

SZCZEPANOWSKI

**CPC
6128
FÜR
EINSTEIGER**

EIN DATA BECKER BUCH

SZCZEPANOWSKI

**CPC
6128
FÜR
EINSTEIGER**

EIN DATA BECKER BUCH

ISBN 3-89011-143-2

Copyright © 1985 DATA BECKER GmbH
Merowingerstraße 30
4000 Düsseldorf

Alle Rechte vorbehalten. Kein Teil dieses Buches darf in irgendeiner Form (Druck, Fotokopie oder einem anderen Verfahren) ohne schriftliche Genehmigung der DATA BECKER GmbH reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.*

Wichtiger Hinweis:

Die in diesem Buch wiedergegebenen Schaltungen, Verfahren und Programme werden ohne Rücksicht auf die Patentlage mitgeteilt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt und dürfen nicht gewerblich genutzt werden.

Alle Schaltungen, technischen Angaben und Programme in diesem Buch wurden von dem Autoren mit größter Sorgfalt erarbeitet bzw. zusammengestellt und unter Einschaltung wirksamer Kontrollmaßnahmen reproduziert. Trotzdem sind Fehler nicht ganz auszuschließen. DATA BECKER sieht sich deshalb gezwungen, darauf hinzuweisen, daß weder eine Garantie noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernommen werden kann. Für die Mitteilung eventueller Fehler ist der Autor jederzeit dankbar.

KAPITEL 1:**DIE TASTATUR**

Allgemeines zur Tastatur	6
Los geht's	7
Drei-Finger-Griff	8
Die Cursor-Tasten	9
Editieren mit den Cursor-Tasten	13
Löschen des Bildschirms	16
Die Tasten mit Buchstaben	17
Groß/Kleinschrift	19
Die Leertaste	21
Die DEL-Taste	22
Die CLR-Taste	26
Einstellbare Zeichendichte	28
Der Kopiercursor	30

KAPITEL 2:**DER ERSTE BEFEHL**

Die RETURN-Taste	41
Der PRINT-Befehl	43
Rechnen mit PRINT	44
Die Klammerrechnung	48
Exponentialschreibweise	49
Textausgabe mit PRINT	51
Vereinfachte PRINT-Eingabe	53
PI und Potenzierung	54
Kombinieren von Strings mit Zahlen	57
Trennen von Befehlen	59

KAPITEL 3:**DAS ERSTE PROGRAMM**

Ein Programm, was ist das?	62
Die Zeilennumerierung	63
Programmstart	66
Programmänderung	67
Verzweigung	72
Speichern und Laden von Programmen	74
Löschen eines Programms	76

KAPITEL 4:**PROGRAMMIERHILFEN**

Automatische Zeilennumerierung	77
Umnummerierung von Zeilen	79
Gezieltes Löschen von Zeilen	83
Funktionstasten	85

KAPITEL 5:**BASIC EINFÜHRUNG**

Problembeschreibung zur Adressenverwaltung	89
Dateiorganisation	90
Rechnerinterne Speicherung der Daten	92
Variablen	93
Variablenverarbeitung	94
Tabellen	96
Dateneingabe über Tastatur	101
Schleifen	103
Erste Reaktionen des Programms	108
Unterprogramme	111
Das Menue	113

Die Abfrage mit IF	115
ASCII-Code	118
Berechnetes GOTO	120
Adressen eingeben	123
Adressen ändern	127
Adressen löschen	132
Adressen ausgeben	136
Datei sichern	142
Datei laden	144
Programm beenden	146

KAPITEL 6:**FARBE UND GRAFIK**

Die Grafikbetriebsarten	148
Die Bildschirmfarben	149
Die Rahmenfarbe	150
Einstellen der Blinkgeschwindigkeit	152
Ändern der Zeichenfarbe	153
Ändern der Hintergrundfarbe	158
Hochauflösende Grafik	160
Zeichnen von Punkten	161
Zeichnen eines Kreises	162
Zeichnen von Linien	164
Relatives Zeichnen	166

KAPITEL 7:**TONERZEUGUNG**

Der SOUND-Befehl	169
Die Tonhöhe	170
Musik aus Frequenzen	171
Rauschen	175

KAPITEL 8:**DAS DISKETTENLAUFWERK**

Die Disketten	177
Das Formatieren	178
Dateinamen	179
Programme als ASCII-Dateien	180
Geschützte Programme	182
Sichern von Speicherausügen	184
Programmverknüpfung	185
Löschen von Dateien	188
Umbenennen von Dateien	189

KAPITEL 9:**NOCH MEHR BEFEHLE**

Joystickabfrage	190
Zufallszahlen	192
LEFT\$	195
RIGHT\$	196
MID\$	197
Ermitteln der Stringlänge	198
INSTR	200

ANHANG

Reservierte Worte	203
Fehlermeldungen	205
Stichwortverzeichnis	210

Kapitel 1: DIE TASTATUR



In diesem Kapitel lernen Sie die Handhabung der 74 Tasten Ihres CPC6128. Mit Hilfe dieser Tasten werden Sie alle verborgenen Möglichkeiten Ihres Heimcomputers erschließen. Sei es die Eingabe von Texten, das Erstellen von Bildern (Grafiken), das Abschicken von Anweisungen oder die Berechnung mathematischer Probleme, all dies wird für Sie nach diesem Kapitel kein Neuland mehr sein. Wenn Sie einmal alle Tasten beherrschen, wird der CPC6128 Ihnen ein folgsamer Partner bei der Lösung vieler Probleme sein.

Das Beherrschen der Tastatur dient nicht nur Computer-Freaks, die einmal jedes Bit des Rechners mit dem Vornamen kennen möchten, sondern auch denen, die ihren Computer nur mit den reichhaltig angebotenen Fertigprogrammen "füttern" wollen. Jemand, der sich zu den zuletzt genannten Anwendern zählt, darf bei einer Meldung eines Standardprogramms wie z.B. "DRUECKEN SIE DIE RETURN-TASTE ZUM AUSDRUCK DES BILDSCHIRMINHALTS" nicht verzweifelt in seinem Handbuch wühlen.

Kurzum sollte jeder, der auch nur gelegentlich an seinem Gerät arbeitet, mit der Tastatur vertraut sein. Der Vater eines computerfaszinierten Sohnes z.B. sollte zumindest wissen, wie das beste Spielprogramm des Sohnmanns geladen wird, wenn der gerade außer Haus ist.

Doch keine Angst, Sie müssen sich nicht vorher zu einem Schreibmaschinenkursus an der Volkshochschule anmelden. Die meisten auch noch so erfahrenen Freizeitprogrammierer arbeiten mit dem "Zweifinger-Suchsystem". Wer eine gewisse Zeit mit der Tastatur vertraut ist, wird sich bald darüber wundern, wie flink er über die Tasten saust.

In diesem Kapitel werden die weißen Hilfstasten besonders ausführlich beschrieben. Diese Tasten sind sehr wichtig, da z.B. Texte bearbeitet, Programme unterbrochen und Befehle an den Rechner übermittelt werden können.

ALLGEMEINES ZUR TASTATUR

Auf den ersten Blick erweckt die Tastatur des CPC6128 den Eindruck einer üblichen Schreibmaschinentastatur. Doch wenn Sie genauer hinsehen, werden Sie leichte Abweichungen feststellen:

Die Buchstaben 'Y' und 'Z' sind gemäß der amerikanischen ASCII-Norm vertauscht.

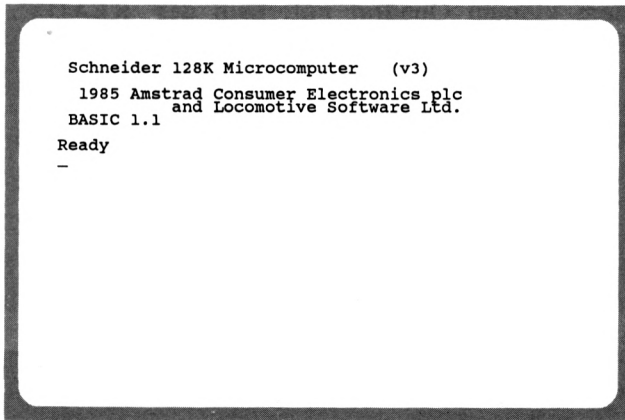
Die Tastatur weist keine Umlaute (ö,ä,ü) und kein scharfes 's' (ß) vor, da diese Zeichen im amerikanischen Zeichensatz nicht enthalten sind.

Es gibt zusätzliche Tasten (Hilfstasten), deren Funktion später erläutert wird.

Sie sollten keine Experimente mit der Tastatur machen, bevor Sie nicht mit deren Funktion vertraut sind. Zwar können durch Fehlbedienungen keine Rauchwolken aus dem Rechner aufsteigen, doch Sie können durch überraschende Mißerfolge

verblüfft werden. Warten Sie also geduldig auf die praktische Einweisung auf den nächsten Seiten.

LOS GEHT'S



```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
  BASIC 1.1 and Locomotive Software Ltd.
Ready
-
```

Nun wird es Zeit, die schlafende Technik in Ihrem CPC6128 zum Leben zu erwecken. Schalten Sie dazu den Rechner ein. Sie finden zwei Schalter, einen am Monitor und einen am Rechner. Beide müssen eingeschaltet werden. Nach jedem Einschalten erscheint eine Meldung, die besagt, daß der Rechner nun bereit ist, Ihren Kommandos zu folgen. Die erste Zeile der Meldung weist darauf hin, daß Sie mit einem SCHNEIDER-Computer arbeiten, der 128 KByte (ca 131070 Zeichen) Hauptspeicher enthält. Von diesen 131070 Speicherzellen stehen Ihnen ca 40000 für eigene BASIC-Programme zur Verfügung. Der Rest der ersten 64 KByte des Speichers wird vom Rechner für das Betriebssystem und die Programmiersprache BASIC benötigt. Die zweiten 64KByte können als Datenspeicher genutzt werden.

Die nächste Zeile besagt die eigentliche Herkunft des Rechners, die Firma AMSTRAD in Großbritannien. Von der Firma LOCOMOTIVE SOFTWARE stammt das integrierte BASIC, das, wie sich später noch herausstellen wird, überaus leistungsfähig ist. Dieses BASIC hat die Versionsnummer 1.1. Sollten einmal

Änderungen oder Verbesserungen vorgenommen werden, so ändert sich auch diese Nummer entsprechend. In der letzten Zeile besagt die Meldung 'READY', daß das Betriebssystem nun Kommandos erwartet. Diese READY-Meldung signalisiert, daß Ihnen der Rechner zur Verfügung steht.

Nun zu dem kleinen Quadrat unterhalb des 'READY'. Dies ist eine Orientierungshilfe auf dem Bildschirm. Angenommen, Sie möchten etwas auf dem Bildschirm schreiben, dann ist es notwendig, genau zu wissen, wo das einzugebene Zeichen erscheinen wird. Diese Markierung, die man in der Fachsprache CURSOR ("körsa" gesprochen) nennt, hilft Ihnen also, sich am Bildschirm zurechtzufinden.

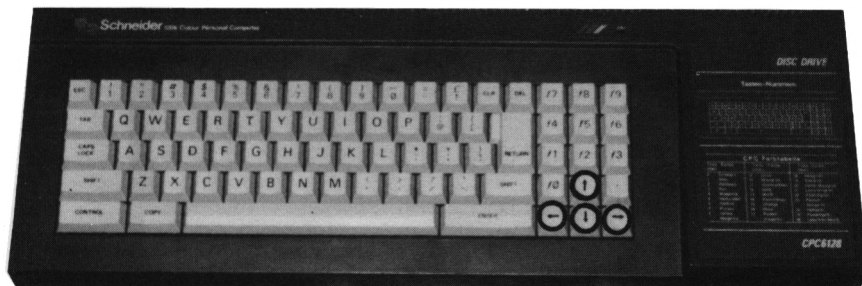
DREI-FINGER-GRIFF



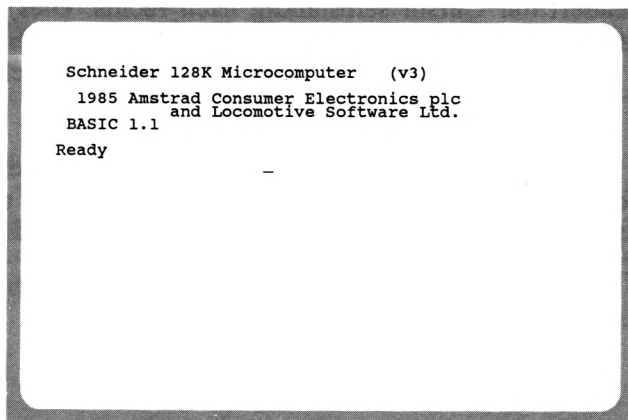
In vielen Situationen ist es notwendig, den Rechner in den Einschaltzustand zu versetzen. Bei vielen anderen Computern muß dieser dazu ein- und wieder ausgeschaltet werden. Dies hat aber eine überflüssige Belastung für die hochwertige Elektronik des Rechners zur Folge. Um dies zu vermeiden, kann der CPC6128 mit einigen Tasten in den Einschaltzustand versetzt werden. Dazu müssen Sie die drei Tasten SHIFT-CTRL-ESC drücken. Achten Sie aber darauf, daß die Taste ESC zuletzt betätigt wird, während die Tasten SHIFT und CTRL gedrückt bleiben. Versuchen Sie nun einmal diesen sogenannten Drei-Finger-Griff. Sie bemerken, daß das gleiche Bild erscheint, das auch dem Einschalten des Rechners folgt.

Das Zurücksetzen des Rechners, auch RESET oder Kalt-Start genannt, sollte vorsichtig gehandhabt werden, das die BASIC-Programme stets dabei gelöscht werden.

