender --> 6   entspricht  6 * 1000 = 6000
Hunderter --> 8   entspricht  8 *  100 =  800
Zehner    --> 4   entspricht  4 *   10 =   40
Einer     --> 6   entspricht  6 *    1 =    6
```

Werden die Produkte addiert, so erhält man als Summe die Zahl 6846.

Nach den mathematischen Regeln läßt sich hierfür schreiben:

```
6 * 1000 =  6 * 10 * 10 * 10
8 *  100 =  8 * 10 * 10
4 *   10 =  4 * 10
6 *    1 =  6
```

Die Mathematik erlaubt dann noch die kürzere Potenzschreibweise. Dabei ist z.B.

$$10 * 10 * 10 = 10^3$$

Festgelegt wurde zusätzlich, daß

$$10^0 = 1$$

ist.

Es kann nun für die Zahl

$$6846 = 6 * 10^3 + 8 * 10^2 + 4 * 10^1 + 6 * 10^0$$

geschrieben werden.

Nicht anders wird im Binär-System gerechnet, außer daß die Basis im Gegensatz zur 10 im Dezimal-System 2 ist, d.h. jede Stelle oder Bit im Binärsystem entspricht einer Zweierpotenz. Die Bits sind von rechts nach links von 0 bis 7 durchnummeriert.

Das erste Bit (Stelle) eines Bytes wird mit Bit "0" bezeichnet und das letzte Bit mit "7". Bit 0 ist dabei ganz rechts und Bit 7 ganz links bei einer Binärzahl zu lesen.

Stellen einer Binärzahl:

```
Binärzahl:   1 0 1 0 1 1 1 0

     Bit:   7 6 5 4 3 2 1 0
```

Bit 0 entspricht hierbei 2^0 und Bit 7 der Wert 2^7.

Betrachten Sie nun die Binärzahl 10011, die entsprechend von links nach rechts zerlegt und in eine Dezimalzahl umgerechnet wird.

Umwandlung einer Binärzahl in eine Dezimalzahl:

$1 * 2^4 = 1 * 2 * 2 * 2 * 2 = 16$

$0 * 2^3 = 0 * 2 * 2 * 2 = 0$

$0 * 2^2 = 0 * 2 * 2 = 0$

$1 * 2^1 = 1 * 2 = 2$

$1 * 2^0 = 1 * 1 = 1$

Die Umrechnung der Binärzahl in eine Dezimalzahl ergibt den Wert von:

$$16 + 0 + 0 + 2 + 1 = 19$$

Wissen müssen Sie weiter, daß der Computer intern immer Blöcke von 8 Bits verarbeitet, die ein Byte genannt werden.

Das Binär-System besitzt, wie vorher gesagt, nur die Ziffern 0 und 1. Auch ein Bit kann nur diese beiden Werte annehmen. Steht es auf 1, so bezeichnet man es als "gesetztes Bit". Einzelne Bits in einem Byte können durch Eingabe von Dezimalzahlen gesetzt werden.
Dieses ist notwendig, da Binärzahlen in BASIC nicht direkt eingegeben werden können. Sollen einzelne Bits eines Bytes gesetzt werden, so ist wie folgt vorzugehen:

Beispiel:

Bit	Berechnung	Dezimal-Wert
0	$1 * 1 = 1$	1
1	$2 * 1 = 2^1$	2
2	$2 * 2 = 2^2$	4
3	$2 * 2 * 2 = 2^3$	8
4	$2 * 2 * 2 * 2 = 2^4$	16
5	$2 * 2 * 2 * 2 * 2 = 2^5$	32
6	$2 * 2 * 2 * 2 * 2 * 2 = 2^6$	64
7	$2 * 2 * 2 * 2 * 2 * 2 * 2 = 2^7$	128

Werden Bit 1, Bit 3, Bit 5 und Bit 6 in einem Byte gesetzt, so ist die Dezimalzahl 106 anzugeben.

Die Berechnung der Zahl erfolgt nach dem Schema:

$$2 + 8 + 32 + 64 = 106$$

21. Benutzereigene Zeichen definieren

Pixel und Zeichen

Die einzelnen Zeichen bestehen aus 8 Byte. Ein Byte stellt eine Zeichenzeile dar. Ist in einem Byte ein Bit gesetzt, so erscheint ein Bildpunkt. Sind z.B. alle Bits des letzten Byte gesetzt, so ist dieses der Unterstreichungsstrich. Das letzte Byte stellt somit die unterste der acht Zeichenzeilen dar. Für einen Buchstaben stehen 8 x 8 Pixel (Bildpunkte) zur freien Gestaltung neuer Zeichen oder Symbole zur Verfügung. Werden sie alle gesetzt, so ergibt sich ein Karosymbol.

Beispiel:

Es soll der Großbuchstabe "F" definiert werden.

Die SYMBOL-Anweisung

Zeichenmatrix für die SYMBOL-Anweisung:

	7 6 5 4 3 2 1 0 Bit	dezimaler Wert des Bytes
1. Byte	X X X X X X X	127
2. Byte	X X X	49
3. Byte	X X X	52
4. Byte	X X X X	60
5. Byte	X X X	52
6. Byte	X X	48
7. Byte	X X X X	120
8. Byte		0
	7 6 5 4 3 2 1 0 Bit	

In der Tabelle sind die gesetzten Bits mit "X" gekennzeichet, d.h. für ein gesetztes Bit erscheint auf dem Monitor ein Pixel.

Die Anweisung SYMBOL erlaubt über die Angabe von 8 Dezimalwerten für die entsprechenden Bytes der 8 Zeichenzeilen eine Neudefinition der Zeichen.

Die SYMBOL-Anweisung lautet für den auszugebenden Buchstaben "F":

```
SYMBOL 163,127,49,52,60,52,48,120,0
```

Die letzten acht Werte in der Liste sind die ermittelten Dezimalzahlen der 8 Bytes, die die Zeichenzeilen festlegen. Der erste Wert ist der ASCII-Wert des Zeichens, welches neu definiert wird und danach mit CHR$(<ASCII-Wert>) aufgerufen werden kann.

Geben Sie das Beispielprogramm ein:

```
10 MODE 0
20 SYMBOL AFTER 37
30 SYMBOL 37,127,49,52,60,52,48,120,0
40 PRINT
50 PRINT CHR$(37),"F"
60 PRINT
```

Starten Sie dieses Programm nun, betrachten Sie das Ergebnis und vergleichen Sie das erste ausgegebene Zeichen mit der oben vorgestellten Zeichenmatrix.

Mit dem Beispielprogramm wird das neudefinierte Zeichen "F" auf den ASCII-Wert 37 des Prozentzeichens (%) gelegt.
Für Ihren Vergleich wurde ein Zeichen gewählt, das Sie selbst überprüfen können. Prüfen Sie nun die beiden ausgegebenen Zeichen und vergleichen Sie diese. Sie erkennen, daß das neudefinierte Zeichen rechtsbündig ist, das andere im Zeichensatz vorhandene aber linksbündig steht. Nicht linksbündig, da bit 7 in allen Bytes nicht gesetzt wird.

Drücken Sie nun SHIFT- mit der Prozenttaste. Auch im Direktmodus bleibt dieses neudefinierte Zeichen erhalten, bis eine weitere SYMBOL AFTER-Anweisung alle eigendefinierten Zeichen wieder löscht.

Die SYMBOL AFTER-Anweisung

Kurzinformation zur SYMBOL AFTER-Anweisung

* Die SYMBOL AFTER-Anweisung gestattet eine Umdefinition aller Zeichen ab dem angegebenen ASCII-Werte bis zum maximalen Wert von 255.

* Darunterliegende Werte können nicht mit SYMBOL umdefiniert werden. Nach dem Einschalten des >> CPC 464 << liegt dieser <ASCII-Wert> bei 240.

* Eine SYMBOL AFTER-Anweisung löscht vorher selbst definierte Zeichen!

Die erworbenen Kenntnisse werden Ihnen einerseits helfen, Zeichen ohne das vorliegende Dienstprogramm zu kreieren, andererseits werden Sie das Programm besser für Ihre Anwendungsfälle auswerten können.

Es stellt ein Hilfsmittel für die Neudefinition von Zeichen dar, ohne größere Überlegungen und mathematische Berechnungen durchführen zu müssen.

Programm: "Zeichendefinition"

Programmlisting

```
10 '---------- Projekt: Zeichen-Definition ---------------
20 ON ERROR GOTO 830' Fehlerbehandlung
30 '---------- Puffer fuer Zeichen ab CHR$(32) festlegen --
40 SYMBOL AFTER 32
50 '---------- Variablendefinition ----------------------
60 DEFINT X,Y,Z
70 DIM A(8,8),Z(8)
80 '--------- Bildschirmaufbau --------------------------
90 MODE 1
100 PAPER 0:INK 0,1
110 PEN 1:INK 1,24
120 CLS:INK 3,0:BORDER 14
130 '--------- Fenster fuer Zeichenmatrix ----------------
140 WINDOW #1,11,18,11,18
150 PAPER #1,2:INK 2,14
160 CLS #1:PAPER #1,0
170 GOSUB 850' --> Matrix-Raster zeichnen
180 '--------- Ueberschrift ------------------------------
190 LOCATE 10,3:PRINT"Zeichen-Definition"
200 LOCATE 25,8:PRINT"altes Zeichen
210 LOCATE 2,5
220 PRINT"Zeichen komplett --> ENTER druecken!"
230 LOCATE 12,8:PRINT"MATRIX"
240 '--------- Umrandung des Zeichens --------------------
250 LOCATE 29,13:PRINT CHR$(150)CHR$(154)CHR$(156)
260 LOCATE 29,14:PRINT CHR$(149)CHR$(32)CHR$(149)
270 LOCATE 29,15:PRINT CHR$(147)CHR$(154)CHR$(153)
280 '--------- Achsen beschriften ------------------------
290 LOCATE 10,9:PRINT"Y"
300 FOR Y=1 TO 8
310 LOCATE 9,19-Y:PRINT STR$(Y)
320 NEXT Y
330 LOCATE 20,19:PRINT"X"
340 FOR X=1 TO 8
350 LOCATE 10+X,19:PRINT RIGHT$(STR$(X),1)
360 NEXT X
370 '- Eingabe des ASCII-Werts des zu aendernden Zeichens -
380 GOSUB 730
```

```
390 LINE INPUT"Welchen ASCII-Wert aendern? ";WERT$
400 WERT=VAL(WERT$)
410 IF WERT <32 OR WERT>255 THEN 380
420 '--------- Ausgabe des Zeichens ----------------------
430 PAPER 3
440 LOCATE 30,14:PRINT CHR$(WERT)
450 PAPER 0
460 '--------- Eingabe der Koordinaten -------------------
470 GOSUB 730
480 LINE INPUT "X-WERT = ";X$
490 IF LEN(X$)=0 THEN 950
500 X=VAL(X$)
510 IF X<1 OR X>8 THEN GOSUB 770:GOTO 430
520 GOSUB 730
530 LINE INPUT "Y-WERT = ";Y$
540 IF LEN(Y$)=0 THEN 950
550 Y=VAL(Y$)
560 GOSUB 730
570 Y=9-Y
580 IF Y<1 OR Y>8 THEN GOSUB 770:GOTO 520
590 LOCATE 25,8:PRINT"neues Zeichen
600 '--------- Punkt auf Matrix setzen --------------------
610 IF A(X,Y)=1 THEN A(X,Y)=-1 ELSE A(X,Y)=1
620 LOCATE #1,X,Y
630 IF A(X,Y)=1 THEN PRINT #1,CHR$(32);:GOTO 650
640 PAPER #1,2:PRINT #1,CHR$(32);:PAPER #1,0
650 GOSUB 850' --> Gitter wiederherstellen
660 '--------- Auswertung der Eingabe ---------------------
670 Z(Y)=Z(Y)+A(X,Y)*2^(8-X)
680 '--------- Zeichen neu definieren ---------------------
690 SYMBOL WERT,Z(1),Z(2),Z(3),Z(4),Z(5),Z(6),Z(7),Z(8)
700 GOTO 430
710 '--------- Unterprogramme -----------------------------
720 '--------- Zeile unten loeschen -----------------------
730 LOCATE 2,24:PRINT STRING$(39,32);
740 LOCATE 2,24
750 RETURN
```

```
760 '--------- Fehlermeldung ausgeben --------------------
770 GOSUB 730
780 PRINT "*** fehlerhafte Eingabe ****
790 FOR i=1 TO 400:NEXT i
800 RETURN
810 '
820 '-------- Fehlerbehandlung --------------------------
830 GOSUB 770:RESUME 430
840 '-------- Matrix-Raster zeichnen --------------------
850 FOR i=113 TO 254 STEP 16
860 MOVE 161,i
870 DRAWR 125,0
880 NEXT i
890 FOR I=161 TO 289 STEP 16
900 MOVE I,113
910 DRAWR 0,127
920 NEXT I
930 RETURN
940 '-------- Abfrage auf Eingabeende -------------------
950 GOSUB 730
960 INPUT "Zeichen fertig (J/N)";ANTW$
970 ANTW$=UPPER$(ANTW$)
980 IF ANTW$<>"J" THEN 430
990 GOSUB 730
1000 INPUT "Weiteres Zeichen definieren (J/N)";ANTW$
1010 ANTW$=UPPER$(ANTW$)
1020 IF ANTW$<>"N" THEN ERASE A,Z:GOTO 70
1030 '---- Programm beenden -----------
1040 LOCATE 1,21
1050 PRINT" Zeichenkette mit dem"
1060 PRINT" neu definierten Symbol:"
1070 GOSUB 730
1080 PRINT STRING$(38,WERT)
1090 LOCATE 1,1
1100 END'Programmende
```

Sie erkennen anhand der REM-Zeilen im Programmlisting, daß dieses Dienstprogramm blockartig aufgebaut ist.

21. Benutzereigene Zeichen definieren

Programmblöcke

* Puffer festlegen
* Variablendefinition
* Bildschirmaufbau
* Eingabe des zu ändernden Zeichens
* Ausgabe des zu ändernden Zeichens
* Eingabe der Koordinaten
* Graphische Darstellung der Eingabe
* Auswertung und Neudefinition
* Hilfsroutinen als Unterprogramme

Programmblock: Puffer festlegen

* Der "aktive" Teil dieses Programmblocks ist die Anweisung in Zeile 70. Die REM-Zeilen, die durch einen Apostroph gekennzeichnet sind, beeinflussen den Programmablauf nicht.
  Mit der Anweisung SYMBOL AFTER 32 ein Puffer für 224 Zeichen eingerichtet. Sie ermöglicht die Neudefinition sämtlicher Zeichen ab dem ASCII-Wert 32.
  Ohne Pufferfestlegung können nur die Zeichen ab dem ASCII-Wert 240 umdefiniert werden.

Programmblock: Variablendefinition

* Ein Zeichen besteht aus 8 x 8 Pixeln, die entweder gesetzt sein können oder nicht. Halbe Punkte existieren nicht. Die Koordinaten der Punkte werden im Programm mit X und Y bezeichnet. Es sind nur ganzzahlige Werte möglich. Die Variablen X und Y werden deshalb auch als Integervariablen definiert.
  Für das zu bearbeitende Zeichen wird ein Feld benötigt. In dem Feld wird geprüft, ob ein Pixel bereits gesetzt ist oder nicht. Das Feld ist entsprechend der Anzahl der Pixel eines Zeichens dimensioniert.
  Mit der DIM-Anweisung wird ein Feld von 8 * 8 Elementen und ein Vektor Z mit 8 Elementen dimensioniert. Der Vektor wird später für die Anweisung SYMBOL benötigt.

Programmblock: Bildschirmaufbau

* Der Bildschirm wird über die MODE 1 -Anweisung auf 40 Zeichen pro Zeile gesetzt. Der Vorteil ist, daß die 8 * 8 Zeichen-Matrix annähernd einem Quadrat gleicht. Die Zeichen und die Hintergrundfarbe werden durch die INK-Anweisungen in die Grundeinstellung "eingelöscht". Die Zeichenfarbe wird hellgelb und die Hintergrundfarbe blau vorgegeben.

Teil: Fenster für Zeichenmatrix

* Ein Bildschirmfenster wird mit der WINDOW-Anweisung für die Zeichen-Matrix definiert und mit der PAPER- INK- und CLS-Anweisung in die Farbe pastellblau eingefärbt. Die GOSUB-Anweisung ruft das Unterprogramm "Matrix-Raster zeichnen" auf.

Teil: Überschrift

* Die Überschriften und Bedienerhinweise werden mit PRINT-Anweisungen auf dem Bildschirm ausgegeben. Mit der LOCATE-Anweisung werden diese an der gewünschten Stelle auf dem Bildschirm plaziert.

Teil: Umrandung

* Mit Graphikzeichen wird ein Feld umrahmt, in das später das Symbol zur optischen Prüfung gesetzt wird.

Teil: Achsen beschriften

* Ist das Matrix-Raster gezeichnet, werden die einzelnen Zeilen und Spalten beschriftet.
  Die STR$-Anweisung gibt die Ziffern als Zeichenkette aus, ohne hierbei ein Leerzeichen an die Ziffer anzufügen, damit der fertige Bildschirmaufbau nicht überschrieben wird.
  Das Leerzeichen vor positiven Zahlen wird für die Beschriftung in der X-Richtung aus dem gleichen Grund ebenfalls entfernt.

  Mit der RIGHT$-Anweisung wird nur das letzte Zeichen, die Ziffer in der mit STR$ erzeugten Zeichenkette, ausgegeben.
  Abschließend werden die Achsen als Eingabehilfe beschriftet.

Programmblock: Eingabe des ASCII-Werts

* Der ASCII-Wert des zu ändernden Zeichens ist einzugeben. Alle Bedienereingaben werden in diesem Programm in der vorletzten Bildschirmzeile abgefragt. Hierzu wird die Eingabezeile gelöscht und der Cursor plaziert.
  Die LINE INPUT-Anweisung liest alle Eingaben, auch die Koordinateneingaben, als Zeichenkette ein. Die VAL-Anweisung wandelt die Koordinateneingabe und den eingegebenen ASCII-Wert wieder in einen numerischen Wert um.
  Versehentliche Fehleingaben, die zu der Fehlermeldung:

  "?Redo from start"

  führen, werden hiermit abgefangen.
  Die ON ERROR GOTO-Anweisung wurde programmiert, um auch den Punkt als falsche Eingabe auszuschließen.
  Der eingegebene ASCII-Wert wird auf Plausibilität geprüft, da Werte kleiner 32 nach der SYMBOL AFTER-Anweisung nicht erlaubt sind.

Programmblock: Ausgabe des Zeichens

* Das dem ASCII-Wert entsprechende Zeichen wird in das umrandete Feld gesetzt.

Programmblock: Eingabe der Koordinaten

* Die eingegebenen Koordinaten des Pixels auf der Matrix werden wie bei der Eingabe des ASCII-Werts ausgewertet. Zusätzlich wird mit der Anweisung

  IF LEN (X$)=0 THEN ...

  geprüft, ob nur die ENTER-Taste betätigt wurde. Trifft dieses zu, so wird zur Abfrage auf Eingabeende gesprungen.
  Mit der Anweisung

  IF X<1 OR X>8 THEN GOSUB ...

wird geprüft, ob der eingegebene Wert im zulässigen Bereich von 1 bis 8 liegt. Liegt die Eingabe nicht in diesem Bereich, so wird sie wiederholt und zur entsprechenden Eingabezeile gesprungen.
Nach der ersten Eingabe der zweiten Koordinate wird der Text "altes Zeichen" durch den Text "neues Zeichen" auf dem Bildschirm ersetzt.

Programmblock: Punkt auf Matrix setzen

* Ein bereits gesetzter Punkt kann wieder gelöscht werden. Es wird mit der Anweisung