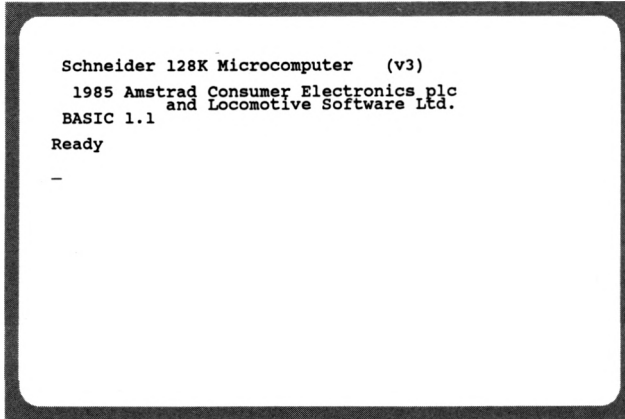
DIE CURSOR - TASTEN



Diese Tasten befinden sich rechts über dem Ziffernblock. Die Pfeile auf diesen Tasten entsprechen der Richtung, in der sich der Cursor auf dem Bildschirm bewegen soll. Betätigen Sie nun einmal die RECHTS-Taste 20 mal. Der Cursor wandert 20 Stellen nach rechts und verweilt wieder geduldig in der Bildschirmmitte.

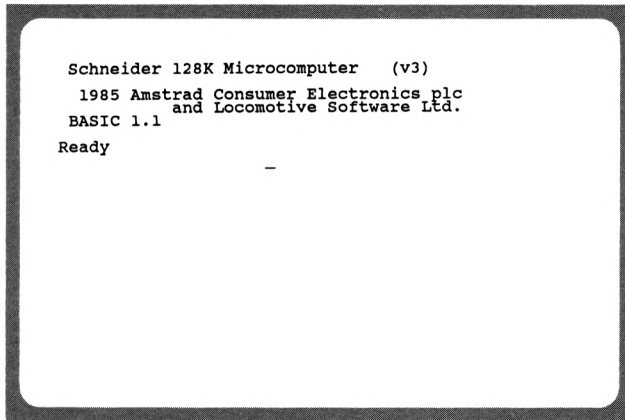


Was aber, wenn der Cursor den rechten Rand dieser Zeile überschreitet? Probieren Sie dies selbst aus, und drücken dazu nochmals 20 mal die RECHTS - Taste. Nach dem 20. Druck auf diese Taste erscheint der Cursor in der 1. Spalte der folgenden Zeile.



Es ist doch recht umständlich, 20 mal auf die Cursor-Taste zu hämmern, um in die Bildschirmmitte zu gelangen. Diese Prozedur können Sie etwas vereinfachen. Wenn Sie die Taste gedrückt halten, so wandert der Cursor selbständig nach rechts. Dies gilt auch für alle anderen Tasten. Probieren Sie dies aus, und halten Sie dazu die CURSOR-RECHTS - Taste gedrückt. Beobachten Sie, wie der Cursor zum rechten Bildschirmrand sprintet.

Um den Cursor nun in die andere Richtung zu bewegen, benutzen Sie die CURSOR-LINKS - Taste. Probieren Sie nun auch diese Taste aus. Der Cursor bewegt sich also jetzt in die andere Richtung.



```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
  BASIC 1.1 and Locomotive Software Ltd.
Ready
```

Selbstverständlich bewegt sich der Cursor auch selbständig nach links, ohne daß immer wieder auf die Taste gedrückt werden muß. Halten Sie dazu einfach die Taste gedrückt.

Wie kann man nun den Cursor wieder in die Ausgangsstellung (unterhalb 'READY') bringen? Dazu könnten Sie z.B. solange die **CURSOR LINKS** - Taste gedrückt halten, bis der Cursor Zeile für Zeile nach oben wandert. Doch dies ist nicht im Sinne des Erfinders. Die Cursor-Tasten **OBEN** und **UNTEN** ermöglichen Ihnen die ebenso schnelle Bewegung des Cursors nach oben und unten wie mit den bisher beschriebenen Tasten nach links und rechts.

Bewegen Sie nun den Cursor zum linken Bildschirmrand. Der Cursor wartet wiederum geduldig, bis Sie ihn an eine andere Stelle bewegen. Peilen Sie nun die **CURSOR-UNTEN** - Taste an und drücken Sie diese Taste dreimal. Sie haben bemerkt, daß der Cursor drei Zeilen nach unten gesprungen ist.

```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
  and Locomotive Software Ltd.
BASIC 1.1
Ready

-
```

Versuchen Sie nun den Cursor in die 3. Bildschirmzeile zu bewegen. Benutzen Sie dazu die CURSOR-OBEN - Taste.

```
Schneider 128K Microcomputer (v3)
- 1985 Amstrad Consumer Electronics plc
  and Locomotive Software Ltd.
BASIC 1.1
Ready
```

Auch hier wandert der Cursor selbständig Zeile für Zeile nach oben oder nach unten, wenn Sie die Taste gedrückt halten.

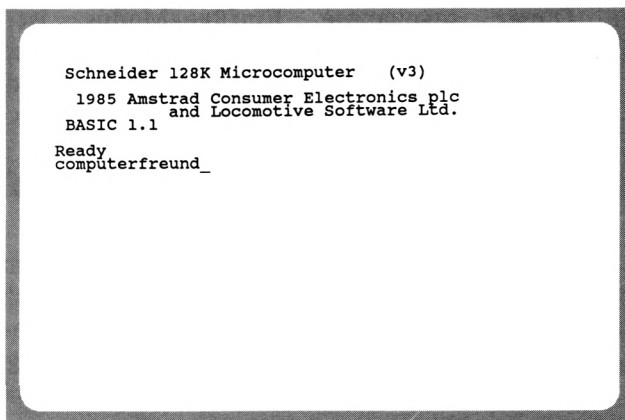
Um ein Gefühl für die Cursor-Tasten zu erlangen, versuchen Sie einmal, mit dem Cursor ein unsichtbares Quadrat zu zeichnen. D.h. Sie bewegen den Cursor rechts / unten / links / oben oder anders herum links / unten / rechts / oben. Das Beherrschen der Cursor-Tasten ist erstrebenswert, da Sie später z.B. Korrekturen

in einem Programm blitzschnell vornehmen können. Doch Übung macht den Meister, und der ist bekanntlich noch nicht vom Himmel gefallen.

EDITIEREN MIT DEN CURSOR-TASTEN

Hier taucht ein Fremdwort aus der Datenverarbeitung auf, dessen Bedeutung dem einen oder anderen Leser, besonders dem Anfänger, sicher nicht bekannt ist. Editieren bedeutet Bearbeiten von Texten, z.B. Texte erstellen und ändern. Man spricht bereits vom Editieren, wenn Sie ein paar Worte auf den Bildschirm schreiben.

Bei vielen anderen Rechnern können Sie den Cursor auf dem Bildschirm beliebig bewegen, Text eingeben und weiterbewegen. Der CPC6128 jedoch hat eine wichtige Eigenart. Sie können den Cursor solange beliebig bewegen, bis Sie irgendein Zeichen der Tastatur gedrückt haben. Von nun an können Sie den Cursor nur innerhalb der eingegebenen Zeichen wandern lassen. Dies ist etwas graue Theorie, die wir nun in die Praxis umsetzen. Versetzen Sie Ihren Rechner zunächst in den Einschaltzustand (SHIFT-CTRL-ESC). Schreiben Sie nun das Wort "computerfreund" auf dem Bildschirm, auch wenn das Suchen der Buchstaben bei Ihnen etwas länger dauert:



```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
  and Locomotive Software Ltd.
BASIC 1.1
Ready
computerfreund_
```

Nun verweilt der Cursor hinter dem letzten Buchstaben dieses Wortes und weigert sich strikt, irgendeine andere Richtung als links zu den eingegebenen Buchstaben einzuschlagen. Versuchen Sie's, auch wenn Sie wenig Erfolg haben werden. Der Cursor dient nun nur noch dazu, den eingegebenen Text zu bearbeiten. Dazu dürfen und können Sie den Text nicht mehr verlassen. Bewegen Sie den Cursor nun ganz nach links, indem Sie die LINKS-Taste gedrückt halten. Am linken Bildschirmrand angekommen stoppt der Cursor, als ob er gegen eine Wand schlägt. Die gleiche Reaktion erfolgt, wenn Sie wieder nach rechts wandern. Hinter dem letzten Buchstaben scheint auch eine Wand zu sein. Warum diese Einschränkung der Cursorbewegung? Um dies zu begründen, müssen Sie sich im Klaren sein, wozu Sie überhaupt etwas auf dem Bildschirm schreiben. Es gibt verschiedene Gründe, wie z.B. die Programmerstellung. Zum Schreiben eines Programms, wozu wir später noch kommen werden, geben Sie einzelne Programmzeilen ein. Jede Programmzeile wird mit einer bestimmten Taste (RETURN) dem Rechner übergeben. Es ist unsinnig, eine angefangene Programmzeile an irgendeiner anderen Stelle des Bildschirms fortzusetzen. Erst die Übergabe der Zeile gibt den Cursor wieder frei.

Kommen wir wieder zum "computerfreund" zurück. An jeder beliebigen Stelle dieses Wortes können Buchstaben eingefügt werden. Dazu muß der Cursor auf den Buchstaben positioniert werden, vor den ein weiterer Buchstabe oder Zeichen eingefügt werden soll. Dieser sogenannte Einfüge-Modus ist ständig eingeschaltet. Bei manchen anderen Rechnern muß dieser Modus zunächst eingeschaltet werden. Ändern wir nun das Wort "computerfreund" um in "heimcomputer-freund". Bewegen Sie dazu den Cursor an den linken Rand des Bildschirms.

```
Schneider 128K Microcomputer (v3)
 1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.
BASIC 1.1
Ready
computerfreund
```

Der Cursor befindet sich nun auf dem Buchstaben 'c', vor dem noch die Silbe "heim" eingegeben werden muß. Drücken Sie dazu nun nacheinander die Buchstaben 'h', 'e', 'i' und 'm'.

```
Schneider 128K Microcomputer (v3)
 1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.
BASIC 1.1
Ready
heimcomputerfreund
```

Mit jedem Buchstaben wird der Text links vom Cursor um eine Stelle nach links geschoben. Der Cursor bleibt jedoch auf dem 'c'. Nun müssen wir noch den Bindestrich einfügen. Sicher wissen Sie schon längst, wie dies geschieht. Sie setzen den Cursor auf das 'f' und drücken die Taste '-' (rechts von der Taste '0').

```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
  and Locomotive Software Ltd.
BASIC 1.1
Ready
heimcomputer-freund
```

Wenn der Cursor nun wieder ganz nach rechts geschoben wird, kann die Eingabe des Textes fortgesetzt werden. Es wurde bereits erwähnt, daß eine Eingabe mit der Taste 'RETURN' abgeschlossen wird. Wenn Sie jedoch in unserem Beispiel diese Taste drücken, meckert der CPC. Mit der Meldung "Syntax error" sagt er "was Du mir hier übermittelst hast, verstehe ich überhaupt nicht". Ist auch klar, denn die einzige Sprache, die Ihr CPC6128 spricht, ist BASIC, und die müssen Sie erst lernen, um mit ihm zu arbeiten. Bis dahin sind es nur noch wenige Seiten.

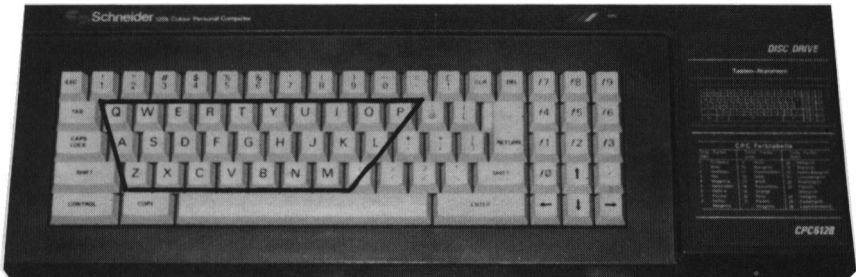
Sie haben nun den Einfügemodus kennengelernt. Wenn Sie noch ein bißchen üben, werden Sie sicher bald das "Einfügen" perfekt beherrschen.

LÖSCHEN DES BILDSCHIRMS

Stellen Sie einmal fest, daß das, was sich auf dem Bildschirm befindet, eigentlich alles unbrauchbares Kauderwelsch ist, so wischen Sie einfach alles weg. HALT!! nicht mit dem Fensterleder, damit können Sie vielleicht den Staub vom Bildschirm wischen, nicht jedoch die eingetippten und somit im Rechner gespeicherten Daten des Bildschirms. Schließlich ist es für einen Rechner, der ca. 1 Million Befehle pro Sekunde abarbeiten kann, ein Kinderspiel, "mal eben" die 1000 Zeichen auf

dem Bildschirm auf Befehl zu löschen. Viele andere Rechner halten zum Löschen des Bildschirms eine Taste bereit. Beim CPC6128 ist es nicht ganz so komfortabel, da hier vier Tastendrucke notwendig sind. Das BASIC des CPC6128 hält einen Befehl bereit, der dem Rechner befiehlt "lösche den gesamten Bildschirm". Dieser Befehl lautet 'CLS', was wie alle Befehle aus dem englischen (Clear Screen) kommt. Wie Sie bereits gelernt haben, müssen Sie die Befehle mit der 'RETURN'-Taste abschließen. Geben Sie also den Befehl 'CLS' gefolgt von der Taste 'RETURN' ein. Ihr CPC gehorcht Ihnen aufs Wort und löscht den Bildschirm. Die Meldung 'Ready' bedeutet wieder, daß der Rechner auf weitere Eingaben wartet. Diesen Befehl sollten Sie sich merken, da er im weiteren Verlauf dieses Buches noch öfter gebraucht wird.

DIE TASTEN MIT BUCHSTABEN

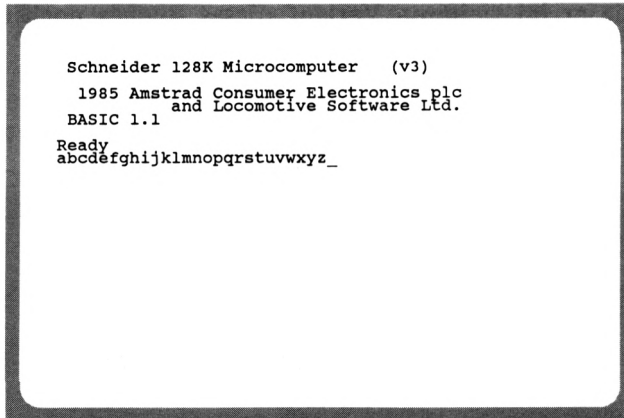


Wie bereits erwähnt, gleicht die Anordnung dieser Tasten bis auf kleine Abweichungen der einer Schreibmaschine. Konzentrieren wir uns zunächst auf diese Buchstaben. Wenn Sie irgendeine dieser Tasten drücken, so erscheint der entsprechende Kleinbuchstabe an der Cursor-Position auf dem Bildschirm. Bevor wir diese Tasten einsetzen, löschen Sie bitte den Bildschirm mit dem Befehl 'CLS'.

Nun haben wir einen "sauberen" Bildschirm, auf dem wir editieren können. Bitte geben Sie nun keine Unmengen von Texten ein in der Hoffnung, daß Ihr Computer alles, zu jeder Zeit

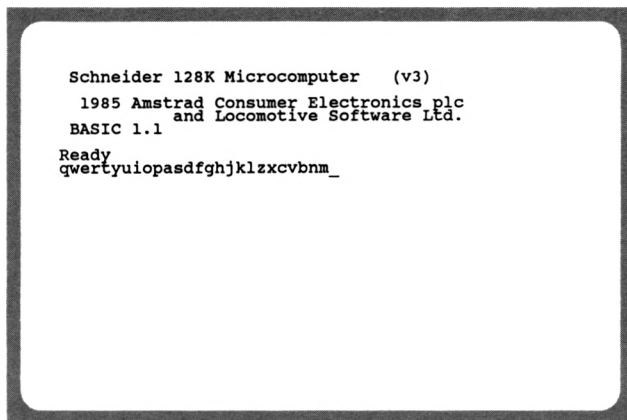
abrufbereit, in seinen Speicher aufnimmt. Ganz so einfach ist es nicht. Das, was Sie eingeben, wird lediglich im flüchtigen Bildschirmspeicher festgehalten. Wie man eingegebene Texte abspeichern kann, erfahren Sie in unserer BASIC-Einführung.

Schreiben Sie jetzt einmal alle Buchstaben auf dem Bildschirm, vielleicht sogar in alphabetischer Reihenfolge. Sie werden feststellen, wie schwer doch zu Anfang der ein oder andere Buchstabe zu finden ist, wenn man nicht gerade Sekretärin ist. Auf dem Bildschirm befinden sich nun alle verfügbaren Kleinbuchstaben.

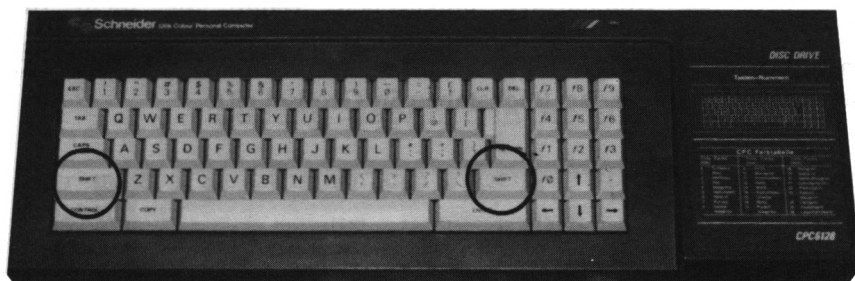


```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
    and Locomotive Software Ltd.
BASIC 1.1
Ready
abcdefghijklmnopqrstuvwxyz_
```

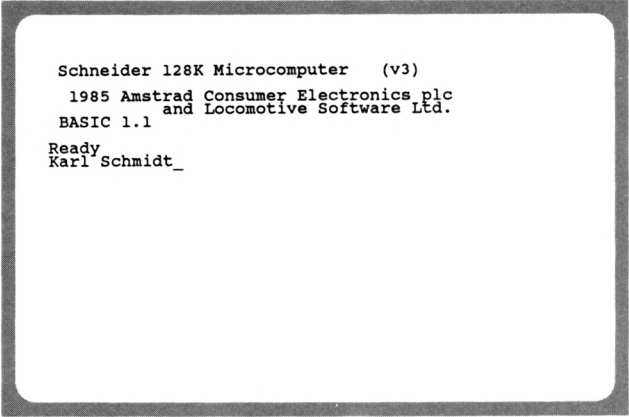
Löschen Sie wieder den Bildschirm und geben nochmals alle Buchstaben ein. Doch nun nicht in alphabetischer Reihenfolge, sondern wie sie auf der Tastatur angeordnet sind. Also zunächst die obere Reihe (Q bis P), dann die mittlere Reihe (A bis L) und schließlich die untere Reihe (Z bis M). Sicher fällt Ihnen diese Reihenfolge viel leichter.



GROSS/KLEINSCHRIFT

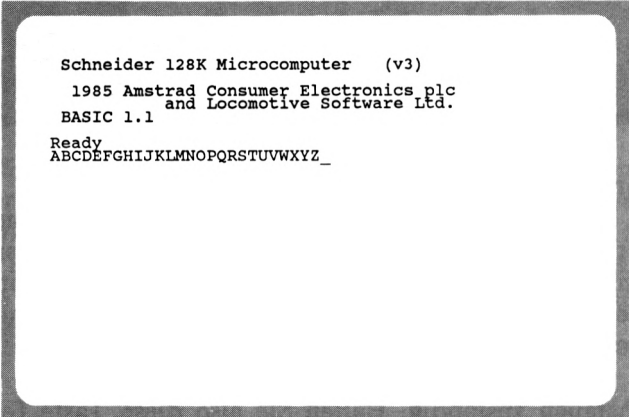


Sie haben bereits festgestellt, daß jeder Buchstabe, den Sie eingeben, auf dem Bildschirm als Kleinbuchstabe erscheint. Oft und vor allem in der Textbearbeitung werden auch Großbuchstaben benötigt. Auf der Schreibmaschine gibt es eine Taste, die zusätzlich gedrückt werden muß, um Großbuchstaben zu drucken. Beim CPC ist dies nicht anders. Die Taste 'SHIFT', die Sie zweimal auf der Tastatur finden, ist für die Großbuchstaben zuständig. Halten Sie also diese Taste fest, wenn Sie Großbuchstaben wünschen. Schreiben Sie nun einmal Namen "Karl Schmidt" auf dem Bildschirm. Die Großbuchstaben werden wie bereits bekannt mit der SHIFT-Taste eingegeben.



```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
    and Locomotive Software Ltd.
BASIC 1.1
Ready
Karl Schmidt_
```

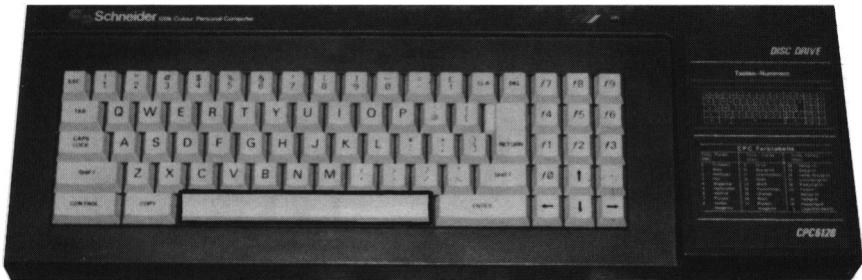
Wir haben bereits alle Kleinbuchstaben auf dem Bildschirm geschrieben. Lassen Sie uns nun auch die Großbuchstaben in alphabetischer Reihenfolge eingeben. Damit Sie nicht die SHIFT-Taste gedrückt halten müssen, gibt es eine weitere Hilfstaste (CAPS LOCK). Wenn Sie diese Taste einmal drücken, erscheinen alle folgenden Buchstaben in Großschrift.



```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
    and Locomotive Software Ltd.
BASIC 1.1
Ready
ABCDEFGHIJKLMN OPQRSTUVWXYZ_
```

Drücken Sie die Taste CAPS LOCK ein weiteres Mal, so können wieder Kleinbuchstaben eingegeben werden.

DIE LEERTASTE

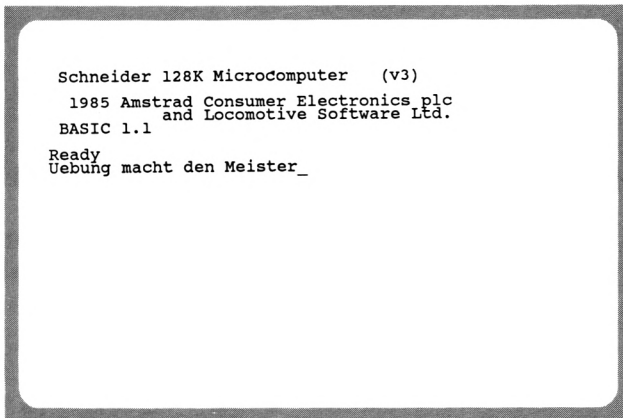


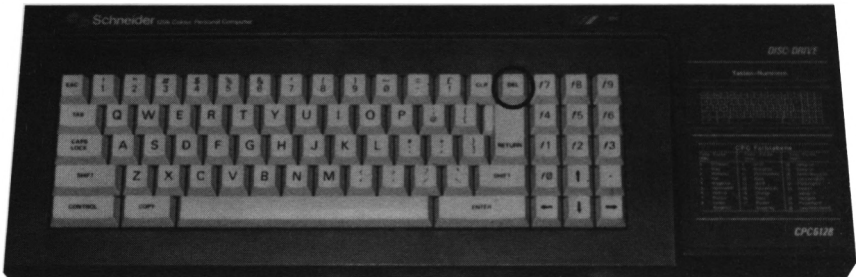
Auch diese Taste finden Sie an jeder Schreibmaschine wieder. Auch bei Ihrem Computer brauchen Sie nicht darauf zu verzichten, Leerzeichen zwischen die Wörter zu setzen. Befindet sich an der Stelle, an der ein Leerzeichen erscheinen soll, bereits ein anderes Zeichen, so wird dieses natürlich gelöscht.

Löschen Sie den Bildschirm und geben den folgenden Satz ein:

Uebung macht den Meister

Hier müssen Sie zwischen den einzelnen Wörtern die Leertaste drücken.



DIE DEL - TASTE

Die Abkürzung dieser Taste sagt bereits viel über die Funktion aus. DEL bedeutet DELETE (löschen). Mit der Taste DEL kann das Zeichen links vom Cursor gelöscht werden. Dies ist besonders hilfreich beim Korrigieren von Tippfehlern. Ein Druck auf die Taste DEL und das falsch eingetippte Zeichen ist wieder gelöscht.

Löschen Sie nun den Bildschirm und tippen den folgenden Satz ab:

Tippfehler sind unverzeihlicj

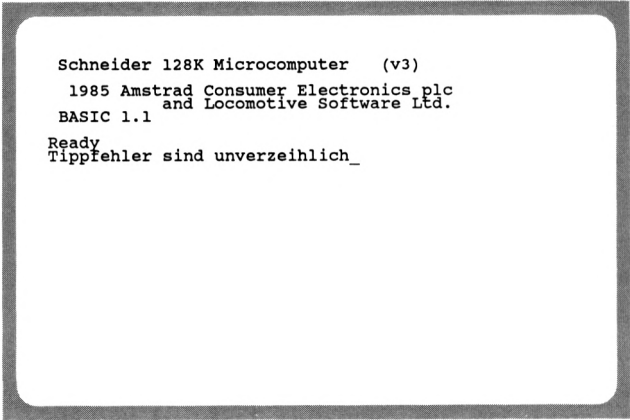
Bei der Eingabe des letzten Buchstabens ist ein Fehler unterlaufen.

```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
  and Locomotive Software Ltd.
BASIC 1.1
Ready
Tippfehler sind unverzeihlic_
```

Tipp-Ex ist hier nicht notwendig. Auf Tastendruck wird der zuletzt eingegebene Buchstabe gelöscht. Drücken Sie dazu die Taste DEL.

```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
  and Locomotive Software Ltd.
BASIC 1.1
Ready
Tippfehler sind unverzeihlic_
```

Nun können Sie hier den richtigen Buchstaben einsetzen, und der Tippfehler ist vergessen.

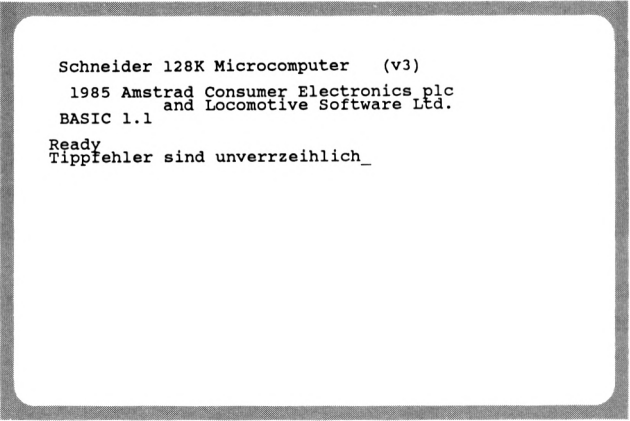


```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
    and Locomotive Software Ltd.
BASIC 1.1
Ready
Tippfehler sind unverzeihlich_
```

Doch dieses Beispiel zeigt nicht den überragenden Vorteil dieser Taste. Spielen wir ein weiteres Beispiel durch. Geben Sie, nachdem Sie den Bildschirm gelöscht haben, den folgenden Satz ein:

Tippfehler sind unverzeihlich

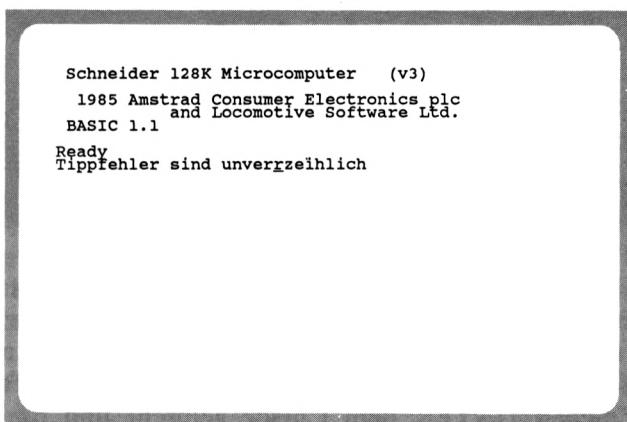
Auch hier hat sich ein Tippfehler eingeschlichen: Zwei 'r' sind hier nicht angebracht.