```
IF A(X,Y)=1 THEN A(X,Y)=-1 ...
```

abgefragt, ob die eingegebene Koordinate bereits als Pixel definiert wurde. Ist bereits ein Pixel definiert, so wird er gelöscht.
Der Cursor wird mit der LOCATE-Anweisung auf die angesprochene Koordinate gesetzt. Hier wird entweder das Matrix-Karo ausgefüllt oder in der Hintergrundfarbe gelöscht.
Wird ein Karo stellvertretend für einen Pixel gesetzt, so wird das Matrix-Raster teilweise gelöscht. Es wird zum Unterprogramm "Matrix-Raster zeichnen" gesprungen, um dieses wieder zu vervollständigen.

Programmblock: Auswertung der Eingabe

* Die Parameter für die SYMBOL-Anweisung werden hier berechnet. Die ermittelte Potenz entspricht hierbei dem gesetzten Bit (Stelle). Vergleichen Sie hierzu Bit und Byte!

Teil: Zeichen neu definieren

* Die SYMBOL-Anweisung definiert das neue Zeichen. Die Parameterangabe erfolgt berechnet über den numerischen Wert der 8 notwendigen Byte, die immer eine Zeichenzeile bestimmen.

21. Benutzereigene Zeichen definieren

Unterprogramme

Teil: Zeile unten löschen

* Dieses Unterprogramm löscht die alte Eingabe, die Eingabezeile und plaziert den Cursor an die Eingabeposition.

Teil: Fehlermeldung ausgeben

* Tritt ein Fehler in der Bedienung auf, so wird eine Fehlermeldung in der Eingabezeile ausgegeben. Benutzt wird hierzu das Unterprogramm "Zeile unten löschen".

Teil: Fehlerbehandlung

* Die ON ERROR GOTO-Fehlerbehandlungsroutine benutzt das Unterprogramm "Fehlermeldung ausgeben". Mit der Anweisung RESUME 430 wird der Programmablauf nach der Fehlerbehandlung fortgesetzt.

Teil: Abfrage auf Eingabeende

* Mit der Abfrage "Zeichen fertig (J/N)" soll der Abschluß der Neudefinition eines Zeichens bestätigt werden. Wird die Abfrage mit J für Ja eingegeben, so kann anschließend ein weiteres Zeichen neu definiert werden.
  Die Anweisung UPPER$ wandelt die eingegebenen Kleinbuchstaben in Großbuchstaben um. Dieses vereinfacht die vorangehende Prüfung vor einer Programmverzweigung.

Teil: Matrix-Raster zeichnen

* Mit der DRAWR-Anweisung wird ein Raster auf die Zeichen-Matrix gelegt. Optisch werden hiermit die 8 x 8 Pixel eines Zeichens nachgebildet, um die Eingabe der Koordinaten zu vereinfachen.

Die DRAWR-Anweisung

Kurzinformation zur DRAWR-Anweisung

* Die DRAWR-Anweisung zeichnet relativ zur aktuellen Graphik-Cursorposition eine Linie.
* Der Startpunkt der Linie wird bestimmt durch die aktuelle Graphik-Cursorposition.
* Die X-Koordinate des Zielpunktes wird durch die X-Koordinate der aktuellen Graphik-Cursorposition plus der relativen Koordinaten bestimmt. Die Y-Koordinate berechnet sich entsprechend.

Die MOVE-Anweisung

Kurzinformation zur MOVE-Anweisung

* MOVE setzt den Graphik-Cursor auf die angegebenen X- und Y-Koordinaten.

22. Textverarbeitung mit dem >> CPC 464 <<

Einführung

Textverarbeitung wird heute ganz groß geschrieben. Sie finden hierzu auf dem Software-Markt eine reichhaltige Palette von unterschiedlich leistungsstarken Programmen.

In einer Reihe von Fachzeitschriften wurden im letzten Halbjahr 1984 über fünfzig Textsysteme vorgestellt, getestet, angeboten, deren Preis bei wenigen hundert DM begannen und irgendwo bei viertausend DM "endeten".

Es stellen sich die Fragen:

* Muß ein Textsystem so teuer sein?

* Reicht ein preiswertes System nicht auch?

Es kommt immer darauf an, welche Ansprüche Sie stellen. Das Projekt "Text" bietet Ihnen für den >> CPC 464 << ein kostenloses Textsystem. Die Einsatzmöglichkeiten sind natürlich beschränkt.

Es reicht aber vollkommen für Ihre Anwendungsfälle aus, wenn Sie nur einfache Standard-Briefe, Formulare oder Rechnungen schreiben, auf Kassette aufbewahren und von der Kassette für zukünftige Zwecke wieder in den Arbeitsspeicher des >> CPC 464 << laden wollen.

Stellen Sie höhere Ansprüche und wollen Sie vielleicht ein Buch schreiben, dann sollten Sie sich selbstverständlich auch ein leistungsfähigeres Textverarbeitungs-System anschaffen.

Die Grundidee zum Projekt "Text"

Die Grundidee, die diesem Programm zugrunde liegt, stammt aus dem "Großen alphaTronic BASIC-Buch" von B. Bollow und N. Bollow Die Einfachheit ist überzeugend und die Leistungsfähigkeit bei sich wiederholenden Texten, in denen wenige Änderungen vorzunehmen sind, ist beeindruckend.

Grundidee:

BASIC-Programme, die ASCII-gespeichert sind, können als sequentielle Dateien aufgefaßt werden. Die so abgespeicherten Programme können eingelesen, verändert und wieder gesichert werden.

Ein Programm muß aber nicht zwangsläufig BASIC-Anweisungen enthalten, sondern darf auch nur aus Kommentarzeilen bestehen. Wie Sie bereits wissen, enthalten Kommentarzeilen beliebige Texte, die aber auch Briefzeilen oder Formularzeilen sein können.

Besteht ein Programm nur aus Kommentar- bzw. Briefzeilen, so wird es im weiteren Text als "Briefzeilenprogramm" bezeichnet. Wird ein Briefzeilenprogramm direkt mit der LIST #8-Anweisung auf dem Drucker ausgegeben, so würden die Zeilennummern und der Apostroph stören und das gelistete Briefzeilenprogramm wäre noch kein echter Brief.

Das Briefzeilenprogramm muß auf der Kassette in ASCII-abgespeichert werden, damit die einzelnen Kommentarzeilen mit den Briefzeilen als Zeichenketten aufgefaßt und nacheinander einzeln mit der LINE INPUT #9-Anweisung von der Kassette in Zeichenkettenvariablen eingelesen werden können.
Die eingelesenen Zeichenketten können nun mit Stringfunktionen zerlegt und die störenden Zeilennummern und der Apostroph entfernt werden. Die übrigbleibenden Zeichenketten entsprechen danach den eigentlichen Briefzeilen, die über ein geeignetes Programm mit der PRINT #8-Anweisung ausgedruckt werden können.

Der Einlesevorgang von der Kassette wird bis zum Textende wiederholt. Geprüft wird das Text- bzw. Briefzeilenprogramm mit der EOF-Anweisung.

Nach diesem Schema reiht sich Briefzeile an Briefzeile und der gewünschte Text wird vollständig ausgedruckt.
Die Bearbeitung der Texte soll mit dem deutschen Zeichensatz erfolgen und als Zusatz eine Einfügefunktion vorhanden sein.

Programmblöcke des Projekts "Text"

Aus der Grundideebeschreibung können notwendige Programmblöcke und -teile abgeleitet und ihre Reihenfolge bestimmt werden:

1. Definition des deutschen Zeichensatzes
2. Schreiben von Texten
3. Ablegen des Briefzeilenprogramms
4. Einlesen des Briefzeilenprogramms
5. Entfernen der Zeilennummern und des Apostrophs
6. Einfügefunktion
7. Druckerausgabe

Definition des deutschen Zeichensatzes

Im Kapitel "Projekt: Benutzereigene Zeichen definieren" ist die Anwendung der beiden BASIC-Anweisungen SYMBOL und SYMBOL AFTER vorgestellt worden. Vorausgesetzt werden hier die dort erworbenen Kenntnisse. Die einfache Art, eigene Zeichen zu definieren, ist leider nicht direkt auf die Tastatur übertragbar. Es entstünde eine Vertauschung der gewünschten und unbedingt für die Druckausgabe vom Zeichensatz des Druckers benötigten ASCII-Werte.
Die Ausgabe entspricht dann nicht dem deutschen Zeichensatz des Druckers. Es muß zunächst mit der KEY DEF-Anweisung eine Umdefinition der Tasten auf einen anderen ASCII-Wert erfolgen und danach den ASCII-Werten die neu definierten Zeichen Ä, ä, Ö, ö, Ü, ü und ß zugeordnet werden.

Die KEY DEF-Anweisung

Kurzinformation zur KEY DEF-Anweisung

Format:

KEY DEF <Tastencode>,<repeat>,<ASCII-Wert>
[,<ASCII-Wert>,<ASCII-Wert>]

* Die KEY DEF-Anweisung definiert einzelne Tasten in das vom Benutzer gewünschte Zeichen um.

* Die neu zu belegende Taste wird im Tastencode, das gewünschte Zeichen wird mit seinem ASCII-Wert angegeben.

* Wird für <repeat> eine Null angegeben, so ist die Wiederholfunktion der Taste ausgeschaltet. Ist sie erwünscht, so ist anstelle der Null eine Eins zu setzen.

* Es sind insgesamt drei verschiedene ASCII-Wert Angaben möglich. Der erste ASCII-Wert wird ausgegeben, wenn die Taste allein gedrückt wird, der zweite, wenn sie in Kombination mit der SHIFT-Taste und der dritte angegebene ASCII-Wert, wenn die Tastenkombination CTRL plus der Taste mit dem vorgegebenen <Tastencode> gleichzeitig gedrückt wird.

Die Tastenbelegung soll der einer deutschen Schreibmaschinentastatur gleichen, d.h. die Tastenbelegung ist:

Tabelle der Umbelegung

Zeichen		Taste mit		Tastencode	ASCII-Wert der Taste alter	neuer
Ä	-->	SHIFT	plus	17	(123)	91
ä	-->			17	(91)	123
Ö	-->	SHIFT	plus	26	(124)	92
ö	-->			26	(64)	124
Ü	-->	SHIFT	plus	19	(125)	93
ü	-->			19	(93)	125
ß	-->			24	(94)	126

Dienstprogramm: "Änderung auf deutschen Zeichensatz"

Für die schnelle und unkomplizierte Umbelegung der Tasten und für die Neudefinition der Zeichen steht Ihnen das nachstehende Dienstprogramm zur Verfügung.

```
5 'Änderungsprogramm für deutschen Zeichensatz
10 KEY DEF 24,1,126
20 KEY DEF 26,1,124,92
30 KEY DEF 17,1,123,91
40 KEY DEF 19,1,125,93
50 SYMBOL AFTER 91
60 SYMBOL 91,195,60,102,102,126,102,102,0
65 SYMBOL 92,195,60,102,102,102,102,60,0
70 SYMBOL 93,102,0,102,102,102,102,60,0
80 SYMBOL 123,108,0,120,12,124,204,118,0
90 SYMBOL 124,102,0,60,102,102,102,60,0
100 SYMBOL 125,102,0,102,102,102,102,63,0
110 SYMBOL 126,60,102,102,108,102,102,124,96
```

Laden Sie dieses Programm von der Programmkassette ein und starten Sie es mit der RUN-Anweisung. Der Zeichensatz ist dann implementiert. Das Dienstprogramm kann anschließend mit der NEW-Anweisung gelöscht werden. Die neue Belegung bleibt bis zum nächsten Ausschalten des Computers oder bis zur nächsten SYMBOL AFTER-Anweisung erhalten.
Das erste Teil-Programm zum "Projekt: Text" ist hiermit abgeschlossen.

Schreiben von Texten

Das Schreiben der Textzeilen ist dem Schreiben einer Kommentarzeile in einem Programm gleich, d.h. die Textzeilen werden ohne zusätzlichen Programmteil direkt eingegeben.

Beispiel:
```
10 'Sehr geehrte Damen und Herren,
20 '
30 'Ihr Angebot vom 30.10.1984 haben wir mit Datum usw.
```

Die drei Kommentarzeilen stellen den Anfang eines Anwortbriefes dar und können beliebig erweitert werden. Der Umfang ist nur durch den Platz im Arbeitsspeicher begrenzt.

Mit der BASIC-Anweisung AUTO können Sie die benötigten Zeilennummern automatisch generieren, so daß lediglich der Apostroph vor die Textzeile gesetzt werden muß.
Ist eine Textzeile beendet, so ist sie mit der ENTER-Taste abzuschließen. Sollten in abgeschlossenen Textzeilen Fehler gefunden werden, so sind sie mit der EDIT-Anweisung oder mit der COPY-Taste als Programmzeile zu editieren.
Ist der Text geschrieben, so sollte zur Kontrolle mit der RUN-Anweisung geprüft werden, ob ein Apostroph in einer Zeile vergessen wurde.
Ein fehlender Apostroph wird vom Interpreter als Syntaxfehler gemeldet und die entsprechende Zeile auf dem Monitor ausgegeben, so daß sie direkt korrigiert werden kann.

Vorgehensweise bei der Texteingabe

* Aufrufen der automatischen Zeilenerzeugung im Direktmodus mit AUTO

* Schreiben der einzelnen Textzeilen mit einem Apostroph am Anfang!

* Textzeilen mit ENTER-Taste abschließen!

* Prüfen der Zeilen mit der RUN-Anweisung

Ablegen des Briefzeilenprogramms

Das Briefzeilenprogramm, Ihr Text, kann nach der Prüfung auf der Datenkassette abgelegt werden.

Die Anweisung hierzu ist SAVE und heißt "retten" oder "sichern". Dahinter ist in Hochkommata der Name des Programms anzugeben. Der Name darf maximal sechzehn Buchstaben lang sein. Am Schluß der Anweisung steht noch ein Komma mit nachfolgendem A. Durch diese Angabe wird das Programm im ASCII-Modus und nicht binär auf der Kassette abgelegt. ASCII-Speicherung ist hier unbedingt erforderlich!

Bevor Sie Ihren Text mit der SAVE-Anweisung sichern, sollten Sie eine Daten-Compact-Kassette einlegen, da sonst nachfolgende Programme auf der Programm-Kassette überschrieben werden!

Geben Sie hierzu im Kommandomodus ein:

SAVE "<Textname>",A

vergessen Sie aber nicht den Kennbuchstaben "A" für ASCII-Abspeicherung und lesen Sie vorher das Zählwerk an Ihrem Datacorder ab, damit Sie sich ein Inhaltsverzeichnis Ihrer Datenkassetten erstellen können.

Einlesen des Briefzeilenprogramms

Das Briefzeilenprogramm kann nicht durch einfaches Programmladen und Starten als Brief gedruckt werden. Die Kommentarzeilen werden dann zwar durchlaufen, aber nicht ausgedruckt, da keine ausführbare BASIC-Anweisung vorhanden ist. Außerdem sollen die Zeilennummern, das Kennzeichen der Kommentarzeilen, der Apostroph, entfernt und die Markierungszeichen gegen individuelle Eingaben ersetzt und danach der Text gedruckt werden.

Hierzu wird ein BASIC-Programm benötigt, das die Funktionen

* Einlesen der abgelegten Programmzeilen
* Prüfen auf Textende
* Suchen des Apostrophs und Entfernen einschließlich der Zeilennummer
* Einfügen individueller Texte an markierten Positionen
* Einstellen des linken Blattrandes für den Druck
* Drucken der Textzeilen

beinhaltet.

Alle die genannten Funktionen wurden kompakt in dem nachstehenden kleinen BASIC-Programm verwirklicht, das Sie aber auch noch individuell erweitern können.

Programmlisting