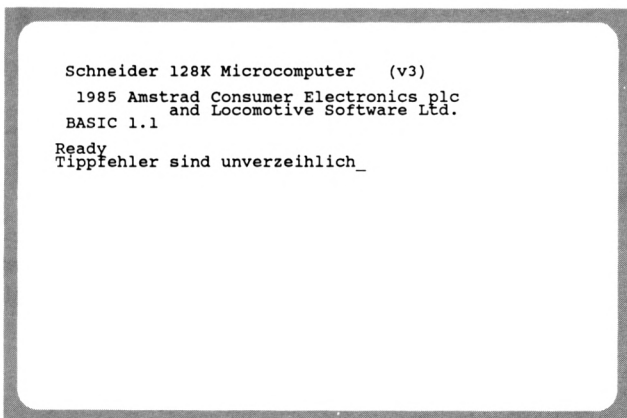
```
Schneider 128K Microcomputer (v3)
  1985 Amstrad Consumer Electronics plc
    and Locomotive Software Ltd.
BASIC 1.1
Ready
Tippfehler sind unvrerrzeihlich_
```


Es ist nicht notwendig, das letzte Wort ab dem Tippfehler nochmal zu schreiben. Wird nämlich ein Zeichen links vom Cursor mit der Taste DEL gelöscht, so werden automatisch alle Zeichen in dieser Zeile, die sich rechts vom Cursor befinden, nachgerückt. Das ist nicht einfach zu erklären, sehen Sie darum selbst:

Bewegen Sie den Cursor hinter das zu löschende 'r' des fehlerhaften Satzes. Dazu drücken Sie die Tasten CURSOR-LINKS.

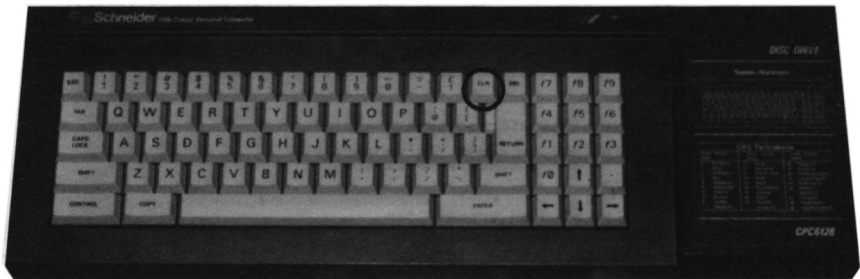


Wenn Sie nun die Taste DEL drücken, wird das 'r' links neben dem Cursor gelöscht und der Rest, einschließlich des Zeichens unter dem Cursor, nach links aufgerückt.



Wenn Sie nun den Cursor zum Ende dieses Satzes bringen, können Sie mit dem Schreiben fortfahren. Halten Sie dazu solange die CURSOR-RECHTS - Taste gedrückt, bis der richtige Punkt erreicht ist.

DIE CLR - TASTE



Die DEL - Taste hat wie gesagt die Funktionen, das Zeichen links vom Cursor zu löschen und das Zeichen unter sowie alle Zeichen rechts vom Cursor nachzurücken. Dies ist nicht die einzige komfortable Editierhilfe des CPC6128. Mit der Taste CLR werden Zeichen UNTER dem Cursor gelöscht und die Zeichen rechts vom Cursor nachgerückt. Dazu wieder ein Beispiel. Geben Sie nochmals einen fehlerhaften Satz ein:

```
Schneider 128K Microcomputer (v3)
 1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.
BASIC 1.1
Ready
Das Wetter ist sehr gut_
```

Nun bemerken Sie, daß das Wetter eigentlich gar nicht so gut ist. Das Wort "sehr" soll entfernt werden. Sicher, Sie könnten es mit der Taste DEL löschen, wir wollen aber auch die Taste CLR kennenlernen. Da CLR das Zeichen unter dem Cursor löscht und die Zeichen rechts vom Cursor nach links aufrückt, müssen wir den Cursor auf das 's' von 'sehr' positionieren.

```
Schneider 128K Microcomputer (v3)
 1985 Amstrad Consumer Electronics plc
and Locomotive Software Ltd.
BASIC 1.1
Ready
Das Wetter ist gehr gut
```

Wenn Sie nun die CLR-Taste fünfmal drücken, wird das Wort 'sehr' gelöscht.

Mit den Tasten DEL und CLR verfügen Sie über zwei komfortable Editierhilfen, die Ihnen bei der Erfassung der Programme und bei sonstigen Texteingaben sehr hilfreich zur Seite stehen werden.

EINSTELLBARE ZEICHENDICHTE

Sicher haben Sie bereits festgestellt, daß eine Zeile auf dem Bildschirm 40 Zeichen enthält. Auf den 25 Zeilen können somit insgesamt 1000 Zeichen untergebracht werden. Bei vielen anderen Computern dieser Klasse wird dies höchst ungern in Kauf genommen. Da wird mit allen möglichen Tricks versucht, anstatt der 40 Zeichen 80 in einer Zeile unterzubringen. Dies erfordert eine technisch aufwendige und dementsprechend teure Erweiterung.

Die Entwickler des CPC hatten sehr lichte Momente, als sie sich entschlossen, die Zeichendichte einer Zeile variabel zu gestalten. So kann der Anwender auswählen, ob er sich an 20, 40 oder gar 80 Zeichen in einer Zeile erfreuen möchte, und zwar auf einfachste Weise mittels eines BASIC-Befehls. Wir haben bereits einen BASIC-Befehl kennengelernt, CLS. Dieser Befehl "putzt" alle Zeichen vom Bildschirm. Der Befehl zum Umschalten der Zeichendichte lautet 'MODE'. Da wir drei Auswahlmöglichkeiten haben, muß dem Befehl eine weitere Ziffer angestellt werden. Diese Ziffer besagt, welche der drei Zeichendichten ausgewählt wird. Somit gibt es drei verschiedene MODE-Befehle:

- MODE 0 - umschalten auf 20 Zeichen pro Zeile
- MODE 1 - umschalten auf 40 Zeichen pro Zeile
- MODE 2 - umschalten auf 80 Zeichen pro Zeile

Lassen Sie uns nicht mehr länger auf den praktischen Einsatz dieses Befehls warten. Schalten Sie nun einmal den Bildschirm auf 'MODE 0'. Beachten Sie, daß Sie diesen Befehl mit der Taste 'RETURN' abschließen, da Sie sonst vergeblich auf eine Reaktion warten, erst die Taste 'RETURN' gibt den Befehl zur Ausführung frei.

"Das ist was für Sehbehinderte" werden Sie vielleicht denken, wenn das überdimensionale 'Ready' erscheint. Sicher ist diese Schrift besser lesbar als etwa die bisherige 40 Zeichen-Darstellung. Man wird sie vor allem in Spielprogrammen einsetzen. Wenn Sie etwa daran denken, diese Schrift z.B. als Programmüberschrift zu nutzen und auf den restlichen Zeilen eine andere Zeichendichte einzusetzen, so muß ich Sie leider enttäuschen. Auf einem Bildschirm kann immer nur eine Schriftart eingesetzt werden, da bei jedem MODE-Befehl der Bildschirm zunächst gelöscht wird.

Kommen wir nun zu der zweiten Alternative, der 80 Zeichen-Darstellung. Den Befehl kennen Sie bereits aus der vorherigen Übersicht. Geben Sie den Befehl 'MODE 2' ein und beachten die Reaktion des Rechners.

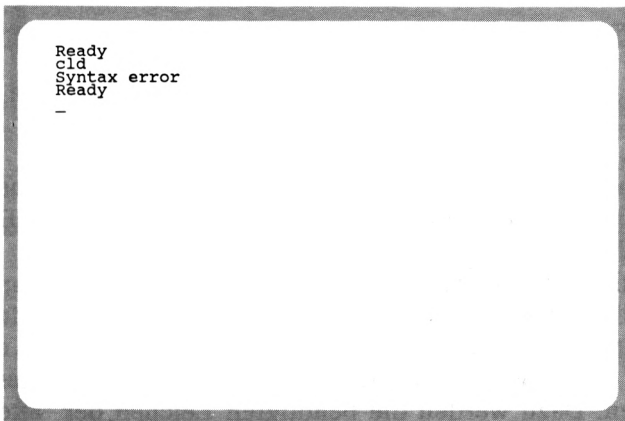
Wenn Sie nun etwas auf dem Bildschirm schreiben, werden Sie feststellen, daß die kleinen Zeichen überraschend gut zu lesen sind. Wenn Sie dies nicht bestätigen können, haben Sie sicher einen Farbmonitor, denn nur der Grünmonitor bietet eine gute Auflösung der 80 Zeichen. Dies und die augenschonende Wirkung sind Vorteile eines Grünmonitors. Aber spätestens nach Einsatz der großen Farbenpracht des CPC werden die Vorteile eines Farbmonitors bewußt. Dies soll nicht etwa bedeuten, daß Sie zwei Monitore einsetzen sollen, auch wenn dies die optimale Lösung wäre. Ein guter Kompromiß ist der Einsatz eines Grünmonitors und eines HF-Modulators, der die Farbenpracht auch auf einem üblichen Fernseher präsentiert.

DER KOPIERCURSOR



Mit jedem Computer können Texte, Programme und Daten festgehalten und zu jeder Zeit wieder geändert werden. Programme etwa werden nicht nur einmal geschrieben, oft müssen Änderungen vorgenommen werden. Damit Sie mit besten Voraussetzungen an die ersten Programmierversuche gehen können, lernen Sie schon jetzt, wie man bereits eingegebene Zeichenfolgen ändern kann.

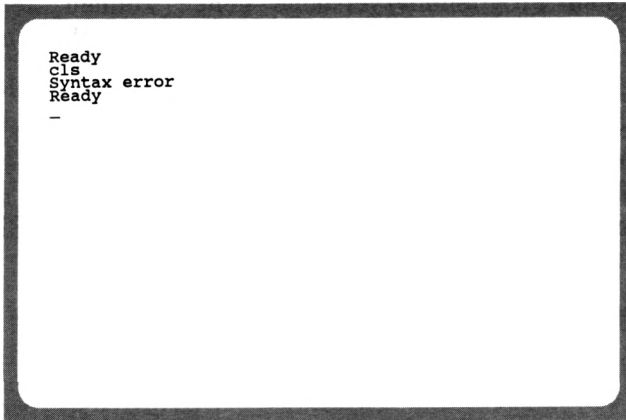
Lassen Sie uns ein Beispiel durchführen. Wir nehmen an, daß Sie den Befehl CLS eingeben möchten. Bei der Eingabe jedoch machen Sie einen Fehler und tippen 'cld'. Sie haben sich also beim letzten Buchstaben des Befehls vertippt. Führen Sie dieses Beispiel nun aus.



```
Ready
cld
Syntax error
Ready
-
```

Nun möchten wir den Befehl aber nicht nochmals eingeben, sondern den noch auf dem Bildschirm gezeigten ändern. Andere vergleichbare Computer verfügen über einen sogenannten Bildschirmeditor. Mit einem derartigen Editor kann man die Befehlszeile mit dem Cursor nochmals aufsuchen, ändern und mit RETURN erneut abschicken. Dies ist beim CPC nicht möglich. Versuchen Sie es trotzdem. Fahren Sie mit dem Cursor auf das falsch eingegebene 'd' und ersetzen es durch ein 's'. Links vom

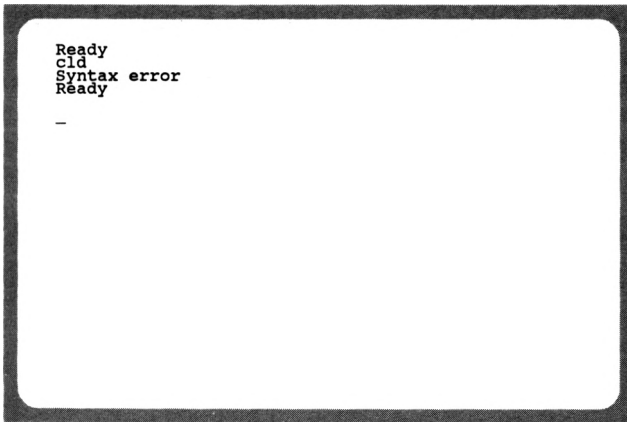
Cursor befindet sich nun der Befehl 'cls'. Drücken Sie nun die Taste RETURN, so wird der Bildschirm nicht etwa gelöscht, sondern erneut die Fehlermeldung ausgegeben.



Der Grund dafür ist, daß alle Buchstaben der Zeichen eines Befehls eingegeben werden müssen, bevor man die RETURN-Taste betätigt. Bereits auf dem Bildschirm enthaltene Zeichen können nicht geändert oder ergänzt werden. Bei sehr langen Befehlszeilen, wie wir sie später noch kennenlernen werden, ist es sehr nachteilig, wenn man stets die gesamte Zeile neu eingeben muß. Mit dem CPC haben Sie eine Möglichkeit, sich diese Mühe zu ersparen. Dazu werden die benötigten Zeichen der "alten" Zeile zu einer anderen Position des Bildschirms kopiert. Um diese Methode zu beherrschen, ist ein wenig Übung nötig. Sie werden nun Schritt für Schritt in diese Technik eingeführt. Löschen Sie dazu zunächst den Bildschirm - diesmal mit dem richtigen Befehl - und geben erneut den falschen Befehl 'cld' ein.

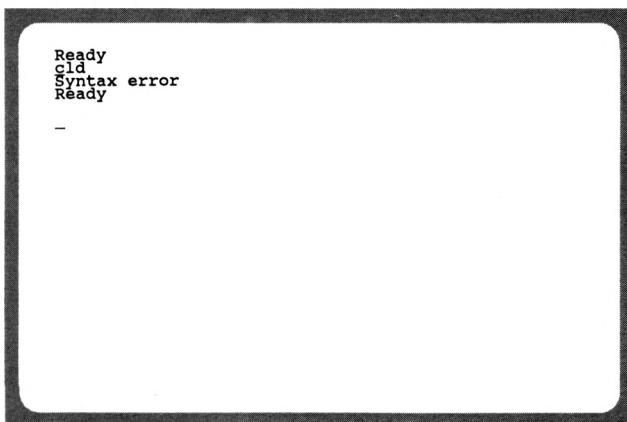
1. Schritt - CURSOR zum Anfang der "neuen" Zeile

Entscheiden Sie sich zunächst, wohin die "alte" Befehlszeile kopiert werden soll und bewegen Sie den Cursor zum Anfang dieser Zeile. In unserem Beispiel soll dies die 5. Bildschirmzeile sein.



2. Schritt - KOPIERCURSОР zum Anfang der "alten" Zeile

Den bisher beschriebenen Cursor wissen Sie bereits zu beherrschen. Was jedoch ist der Kopiercursor und wie wird er bewegt? Der Kopiercursor ist ein zweiter Cursor, der das gleiche Aussehen wie der übliche Cursor hat. Er wird ebenfalls mit den Cursortasten, jedoch zusammen mit SHIFT bewegt. Halten Sie nun die SHIFT-Taste gedrückt und betätigen die Cursor-OBEN - Taste, bis der Kopiercursor die "alte" Zeile erreicht hat.



3. Schritt - mit COPY-Taste gewünschte Zeichen kopieren

Die Taste COPY bewegt den Kopiercursor sowie den normalen Cursor nach rechts und kopiert dabei die Zeichen unter dem Kopiercursor. Dies hört sich komplizierter an, als es eigentlich ist. Probieren Sie es selbst aus und drücken zweimal die COPY-Taste.

```
Ready
cl_
Syntax error
Ready
cl_
```

Nun haben Sie die beiden ersten Zeichen des Befehls CLS kopiert. Der dritte Buchstabe darf nicht kopiert werden, sondern muß neu eingegeben werden. Sie drücken also nun die Taste 's'.

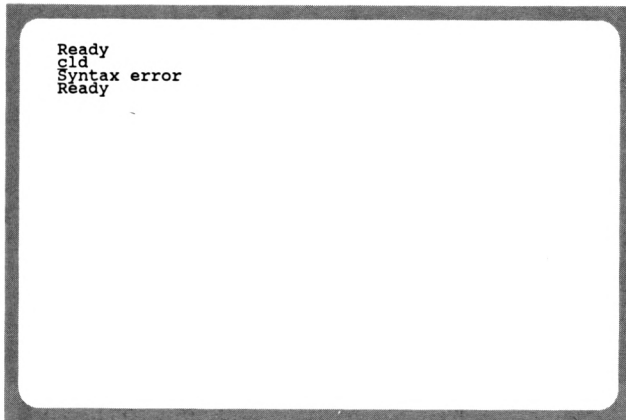
```
Ready
cl_
Syntax error
Ready
cls_
```

Der alte Befehl ist nun kopiert und geändert worden. Nun drücken Sie die Taste 'RETURN' und der Befehl wird ausgeführt. Mit der Taste 'RETURN' verschwindet auch der Kopiercursor.

Neben dieser Methode zur Änderung bestehender Befehlszeilen gibt es eine weitere. Wir haben die "alte" Zeile an eine andere Bildschirmposition kopiert und geändert. Die andere Möglichkeit ist, die "alte" Zeile an Ort und Stelle zu ändern. Der Versuch, lediglich den Cursor an die zu ändernde Stelle zu bringen, schlug fehl. Führen wir nun auch diese zweite Möglichkeit Schritt für Schritt aus, nachdem wir erneut den falschen Befehl 'cld' eingeben.

1. Schritt - CURSOR zum Anfang der "alten" Zeile

Wir positionieren den Cursor also nicht an eine andere Stelle, sondern zum Anfang der zu ändernden Zeile.



2. Schritt - COPY-Taste bis zum ersten falschen Zeichen

Nun drücken Sie die COPY-Taste zweimal. Dabei werden die ersten beiden Zeichen an die gleiche Stelle kopiert, da die Position des Kopiercursors und des normalen Cursors identisch ist. Sie müssen also die COPY-Taste und nicht die Cursor-RECHTS - Taste drücken.

```
Ready
cld
Syntax error
Ready
```

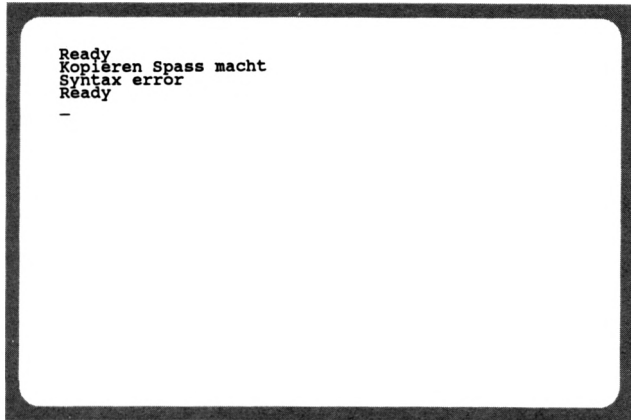
Das dritte Zeichen wird nun wieder nicht übernommen, sondern durch ein 's' ersetzt. Drücken Sie also nun die Taste 's'.

```
Ready
cls
Syntax error
Ready
```

Dieser nun fehlerfreie Befehl kann mit der Taste 'RETURN' ausgeführt werden. Sie haben nun zwei Möglichkeiten zum Ändern von bestehenden Bildschirmzeilen kennengelernt, wobei die zweite Methode etwas einfacher erscheint. Welche von beiden Sie einsetzen, sollten Sie von Fall zu Fall entscheiden.

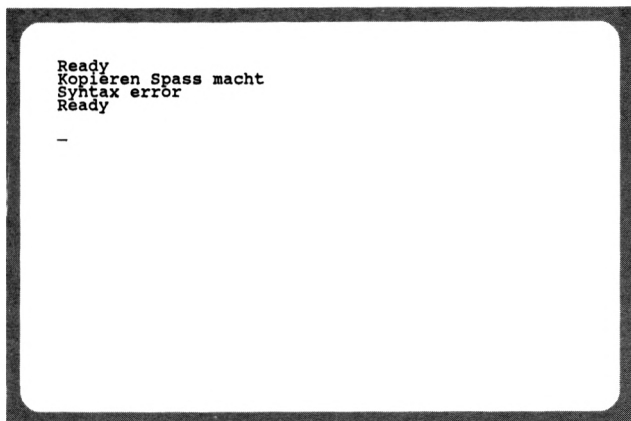
Vielleicht haben Sie doch noch Schwierigkeiten beim Einsatz des Kopiercursors. Lassen Sie uns deshalb ein weiteres, umfang-

reicherer Beispiel durchführen. Geben Sie dazu zunächst die folgende Wortfolge ein, nachdem Sie den Bildschirm gelöscht haben. Drücken Sie danach RETURN.



Aus dieser Wortfolge wollen wir mit Hilfe des Kopiercursors den Satz "Kopieren macht Spass" basteln. Da die Wortfolge keine eigentliche Befehlszeile darstellt, müssen wir den "Syntax error" nach der Taste 'RETURN' in Kauf nehmen. Doch dies soll uns zum Erreichen unseres Ziels nicht irritieren.

Der richtige Satz soll in der 6. Bildschirmzeile erscheinen. Dazu bewegen Sie den Cursor zum Anfang dieser Zeile.



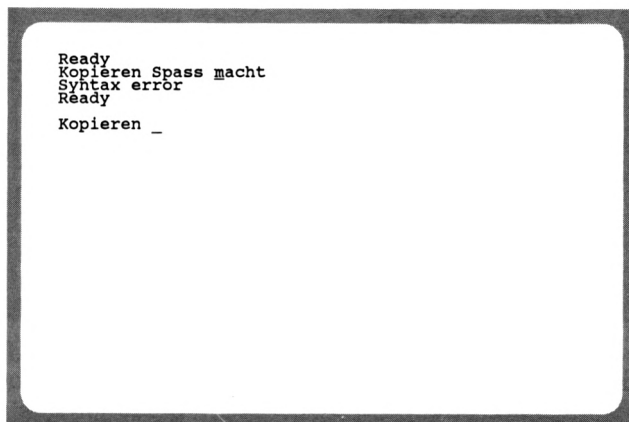
Nun müssen wir den Kopiercursor zum Anfang des ersten zu kopierenden Wortes bringen. Dazu halten Sie die SHIFT-Taste gedrückt und bewegen den Kopiercursor mit den Cursorstasten.

```
Ready
Kopieren Spass macht
Syntax error
Ready
_
```

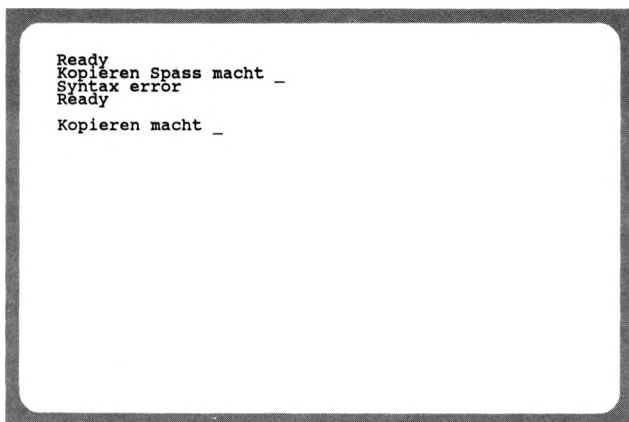
Nun können wir das erste Wort kopieren. Drücken Sie dazu wieder die COPY-Taste, bis der Kopiercursor auf dem 'S' des Wortes 'Spass' steht. Das Leerzeichen soll also auch kopiert werden.

```
Ready
Kopieren Spass macht
Syntax error
Ready
Kopieren _
```

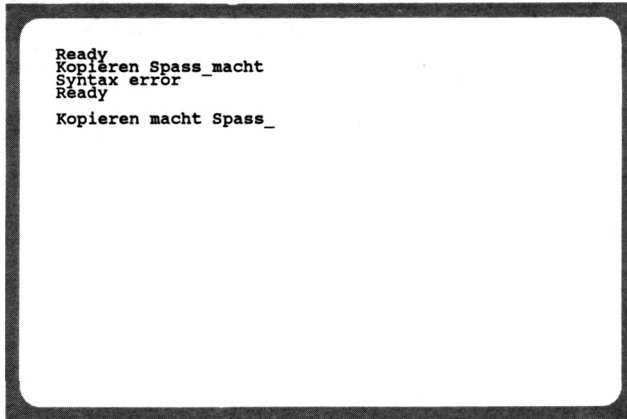
Jetzt soll das zweite Wort kopiert werden. Dazu müssen wir wieder den Kopiercursor zum Anfang des Wortes 'macht' bringen. Drücken Sie also SHIFT und Cursor-RECHTS sechsmal.



Anschließend wird dieses Wort kopiert, indem Sie wieder die COPY-Taste drücken, und zwar sechsmal. Der letzte Tastendruck ist notwendig, damit ein Leerzeichen übernommen wird.



Übernehmen Sie nun selbständig das letzte Wort "Spass". Also Kopiercursor bewegen und COPY-Taste drücken! Nachdem alles erfolgreich verlaufen ist, gleicht Ihr Bildschirm dem folgenden Foto:

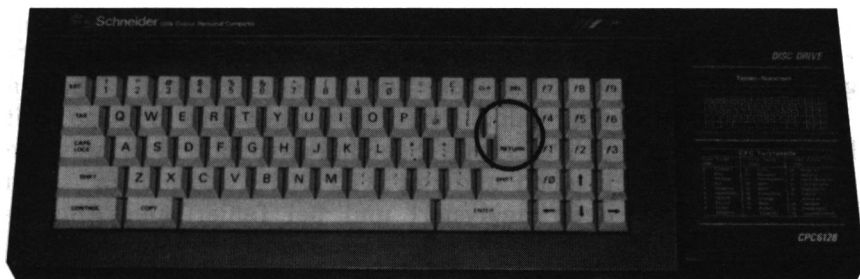


Haben Sie es geschafft? Gratulation! Wenn Sie nun noch kleine weitere Übungen machen, werden Sie diese Technik sicher bald beherrschen. Sie müssen nur immer darauf achten, daß Sie das Bewegen der beiden Cursor nicht verwechseln. Sonst kommen Sie in Situationen, die Sie erst nach einiger Routine zu meistern wissen werden.

KAPITEL 2: DER ERSTE BEFEHL

Zur weiteren Erklärung der Tastatur reicht es nun nicht mehr aus, irgendwelche Zeichen auf dem Bildschirm zu schreiben. Die wenigsten unserer Leser erwarben Ihren CPC6128 für diesen Zweck. Für alle diejenigen, die sich im letzten Kapitel etwas langweilten, wird es in diesem Kapitel interessanter. Wir wollen nun den Computer dafür nutzen, wofür er ja einzig und allein vorgesehen ist, nämlich zur Ausführung von Befehlen.

DIE RETURN-TASTE



RETURN ist die wichtigste Taste an Ihrem Rechner. Wenn Sie diese Taste drücken, wird der eingegebene Text an den Rechner übergeben. Der Rechner interpretiert diese Zeile als Befehl und leitet die notwendigen Schritte ein. Nach Abschluß des Befehls meldet sich der Rechner, wie nach dem Einschalten, mit 'READY' wieder. Er ist also wieder bereit zur Aufnahme weiterer Instruktionen. Versuchen Sie nun einmal, den Rechner zu begrüßen. Geben Sie den Text "GUTEN TAG" ein und drücken anschließend die Taste RETURN. Was meinen Sie, wird Ihr Rechner antworten? Doch sehen Sie selbst.

```
Ready
guten tag
Syntax error
Ready
-
```

Hier erscheint die erste Fehlermeldung Ihres CPC6128, der SYNTAX ERROR (Syntax-Fehler). Die Syntax, also die Zusammensetzung der Zeichen, wird vom Computer nicht verstanden. Zwar gibt es heutzutage Großrechner, die sich weitgehend mit dem Menschen über eingegrenzte Themen "unterhalten" können, jedoch erfordert dies eine Anlage mit hunderttausendfachem Preis und Speicherkapazität. Das, was Sie mit der Tastatur eingeben und mit RETURN "abschicken", wird vom BASIC-Interpreter analysiert und bei fehlerfreier Eingabe ausgeführt.