```
10 '************************************************************
20 '*          Projekt:   TEXT                                 *
30 '************************************************************
40 '
50 MODE 2:BORDER 0:WINDOW #1,1,80,10,24
60 LOCATE 28,6:PRINT"Projekt TEXT"
70 LOCATE 26,7:PRINT STRING$(16,"-")
80 LOCATE 7,10:PRINT "Bitte legen Sie die Kassette mit Ihrem T
ext in den  DATACORDER  ein":LOCATE 7,11
90 PRINT "und spulen Sie diese an die entsprechende Stelle!"
100 LOCATE 7,13
110 INPUT "Drücken Sie danach die ENTER-Taste. ",enter$
120 CLS #1:LOCATE 7,10
130 PRINT "Geben Sie bitte den Textnamen ein!"
140 LOCATE 7,12:INPUT "Textname ";F$
150 CLS #1:LOCATE 7,10
160 INPUT "Linker Rand auf Spalte .. ",SPALTE
170 IF SPALTE <1 OR SPALTE >60 THEN 150
180 CLS #1:LOCATE 7,10
190 PRINT "Drücken Sie nun die Taste 'PLAY' auf Ihrem DATACORD
ER"
200 LOCATE 7,12:PRINT "und danach die ";
210 INPUT "ENTER-Taste.",ENTER$
220 FOR I=0 TO 6:READ X$(I):NEXT I
230 DATA "Tagesdatum     : ","Anrede         : "
240 DATA "Rechnungsbetrag: ","Rechnungsdatum : "
250 DATA "Sorte          : ","Liefertermin   : "
260 DATA "Zahlungstermin : "
270 CLS #1:L=0:OPENIN "!"+F$
280 LINE INPUT #9,A$:'Zeile aus Text einlesen
290 B=INSTR(A$,"'"):A$=MID$(A$,B+1)
300 Z=INSTR(A$,"***"):'Feststellen, ob Zeile Textlücken hat
310 IF Z=0 THEN 360:'Keine Textlücke;   Zeile ausdrucken
320 PRINT #1,X$(L);:L=L+1
330 LINE INPUT #1,B$
340 A$=LEFT$(A$,Z-1)+B$+MID$(A$,Z+3)
350 GOTO 300:'Noch weitere Textlücken in derselben Zeile?
```

```
360 PRINT #8,TAB(SPALTE);A$
370 IF EOF THEN 390
380 GOTO 280'Nächste Textzeile bearbeiten
390 CLOSEIN
400 CLS #1:PRINT #8,CHR$(12);
410 INPUT #1," Weiterer Text J/N";ANTW$
420 IF UPPER$(ANTW$)="J" THEN CLS #1:RESTORE:GOTO 80
430 MODE 1
440 END' ****              Programmende                    *****
```

Einige neue BASIC-Anweisungen finden Sie in dem Programmlisting, die in Kurzinformationen erläutert werden sollen.

Einlesen der abgespeicherten Programmzeilen

In Zeile 270 finden Sie die OPENIN-Anweisung:

```
270 ...  OPENIN "!"+F$
```

Die OPENIN-Anweisung

Kurzinformation zur OPENIN-Anweisung

* OPENIN eröffnet einen Datenfile, legt einen Datenpuffer von 2 k an und liest die abgespeicherten Daten als Datenblöcke von Kassette mit dem laufenden Programm ein.

* Falls der erste Buchstabe des Dateinamens ein Ausrufezeichen "!" ist, werden keine Anweisungen zur Bedienung des Datacorders auf dem Monitor ausgegeben.

Die nachstehende Variable F$ übergibt im konkreten Fall den Textnamen an die OPENIN-Anweisung.

Prüfen auf Textende

Wird von der Kassette eine Datei, in diesem Fall eine Textdatei, eingelesen, so muß das Dateiende abgefragt werden.

Die entsprechende Anweisung hierfür steht in Zeile 370:

```
370 IF EOF THEN 390
```

Die EOF-Anweisung

Kurzinformation zur EOF-Anweisung:

* Mit EOF wird das Dateiende abgefragt. Wird das Dateiende erreicht, so ergibt diese Funktion den Wert -1 (wahr).

* EOF muß vor jeder erneuten LINE INPUT #9-Anweisung abgefragt werden, da sonst bei Dateiende die Fehlermeldung "EOF met in <Zeilennummer>" ausgegeben wird und ein Programmabbruch erfolgt.

Wenn das Dateiende erreicht ist, wird mit der IF THEN-Anweisung zur Zeile 390 verzweigt. Ist das Textende erreicht, so muß die eröffnete Datei mit der CLOSEIN-Anweisung in Zeile 390 wieder geschlossen werden.

Die CLOSEIN-Anweisung

Kurzinformation zur CLOSEIN-Anweisung:

* Die CLOSEIN-Anweisung schließt einen Eingabefile vom Datacorder, der vorab mit OPENIN eröffnet wurde.

* Ohne vorherige CLOSEIN-Anweisung kann kein neuer Eingabefile mit OPENIN eröffnet werden.

* Außerdem schließt die END-Anweisung alle eröffneten Files.

Die weiteren Programmfunktionen "Apostroph suchen", "Ersetzen von Markierungszeichen gegen individuelle Texte" werden über Ihnen bereits bekannte Stringfunktionen verwirklicht.

Wiederholung Zeichenkettenfunktionen

Die INSTR-Anweisung

Mit dieser Anweisung können bestimmte Zeichen oder Zeichenfolgen in einer Zeichenkette, einem String, gesucht werden.

INSTR(X$,Y$)

Der Suchbegriff, also die gesuchte Zeichenfolge ist in Y$ enthalten. Im String X$ wird die Zeichenfolge Y$ gesucht.

X$ und ebenso Y$ können String-Variablen, String-Konstanten oder String-Ausdrücke sein. Die Funktion INSTR liefert das Ergebnis Null, wenn Y$ nicht gefunden wird oder die Länge von X$ gleich Null ist.

Die MID$-Anweisung

Hiermit kann ein Teil einer Zeichenkette bearbeitet werden.

MID$(X$,I)

überträgt die Zeichenkette X$ beginnend mit dem I-ten Zeichen. I kann Werte zwischen 0 und 255 annehmen. Ist I größer als die gesamte Zeichenkette X$, so wird ein String der Länge 0 (Null) übertragen.

MID$(X$,I)= Y$

Die Zeichenkette X$ wird vom I-ten Zeichen an durch die Zeichenkette Y$ ersetzt. Alle Zeichen von Y$ werden übertragen, falls X$ lang genug ist.
Die Kombination dieser kurz wiederholten String-Anweisungen erlaubt in dem Programm das Suchen und Ersetzen.

Apostroph und Zeilennummer entfernen

In der Zeile 290 wird die Stelle gesucht, auf der in der jeweils bearbeiteten Zeile des Briefzeilenprogramms der Apostroph steht. Mit der MID$-Anweisung wird der Inhalt der Zeichenkettenvariablen A$ in folgender Weise verändert:

* Der Apostroph steht auf der Position, die durch die Anweisung

```
B=INSTR(A$,"'")
```

  ermittelt und an die numerische Variable B übergeben wird.

* Alle Zeichen rechts vom Apostroph, mit der Stelle B + 1 beginnend, werden in der Zeichenkettenvariablen A$ linksbündig abgelegt. Hierdurch wird der Inhalt der Variablen A$ nach links verschoben.

Auf diese Weise wird zeilenweise die Anweisungsnummer und der Apostroph aus jeder Zeile des Briefzeilenprogramms entfernt. Die Programmzeilen werden damit zu Briefzeilen!

Beispiel:

Aus der Kommentarzeile

```
10 'Sehr geehrte Damen und Herren,
```

werden entfernt:

1. Die Anweisungsnummer 10
2. Das folgende Leerzeichen
3. Der Apostroph.

In der Zeichenkettenvariablen A$ finden Sie jetzt den Text:

```
Sehr geehrte Damen und Herren
```

Einfügen individueller Texte an markierten Positionen

In fast allen Anwendungsfällen gibt es "Lücken" im Brieftext, die man für jeden Briefempfänger individuell ausfüllen möchte.

Anwendungsbeispiel

Nehmen Sie z.B. eine im ASCII-Modus gespeicherte "Mahnung".

```
10 'TEUFEL & PARTNER
20 '
30 'Holzkohlengroßhandlung
40 'Engelstraße 6
50 '4712 Hexenhausen 2
60 '
70 '
80 'Ihr Zeichen      Unser Zeichen      Bestellung     Datum
90 '
100 'KQ               SW/F3              Brennstoff     ***
110 '-----------------------------------------------------
120 '
130 'Sehr geehrte***
140 '
150 'unsere Rechnung über *** DM
160 'vom *** ist inzwischen fällig.
170 'Der Rechnungsbetrag ergibt sich aus einer *** - Lieferung
180 'vom ***
190 '
200 'Wir bitten Sie, den ausstehenden Betrag bis zum ***
210 'zu überweisen.
220 '
230 'Falls sich unser Schreiben mit Ihrer Überweisung kreuzte,
240 'betrachten Sie dieses Schreiben als gegenstandslos.
250 '
260 '
270 'Mit freundlichen Grüßen
```

Vorüberlegungen:

* Der Druck soll immer dann anhalten, wenn "***" gefunden wird.

* Der einzusetzende Text soll abgefragt und eingelesen werden.

Gesucht wird die einzelne Einfügeposition nach dem gleichen Schema, wie bei dem Suchen des Apostrophs. Mit der MID$- und RIGHT$-Anweisung wird in Zeile 340 der eingegebene Text an der Position der Markierungszeichen in die Textzeile eingefügt.

Der ausgedruckte Mahnbrief:

TEUFEL & PARTNER

Holzkohlengroßhandlung
Engelstraße 6
4712 Hexenhausen 2

Ihr Zeichen	Unser Zeichen	Bestellung	Datum
KQ	SW/F3	Brennstoff	5.8.84

Sehr geehrter Herr Engel

unsere Rechnung über 725.88 DM
vom 8.4.1984 ist inzwischen fällig.
Der Rechnungsbetrag ergibt sich aus einer Koks - Lieferung
vom 5.4.1984

Wir bitten Sie, den ausstehenden Betrag bis zum 19.8.1984
zu überweisen.

Falls sich unser Schreiben mit Ihrer Überweisung kreuzte,
betrachten Sie dieses Schreiben als gegenstandslos.

Mit freundlichen Grüßen

Zu beachten sind die Abfragen der Ergänzungen. Diese Abfragen erleichtern die Eingabe und helfen die markierten Einfügepositionen als Textlücke im Standardtext auszufüllen.

Fest vorgegeben sind in dem Beispieltext:

Textlücke	Ergänzung
*** (1)	Tagesdatum
*** (2)	Anrede
*** (3)	Rechnungsbetrag
*** (4)	Rechnungsdatum
*** (5)	Sorte
*** (6)	Liefertermin
*** (7)	Zahlungstermin

Diese Ergänzungen können individuell gesetzt und geändert werden. Die Anzahl der Ergänzungen ist ohne Dimensionierung von X$ auf elf beschränkt.

Drucken der Textzeilen

Die Textzeilen werden mit der PRINT #8-Anweisung über die parallele Schnittstelle zum Drucker ausgegeben. Damit der linke Blattrand einstellbar ist, wurde eine Abfrage für den Wert programmiert. Sie finden die Abfrage in Zeile 160 und die dazugehörige Anweisung in Zeile 360:

```
360 PRINT #8,TAB(SPALTE);A$
```

Die Variable A$ enthält die zu druckende Textzeile.

Das "Projekt Text" ist nun vorgestellt und Sie können ausprobieren, ob es für Ihre Standardtexte ausreichend ist, oder ob ein professionelles Textprogramm angeschafft werden muß.

Einführung

Daten als Datensätze in Feldern abzulegen, zu sichern, zu sortieren und in der gewünschten Reihenfolge gezielt wieder auszugeben, alle diese Punkte der Datenver- und -bearbeitung sind in den letzten Kapiteln schon besprochen und von Ihnen erarbeitet.

In dem hier vorgestellten Projekt "Kartei" sollen die genannten Manipulationen von Daten, kombiniert zu einem Adressverwaltungs- bzw. zu einem Karteiprogramm, zusammengefaßt werden.

Das Projektprogramm "Kartei" ist so konzipiert, daß es

* Ihnen Anregungen für eigene individuelle Karteiprogramme,

* eine von vielen möglichen Lösungen für eine Adressverwaltung und

* alle denkbaren Erweiterungen bietet.

Das Projekt "Kartei" soll exemplarisch am konkreten Beispiel Erfahrungen vermitteln, die übertragbar auf andere Dateiprogramme mit anderen Inhalten und Namen sind.

Außerdem werden Sie damit das Basiswissen für den Themenbereich "Dateien und Diskettenstation", der in einem weiteren Buch aus der Edition HEIM zum >> CPC 464 << aufgegriffen wird, erlangen.

Das Projekt: "Kartei"

Die Leistungsmerkmale des Kartei-Programms

Vorgestellt wird für Ihre private oder geschäftliche Nutzung eine Adressverwaltung mit den Leistungsmerkmalen:

* Verwaltung von bis zu 200 Adressen
* Menütechnik für die Benutzerführung
* Eingabe der Adressen im Dialog
* Verarbeiten der Sonderzeichen

* Sichern der Adressen auf dem Datacorder
* Einlesen von gespeicherten Adressdateien
* Suchen von Adressen nach frei wählbaren Suchbegriffen
* Blättern der Adresskartei
* Löschen von einzelnen Adressen

Realisiert wurde das Karteiprogramm mit Ihnen bekannten BASIC-Anweisungen, damit das Hauptaugenmerk auf den Programmablauf gelegt und hierdurch von Ihnen eine unkomplizierte Modifizierung des Programms vorgenommen werden kann.

Das umfangreiche Programm ist komplett als Programmlisting abgedruckt. Sie sollten aber, um Tippfehler zu vermeiden, dem Einlesen des Karteiprogramms von der Programmkassette den Vorzug geben.
Probieren Sie nun das Programm "Kartei" mit Ihren Daten zuerst einmal aus.

Hinweise zum ersten Menü

Hierzu einige zusätzliche Hinweise zu dem vom Programm ausgegebenen ersten Menü:

* Wählen Sie den gewünschten Programm-Teil durch Eingabe einer Kennziffer. In Ihrem Fall ist es die Kennziffer 1, da noch keine Adressdatei existiert.

* Nach der Eingabe der Kennziffer erscheint das zweite Menü, in dem die einzugebende Adresse abgefragt wird. Die Eingaben sind, wie Sie es bereits von anderen Programmen her kennen, korrigierbar, solange nicht die Eingabe mit der ENTER-Taste abgeschlossen ist.

Weitere Erläuterungen sind nicht notwendig, da innerhalb des Programms zu jedem Programmteil entsprechende Benutzerhinweise ausgegeben werden.

Programmlisting