Ihr CPC6128 ist mit der Programmiersprache BASIC ausgerüstet und versteht somit nur Befehle dieser Sprache. BASIC ist die am meisten verbreitete Programmiersprache, die gerade dem Anfänger optimale Voraussetzungen zum Programmieren bietet. Die Befehle sind weitgehend der englischen Sprache angelehnt. Wie schnell kann ich diese Sprache erlernen? Dies ist eine beliebte Frage, die jedoch nicht hundertprozentig beantwortet werden kann. Viele Kriterien spielen hier eine Rolle. Einmal ist die Lernfähigkeit des Anfängers ausschlaggebend, zum anderen ist die Zeit von Bedeutung, die man am Rechner investiert. Da gibt es Leute, die jede freie Minute dem hochverehrten Computer widmen. Zeitungsanzeigen wie "Verkaufe meinen Computer, Scheidung droht" sind keine Seltenheit mehr. Grundsätzlich gilt es, ein gesundes Maß an Zeit zu opfern, um das Hobby der Pro-

grammierung in BASIC zur hellen Freude werden zu lassen. Mit einem gewissen Grundwissen in Mathematik, das vielleicht bis zu den Grundlagen der Algebra reicht, können bei einem wöchentlichen Zeitaufwand von ca. 10 Stunden schon nach etwa drei Monaten die ersten Erfolgserlebnisse verbucht werden. Zwar trägt das logische und abstrakte Denkvermögen mit zum Erfolg bei, jedoch wird sich dies auch von Programm zu Programm ständig weiterentwickeln. Zum Erstellen von anspruchsvollen Programmen wie z.B. einer kleinen Textverarbeitung oder einer Dateiverwaltung sind neben der Programmierkenntnis die Programmiererfahrung unerlässlich. Von Programm zu Programm steigert sich erfahrungsgemäß die Qualität des Programmierstils. Ein kleiner Anhaltspunkt für Sie: Viele haben schon drei Monate nach Erwerb ihres ersten Computers viel Freude am Entwickeln von kleinen Spielen gehabt. Solange die Begeisterung da ist, werden auch Sie bald einer der vielen Hobby-Programmierer sein.

DER PRINT-BEFEHL

Ohne diesen Befehl ist kaum ein Programm denkbar. Er übernimmt die gesamte Ausgabe in einem Programm. Ob Sie nun ein Rechenergebnis auf dem Bildschirm oder eine Adresse auf das Kassettenlaufwerk ausgeben wollen, ohne diesen Befehl läuft nichts. Auch die gesamte Druckausgabe wird vom Befehl 'PRINT' übernommen.

Zur Zeit wenden wir uns nur dem Direkt-Modus zu, wir schreiben also noch keine Programme. Direkt-Modus bedeutet, Sie geben Befehle ein, schließen sie mit RETURN ab und erwarten ein sofortiges Ergebnis. Dies wollen wir nun erstmalig ohne Fehlermeldung durchführen. Geben Sie nun den Befehl 'print 10' gefolgt von RETURN ein.

```
Ready  
print 10  
10  
Ready  
-
```

PRINT leitet also alles, was dem Befehl als Parameter angefügt wird, zum Bildschirm (die Ausgabe auf externe Geräte erfolgt mit einer anderen Form des PRINT). Parameter sind Bestandteile eines Befehls, die genau beschreiben, welche Auswirkungen der Befehl haben soll.

Hier war die Zahl '10' der Ausgabeparameter. 'print 10' bedeutet also 'zeige die Zahl 10 auf dem Bildschirm'. Natürlich können Sie nicht nur Zahlen, sondern auch andere beliebige Zeichen ausgeben. Wie vielseitig dieser Befehl ist, stellen Sie spätestens im weiteren Verlauf dieses Kapitels fest.

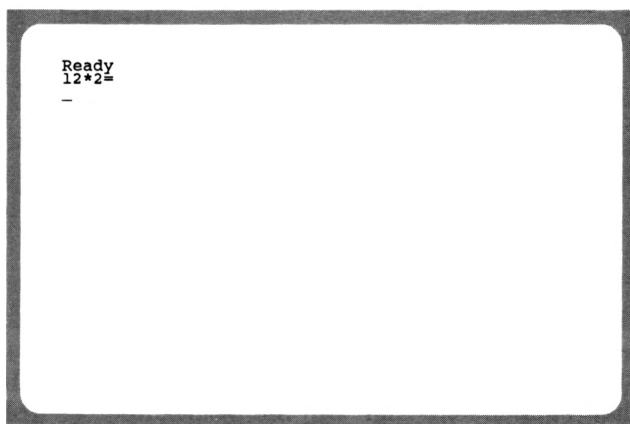
RECHNEN MIT PRINT

Wenn Sie einmal ausrechnen möchten, wieviel Lohnsteuer Sie wohl zurückerstattet bekommen und gerade keinen Taschenrechner zur Hand haben, schalten Sie den CPC6128 ein! Ich selbst schalte auch oft den Rechner nur für kleine Berechnungen ein. Viele meiner Bekannten haben dann oft ironisch gefragt: "Was, rechnen kann man damit auch?". Das wäre ja auch ein Unding, wenn man mit einem Computer, der immerhin das zwanzigfache eines Taschenrechners kostet, nicht rechnen könnte.

Doch nun zur Sache. Wenden wir uns zunächst den vier Grundrechenarten zu. Hierfür gibt es auf der Tastatur vier Symbole:

+	für Addition
-	für Subtraktion
*	für Multiplikation
/	für Division

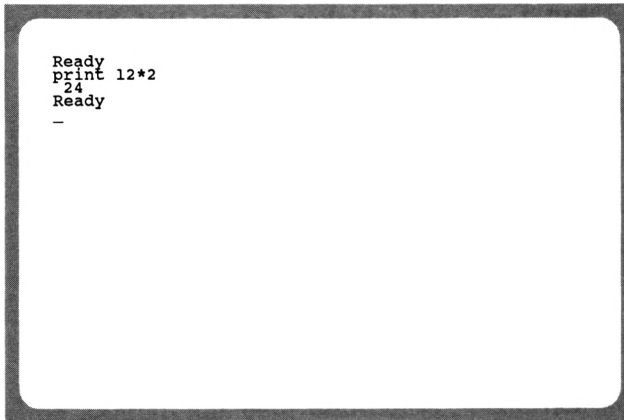
Nur ist das Rechnen mit Ihrem Computer nicht vergleichbar mit einem Taschenrechner. Hier können Sie z.B. nicht '12*2=' eingeben, da bekanntlich ohne Befehl nichts läuft. Es gibt immer experimentierfreudige Leser, die dies trotzdem versuchen werden. Was dann geschieht, ist etwas verwirrend. Versuchen Sie es selbst. Geben Sie '2*12=' ein und schicken diesen "Befehl" mit der Taste 'RETURN' ab.



Was dann geschieht, ist ohne vorzugreifen schwer zu erklären. Der Rechner zeigt nicht das Ergebnis, sondern meldet sich mit READY wieder. Der Cursor springt zum Anfang der nächsten Zeile. Aber was geschieht dazwischen? Der Rechner muß etwas ausgeführt haben, da er sonst eine Fehlermeldung ausgegeben hätte. Ich möchte hier nicht zu weit ausholen. Doch ich muß etwas vorgeifen. Ein Programm besteht aus mehreren Zeilen, die nach Programmstart hintereinander abgearbeitet werden. Jede dieser Zeilen ist am Anfang mit einer Zahl gekennzeichnet, die die Reihenfolge beim Ablauf bestimmt. In diesem Fall wurde die Programmzeile 12 mit dem Inhalt '*2=' eingegeben. Dies ist zwar kein korrekter Befehl, jedoch wird das nicht bei der Eingabe der

Zeile erkannt, sondern erst beim Ablauf des Programms. Doch lassen Sie uns dies zunächst einmal ignorieren.

Wie wird nun '12*2' korrekt errechnet und ausgegeben? Die Lösung ist einfach: Geben Sie dazu den Befehl 'print 12*2' ein. Beachten Sie: Immer wenn Sie einen Befehl eingeben, müssen Sie diesen mit RETURN abschließen. Was Sie auf dem Bildschirm schreiben, ist dem Rechner egal. Ihn interessiert nur das, was Sie ihm mit RETURN zur Ausführung überlassen.

A screenshot of a terminal window with a dark border. The text inside is as follows:

```
Ready 12*2
print 24
Ready
-
```

Das Ergebnis entlockt Ihnen vielleicht das erste "AHA". Ihr Rechner hat zum ersten Mal für Sie gearbeitet. Er folgte treu Ihrer Anweisung, 12 mit 2 zu multiplizieren und das Ergebnis auf dem Bildschirm anzuzeigen.

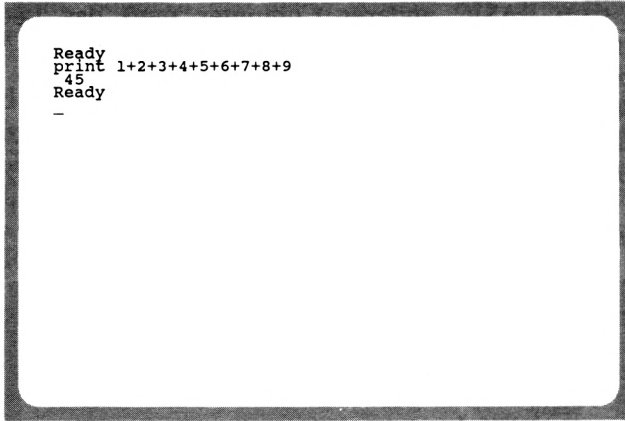
Nebenbei möchte ich bemerken, daß Programmiersprachen eigentlich nur kompliziert erscheinen, weil Sie die Anweisungen an den Rechner wesentlich verkürzen müssen. Jede Programmiersprache muß bis ins letzte Detail klar organisiert sein. Eine Programmiersprache, die Anweisungen wie 'RECHNE 2*12 UND ZEIGE ERGEBNIS' akzeptiert, gibt es nicht und wird es auch nie geben. Verdrängen Sie bitte derartige Erwartungen. Ganz so einfach ist das Programmieren auch nicht. Wenn Sie es beherrschen, dürfen Sie sich nicht umsonst selbst auf die Schulter klopfen.

Entgegen dem BASIC anderer Computer muß das Leerzeichen hinter dem PRINT hier unbedingt angegeben werden, da sonst ein 'Syntax error' gemeldet wird. Doch nicht nur hier, sondern grundsätzlich nach jedem Befehl. Dies werden Sie im weiteren Verlauf des Buches schnell erlernen. Doch nun weiter mit den Grundrechenarten. Berechnen Sie nun hintereinander '12+2', '12-2', '12*2' und '12/2'. Ihr Bildschirm sollte nachher dem folgenden Bildschirm gleichen:

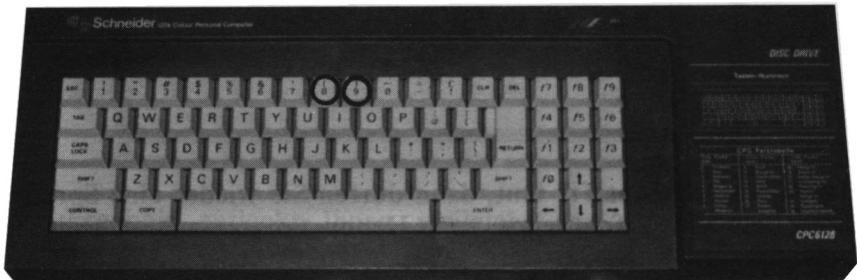
```
Ready
print 12+2
14
Ready
print 12-2
10
Ready
print 12*2
24
Ready
print 12/2
6
Ready
-
```

Haben Sie es geschafft? Gut, dann dürfen Sie wieder eine Sprosse höher auf die Erfolgsleiter zum Hobby-Programmierer steigen. Nein, dann lesen Sie bitte den letzten Abschnitt nochmals aufmerksam durch.

Neben solch simplen Berechnungen können Sie auch lange Berechnungen bis zu 255 Zeichen anstellen! Beachten Sie jedoch die Hierarchie bei solchen Berechnungen: Punktrechnung geht stets vor Strichrechnug. Geben Sie nun dem Rechner die Aufgabe, das Ergebnis von '1+2+3+4+5+6+7+8+9' zu ermitteln.



DIE KLAMMERRECHNUNG

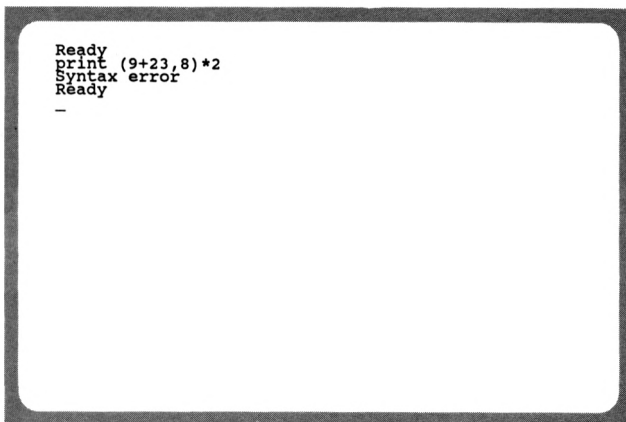


Es besteht also die Möglichkeit, beliebige Kettenrechnungen durchzuführen. Ein praktisches Beispiel: Es soll der Gesamtpreis von drei Teppichstücken ermittelt werden. Der Quadratmeterpreis beträgt 23,80 DM. Das erste Stück umfaßt 2.45 m * 2.80 m, das zweite Stück 4.50 m * 3.85 m und das dritte Stück 2.75 m * 4.80 m. Zusätzlich soll 14% Mehrwertsteuer addiert werden. Wie hoch ist nun der Gesamtpreis für diese drei Teppichstücke? Wenn Sie die Klammern richtig setzen, genügt ein PRINT-Befehl:

```
print (2.45*2.8+4.5*3.85+2.75*4.8)*23.8*1.14
```


Das sieht komplizierter aus, als es eigentlich ist. Innerhalb der Klammer werden die Quadratmeter ausgerechnet. Dann wird die Quadratmeteranzahl mit dem Quadratmeterpreis multipliziert. Schließlich wird noch die Mehrwertsteuer hinzugerechnet. Und das alles mit einem Befehl! Bei korrekter Eingabe erhalten Sie den Wert 1014.32982, also den Preis 1014.33 DM.

Wie Sie bereits erkannt haben, wird der Punkt und nicht das Komma als Dezimalpunkt benutzt. Das ist zwar eine amerikanische Norm, wird jedoch auch an deutschen Rechnern praktiziert. Wenn Sie das Komma in Verbindung mit der Klammerrechnung eingeben, z.B. `print (9+23,8)*2`, so erscheint die Fehlermeldung **SYNTAX ERROR**.

A screenshot of a terminal window with a dark border. The text inside is as follows:

```
Ready  
print (9+23,8)*2  
Syntax error  
Ready  
-
```

EXPONENTIALSCHREIBWEISE

Der Rechner stellte das Ergebnis mit 5 Nachkommastellen dar. Es ist eine sogenannte Fließ- oder Gleitkommazahl. Grundsätzlich wird auf neun Stellen genau gerechnet. Ergibt sich eine Zahl mit drei Stellen vor dem Komma, so wird automatisch auf 6 Stellen nach dem Komma gerechnet. Sollte die Zahl 999999999 überschreiten, so wird wieder auf neun Stellen genau gerechnet. Eine größere Zahl erhält dann einen Zusatz, der die Größe des echten Ergebnisses kennzeichnet. Geben Sie z.B. einmal die Rechnung

PRINT 999999999+1 ein. Das Ergebnis ist eine zehnstellige Zahl, die nicht mehr normal dargestellt werden kann.

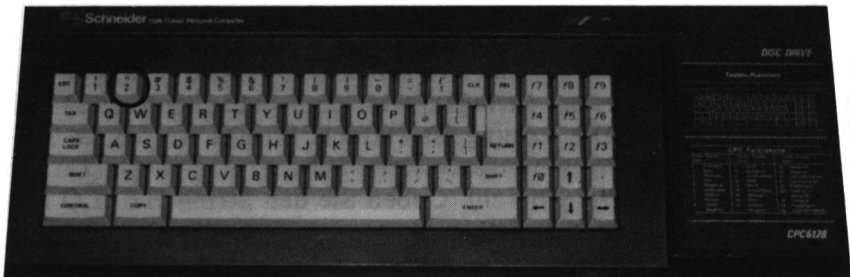
```
Ready
print 999999999+1
  1E+09
Ready
-
```

Verzweifeln Sie nicht bei diesem Ergebnis. Es bedeutet einfach, daß das Ergebnis '1 * 10 hoch 9' ergibt, also eine 1 mit 9 Nullen. 'E+09' bedeutet Basis 10, Exponent +9. Der Exponent kann aber auch negativ sein. Die Rechnung 'PRINT 9/30000000000' bestätigt dies.

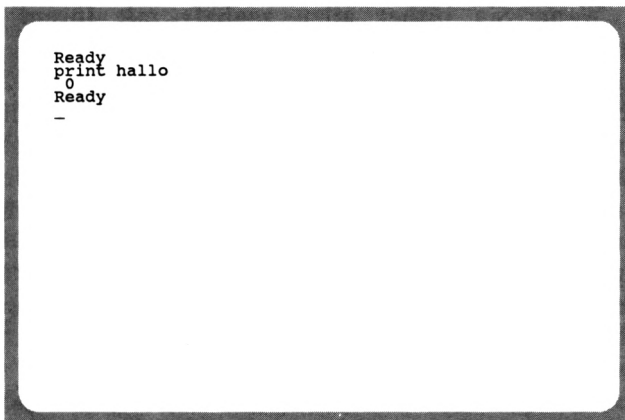
```
Ready
print 9/30000000000
  3E-10
Ready
-
```

Obwohl das Ergebnis eigentlich 0.0000000003 lauten sollte, benutzt der Rechner die Exponentialschreibweise. '3E-10' ist gleichbedeutend mit '3 * 10 hoch -10'. Die Zahl 3 befindet sich also an zehnter Stelle nach dem Komma. Das ist vorerst alles, was Sie über das Rechnen mit den vier Grundrechenarten im Direkt-Modus wissen sollten. Weitere mathematische Funktionen finden Sie in Ihrem Handbuch zum CPC6128.

TEXTAUSGABE MIT PRINT



Neben Zahlen können auch Texte, sogenannte **Strings** (Zeichenketten), mit PRINT ausgegeben werden. Voreilige Versuche wie PRINT HALLO werden kläglich scheitern.



Obwohl der Befehl nicht den gewünschten Erfolg verbuchen kann, erscheint keine Fehlermeldung. Der Rechner hat den Befehl also ausgeführt. Nur was hat er ausgeführt? Woher die Null? Dies sind Probleme, die bei den ersten Gehversuchen mit BASIC immer wieder auftreten. Die Fragen sind, ohne vorzugreifen, schwer zu beantworten. Es wird der Inhalt einer Variablen (rechnerinterner Zahlenspeicher) angezeigt. Die Bezeichnung dieses Speichers ist HALLO. Da wir in der Variablen HALLO nicht abgelegt haben, wird der Wert 0 ausgegeben. Sollten Sie dies nicht ganz verstanden haben, so ist das kein Grund zur Beunruhigung. Die Variablen werden später noch ausführlich behandelt.

Doch wie kann man nun Zeichenketten (bleiben wir im weiteren Verlauf beim Fachwort Strings), wie kann man Strings nun mit PRINT ausgeben? Die Lösung ist einfach: Strings werden in Anführungszeichen eingeschlossen. Setzen Sie dies nun in die Tat um und ändern den letzten Befehl. Geben Sie den String HALLO nun wie folgt in Anführungszeichen an:

PRINT "HALLO"

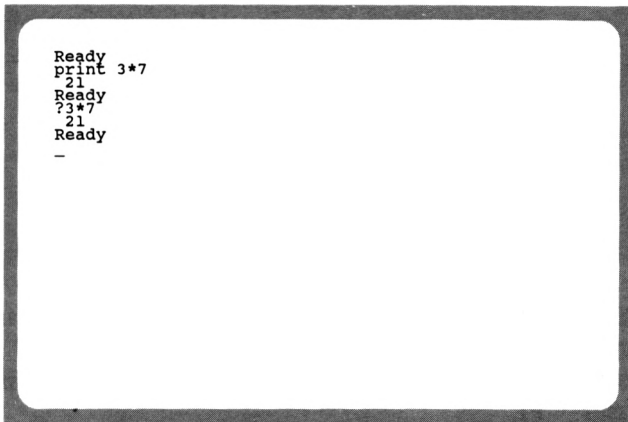
Na also, nach dem ersten Fehlversuch hat es nun doch geklappt. Der Rechner wünscht Ihnen einen guten Tag (er macht bekanntlich all das, was Sie von ihm verlangen=).

Übrigens: Wenn Sie bisher allzu zaghaft mit Ihrem Rechner umgegangen sind und jeden Befehl vorher dreimal überlegten, um ja nichts kaputt zu machen, so kann ich Sie beruhigen. Kein noch so fehlerhafter Befehl kann Ihrem Rechner Qualmwolken entlocken. In kaum einem anderen Hobby ist die Weisheit "aus Fehlern lernt man" so angebracht wie bei Ihrem. Ein Elektronik-Bastler, der die 45 Mark- Endstufe eines selbstgebauten Verstärkers falsch einlötet, beißt sicher in den Teppich. Wenn Sie jedoch einen Fehler in Ihrem 23 KByte-Programm verursachen, so kann er in fünf Minuten wieder korrigiert sein. Der einzige und gleichzeitig schlimmste Fehler ist, wenn Sie z.B. acht Stunden hintereinander an einem Programm gearbeitet haben und die Mutter den defekten Staubsauger einschaltet. Ein Kurzschluß hat hier große Auswirkung, nämlich acht Stunden verschenkte Arbeit.

Auch können Sie z.B. eine wichtige und wertvolle Kassette durch unsachgemäße Behandlung zerstören. Doch wenn Sie die wichtigsten Richtlinien zur Datensicherung beachten, kann nichts schiefgehen.

VEREINFACHTE PRINT-EINGABE

Anstelle des Befehls PRINT kann man auch ein Fragezeichen eingeben. Geben Sie anstelle des Befehls PRINT das Fragezeichen ein, so ersparen Sie sich damit etwas Arbeit, zumal der Befehl PRINT der am häufigsten verwendete Befehl ist. Es ist also egal, ob Sie nun 'PRINT 3*7' oder '? 3*7' eingeben. Probieren Sie es selbst einmal aus. Hinter dem Fragezeichen ist kein Leerzeichen erforderlich.



Es liegt nun an Ihnen, ob Sie den PRINT-Befehl ausschreiben oder das Fragezeichen verwenden. Bei Berechnungen, die "mal eben" eingetippt werden, ist das Fragezeichen recht praktisch. Man kann damit ebenso schnell rechnen wie mit einem Taschenrechner.

PI UND POTENZIERUNG

Wenden wir uns nun wieder der Mathematik zu. Die Konstante PI dürfte jedem bekannt sein. Bei der Kreis- und Kugelberechnung wird sie meistens benötigt. Welchen Wert hat denn nun PI? War es 3.1412 oder 3.1214 oder? Zerschneiden Sie sich nicht den Kopf. Meist wird mit 3.14 gerechnet, also auf zwei Stellen nach dem Komma. Der CPC6128 hält diese Konstante für Berechnungen bereit. Rufen Sie das PI einmal auf den Bildschirm. Geben Sie dazu 'PRINT PI' ein.

```
Ready
print pi
3.14159265
Ready
-
```

PI auf acht Stellen genau! Das ist vollkommen ausreichend. Berechnen Sie nun einmal den Umfang eines Kreises mit dem Durchmesser von 12 cm. Die Formel lautet: Umfang = Durchmesser * PI.

```
Ready
Print 12*pi
37.6991119
Ready
-
```

Der Kreisumfang beträgt also rund 37.7 cm.

Nun zur Potenzierung. Potenzierung bedeutet, eine Zahl mit sich selbst malzunehmen. Der Exponent (die Hochzahl) bestimmt, wie oft die Basis (Grundzahl) mit sich selbst multipliziert werden soll.

2 hoch 3 bedeutet $2*2*2$, also 8
10 hoch 4 bedeutet $10*10*10*10$, also 10000

Im ersten Beispiel ist '2' die Basis und '3' der Exponent. Doch nun Schluß mit dem Mathematik-Nachhilfeunterricht. Beim CPC6128 wird zur Potenzierung zwischen der Basis und dem Exponent ein 'Pfeil nach oben' gesetzt. Rechnen Sie nun einmal 2 hoch 8 aus.

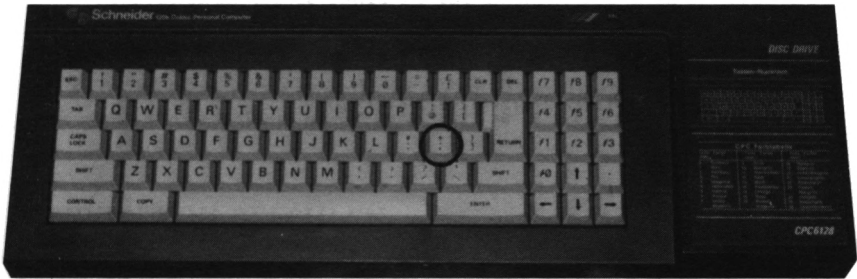
```
Ready  
print 2*8  
256  
Ready  
-
```

Das war sicher nicht schwer. Berechnen Sie nun die Kreisfläche des Kreises mit dem Durchmesser von 12 cm. Die Formel lautet: Kreisfläche = Radius hoch 2 mal PI (Radius ist der halbe Durchmesser).

```
Ready  
Print 6*2**2*3.141592653589793238462643383279502884197169399375105820974944597408  
113.097338  
Ready  
-
```

Die Kreisfläche beträgt also ca. 113 qcm. Doch nun genug von der trockenen Mathematik. Wenden wir uns nun wieder interessanteren Sachen zu.

KOMBINIEREN VON STRINGS MIT ZAHLEN



Die Tasten ';' und ',' spielen eine große Rolle, wenn Strings und Zahlen kombiniert werden sollen. Aber welche? Wenn Sie z.B. eine Zeile wie "2.5 mal 2.5 = 6.25" ausgeben wollen, bei der das Ergebnis gleichzeitig errechnet werden soll, so wird ein String und eine anschließende Berechnung benötigt. Die Lösung sieht dann wie folgt aus:

```
Ready
print "2.5 mal 2.5 =" ; 2.5*2.5
2.5 mal 2.5 = 6.25
Ready
-
```

Das Semikolon trennt also Strings von Zahlen. Doch nicht nur das, auch Strings und Zahlen bzw. Berechnungen können untereinander durch das Semikolon getrennt werden. So können Sie z.B. mehrere Berechnungen mit einem PRINT-Befehl ausführen und die Ergebnisse nebeneinander anzeigen. Versuchen Sie nun,

die Potenzen der Zahl '2' vom Exponent '1' bis zum Exponent '8' zu berechnen und nebeneinander in einer Zeile anzuzeigen.

```

Ready
print 2^1;2^2;2^3;2^4;2^5;2^6;2^7;2^8;
2 4 8 16 32 64 128 256
Ready
-

```

Ein mit SEMIKOLON abgeschlossener PRINT-Befehl hat zur Folge, daß ein weiterer PRINT-Befehl nicht in der nächsten Zeile, sondern in der gleichen Zeile hinter dem ersten PRINT erfolgt. Doch hierzu erfahren Sie später mehr.

Geben Sie nochmals den letzten Befehl ein, ersetzen aber alle Semikoli durch Kommata.