```
10 '************************************
20 '*   Karteiprogramm                 *
30 '************************************
40 '**** Dimensionierung **************
50 DIM A$(200,5)
60 LZEIL=1:S=199
70 '**** Bildschirmeinstellung   ******
80 MODE 1:PRINT STRING$(39,154)
90 WINDOW #1,1,40,8,21:WINDOW #2,1,40,24,25:WINDOW #3,1,40,4,6
100 WINDOW #4,1,40,25,25:WINDOW #5,1,40,20,20
110 LOCATE 9,2:PRINT"*** Karteiprogramm ***"
120 PRINT STRING$(39,154)
130 LOCATE #2,1,1:PRINT#2,STRING$(39,154)
140 '*** Hilfstexte einlesen **********
150 RESTORE:FOR I=1 TO 5:READ TEXT$(I):NEXT I
160 DATA "Vorname ...:","Nachname ..:"
170 DATA "Strasse ...:","PLZ Ort ...:","Tel. ......:"
180 '*** Befehlseingabe ***************
190 GOSUB 1330'---Befehlsliste ausgeben
200 T$=INKEY$:IF T$<"1" OR T$>"6" THEN 200
210 ON VAL(T$) GOSUB 240,480,740,890,1000,1170
220 GOTO 190
230 '* Programmteil: Adresseneingabe  *
240 CLS #1:LOCATE #3,6,2
250 PRINT#3,"Programmteil: Adresseneingabe"
260 IF LZEIL<=S THEN 360
270 FOR I=1 TO LZEIL'----freie Stelle suchen
280 IF A$(I,1)<>"" THEN 310
290 FOR K=2 TO 5:IF A$(I,K)<>"" THEN 310:NEXT K
300 GOTO 380
310 NEXT I
320 CLS #4:LOCATE #4,9,1
330 PRINT #4,"Der Speicher ist belegt!"CHR$(7)
340 FOR i=1 TO 1500:NEXT I:CLS #3:RETURN
```

```
350 '------ Adresseneingabe  ---------
360 I=LZEIL:LZEIL=LZEIL+1
370 CLS #4:LOCATE #4,5,1
380 PRINT #4,"*** Bitte Adresse eingeben ***"
390 FOR K=1 TO 5
400 LOCATE #1,4,2+2*(K-1):PRINT #1,TEXT$(K)
410 NEXT K
420 FOR K=1 TO 5
430 LOCATE #1,16,2+2*(K-1):LINE INPUT #1,A$(I,K)
440 NEXT K
450 IF A$(I,5)<>"" THEN A$(I,5)="Tel.: "+A$(I,5)
460 CLS #3:CLS #1:RETURN
470 '* Programmteil: Adr.-Teil suchen *
480 CLS #1:CLS #3
490 LOCATE #3,7,2:PRINT #3,"Programmteil: Adressen-Suchen"
500 LOCATE #5,7,1:PRINT #5,"Suchen nach Kennziffer: "
510 FOR I=1 TO 5
520 LOCATE #1,7,2+2*(i-1):PRINT #1,TEXT$(I) I
530 NEXT I
540 T$=INKEY$:IF T$<"1" OR T$>"5" THEN 540:'Eingabe
550 LOCATE #4,13,1:PRINT#4,"Suchbegriff"
560 K=VAL(T$):CLS #5:LOCATE #5,7,1:PRINT #5,TEXT$(K);
570 LINE INPUT #5,T$:CLS #1
580 INDEX1=1:'----- Suchen des Teils ----
590 FOR I=INDEX1 TO LZEIL
600 IF INSTR(A$(I,K),T$)<>0 THEN 650
610 NEXT I
620 LOCATE #2,5,2
630 PRINT #2,"*** Suchwort nicht gefunden! ***"CHR$(7)
640 FOR I=1 TO 1500: NEXT I:CLS #3:RETURN
650 LOCATE #1,4,2:PRINT#1,A$(I,1)" "A$(I,2):'Ausgabe
660 FOR L=3 TO 5:LOCATE #1,4,L:PRINT#1,A$(I,L):NEXT L
670 CLS #4:LOCATE #4,5,1:PRINT #4,"Richtige Adresse J/N ?"
680 T=INKEY(45):IF T=-1  AND INKEY(46)=-1 THEN 680
690 IF T=-1 THEN 720
700 IF INKEY(45)<>-1 THEN 700
710 Z=1:RETURN'----- richtige Adresse
720 CLS #1:INDEX1=I+1:GOTO 590
```

```
730 '* Programmteil: Kartei blaettern *
740 CLS #1:CLS #3:LOCATE #3,4,2
750 PRINT#3,"Programmteil: Kartei blaettern"
760 CLS #4:LOCATE #4,10,1:PRINT #4,"Weiter J/N ?"
770 FOR I=1 TO LZEIL-1
780 IF A$(I,1)="" AND A$(I,2)="" THEN 840
790 LOCATE #1,4,2:PRINT#1,A$(I,1)" "A$(I,2)
800 FOR L=3 TO 5:LOCATE #1,4,L:PRINT#1,A$(I,L):NEXT L
810 T=INKEY(45):IF T=-1 AND INKEY(46)=-1 THEN 810
820 IF T=-1 THEN 850
830 CLS #1
840 NEXT I
850 CLS #4:LOCATE #4,13,1
860 PRINT #4,"Kartei - Ende!"CHR$(7)
870 FOR L=1 TO 1500:NEXT L:RETURN
880 '* Programmteil: Adr. loeschen   *
890 CLS #1:LOCATE #3,6,2
900 PRINT#3,"Programmteil: Adressen loeschen"
910 Z=0:GOSUB 500
920 IF Z<>1 THEN RETURN
930 CLS #4:LOCATE #4,5,1
940 PRINT#4,"Loeschen J/N ?"
950 T=INKEY(45):IF T=-1 AND INKEY(46)=-1 THEN 950
960 CLS #4:IF T=-1 THEN RETURN
970 FOR K=1 TO 5:A$(I,K)="":NEXT K
980 CLS #1:RETURN
990 '* Programmteil: Adr. speichern  *
1000 CLS #1:CLS #3:CLS #4
1010 PRINT#3,"Programmteil: Adressen speichern"
1020 LOCATE #4,5,1:PRINT#4,"Fuer Abbruch bitte XXX eingeben"
1030 LOCATE #1,4,2
1040 PRINT #1,"Druecken Sie die Tasten REC und PLAY."
1050 LOCATE #1,4,4
1060 PRINT #1,"Geben Sie dann den Karteinamen ein."
1070 LOCATE #1,4,10:INPUT #1,"Karteiname ?",T$
1080 IF UPPER$(T$)="XXX" THEN RETURN
1090 CLS #4:LOCATE #4,4,1
1100 PRINT#4,"Kartei ";T$" wird gespeichert"
```

```
1110 T$="!"+T$:OPENOUT T$
1120 FOR I=1 TO LZEIL
1130 FOR K=1 TO 5:WRITE #9,A$(I,K):NEXT K
1140 NEXT I
1150 CLOSEOUT:CLS #1:RETURN
1160 '* Programmteil: Adr. einlesen   *
1170 LZEIL=1:CLS #1:CLS #4
1180 CLS #3:LOCATE #3,4,2
1190 PRINT#3,"Programmteil: Adressen einlesen"
1200 LOCATE #4,5,1:PRINT#4,"Fuer Abbruch bitte XXX eingeben"
1210 LOCATE #1,4,2:PRINT #1,"Druecken Sie die Taste PLAY."
1220 LOCATE #1,4,4
1230 PRINT #1,"Geben Sie dann den Karteinamen ein."
1240 LOCATE #1,4,10:INPUT #1,"Karteiname ? ",T$
1250 IF UPPER$(T$)="XXX" THEN RETURN
1260 CLS #4:LOCATE #4,4,1:PRINT#4,"Kartei ";T$" wird gesucht"
1270 T$="!"+T$:OPENIN T$
1280 IF EOF THEN 1310
1290 FOR K=1 TO 5:INPUT #9,A$(LZEIL,K):NEXT K
1300 LZEIL=LZEIL+1:GOTO 1280
1310 CLOSEIN:CLS #1:CLS #4:RETURN
1320 '* Befehlsliste ausgeben   *******
1330 RESTORE 1330:CLS #3
1340 CLS #1:LOCATE #1,7,1:PRINT #1,"M E N U E"
1350 FOR I=1 TO 6:READ TEXT$
1360 LOCATE #1,7,3+2*(I-1):PRINT#1,TEXT$
1370 NEXT I:CLS #4
1380 LOCATE #4,7,1:PRINT #4,"Bitte Kennziffer  eingeben!"
1390 RETURN
1400 DATA "Adresse eingeben ......: 1"
1410 DATA "Adressen-Teil suchen ..: 2"
1420 DATA "Kartei blaettern ......: 3"
1430 DATA "Adresse loeschen ......: 4"
1440 DATA "Adressen speichern ....: 5"
1450 DATA "Adressen einlesen .....: 6"
```

Programmblöcke des Karteiprogramms

Hilfen für die Analyse des Programms

Sie finden anhand der Kommentarzeilen die Programmblöcke:

Programmblöcke	Zeilenbereich
* Dimensionierung	50 - 60
* Bildschirmeinstellung	80 - 130
* Hilfstexte einlesen	150 - 170
* Befehlseingabe	190 - 220
* Adresseneingabe	240 - 460
* Adressenteil suchen	480 - 720
* Kartei blättern	740 - 870
* Adressen löschen	890 - 980
* Adressen speichern	1000 - 1150
* Adressen einlesen	1170 - 1310
* Befehlsliste ausgeben	1330 - 1450

Kurzinformationen zu den Programmblöcken

Dimensionierung

* Definiert wird ein zweidimensionales Feld mit 200 x 5 Feldelementen. Die Anzahl der Feldelemente kann durch die entsprechende Änderung bei der DIM-Anweisung bei einer durchschnittlichen Adressenlänge von insgesamt 100 Zeichen pro Adresse auf 400 x 5 erhöht werden.
  Außerdem muß die Variable S in der Zeile 60 auf 399 abgeändert werden.

Bildschirmaufbau

* Insgesamt fünf Bildschirmfenster werden mit den fünf WINDOW-Anweisungen definiert und eine graphische Gestaltung vorgenommen.

Hilfstexte einlesen

* Die Hilfstexte für die Menüs werden aus DATA-Zeilen in die Feld-Variable TEXT$(I) eingelesen.

Befehlseingabe

* Die eingegebene Kennziffer wird mit INKEY$ abgefragt. Sie wird eingelesen und auf Plausibilität geprüft. Die ON GOSUB-Anweisung bewirkt den von der Kennziffer abhängigen Aufruf des angewählten Programmteils.

Adresseneingabe

* Vorab wird mit der Anweisung

```
IF LZEIL<=S THEN 360
```

in Zeile 260 geprüft, ob der letzte mögliche Adressenplatz des dimensionierten Feldes bereits belegt ist. Ist das der Fall, so wird nach einem freien Adressenplatz in dem dimensionierten Feld gesucht. Gefunden wird ein freier Platz in diesem Fall nur dann, wenn zuvor eine Adresse gelöscht wurde. Wurde keine Adresse gelöscht, also kein freier Platz gefunden, so erscheint die Meldung "Der Speicher ist belegt!".
Wird dagegen bei der Suche ein freier Patz gefunden, so kann eine weitere Adresse eingeben werden.

Adressenteil suchen

* Es kann gezielt nach einem bestimmten Adressenteil gesucht werden. Wird er gefunden , so wird die gesamte Adresse ausgegeben.
Die kurzen Zugriffszeiten entstehen durch ein gezieltes Suchen mit der INSTR-Anweisung allein bei den vorhandenen Inhalten der dem Adressenbereich zugehörigen Feldvariablen.

Kartei blättern

* Die Reihenfolge richtet sich nach der Eingabefolge der Adressen, d.h. die zuerst eingegebene Adresse erscheint zuerst. Eine Unterbrechung des Blätterns ist über die INKEY-Anweisung realisiert. Die Ausgabe der Adressen erfolgt, ähnlich einer Karteikarte, einzeln, so daß hier die Grundforderung an ein "ansehbares" Karteiprogramm mit einem stehenden Bildschirm erfüllt ist.
  Eine alphabetische Sortierung ist eine denkbare Erweiterungsmöglichkeit, die Sie unproblematisch einbinden können. Die erforderlichen praktischen Hinweise finden Sie hierzu in dem Kapitel "Sortieren von Zahlen und Zeichenketten".

Adressen löschen

* Das Unterprogramm "Adressen suchen" wird hierbei mit einbezogen, so daß die zu löschende Adresse zunächst gesucht, dann abgefragt und, falls gewollt, gelöscht wird. Den Feldelementen der Adresse werden hierbei je ein "Nullstring" zugeordnet, so daß die vorherigen Inhalte überschrieben werden.

Adressen speichern

* Die OPENOUT-Anweisung in Zeile 1110 eröffnet einen Datenfile für die Ausgabe auf dem Datacorder. Hierbei wird ein Puffer von 2 k angelegt, in dem die Datensätze vor der Ausgabe auf dem Datacorder zwischengespeichert werden.
  Das Ausrufezeichen "!" vor dem Dateinamen unterdrückt die englischen Hinweise für den Datacorderbetrieb auf dem Bildschirm.

  Das Schreiben der Daten geschieht mit der Anweisung:

```
WRITE #9,A$(I,K)
```

  Hiermit wird der Inhalt der Variable A$(I,K) in den Puffer geschrieben. Ist er gefüllt, so werden die Datensätze auf der Kassette abgelegt.

Die CLOSEOUT-Anweisung leert den Puffer vor dem Programmende. Die zwischengespeicherten "Rest-Daten" würden ohne diese Anweisung nicht abgespeichert und es könnte keine neue Adressdatei eröffnet werden.

Adressen einlesen

* Die Datei wird mit der INPUT #9-Anweisung eingelesen und mit der OPENIN-Anweisung ein Puffer angelegt. Die Datensätze werden blockweise eingelesen. Die EOF-Anweisung in Zeile 1280 fragt vor der INPUT #9-Anweisung das Dateiende ab.
  Die eröffnete Datei wird nach dem Lesevorgang mit der CLOSEIN-Anweisung geschlossen.

Befehlsliste ausgeben

* Die Texte für das Haupt-Menü werden aus DATA-Zeilen eingelesen und mit LOCATE-Anweisungen auf dem Bildschirm positioniert.

Einführung

Ein oder auch mehrere Fehler in einem Programm sind bei der Programmentwicklung fast eine Normalität. Die Fehler können einerseits nützlich, andererseits aber sehr lästig sein. Nützlich, weil aus jeder Fehlerbeseitigung neue Erkenntnisse abgeleitet werden können, die der eigenen Programmiererfahrung und somit den späteren Programmen zugute kommen.

Lästig, da der Zeitaufwand und die Enttäuschung groß sein kann, wenn ein kreiertes Programm nicht "läuft". Häufig handelt es sich hierbei nicht um logische Programmfehler, sondern im nachhinein so trivial erscheinende, daß sie vermeidbar gewesen wären.

Soft- und Hardware Fehler

Hier sind zunächst vier unterschiedliche Fehlerarten zu unterscheiden:

Fehlerarten:

* Syntax - Fehler > Software-Fehler
* Fehler in der Programmlogik > Software-Fehler
* Fehler durch den Programmbenutzer > Software-Fehler
* Fehler der Technik > Hardware-Fehler

Ein Syntax-Fehler liegt immer dann vor, wenn eine BASIC-Format-Vorgabe nicht beachtet wird, z.B. wenn REED anstelle von READ geschrieben wird, oder wenn beim Setzen von Klammern die Zahl der öffnenden Klammern mit der Zahl der schließenden Klammern nicht übereinstimmt.

Logische Programmfehler können z.B. Sprünge in eine FOR NEXT-Schleife sein, nicht mit RETURN abgeschlossene Unterprogramme oder aber auch eine WHILE WEND-Schleife, deren Bedingung niemals erfüllt, d.h. "true" werden kann.

Diese zwei Fehlerarten werden meist während der Testphase des Programmes erkannt und beseitigt. Das ist für die Lauffähigkeit eines Programms absolute Vorbedingung.

Fehler, die durch den Programmbenutzer bzw. bei der Nutzung des Programms auftreten, lassen auf eine zu kurze Testphase oder auf eine Nichtbeachtung aller Programmöglichkeiten schließen. Diese Fehlerart ist häufig anzutreffen, wenn z.B. Benutzereingaben über die Tastatur in das laufende Programm gefordert werden. Hier hilft nur eine sorgfältige Prüfung aller denkbaren Programmvorgänge.

Hardware-Fehler sind die problematischsten, da sie meistens nicht eigenhändig behoben werden können und zu einem unerwarteten Abbruch der Programmierarbeit führen.

Die einfachste, denkbare Fehlerursache ist hierbei ein Netzausfall oder das Durchbrennen einer Sicherung im Terminal oder in der Peripherie des Computers.

Ihr >> CPC 464 << wertet die oben genannten "Software-Fehler" aus und meldet die meisten davon dem Programmierer durch Ausgabe einer Fehlermeldung.

Um Ihnen sämtliche originale Fehlermeldungen auch auf dem Monitor zu bieten, wurde ein Programm erstellt, das nacheinander alle diese Fehlermeldungen ausgibt.

Das BASIC-Programm finden Sie auf der nächsten Seite.

Programm: "Fehlermeldungen des >> CPC 464 <<"

```
10 REM PROGRAMM GIBT ALLE MOEGLICHEN
20 REM FEHLERMELDUNGEN VOM >>CPC 464<<
30 REM AUF DEM MONITOR AUS
40 CLS
50 MODE 1
60 LOCATE 5,2
70 PRINT "**** Alle Fehlermeldungen ****
"
80 LOCATE 2,4
90 PRINT "Bitte Punkt im Ziffernblock dr
uecken!"
100 LOCATE 2,23
110 PRINT "Bei Fehler 2 bitte ESC-Taste
druecken!"
120 i=0
130 KEY 138,"goto 140"+CHR$(13)
140 i=i+1
150 WINDOW #2,3,40,10,20
160 CLS #2
170 LOCATE #2,5,1
180 PRINT #2,"Fehler Nr."I
190 WINDOW SWAP 0,2
200 LOCATE 1,4
210 ERROR i
```

Sie konnten während des Programmablaufs erkennen, daß neben der Fehlermeldung auch eine Fehlernummer ausgegeben wurde. Diese Fehlernummer ist der entsprechenden Fehlermeldung fest zugeordnet, so daß z.B. Fehlernummer 6 die Fehlermeldung "Overflow in <Zeilennummer>" eindeutig kennzeichnet.

Unter BASIC kann die Fehlernummer in einer Fehlerbehandlungsroutine mit der ERR-Anweisung abgefragt werden. Die ERL-Anweisung ermittelt die Zeilennummer, in der der Fehler aufgetreten ist, so daß über beide Angaben ein Beheben des Fehlers durch das Programm selbst möglich wird.

24. Fehlermeldungen und Fehlercodes

Nachfolgend finden Sie aufgeführt:

* Alle Fehlermeldungen mit der dazugehörigen Fehlernummer
* Die Bedeutung bzw. die Übersetzung der Fehlermeldung
* Eine Erläuterung zur Entstehung eines Fehlers
* Programmbeispiele, die eine Fehlermeldung erzeugen
* Querverweise zu BASIC-Anweisungen

Falls in Ihrem Programm eine Fehlermeldung erscheint, so sollten Sie diesen Teil des Buches unbedingt nutzen, um den Fehler möglichst schnell lokalisieren und beheben zu können.

Liste der Fehlercodes

Fehlernummer: 1

Fehlermeldung: "Unexpected NEXT in <Zeilennummer>"

Bedeutung: NEXT wurde ohne dazugehörige FOR-Anweisung gefunden.

Erläuterung:

Diese Fehlermeldung tritt auf, wenn die zur NEXT-Anweisung zugehörige Laufanweisung FOR nicht programmiert wurde. Eine weitere Ursache für diese Fehlermeldung kann ein Hineinspringen in die FOR NEXT - Schleife sein, z.B. mit der GOTO-Anweisung.

Vergleichen Sie hierzu: FOR NEXT

Fehlernummer: 2

Fehlermeldung: "Syntax error in <Zeilennummer>"

Bedeutung: BASIC-Schreibfehler

Erläuterung:

Ein Syntaxfehler wird gemeldet, wenn eine BASIC-Rechtschreibregel nicht beachtet wird, z.B. REED anstelle von READ, oder wenn beim Setzen von Klammern die Zahl der sich öffnenden Klammern mit der Zahl der sich schließenden Klammern nicht übereinstimmt.

Vergleichen Sie hierzu: EDIT

Fehlernummer: 3

Fehlermeldung: "Unexpected RETURN in <Zeilennummer>"

Bedeutung: Bei einer RETURN-Anweisung fehlt die dazugehörige GOSUB-Anweisung

Erläuterung:

Diese Fehlermeldung wird ausgegeben, wenn versucht wird, aus einem Unterprogramm ohne vorherigen ordnungsgemäßen Einsprung in dieses Unterprogramm zurückzukehren, Wie bei der FOR NEXT-Schleife das Paar FOR und NEXT zusammengehören, so gehört auch das Paar GOSUB-RETURN zusammen. Fehlt die GOSUB-Anweisung, wird die oben aufgeführte Fehlermeldung auf dem Bildschirm angezeigt. Der Fehler tritt ebenfalls auf, wenn von außen in ein Unterprogramm hineingesprungen wird, d.h. also, wenn die GOSUB-Anweisung nicht ordnungsgemäß durchlaufen wird.

Vergleichen Sie hierzu: RETURN, GOSUB

Fehlernummer: 4

Fehlermeldung: "DATA exhausted in <Zeilennummer>"

Bedeutung: Es fehlen Daten für die DATA-Anweisung

Erläuterung:

Auch hier haben wir wieder zumindest ein Pärchen, bestehend aus einer oder mehreren READ-Anweisungen und einer oder mehreren DATA-Anweisungen, vor uns. "Gelesen" werden DATA-Anweisungen ausschließlich mit einer oder mehreren READ-Anweisungen. Die Fehlermeldung tritt dann auf, wenn alle vorhandenen DATA-Anweisungen bereits gelesen worden sind und ein erneuter Leseversuch unternommen wird.
Mit anderen Worten, der Fehler wird angezeigt, wenn für eine READ-Anweisung keine DATA-Anweisung mehr vorhanden ist.

Vergleichen Sie hierzu: READ, DATA

Fehlernummer: 5

Fehlermeldung: "Improper argument in <Zeilennummer>"

Bedeutung: Es wurde ein unerlaubtes Argument angegeben.

Erläuterung:

Diese Fehlermeldung kann die verschiedensten Ursachen haben. Sie bedeutet, daß der Wert eines Arguments einer Funktion nicht erlaubt ist oder bestimmte Parameter einer Anweisung falsch sind.

Beispiel:

```
PRINT SQR(-25)
Improper argument
```

MODE 5
Improper argument

WIDTH 400
Improper argument

Hinweis:

Wenn bei ON <M> GOTO ... oder auch bei ON <M> GOSUB ... der Wert des Ausdrucks gleich Null oder größer als die Anzahl der Listenelemente (aber <=255) ist, so wird keine Fehlermeldung angezeigt, sondern es wird mit der nächsten, ausführbaren Anweisung weitergearbeitet. Das ist insofern von Bedeutung, als der Programmierer die notwendigen Plausibilitätsprüfungen selbst durchführen muß, wenn er nicht Opfer von Fehleingaben oder Rechenfehlern werden will.

Fehlernummer: 6

Fehlermeldung: "Overflow in <Zeilennummer>"

Bedeutung: Überlauf

Erläuterung:

Liegt das Ergebnis einer Berechnung durch BASIC außerhalb des zulässigen Bereichs, so wird diese Fehlermeldung als Warnung ausgegeben, ohne jedoch einen Programmabbruch zu verursachen.

Beispiel:

```
10 A=1E+38
20 PRINT A+A
30 PRINT "Programmende"
```

Ergebnis:

```
Overflow
 1.70141E+38
Programmende
Ready
```

Fehlernummer: 7

Fehlermeldung: "Memory full in <Zeilennummer>"

Bedeutung: Der Speicher des >> CPC 464 << ist voll.

Erläuterung:

Diese Fehlermeldung bedeutet, daß der Speicher des >> CPC 464 << vollständig gefüllt ist. Dies kann folgende Ursachen haben:

1) Das Programm ist zu groß.

2) Das Programm enthält zu viele FOR NEXT-Schleifen.

3) Das Programm enthält zu viele Verschachtelungen von Unterprogrammen.

4) Im Programm treten zu viele Variablen auf.

5) Durch die MEMORY-Anweisung wurde der für BASIC verfügbare Speicherplatz zu klein bemessen.

Hinweis:

Ein geöffneter Kassettenfile benötigt einen Puffer im Speicher des >> CPC 464 <<, was dazu führen kann, daß die Werte für MEMORY eingeschränkt sind.

Vergleichen Sie hierzu: FRE, HIMEM, MEMORY

Fehlernummer: 8

Fehlermeldung: "Line does not exist in <Zeilennummer>"

Bedeutung: Eine angesprochene Zeilennummer existiert nicht.

Erläuterung:

Die angesprochene Zeilennummer ist im Programm nicht vorhanden. Fehlermeldungen dieser Art kommen vor, wenn zu einer nicht vorhandenen Zeile verzweigt wird. Sie erscheint häufig im Zusammenhang mit GOTO oder GOSUB, oder auch bei IF THEN ELSE.

Vergleichen Sie hierzu: DELETE, EDIT, LIST, RENUM

Fehlernummer: 9

Fehlermeldung: "Subscript out of range in <Zeilennummer>"

Bedeutung: Ein Index liegt außerhalb des festgelegten Bereichs.

Erläuterung:

Diese Fehlermeldung bedeutet, daß ein Index außerhalb des zulässigen Bereichs liegt. Sie wird ausgegeben, wenn bei der Dimensionierung eines Feldes mit der DIM-Anweisung zu kleine Werte gewählt wurden.

```
10 DIM A(2)
20 A(4)=55
```

Ergebnis: Subscript out of range in 20

Vergleichen Sie hierzu: DIM

Fehlernummer: 10

Fehlermeldung: "Array already dimensioned in <Zeilennummer>"

Bedeutung: Eine Feldvariable wurde bereits dimensioniert.

Erläuterung:

Diese Fehlermeldung besagt, daß bereits ein Feld mit gleichem Namen dimensioniert wurde. In diesem Fall muß entweder ein anderer Variablenname gewählt werden oder das ursprüngliche Feld gelöscht werden.
Eine wiederholte Dimensionierung kann auch "verdeckt" vorkommen, so daß es dem Programmierer anfänglich gar nicht auffällt. Hierzu folgendes Beispiel:

```
100 FOR I=0 TO 10
110 A(I)=I*I
120 PRINT A(I);
130 NEXT I
140 DIM A(100)
150 PRINT "Neuer Abschnitt"
```

Ergebnis:

```
0  1  4  9  16  25  36  49  64  81  100
Array already dimensioned in 140
```

Im ersten Programmteil (Anweisungsnummern 100 bis 130) wird ein Feld mit 11 Elementen bearbeitet, und zwar von A(0) bis A(10). Damit ist die intern vorgegebene Dimensionierung ausgenutzt. In der Anweisungsnummer 140 wird die Variable erneut, und damit unzulässigerweise ein zweites Mal dimensioniert. Das Ergebnis ist die oben angezeigte Fehlermeldung:

"Array already dimensioned in 140".

Vergleichen Sie hierzu: DIM, ERASE

Fehlernummer: 11

Fehlermeldung: "Division by zero in <Zeilennummer>"

Bedeutung: Es wurde durch Null dividiert.

Erläuterung:

Eine Division durch Null "0" ist nach den Regeln der Mathematik nicht zulässig und wird deshalb auch durch BASIC als mathematischer Fehler angezeigt.

Beispiel:

```
10 INPUT WERT
20 PRINT WERT/ZAHL
30 PRINT "Programm laeuft weiter"
40 END
```

Ergebnis:

```
? 78
Division by zero
 1.701411E+38
Programm laeuft weiter
Ready
```

Fehlernummer: 12

Fehlermeldung: "Invalid direct command in <Zeilennummer>"

Bedeutung: Diese direkte Eingabe ist im Kommandomodus unzulässig.

Erläuterung:

Es wird im Direkt-Modus eine BASIC-Anweisung eingegeben, die nur in einem Programm angewendet werden darf.

Beispiel:

```
DEF FN C(A,B)=A*A + B*B
Invalid direct command
```

Fehlernummer: 13

Fehlermeldung: "Type mismatch in <Zeilennummer>"

Bedeutung: Die Zuweisung zu einer Variablen ist nicht typgerecht.

Erläuterung:

Bei einer Zuweisung von Variablen stimmen die Typen nicht überein und führen zur entsprechenden Fehlermeldung.

Beispiel 1:

```
A$=3
Type mismatch
Ready
```

Beispiel 2:

```
LINE INPUT A
Type mismatch
Ready
```

Bei einer LINE INPUT-Anweisung muß eine Zeichenkettenvariable folgen.

Vergleichen Sie hierzu: DEFINT, DEFSTR, DEFREAL

Fehlernummer: 14

Fehlermeldung: "String space full in <Zeilennummer>"

Bedeutung: Zeichenkettenbereich ist belegt.

Erläuterung:

Es wurden zuviele Zeichenketten definiert, so daß der für Zeichenketten reservierte Speicherbereich auch nach einer "garbage collection" nicht groß genug ist. "Garbage collection" bedeutet ein Aufräumen des Arbeitsspeichers.
Die für Zeichenketten reservierte Speicherbereiche werden gelöscht, wenn diese nicht mehr benötigt werden. Das Aufräumen wird vom BASIC-Interpreter selbst durchgeführt.

Beachten Sie, daß während einer "garbage collection" weder Eingaben noch Berechnungen durchgeführt werden können.

Vergleichen Sie hierzu: FRE

Fehlernummer: 15

Fehlermeldung: "String too long in <Zeilennummer>"

Bedeutung: Eine Zeichenkette besitzt mehr als 255 Zeichen.

Erläuterung:

In einem Programmablauf kann es durch Aneinanderhängen (Addition) von Zeichenkettenausdrücken zu dieser Fehlermeldung kommen.

Fehlernummer: 16

Fehlermeldung: "String expression too complex in <Zeilennummer>"

Bedeutung: Ein Zeichenkettenausdruck ist zu kompliziert.

Erläuterung:

Diese Fehlermeldung wird ausgegeben, wenn ein Stringausdruck für BASIC zu kompliziert ist.
Dieses tritt ein, wenn zu viele Verschachtelungen enthalten sind.

In einem derartigen Fall sollte der Ausdruck aufgeteilt werden, so daß sich kleinere, für BASIC überschaubare Ausdrücke ergeben.

Fehlernummer: 17

Fehlermeldung: "Cannot CONTinue in <Zeilennummer>"

Bedeutung: Der Programmablauf kann nicht fortgesetzt werden.

Erläuterung:

Diese Fehlermeldung tritt nur beim Befehl CONT auf, der z.B. nach STOP oder nach einem Fehler eine Fortsetzung des Programms bewirkt. Eine Programmfortsetzung ist jedoch nicht immer möglich.

Dies ist immer dann der Fall, wenn das Programm nach einem Abbruch geändert wurde.

Vergleichen Sie hierzu: CONT, STOP

Fehlernummer: 18

Fehlermeldung: "Unknown user function in <Zeilennummer>"

Bedeutung: Eine aufgerufene Funktion wurde noch nicht definiert.

Erläuterung:

Wird eine vom Benutzer zu definierende Funktion mit FN aufgerufen, ohne daß diese zuvor mit DEF FN definiert wurde, so kann die entsprechende Berechnung nicht durchgeführt werden und es wird die entsprechende Fehlermeldung ausgegeben.

Vergleichen Sie hierzu: DEF FN

Fehlernummer: 19

Fehlermeldung: "RESUME missing in <Zeilennummer>"

Bedeutung: In einer Fehlerbehandlungsroutine fehlt die RESUME-Anweisung.

Erläuterung:

Im Zusammenhang mit der ON ERROR GOTO-Anweisung muß eine Fehlerbehandlungsroutine mit RESUME abgeschlossen werden. Fehlt die RESUME-Anweisung, so ist eine Wiederaufnahme des regulären Programmablaufs nicht möglich.

Vergleichen Sie hierzu: RESUME, ON ERROR GOTO, ERR, ERL

Fehlernummer: 20

Fehlermeldung: "Unexpected RESUME in <Zeilennummer>"

Bedeutung: RESUME steht nicht innerhalb einer Fehlerbehandlungsroutine.

Erläuterung:

Eine RESUME-Anweisung ist nur innerhalb einer Fehlerbehandlungsroutine zulässig, die mit ON ERROR GOTO vordefiniert sein muß. Ein Einsprung mitten in eine Fehlerbehandlungsroutine ist meist ein Programmierfehler wie in folgendem Programmbeispiel:

```
10 ON ERROR GOTO 90
20 DIM A(20)
30 FOR I=1 TO 15
40 A(I)=I
50 PRINT A(I)
60 GOTO 90:REM *** Programmfehler ***
70 NEXT
80 END
90 ON ERROR GOTO 0
100 PRINT ERR,ERL:RESUME 100
```

Ergebnis:

```
 1
 20          100
Unexpected RESUME in 100
```

Vergleichen Sie hierzu: RESUME, ON ERROR GOTO, ERR, ERL

Fehlernummer: 21

Fehlermeldung: "Direct command found"

Bedeutung: Direkt-Kommando beim Laden von Kassette gefunden.

Erläuterung:

Sollen mit LOAD Programme vom Datacorder geladen werden und wird eine Zeile ohne Zeilennummer erreicht, so wird diese als Direktanweisung interpretiert und die Fehlermeldung "Direct command found" ausgegeben. In der Regel sind in einem derartigen Fall die gelesenen Daten kein Programm, sondern Texte.

Fehlernummer: 22

Fehlermeldung: "Operand missing in <Zeilennummer>"

Bedeutung: Es fehlt in einer Anweisung ein Operand.

Erläuterung:

Diese Fehlermeldung tritt auf, wenn in einem Ausdruck, bei einem Funktionsaufruf oder auch bei Anweisung ein Operand fehlt. Die Fehlermeldung tritt nicht nur in einem Programm, sondern auch im Direktmodus auf.

Beispiel:

```
PRINT 3*4.75+
Operand missing
Ready
```

Fehlernummer: 23

Fehlermeldung: "Line too long in <Zeilennummer>"

Bedeutung: Eine Zeile ist zu lang.

Erläuterung:

Vom Benutzer direkt oder im Programm eingegebene Zeilen werden von BASIC in eine eigene Form umgewandelt. Diese kann nur eine bestimmte Größe annehmen, andernfalls wird die Fehlermeldung "Line too long" ausgegeben. Diese Fehlermeldung tritt nur in sehr seltenen Fällen auf. In einem derartigen Fall sollte man eine Zeile in zwei oder mehrere aufteilen, was meist auch der Übersichtlichkeit des Programms dient.

Fehlernummer: 24

Fehlermeldung: "EOF met in <Zeilennummer>"

Bedeutung: Das Ende einer Datei wurde bereits erreicht.

Erläuterung:

Wird beim Lesen einer Datei vom DATACORDER nicht das Ende der Datei beachtet, sondern über das Ende der Datei hinaus gelesen, so ist dies nicht möglich und führt zu dieser Fehlermeldung.
Das Ende einer Datei kann mit EOF abgefragt und mit einer IF THEN-Anweisung die INPUT-Anweisung im Programmablauf übersprungen werden.

Vergleichen Sie hierzu: EOF, OPENIN, INPUT, LINE INPUT

Fehlernummer: 25

Fehlermeldung: "File type error in <Zeilennummer>"

Bedeutung: Der Typ einer Datei ist nicht korrekt.

Erläuterung:

Werden Daten, z.B. Text-Dateien, vom DATACORDER eingelesen, so muß die Aufzeichnungsart beachtet werden, d.h. sie können nicht mit der LOAD-Anweisung geladen werden. Außerdem können nur Dateien, die im ASCII-Code abgelegt wurden, mit der OPENIN-Anweisung gelesen werden.

Vergleichen Sie hierzu: LOAD, RUN, SAVE, OPENIN, OPENOUT

Fehlernummer: 26

Fehlermeldung: "NEXT missing in <Zeilennummer>"

Bedeutung: Eine FOR NEXT-Schleife ist unvollständig.

Erläuterung:

Wird zur mehrfachen Wiederholung eines Programms eine Schleife mit der dazugehörigen FOR-Anweisung begonnen, so muß das Schleifenende mit NEXT gekennzeichnet werden.

Die Fehlermeldung tritt insbesondere dann auf, wenn zwar eine NEXT-Anweisung vorhanden ist, die angegebene Laufvariable aber nicht mit der in der FOR-Schleife übereinstimmt. Weiterhin muß bei geschachtelten Schleifen bei Angabe mehrerer Parameter nach NEXT auf die richtige Reihenfolge geachtet werden.

Beispiel:

```
10 FOR I=1 TO 10
20 FOR K=1 TO 3
30 PRINT I*K
40 NEXT I
```

Ergebnis:

NEXT missing in 10

In Zeile 40 wurden versehentlich die Laufvariablen K vergessen.

Vergleichen Sie hierzu: FOR NEXT

Fehlernummer: 27

Fehlermeldung: "File already open in <Zeilennummer>"

Bedeutung: Der angesprochene File ist bereits eröffnet.

Erläuterung:

Beim Ablegen bzw. Lesen von Dateien mit dem DATACORDER ist es nicht möglich, eine OPENOUT- oder OPENIN-Anweisung mehrfach zu geben, ohne vorher die bereits geöffnete Datei wieder zu schließen. Das Schließen erfolgt mit der CLOSEIN- bzw. mit der CLOSEOUT-Anweisung. Nach dieser Anweisung kann ein neuer FILE eröffnet werden.

Beispiel:

```
10 OPENOUT "NAMEN"
20 OPENOUT "TELEFONNUMMERN"

  (weiteres Programm)
```

führt beim Starten des Programms zur Fehlermeldung:

"File already open in 20"

Vergleichen Sie hierzu: CLOSEIN, CLOSEOUT, OPENIN, OPENOUT

Fehlernummer: 28

Fehlermeldung: "Unknown command"

Bedeutung: Eine Anweisung ist in BASIC nicht definiert.

Erläuterung:

Diese Fehlermeldung kommt in der Regel bei alleiniger Arbeit mit dem BASIC-Interpreter nicht vor. Sie kann aber bei Kommunikation mit weiteren Geräten auftreten, wenn Anweisungen gegeben werden, die BASIC nicht bekannt sind.

Fehlernummer: 29

Fehlermeldung: "WEND missing in <Zeilennummer>"

Bedeutung: In einer WHILE WEND-Schleife fehlt das WEND.

Erläuterung:

Beim Programmieren einer WHILE WEND-Schleife ist ebenso wie bei der FOR NEXT-Schleife ein Kennzeichnen des Schleifenendes notwendig. Tritt die Fehlermeldung "WEND missing" auf, so fehlt zu einer WHILE-Anweisung die entsprechende WEND-Anweisung als Schleifenabschluß. Hierbei muß beachtet werden, daß sich ein WEND immer auf das vorhergehende WHILE bezieht und deshalb zu jeder WHILE-Anweisung eine WEND-Anweisung notwendig ist.

Beispiel:

```
10 MODE  1
20 WHILE SUMME<50
30 INPUT "Ihre neue Zahl : ";ZAHL
40 SUMME=SUMME+ZAHL
50 PRINT "      SUMME = ";SUMME
60 REM *** WEND VERGESSEN ***
70 END
```

Ergebnis:

Das Programm bricht in Zeile 20 mit der Fehlermeldung "WEND missing in 20" ab, da das Schleifenende nicht mit der WEND-Anweisung gekennzeichnet wurde.
Dieser Fehler tritt nicht auf, wenn Zeile 60 eingefügt wird:

```
60 WEND
```

Vergleichen Sie hierzu: WHILE WEND

Fehlernummer: 30

Fehlermeldung: "Unexpected WEND in <Zeilennummer>"

Bedeutung: Bei einer WEND-Anweisung fehlt die zugehörige WHILE-Anweisung.

Erläuterung:

Diese Fehlermeldung besagt, daß ein durch WEND zu kennzeichnendes Schleifenende angetroffen wurde, ohne daß eine dazugehörige Schleife vorhanden ist.
Dem BASIC-Interpreter ist in diesem Fall keine Bedingung vorgegeben, die er beim Erreichen der WEND-Anweisung überprüft, bevor BASIC einen Programmteil erneut durchläuft.

Vergleichen Sie hierzu: WHILE WEND

Einführung

Viele Menschen betrachten lieber ein Bild, eine Skizze, eine Graphik, um sich über einen Sachverhalt zu unterrichten, als daß sie eine entsprechende Beschreibung über diesen Sachverhalt lesen. Wenn es sich allerdings um ganz kurze Informationen handelt, sehen die Dinge bereits etwas anders aus. Hierzu ein kurzes Beispiel aus dem Alltag:

Sie fahren im Auto und betrachten die Verkehrsschilder genau. Ihnen fällt aufgrund der Gewöhnung dabei gar nicht mehr auf, daß einige Schilder in Wahrheit nur Bilder sind, andere Schilder enthalten Bilder gemischt mit Texten, andere wiederum nur Texte. Diese unterschiedliche Art der Darstellung ist auch notwendig, denn nicht jeder verkehrswichtige Sachverhalt läßt sich nur durch ein einziges Bild eindeutig beschreiben.

Offensichtlich ist also eine große Zahl von Menschen besser in der Lage, sich über Sachverhalte zu unterrichten, wenn sie ihnen in bildhafter oder graphischer Form angeboten werden.
Arbeitsorganisationen machen sich derartige Überlegungen zunutze, wenn sie Arbeitsabläufe im beruflichen Alltag mit graphischen Symbolen, also bildhaft wiedergeben.

Gleiche Überlegungen lassen sich auch im Zusammenhang mit Problemen der elektronischen Datenverarbeitung anstellen:

Wenn man im Computer ein Programm ablaufen läßt, so zwingt man ihn dazu, einen vorgegebenen Arbeitsablauf in einzelnen Arbeitsschritten auszuführen. Denn das aus dem Griechischen stammende Wort "Programm" bedeutet ja nichts anderes als (wörtlich übersetzt) "Vorschrift", hier "Arbeitsvorschrift". Die Aufeinanderfolge von Arbeitsschritten in einem Computer läßt sich nun ebenfalls durch graphische Symbole ausdrücken.

Verständlicherweise wurden diese Symbole ebenso genormt wie die Straßenverkehrs-Symbole, und dies aus guten Gründen: Ohne eine Vereinheitlichung wäre die Benutzung von Symbolen sinnlos, denn an die Stelle der Verdeutlichung träte das Chaos: Man müßte raten, was jeweils gemeint ist.

25. Programmablaufpläne

Sie kennen den Ausdruck "Schilderwald" für eine Situation im Straßenverkehr, die Sie nicht sofort beherrschen, und bei der Sie "raten müssen, was denn nun eigentlich gemeint sei". Hier ist also trotz der Vereinheitlichung (Normung) dennoch keine Verdeutlichung gegeben: Das Überangebot stiftet nur Verwirrung.

Gleiches gilt auch für graphische Darstellungen in der elektronischen Datenverarbeitung. Ein Überangebot an graphischen Informationen verwirrt.

Aus diesem Grund hat man bei den graphischen Darstellungen der EDV die Möglichkeit, in mehreren Stufen vorzugehen. Man wird ein kompliziertes Programm (einen komplizierten Arbeitsablauf) zuerst einmal in groben Zügen darstellen und dabei eben nicht jeden einzelnen Arbeitsschritt des Computers berücksichtigen. Man spricht von einem Blockdiagramm oder auch von einem Flußdiagramm.

Es ist gewissermaßen so, als ob man nur die D-Zug-Stationen einer Reise aufführt und nicht auch alle anderen nur möglichen Stationen.
In einer zweiten Stufe erst wird man bei einem komplizierten Programm einen genauen Programmablaufplan zeichnen (Feindiagramm).
Man bedient sich, wie erwähnt, genormter Symbole, wie sie in der DIN-Norm 66001 niedergelegt sind. Auf der folgenden Seite geben wir die wichtigsten Symbole dieser DIN-Norm wieder.

Symbolgruppen

Mit Programmablaufplänen werden Abläufe von Operationen (Verarbeitungsschritten) im Computer unter Verwendung von jeweils vorhandenen Daten beschrieben. Die Verwendung dieser Programmablaufpläne besteht vor allem aus Symbolen für

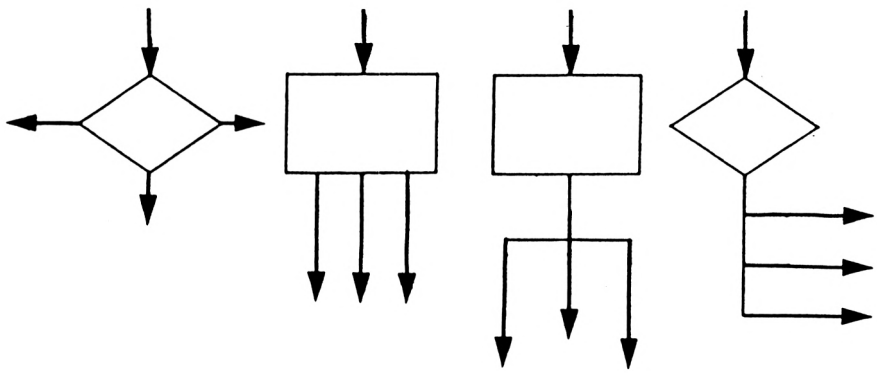
- Operationen
- Eingabe
- Ausgabe
- Ablauflinien

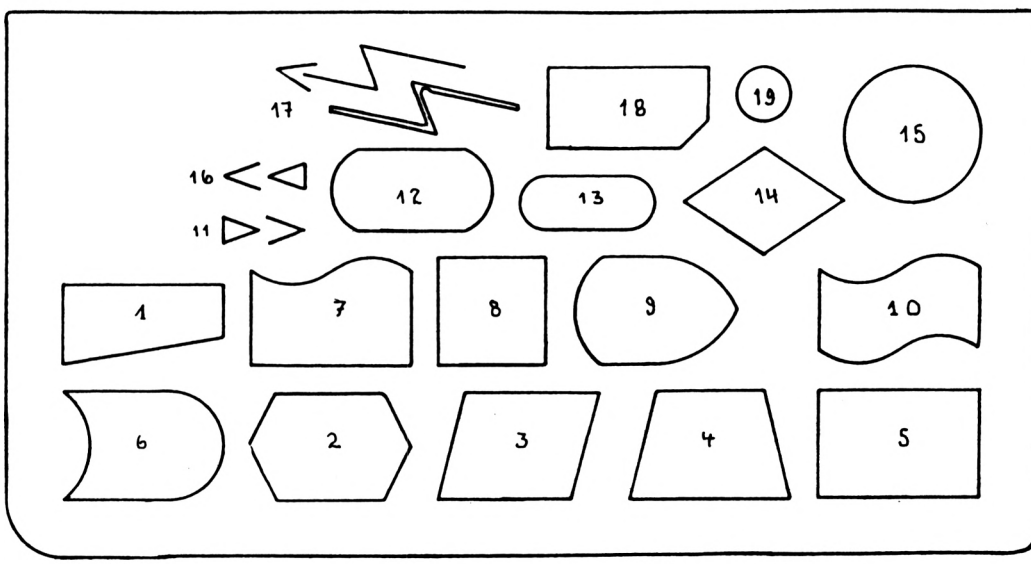
Weitere Symbole dienen der verfeinerten Darstellung.

Anordnung der Symbole

Jedes Symbol des Programmablaufplanes hat nur einen Eingang und einen oder mehrere Ausgänge.

Beispiele:





1 Eingeben von Hand (Datenerfassung)

2 Programm-Modifikation (z.B. programmierte Schalter)

3 Datenträger allgemein für Darstellungen, bei denen der Datenträger nicht näher beschrieben werden kann

4 Operationen von Hand, Eingreifen von Hand

5 Bearbeiten allgemein

6 Vom Leitwerk der Datenverarbeitungsanlage gesteuerter Datenträger

7 Papierausdruck

8 Ausführung einer Hilfsfunktion

9 Anzeige in optischer oder akustischer Form

10 Lochstreifen

11 Pfeil zur Darstellung eines gerichteten Informationsflusses

12 Trommelspeicher, Plattenspeicher (mit graphischer Ergänzung)

13 Grenzstelle

14 Verzweigung

15 Magnetband (mit graphischer Ergänzung)

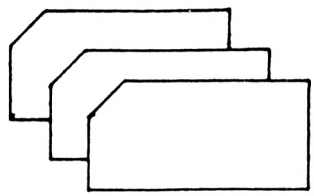
16 wie Nr. 11

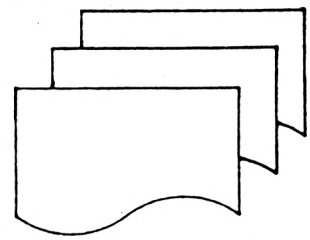
17 Datenübertragung (Ergänzung durch gerichteten Pfeil)

18 Lochkarten

19 Übergangsstelle

Die Wiederholung gleichartiger Symbole stellen Sie dabei wie folgt dar:

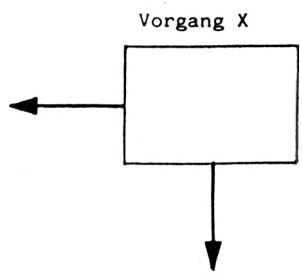


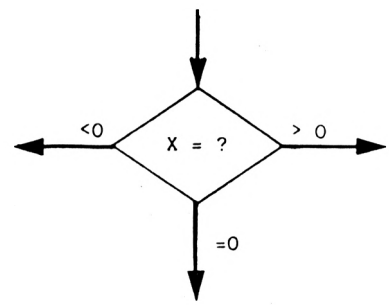


Die Symbole können beschriftet werden. Dabei ist zwischen einer inneren und einer äußeren Beschriftung zu unterscheiden.

Die Außenbeschriftung darf nur dann angewendet werden, wenn eine Beziehung zu anderen Teilen der Dokumentation hergestellt werden muß, bzw. wenn bei mehreren Abzweigungslinien aus einem Symbol die Bedingungen beschrieben werden müssen, nach denen verzweigt werden soll.

Beispiel:





Die Innenbeschriftung eines Symbols soll nur dann angewendet werden, wenn auch ein unmittelbarer Zusammenhang zu weiteren Symbolen besteht und ein Hinweis auf andere Abläufe gegeben werden muß.

25. Programmablaufpläne

Um ein allgemein verständliches Beispiel eines Ablaufplanes zu geben, das zugegebenermaßen mit der elektronischen Datenverarbeitung nicht unmittelbar etwas zu tun hat, zitieren wir aus dem ersten Band (S. 15) des im gleichen Verlag erschienenen zweibändigen großen Alphatronic BASIC-Buchs von Bodo und Norbert Bollow:

Zusammenfassung

Sie werden festgestellt haben, daß wir in unserem Buch mit Programmablaufplänen sehr sparsam umgegangen sind. Und das hat auch seinen Grund:

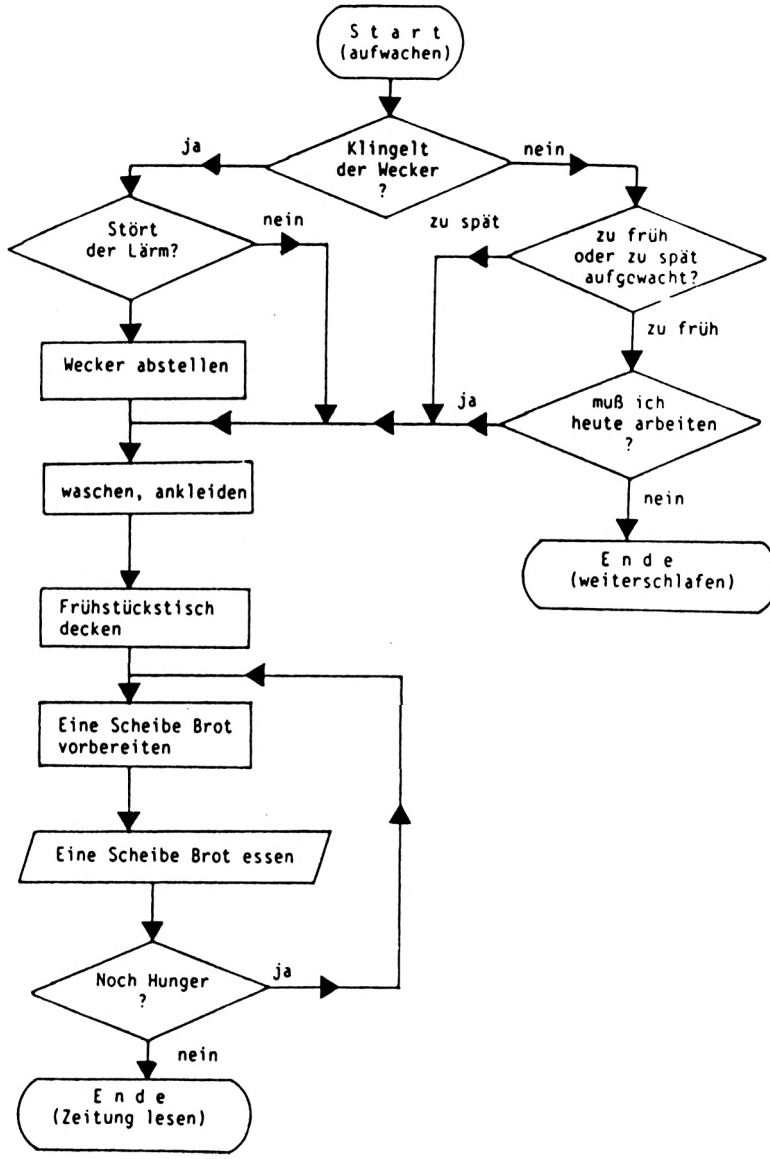
Unsere Übungsprogramme sind im allgemeinen sehr kurz. Sie können sie unmittelbar an Ihrem >> CPC 464 << eingeben und ausprobieren. Diese Möglichkeit, die sie heute haben, gab es in der Zeit, als nur die Großcomputer existierten, nicht:

Der Programmierer mußte seine Programme am Schreibtisch entwerfen und auch am Schreibtisch testen. Man sprach geradezu von "Schreibtischtests", die einem Maschinentest vorausgehen mußten. Verständlicherweise, denn die Maschinenstunden am Großrechner waren sehr teuer (bis zu einigen tausend DM!). Und es vergingen oftmals einige Tage, bis er das Ergebnis eines Maschinenlaufs in Form einer Liste auf dem Papier vor sich liegen hatte. Hier half ihm nur sein Programmablaufplan, wenn er noch mit der Lösung seines Problems fertig werden wollte.

Sie haben da, wie gesagt, ganz andere Möglichkeiten:

Sofort nach dem Start des Programms haben sie Erfolg oder etwas weniger Erfolg! Sie können mit dem >> CPC 464 << im Dialog arbeiten. Ihr Interpreter erlaubt Ihnen diese Vorgehensweise. Und bei kurzen Programmen, die auf einer Bildschirmseite vor Ihnen stehen, haben Sie auch einen ausreichenden Überblick. Sie lernen daher schnell und ohne Ballast und können auf winzige Programmablaufplänchen verzichten.

Bei großen Programmen sieht es natürlich anders aus!


S t a r t
(aufwachen)
Klingelt der Wecker ?
ja
nein
Stört der Lärm?
nein
zu spät
zu früh oder zu spät aufgewacht?
zu früh
Wecker abstellen
ja
muß ich heute arbeiten ?
nein
E n d e
(weiterschlafen)
waschen, ankleiden
Frühstückstisch decken
Eine Scheibe Brot vorbereiten
Eine Scheibe Brot essen
Noch Hunger ?
ja
nein
E n d e
(Zeitung lesen)

26. ASCII-Werte Tabelle

ASCII-Werte Tabelle Teil 1

DEC	OCTAL	HEX	Name	Bedeutung / Wirkung
0	000	00	NUL	ohne Wirkung
1	001	01	SOH	Drucken der Steuerzeichen als Symbole
2	002	02	STX	Textcursor ausschalten
3	003	03	ETX	Textcursor einschalten
4	004	04	EOT	entspricht MODE-Anweisung
5	005	05	ENQ	setzt Zeichen auf Graphikcursorposition
6	006	06	ACK	Einschalten des Textbildschirms
7	007	07	BEL	Signal ausgeben, leert die Warteschlangen
8	010	08	BS	Cursor ein Zeichen zurück
9	011	09	HT	Cursor ein Zeichen vor
10	012	0A	LF	setzt den Cursor eine Zeile nach unten
11	013	0B	VT	setzt den Cursor eine Zeile nach oben
12	014	0C	FF	entspricht CLS
13	015	0D	CR	Cursor an Zeilenanfang
14	016	0E	SO	entspricht PAPER-Anweisung
15	017	0F	SI	entspricht PEN-Anweisung
16	020	10	DLE	Zeichen unter Cursor löschen
17	021	11	DC1	Zeile bis Cursor löschen
18	022	12	DC2	Zeile ab Cursor löschen
19	023	13	DC3	Bildschirm bis Cursor löschen
20	024	14	DC4	Bildschirm ab Cursorposition löschen
21	025	15	NAK	Textbildschirm abschalten, siehe ACK
22	026	16	SYN	Transparentmodus setzen
23	027	17	ETB	Graphikfarbstift setzen
24	030	18	CAN	PAPER und PEN-Farben tauschen
25	031	19	EM	entspricht SYMBOL-Anweisung
26	032	1A	SUB	entspricht WINDOW-Anweisung
27	033	1B	ESC	im Programmablauf keinen Einfluß
28	034	1C	FS	entspricht INK-Anweisung
29	035	1D	GS	entspricht BORDER-Anweisung
30	036	1E	RS	entspricht LOCATE 1,1
31	037	1F	US	entspricht der LOCATE-Anweisung
32	040	20	SP	setzt Leerzeichen "<space>"

ASCII-Werte Tabelle Teil 2

DEC	OCTAL	HEX	ASCII-Zeichen	Tastencode	Block
32	040	20	<space>	47	
33	041	21	!	64	
34	042	22	"	65	
35	043	23	#	57	
36	044	24	$	56	
37	045	25	%	49	
38	046	26	&	48	
39	047	27	'	41	
40	050	28	(	40	
41	051	29	)	33	
42	052	2A	*	29	
43	053	2B	+	28	
44	054	2C	,	39	
45	055	2D	-	25	
46	056	2E	.	31	7
47	057	2F	/	30	
48	060	30	0	32	15
49	061	31	1	64	13
50	062	32	2	65	14
51	063	33	3	57	5
52	064	34	4	56	20
53	065	35	5	49	12
54	066	36	6	48	4
55	067	37	7	41	10
56	070	38	8	40	11
57	071	39	9	33	3
58	072	3A	:	29	
59	073	3B	;	28	
60	074	3C	<	39	
61	075	3D	=	25	
62	076	3E	>	31	
63	077	3F	?	30	
64	100	40	@	26	
65	101	41	A	69	
66	102	42	B	54	

67	103	43	C	62
68	104	44	D	61
69	105	45	E	58
70	106	46	F	53
71	107	47	G	52
72	110	48	H	44
73	111	49	I	35
74	112	4A	J	45
75	113	4B	K	37
76	114	4C	L	36
77	115	4D	M	38
78	116	4E	N	46
79	117	4F	O	34
80	120	50	P	27
81	121	51	Q	67
82	122	52	R	50
83	123	53	S	60
84	124	54	T	51
85	125	55	U	42
86	126	56	V	55
87	127	57	W	59
88	130	58	X	63
89	131	59	Y	43
90	132	5A	Z	71
91	133	5B	[	17
92	134	5C	\	22
93	135	5D	]	19
94	136	5E	↑	24
95	137	5F		32
96	140	60	‾	
97	141	61	a	69
98	142	62	b	54
99	143	63	c	62
100	144	64	d	61
101	145	65	e	58
102	146	66	f	53
103	147	67	g	52
104	150	68	h	44
105	151	69	i	35
106	152	6A	j	45
107	153	6B	k	37

108	154	6C	l	36
109	155	6D	m	38
110	156	6E	n	46
111	157	6F	o	34
112	160	70	p	27
113	161	71	q	67
114	162	72	r	50
115	163	73	s	60
116	164	74	t	51
117	165	75	u	42
118	166	76	v	55
119	167	77	w	59
120	170	78	x	63
121	171	79	y	43
122	172	7A	z	71
123	173	7B	{	17
124	174	7C	|	26
125	175	7D	}	19
126	176	7E	~	

Graphik-Zeichen des >> CPC 464 <<

ASCII-Werte größer 126 definieren Graphikzeichen. Der höchste zulässige ASCII-WERT ist hierbei 255.

Um die Graphik-Symbole und die Sonderzeichen auf dem Monitor mit der PRINT-Anweisung direkt zu schreiben, geben Sie bitte nachstehendes Programm ein und starten es mit RUN:

Programm: GRAPHIK-ZEICHEN

```
10 REM **** ASCII-Wert und GRAPHIK-Zeichen ****
20 MODE 2
30 LOCATE 15,1
40 PRINT "GRAPHIK-ZEICHEN des >> CPC 464 <<"
50 PRINT STRING$(79,"_")
60 FOR I=126 TO 255
70 PRINT I;"=";CHR$(I),
80 NEXT I
90 GOTO 90
```

26. ASCII-Werte Tabelle

Der in der Tabelle aufgeführte Tastencode wird bei den Anweisungen INKEY, KEY DEF und JOY als Angabe bzw. Abfrage verwendet. Vergleichen Sie die Angabe und das Format hierzu bei den entsprechenden Anweisungen.

Die Farben werden bei der BORDER-Anweisung direkt mit dem numerischen Wert des Farbcodes angegeben. Für die Anweisungen CLG, DRAW, DRAWR, PAPER, PEN, PLOT, PLOTR ist der Code der zweiten Tabelle zu verwenden.
Mit der INK-Anweisung kann die Hintergrundfarbe und die Zeichenfarbe nach dem gewählten Code der Code-Tabelle bestimmt und gesetzt werden.
Der angegebene Farbcode der ersten Tabelle ist nicht abhängig von der Formatanweisung MODE.
Die über den Code der zweiten Tabelle gewählte Farbe ist von der MODE-Anweisung beeinflußt. Nach dem Einschalten des >> CPC 464 << benutzt PAPER den Code 0 und PEN den Code 1, d.h. die Farbe des Bildschirmfensters ist gleich 1 (blau) und die der Zeichen ist 24 (hellgelb).

Farbcode - Tabelle

Farbcode	Farbe	Farbcode	Farbe
0	schwarz	14	pastellblau
1	blau	15	orange
2	hellblau	16	rosa
3	rot	17	pastellmagenta
4	magenta	18	hellgrün
5	hellviolett	19	seegrün
6	hellrot	20	hellcyan
7	purpur	21	lindgrün
8	hellmagenta	22	pastellgrün
9	grün	23	pastellcyan
10	cyan	24	hellgelb
11	himmelblau	25	pastellgelb
12	gelb	26	hellweiß
13	weiß		

Code - Tabelle

<Code>			MODE 0	MODE 1	MODE 2
0	=	Farbcode	1	1	1
1	=	Farbcode	24	24	24
2	=	Farbcode	20	20	1
3	=	Farbcode	6	6	24
4	=	Farbcode	26	1	1
5	=	Farbcode	0	24	24
6	=	Farbcode	2	20	1
7	=	Farbcode	8	6	24
8	=	Farbcode	10	1	1
9	=	Farbcode	12	24	24
10	=	Farbcode	14	20	1
11	=	Farbcode	16	6	24
12	=	Farbcode	18	1	1
13	=	Farbcode	22	24	24
14*	=	Farbcode	1><24	20	1
15*	=	Farbcode	16><11	6	24

Die mit "*" gekennzeichneten Codes bewirken im MODE 0 ein Wechseln zwischen den zwei Farben mit den angegebenen Farbcodes "><".

In der nachstehenden alphabetischen Aufstellung finden Sie alle BASIC-Kommandos, Anweisungen und geschützten BASIC-Worte, die der Interpreter als Anweisungen bzw. als Kommandos interpretiert. Die Worte heißen geschützt, da sie nicht für Variablen- bzw. Funktionsnamen verwendet werden dürfen.

Buchstabe: A

ABS	AFTER	AND	ASC
ATN	AUTO		

Buchstabe: B

BIN$	BORDER		

Buchstabe: C

CALL	CAT	CHAIN	CHAIN MERGE
CHR$	CINT	CLEAR	CLG
CLOSEIN	CLOSEOUT	CLS	CONT
COS	CREAL		

Buchstabe: D

DATA	DEF FN	DEFINT	DEFREAL
DEFSTR	DEG	DELETE	DI
DIM	DRAW	DRAWR	

Buchstabe: E

EDIT	EI	ELSE	END
ENT	ENV	EOF	ERASE
ERL	ERR	ERROR	EVERY
EXP			

Buchstabe F:

FIX	FN	FOR	FRE

Buchstabe G:

GOSUB	GOTO

Buchstabe H:

HEX$	HIMEM

Buchstabe I:

IF	INK	INKEY	INKEY$
INP	INPUT	INSTR	INT

Buchstabe J:

JOY

Buchstabe K:

KEY	KEY DEF

Buchstabe L:

LEFT$	LEN	LET	LINE INPUT
LIST	LOAD	LOCATE	LOG
LOG10	LOWER$		

Buchstabe M:

MAX	MEMORY	MERGE	MID$
MIN	MOD	MODE	MOVE
MOVER			

Buchstabe N:

NEW	NEXT	NOT

Buchstabe O:

ON	ON BREAK GOSUB	ON BREAK STOP	ON ERROR GOTO
ON GOSUB	ON GOTO	ON SQ GOSUB	OPENIN
OPENOUT	OR	ORIGIN	OUT

Buchstabe P:

PAPER	PEEK	PEN	PI
PLOT	PLOTR	POKE	POS
PRINT	PRINT USING		

Buchstabe R:

RAD	RANDOMIZE	READ	RELEASE
REM	REMAIN	RENUM	RESTORE
RESUME	RETURN	RIGHT$	RND
ROUND	RUN		

Buchstabe S:

SAVE	SGN	SIN	SOUND
SPACE$	SPC	SPEED INK	SPEED KEY
SPEED WRITE	SQ	SQR	STEP
STOP	STR$	STRING$	SWAP
SYMBOL	SYMBOL AFTER		

Buchstabe T:

TAB	TAG	TAGOFF	TAN
TEST	TESTR	THEN	TIME
TO	TROFF	TRON	

Buchstabe U:

UNT	UPPER$	USING

Buchstabe V:

VAL	VPOS

Buchstabe W:

WAIT	WEND	WHILE	WIDTH
WINDOW	WINDOW SWAP	WRITE	

Buchstabe X:

XOR	XPOS

Buchstabe Y:

YPOS

Buchstabe Z:

ZONE

* CPU — Z80A mit 4 MHz Taktfrequenz

Z80 CPU - der weltweit meistverwendete Mikroprozessor in der Homecomputer-Klasse, mit dem ein Großteil der vorhandenen Software genutzt werden kann. Das Betriebssystem CP/M ist nutzbar.

* Arbeitsspeicher: 64 KByte RAM, davon unter BASIC für den Benutzer frei verfügbar 43533 Bytes.

32 KByte ROM Systemspeicher

Der >> CPC 464 << verfügt standardmäßig über 64 KByte RAM, von denen über 42 KByte dem Benutzer zur Verfügung stehen. Dies wird durch die ROM Overlay-Technik erreicht. Der zur Verfügung stehende Speicherplatz erlaubt es, auch größere Programme zu entwickeln.

* Tongenerator: AY-3-8912
- 3 Kanäle
- 8 Oktaven

Musik aus 3 Kanälen, vorhanden sind 8 Oktaven. Für jeden Kanal kann unabhängig Tonhöhe und Lautstärke über eine entsprechende Hüllkurve gewählt werden. Die Ausgänge zum Anschluß an eine Stereo-Anlage sind vorhanden. Intern wird der Ton über einen MONO-Lautsprecher wiedergegeben.

* Externer Speicher: Integrierter Datacorder
- softwaremäßig steuerbar
- Aufzeichnung 1000 bzw. 2000 Baud

Ein Magnetbandspeicher steht mit dem eingebauten Datacorder zur Verfügung. Damit können Programme gelesen und abgespeichert werden, ohne sich mit den leidigen Problemen der Kompatibilität, des Anschließens und des richtigen Pegels abgeben zu müssen. Programme können entweder mit 1K Baud oder 2 K Baud geschrieben werden. Die Lesegeschwindigkeit wird softwaremäßig automatisch eingestellt.

* Bildschirm: 6845 Bildschirmsteuerung (CRTC)
- Modi 3; 20, 40, 80 Spalten
- Graphik max. 640 x 200 Punkte
- Farben 27
  darstellbar max. 16

Bis zu 8 Bildschirmfenster sind frei und überlappend definierbar. Zudem noch ein Graphikfenster.
80-spaltiger Textmodus, 256 Zeichen frei definierbar, eine Palette von 27 Farben, gleichzeitig darstellbar sind max. 16 Farben. Die Auflösung beträgt max. 640 x 200 Bildpunkte.

* Schnittstellen: Centronics-kompatibel (parallel)
Joystick 9-polig
RGB und Sync
Stereo-Ausgang

Ein Centronics-kompatibler Druckeranschluß nach Standard ist vorhanden. Die Busy Signale werden für handshake verwendet.

* Tastatur: Schreibmaschinenähnlich mit 74 Tasten
abgesetzter Ziffernblock
Sondertasten
Funktionstasten zur freien Belegung

Bis zu 120 Zeichen können auf eine Funktionstaste gelegt werden. Vier Cursor-Funktionstasten, abgesetzter Ziffernblock, COPY-, CAPS-LOCK-, ESC-, CLR-, TAB-, CTRL- und DEL-Taste sowie zwei ENTER-Tasten sind vorhanden.

* Erweiterungen: ROM-Erweiterungen bis zu 240 Bausteine.

Über die Firmware können bis zu 240 ROM-Erweiterungen zu je 16 K angesprochen werden.

RAM-Erweiterung bis 9 MByte

Wahlweise anschließbare Diskettenstation

* BASIC-Interpreter	Sehr interessanter BASIC-Befehlssatz Grundbefehlssatz entspricht dem BASIC-Standard-Befehlssatz! - mehr als 150 BASIC-Befehle - erweitert in vielen Bereichen! - Sound-, Graphik-BASIC-Befehle

DAS STANDARD BASIC-BUCH zum Schneider-Computer CPC 464
E. Unger Best.-Nr. B-201

NEUERSCHEINUNG. Das Standardwerk zum Schneider-Computer **CPC 464!**

Auf ca. 290 Seiten eine klare und verständliche Einführung in die Programmiersprache BASIC mit allen BASIC-Kommandos und einer ausführlichen Beschreibung der Arbeit mit dem Datacorder, dem Cassettenteil des Computers. Das Buch überschüttet den Lernenden nicht mit einer Fülle von Informationen, sondern führt ihn behutsam an den Lernstoff heran. Der Leser wird in die Lage versetzt, unmittelbar am Computer arbeitend, das Gelernte in die Praxis umzusetzen. Die Anleitungen sind durch die Art der Aufgabenstellung und die schrittweise Darstellung der Lösungswege auf die Bedürfnisse des Selbstunterrichts abgestellt.

Das Buch ist praxisbezogen: Schon nach wenigen Kapiteln ist der Leser imstande BASIC-Programme zu verstehen und eigene kleine Programme zu schreiben. Der Einstieg in die neue Materie **„Logik des Programmierens"** wird Ihnen durch sorgfältige Aufbereitung des Lehrstoffes so leicht wie möglich gemacht.

Alle englischen Fehlermeldungen des **CPC464-BASIC** werden in das Deutsche übersetzt und eingehend erläutert. Die möglichen Fehlerursachen werden aufgezeigt und – wenn nötig – durch Programm-Beispiele erklärt.

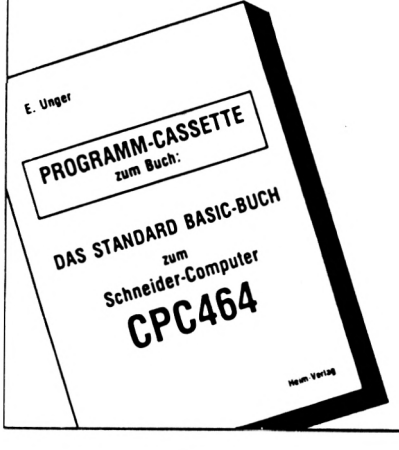
Ein Spitzenbuch mit über 50 praxisnahen Übungs- und Anwenderprogrammen.

Alle diese Programme finden Sie auf einer **Cassette**, die Ihnen die riesige Mühe des Eintippens erspart. Lesen Sie dazu auch die Beschreibung zur **Programm-Cassette** in diesem Prospekt.

Sie erhalten ein umfassendes und außerordentlich reichhaltiges Fachbuch, das Ihnen das Tor zu Ihrem neuen Computer, dem Schneider **CPC464** öffnet. Ein Buch also, das Sie begeistern wird.

Buch und Cassette im Verbund sind die richtigen Begleiter auf Ihrem Weg zum Experten am Schneider-Computer **CPC464.**

ca. 290 Seiten 68,– DM



DIE PROGRAMM-CASSETTE zum Buch:

DAS STANDARD BASIC-BUCH zum Schneider Computer **CPC464**
E. Unger Best.-Nr. C-204

Die **Programm-Cassette** enthält die Beispielprogramme des **STANDARD BASIC-BUCHS** und ist hervorragend geeignet, Ihnen die praktische Arbeit und das Training am Computer ganz wesentlich zu erleichtern. **PROGRAMM-CASSETTE** laden und mit dem Schneider-Computer üben, das macht den Meister.

Die Palette der Programme geht von einfachen BASIC-Übungsprogrammen bis zu nützlichen Anwenderprogrammen wie zum Beispiel:

- ★ **Sortierprogramm**
- ★ **Programm zur Umrechnung fremder Währungen**
- ★ **Programm zum Erstellen eines beliebigen Zeichenvorrats** (Tasten können beliebig belegt werden)
- ★ **Dienstprogramme** (Hiermit können sämtliche mathematischen, chemischen, physikalischen sowie internationale Zeichensätze mit dem Computer vereinbart und genutzt werden)
- ★ **Textverarbeitung** (Schreiben von Texten, Ablegen und Einlesen eines Briefzeilenprogrammes, Einfügungen, Druckerausgabe)
- ★ **Kartei** (Verwalten von bis zu 200 Adressen, elegante Benutzerführung, Adresseneingabe im Dialog, Sichern der Adressen auf dem Datacorder, Einlesen von gespeicherten Adressdateien, Suchen von Adressen nach frei wählbaren Suchbegriffen, Blättern der Adressdatei, Löschen einzelner Adressen, Neueingabe von Adressen
- ★ Mein Kassenbuch (Einnahmen-Ausgaben Abrechung)
- ★ Mein Telefonbuch (Auswahl aus erfaßten Tel.-Nummern nach Namen und Rufnummern)
- ★ und viele nützliche Programme mehr...

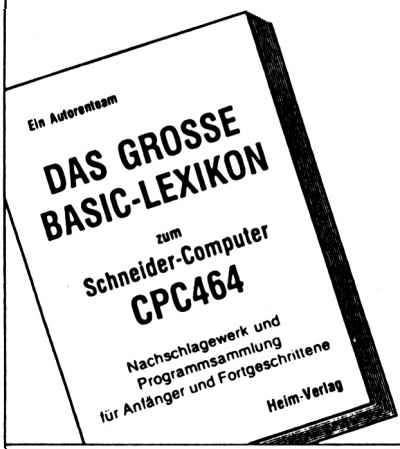
Die Cassette enthält über 50 Programme, die Sie nicht mehr mit riesiger Mühe einzutippen brauchen. Vor allem aber ersparen Sie sich viele Fehler und viele Korrekturen. Sie können **„Gleich zur Sache"** kommen.

Die Cassette hilft Ihnen, schon nach kurzer Zeit BASIC-Programme zu verstehen, selbst zu verändern (erweitern oder kürzen) und so schneller als sonst üblich, eigene kleine Programme zu schreiben.

Auch diese Cassette wird Sie begeistern, sie gehört zu Ihrem Schneider-Computer genauso wie das **STANDARD BASIC-BUCH.**

Weniger als 1,50 DM je Programm

Über 50 Übungs- und Anwenderprogramme **74,– DM**



DAS GROSSE BASIC-LEXIKON zum Schneider-Computer CPC464

NACHSCHLAGEWERK UND PROGRAMMSAMMLUNG FÜR ANFÄNGER UND FORTGESCHRITTENE

Erstellt von einem Autorenteam — Best.-Nr. B-203

NEUERSCHEINUNG. Mit der Erfahrung viele Programmiererjahre hat das Autorenteam für Sie ein Nachschlagewerk von besonderer Qualität geschaffen.

Im **GROSSEN BASIC-LEXIKON** finden Sie den gesamten umfangreichen Befehlssatz (ca. 180 Befehle und Funktionen) des Schneider-Computers **CPC464**. Es kommt aber noch viel mehr hinzu.

Alle Befehle und Funktionen sind nach einem klaren, leicht verständlichen Schema in alphabetischer Ordnung wie folgt dargestellt:

1. **BASIC**-Schlüsselwort
2. **FORMAT** (Schreibvorschrift))
3. **ZWECK** (Wofür wird das Schlüsselwort benötigt)
4. **ANWENDUNG** (Ausführliche Erläuterung des Schlüsselwortes und Vorbereitung des Programmbeispiels)
5. **PROGRAMM-BEISPIEL**
6. **ERGEBNIS** (Das Ergebnis des Beispielprogramms wird – wenn nötig – besprochen)
7. **VERGLEICHSHINWEISE** (Es wird auf vergleichbare Schlüsselwörter verwiesen)

Dieses Nachschlagewerk hat also neben seinem Charakter als Lexikon noch den unschätzbaren Vorteil, auch eine wertvolle Programmsammlung zu sein mit einem hohen Nutzwert für Sie als Programmierer.

Sie können zu jedem Zeitpunkt eingehende Informationen zum gewünschten Schlüsselwort nachschlagen.

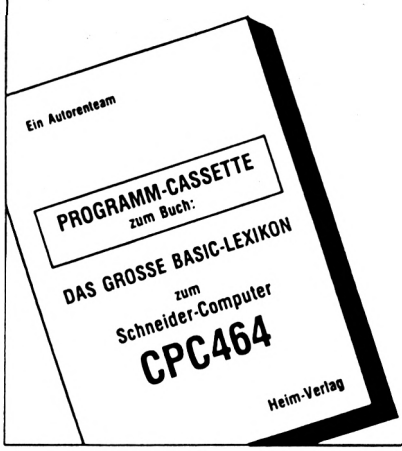
Eine Stärke des Buches sind die auf das Schlüsselwort ausgerichteten Programm-Beispiele, denn dort finden Sie – eingebettet in das Programm – die Art und Weise, wie das Schlüsselwort zu benutzen ist.

Deshalb ist beim Lernen und Erstellen eigener Programme für Ihren Schneider **CPC 464 DAS GROSSE BASIC-LEXIKON** eine wertvolle und unerläßliche Hilfe.

Alle diese Programme finden Sie ebenfalls auf einer **PROGRAMM-CASSETTE**. Damit haben Sie auch hier wieder die Möglichkeit auf bequeme Art und Weise am Schneider Computer zu üben und zu lernen.

Eine wertvolle Buchseite für weniger als 30 Pfennige.

ca. 200 Seiten — 58.– DM



DIE PROGRAMM-CASSETTE zum Buch:

DAS GROSSE BASIC-LEXIKON zum Schneider-Computer CPC 464

Erstellt von einem Autorenteam — Best.-Nr. C-205

DIE PROGRAMM-CASSETTE enthält die Beispielprogramme des **GROSSEN BASIC-LEXIKONS** und ist hervorragend geeignet, Ihnen die praktische Arbeit und das Training am Schneider-Computer ganz wesentlich zu erleichtern.

Über 150 Programmbeispiele erläutern Ihnen am Bildschirm, wie jedes Programm-Schlüsselwort sachgerecht in entsprechende Programm-Anweisungen eingebracht wird.

Sie haben somit die Möglichkeit in alphabetischer Ordnung gezielt auf das gewünschte Schlüsselwort zuzugreifen und dort das geeignete Beispiel für Ihren Anwendungsfall zu finden.

In dem der Cassette beigefügten Bedienungsheft finden Sie Vorschläge für die Nutzung der Cassette beim Lernen und Üben. So enthält dieses Heft ausführliche Hinweise zur gezielten Wiederholung des Lernstoffes in zusammengehörigen Gruppen verwandter Schlüsselwörter.

Auch diese Cassette ist eine wertvolle Ergänzung Ihrer Lernmittel aus dem **Heim-VERLAG** zum Schneider-Computer **CPC464**.

Weniger als 50 Pfennige je Programmbeispiel.

Über 150 Programmbeispiele — 74.– DM

BASIC leicht und schnell gelernt am Schneider-Computer CPC464
Prof. Dr. W. Voß — Best.-Nr. B-202

NEUERSCHEINUNG. Ein weiteres Spitzenbuch zum Schneider **CPC464** für Einsteiger. Es wird keine Programmiererfahrung vorausgesetzt. Auf der Grundlage langjähriger Unterrichtserfahrung hat der Autor 16 Lerneinheiten entwickelt. Zugeschnitten auf den Schneider **CPC 464** wird der Leser Schritt für Schritt in die „Geheimnisse der BASIC-Programmierung eingeführt.

Dies gelingt deshalb so besonders gut, weil das Erlernen der notwendigen Grundkenntnisse vor allem anhand einfacher Programmierbeispiele erfolgt (Jeder weiß: „Übung macht den Meister!").

Anhand praktischer Beispiele wird unter Verzicht auf das zumeist unverständliche „Fach-Chinesisch" der Leser mit seinem Schneider **CPC464** umgehen und ihn beherrschen lernen.

In allen Themen wurde auf eine sorgfältige und leicht verständliche Aufbereitung großer Wert gelegt: Noch nie war es einfacher, die Programmiersprache BASIC in ihren Grundlagen zu erlernen.

Eine wertvolle Buchseite für weniger als 23 Pfennige.

ca. 300 Seiten — **68,– DM**

AUTOREN GESUCHT

Sie

...sind an mehr Information interessiert

...können Programme zur Veröffentlichung anbieten

...können über Anwendungen berichten

...wollen über ein Thema zu Microcomputern schreiben

Wir der *Heim*-Verlag suchen

...Ihre Erfahrungen an

...Ihre Programme aus allen Bereichen

...Ihre praktischen Anwendungen

...Ihre Berichte, Tips + Tricks

...Ihre selbstprogrammierten Spiele

...Routinen und Utilitys, die sich Anwender selbst geschrieben haben

Bei Veröffentlichung zahlen wir ein HONORAR.

Schreiben Sie uns.

Bücher und Programm-Disketten aus dem ***Heim*-Verlag** gibt es bei allen FACHHÄNDLERN, in den Computer-Abteilungen der KAUFHÄUSER und im BUCHHANDEL.

***Heim*-Verlag · Heidelberger Landstr. 194**
6100 Darmstadt-Eberstadt · ✆ 0 61 51-5 53 75

E. Unger

DAS STANDARD BASIC-BUCH zum

Ein BASIC-Lehrgang im Dialog

Eine Edition aus dem Heim-Verlag

Heim

Standard-Basic-Buch · Schneider-Computer CPC 464

E. Unger

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/

[FRA] Ce document a été préservé numériquement à des fins éducatives et d'études, et non commerciales.
[ENG] This document has been digitally preserved for educational and study purposes, not for commercial purposes.
[ESP] Este documento se ha conservado digitalmente con fines educativos y de estudio, no con fines comerciales.