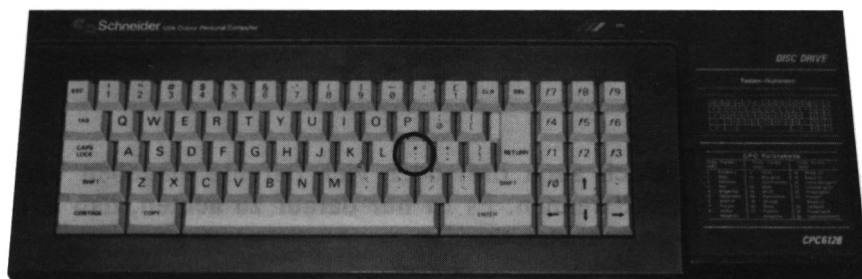
```

Ready
print 2^1,2^2,2^3,2^4,2^5,2^6,2^7,2^8
2 4 8
16 32 64
128 256
Ready
-

```

Das Komma trennt auch die einzelnen Werte, aber mit größerem Abstand. Wenn Sie die Abstände auszählen, so werden Sie feststellen, daß der Abstand zwischen den Ziffern jeweils 13 Zeichen beträgt. Dadurch wird eine leicht formatierte Ausgabe von Zahlenkolonnen ermöglicht. Zur Trennung von Strings wird das Komma nur selten genutzt.

TRENNEN VON BEFEHLEN



Eine Befehlszeile kann aus maximal 255 Zeichen bestehen. BASIC-Befehle erreichen diese Länge sehr selten. Mit dem Doppelpunkt kann man nun Befehle voneinander trennen und somit mehrere Befehle in einer Programmzeile oder einer direkt eingegebenen Befehlszeile darstellen. Ein Beispiel soll hier Klarheit schaffen. Sie wollen zwei Berechnungen mit einem RETURN durchführen. Die Ergebnisse sollen untereinander erscheinen. In unserem Beispiel berechnen wir einmal $30 \uparrow 2 * \text{PI}$ und einmal $30 * \text{PI}$. Versuchen Sie nun zwei PRINT-Befehle in einer Zeile unterzubringen, getrennt durch den Doppelpunkt.

```
Ready
print 30*2*pi:print 30*pi
2827.43339
94.2477796
Ready
-
```

Sie können natürlich auch drei oder vier Befehle eingeben und durch Doppelpunkt trennen. Es ist nur darauf zu achten, daß 255 Zeichen nicht überschritten werden.

Hier läßt sich auch sehr gut die Wirkung des Semikolons nach dem ersten PRINT-Befehl verdeutlichen. Löschen Sie den Bildschirm und geben Sie die beiden PRINT-Befehle nochmals ein. Setzen Sie hinter das erste PRINT ein Semikolon.

```
Ready
print 30*2*pi;;print 30*pi
2827.43339 94.2477796
```

Das Semikolon hinter dem ersten PRINT bewirkt, daß diese Zeile noch nicht abgeschlossen wird. Das zweite PRINT erfolgt unmittelbar hinter dem ersten PRINT. Die leeren Stellen zwischen den Zahlen kommen dadurch zustande, daß vor der ersten Ziffer der Zahl ein Zeichen für das Vorzeichen freigehalten wird. Eine positive Zahl wird jedoch nicht mit '+' gekennzeichnet, sondern nur eine negative Zahl erhält als erstes Zeichen ein Minus (-).

Beachten Sie, daß das Leerzeichen nach dem Doppelpunkt keine Bedeutung hat. Ob hier ein Leerzeichen, fünf oder überhaupt keins enthalten ist, hat keine Auswirkung auf den Befehl.

KAPITEL 3:

DAS ERSTE PROGRAMM

Mit dem ersten Programm ist nicht etwa die ARD gemeint, sondern Ihre ersten Gehversuche bei der Programmierung in BASIC. Verdrängen Sie Ihren eventuellen Unmut, dieses "heiße Eisen" anzufassen. BASIC ist eine Programmiersprache für den Einsteiger, vor der niemand allzu großen Respekt zu haben braucht. Die Tatsache, daß in Kaufhäusern und Computerläden oft Kinder anzutreffen sind, die nach der Schule ihre Zeit mit BASIC vertreiben, sollte die letzten Hemmungen beiseite schieben. Seien Sie kein ewiger Anwender, der seine ganze Zeit den Fertigprogrammen widmet. Mal ehrlich, haben Sie nicht auch oft daran gedacht, eigene Programme zu entwickeln? Dieses und das folgende Kapitel soll für Sie ein kleiner Schritt in die große und interessante Welt der Programmierung sein.

EIN PROGRAMM, WAS IST DAS?

Auf Fachchinesisch wird ein Programm als eine "Folge von Befehlen zur Lösung einer bestimmten Aufgabe" definiert. Diese Folge von Befehlen ist es also, die in logisch richtiger Zusammensetzung ein Programm ergibt. Der Weg zu einem Programm führt also von der Aufgabenstellung über die logische Befehlsfolge zum Ziel, nämlich dem Programm. Logisch richtig bedeutet, daß nicht nur sämtliche BASIC-Befehle bekannt sein müssen, sondern daß sie erst durch ausgetüftelte Zusammensetzung zur Problemlösung führen. So ist es z.B. sinnlos, eine Unmenge von englischen Vokabeln zu kennen, wenn Sie diese nicht durch sinnvolle Zusammensetzung zur Kommunikation nutzen können.

DIE ZEILENNUMERIERUNG

Ein Programm ist also eine Folge von Befehlen. Doch wodurch wird die Reihenfolge bestimmt? Nun, bei der Programmiersprache BASIC wird jede Befehlszeile (auch Statement genannt) mit einer Nummer versehen, die die Reihenfolge bestimmt, mit der das Programm ablaufen wird. Stellen Sie sich z.B. Ihr im Gehirn gespeichertes Programm zur Lösung der Rechenaufgabe '48/12' mit dem Taschenrechner vor. Sie werden feststellen, daß auch hier eine gewisse Reihenfolge eingehalten werden muß:

1. Suche Taschenrechner.
2. Schalte Rechner ein.
3. Wenn Batterie leer, dann Punkt 12.
4. Drücke die Taste '4'.
5. Drücke die Taste '8'.
6. Drücke das Operationzeichen für Division.
7. Drücke die Taste '1'.
8. Drücke die Taste '2'.
9. Drücke die Taste '='.
10. Lese Ergebnis ab.
11. Speicher Ergebnis im Kleinhirn.
12. Schalte Rechner aus.

Erstaunlich, aus wieviel Einzelschritten doch eine so simple Tätigkeit zerlegt werden kann. Sie erkennen, daß alle Schritte nummeriert sind, um die Reihenfolge festzulegen. Doch dies ist nicht der einzige Grund. Der Schritt Nummer 3 schneidet eine wichtige Programmierlogik an, den Sprungbefehl. Wenn eine bestimmte Bedingung erfüllt ist (Batterie leer?), verzweigt das Programm zum Schritt 12. Diese Zahl ist die sogenannte Adresse des Befehls. Ohne Zeilennummer wäre es also nicht möglich, bestimmte Befehle gezielt anzuspringen.

Wir haben im vorherigen Kapitel bereits einen Befehl kennengelernt, den PRINT-Befehl, mit dem Daten auf dem Bildschirm ausgegeben werden. Diesen Befehl setzten wir jedoch nur im Direkt-Modus ein, d.h. der Befehl wird nach dem RETURN direkt ausgeführt. Wir wollen nun eine Folge von drei PRINT-Befehlen als Programm codieren. Was ist zu tun? Richtig, jeder

Befehl erhält eine Zeilennummer, die bestimmt, in welcher Reihenfolge das Programm abgelaufen wird.

DIE ZEILENNUMMERN DÜRFEN BEIM CPC6128 IM BEREICH VON 0 BIS 65535 LIEGEN. DIE SCHRITTWEITE IST OHNE BEDEUTUNG.

Die Schrittweite bestimmt, um welche Zahl die Folgezeile größer als die vorherige Zeile ist. So kann ein vierzeiliges Programm z.B. aus den Zeilennummern 1, 8, 10, 20 bestehen. Auch die Reihenfolge 100 ,200 ,300 ,400 ist möglich. Die Schrittweite darf also innerhalb eines Programms variieren.

Doch wozu ist eine Schrittweite von größer als 1 nützlich. Ist doch klar, wenn zwischen zwei Zeilen in einem Programm noch eine Zeile eingefügt werden soll, so muß die Schrittweite zwischen diesen beiden Zeilen größer als 1 sein, da Zeilennummern immer ganzzahlig sind. Zwischen der Zeile 11 und der Zeile 12 kann keine Zeile mehr eingefügt werden. Zwischen den Zeilennummern 11 und 15 jedoch gibt es mehrere Möglichkeiten, eine Zeile "einzumogeln". Diese Zeile erhält die Nummer 12, 13 oder 14 und wird automatisch zwischen den Zeilen 11 und 15 eingeordnet. In der Praxis hat sich eine Schrittweite von 10 bewährt.

Doch nun zu unserer Aufgabe, die wie folgt aussehen soll:

1. Lösche den Bildschirm.
2. Gebe den Text "18 GETEILT DURCH 6 ERGIBT:" aus.
3. Gebe das Ergebnis unmittelbar hinter dem Text aus.

Zur Lösung der Aufgabe sollen drei BASIC-Zeilen verwendet werden. Wie geht man nun diese Aufgabenstellung an? Zunächst muß die erste Zeilennummer ausgewählt werden. In unserem Beispiel soll dies die Zeile 100 sein. Wir geben dazu die Zahl 100 gefolgt von dem Befehl zum Löschen des Bildschirms ein. Erinnern wir uns: Der Befehl zum Bildschirmlöschen heißt 'CLS'. Diese Zeile sollte dann wie folgt aussehen:


```
Ready  
100 cls  
-
```

HINTER JEDER ZEILENNUMMER MUSS EIN LEERZEICHEN ENTHALTEN SEIN.

Dies ist nun die erste Zeile unseres Programms. Die zweite Zeile soll eine Zeichenkette ausgeben. Dies erfolgt bekanntlich mit dem Befehl PRINT. Wir arbeiten mit einer Schrittweite von 10. Die zweite Zeile erhält demnach die Nummer 110. Geben Sie diese Zeile nun ein.

```
Ready  
100 cls  
110 print "18 geteilt durch 6 ergibt";  
-
```

Hinter dem String wird hier ein Semikolon gesetzt, damit der folgende PRINT noch in dieser Zeile, also hinter dem Text ausgegeben wird.

Die dritte Zeile sollten Sie nun alleine ermitteln. Es soll 18/6 berechnet und ausgegeben werden.

```
Ready
100 cls
110 print"18 geteilt durch 6 ergibt";
120 print 18/6
-
```

Nun ist das Programm fertig und muß nur noch gestartet zu werden.

PROGRAMMSTART

Der CPC6128 hat die soeben eingegebenen Programmzeilen nicht nur auf dem Bildschirm, sondern auch in seinem BASIC-Speicher festgehalten. Auch wenn Sie den Bildschirm löschen, geht das Programm nicht verloren. Es kann zu jeder Zeit gestartet werden.

DER BEFEHL 'RUN' STARTET EIN BASIC-PROGRAMM

Geben Sie nun den Befehl 'RUN' ein und beachten Sie, was auf dem Bildschirm geschieht.

```
18 geteilt durch 6 ergibt 3  
Ready  
-
```

Dieses Programm ist für den Anfänger ein erstes Erfolgserlebnis.

Sie können dieses Programm nun so oft starten, wie es Ihnen gefällt. Überzeugen Sie sich selbst: Geben Sie nochmals 'RUN' ein und nochmals und....

PROGRAMMÄNDERUNG

Es kann durchaus vorkommen, daß an einem Programm Änderungen vorgenommen werden müssen. Stellen Sie sich vor, Sie möchten mit dem zuletzt eingegebenen Programm nicht 18/6, sondern 24/6 berechnen. Dazu müssen Sie nicht etwa alle drei Zeilen nochmals eingeben, sondern Sie können die bestehenden Zeilen an den entsprechenden Stellen abändern. Dazu müssen Sie aber erst wieder das Programm auf den Bildschirm holen.

DER BEFEHL 'LIST' ZEIGT DIE PROGRAMMZEILEN AUF DEM BILDSCHIRM AN

"Schon wieder ein neuer Befehl" werden Sie vielleicht denken. Doch dies wird mit Sicherheit nicht der letzte sein. Um in BASIC zu programmieren, müssen Sie mit den wichtigsten Befehlen vertraut sein.

Geben Sie nun den Befehl 'LIST' gefolgt von der Taste RETURN ein und beobachten, was geschieht. Ihre soeben eingegebenen Programmzeilen erscheinen auf dem Bildschirm. Sicher haben Sie bemerkt, daß Ihr Rechner die Befehle des Programms in Großschrift verwandelt hat. Dies dient der Übersichtlichkeit des Listings. Als "Listing" bezeichnet man übrigens ein ausgegebenes Programm auf dem Bildschirm oder auf dem Drucker.

Was Sie noch wissen sollten: Wenn größere Programme aufgelistet werden, so "scrollt" der Bildschirm. Scrollen bedeutet, es werden von unten Zeilen nachgeschoben, wobei gleichzeitig die obersten Zeilen verschwinden. Um dieses Scrollen anzuhalten, drücken Sie die Taste 'ESC'. Eine beliebige andere Taste setzt das Listing fort. Drücken Sie zweimal hintereinander die Taste 'ESC', um das Listing abubrechen.

Kommen wir nun auf die Programmänderung zurück. Um die Zeilen nun zur Berechnung von 24/6 abzuändern, haben Sie zwei Möglichkeiten:

1. Die Methode mit dem Copy-Cursor
2. Die Methode mit dem Befehl EDIT

Mit dem COPY-CURSOR sind Sie schon zu Anfang des Buches konfrontiert worden. Dies soll uns jedoch nicht daran hindern, diese Methode nochmals durchzuspielen. Die zu ändernde Zeile befindet sich auf dem Bildschirm. Bewegen Sie den COPY-CURSOR mit Hilfe der SHIFT-Taste zum Anfang dieser Zeile:

```
Ready
list
100 CLS
110 PRINT "18 geteilt durch 6 ergibt";
120 PRINT 18/6
Ready
_
```

Drücken Sie nun die COPY-Taste bis zum ersten zu ändernden Zeichen:

```
Ready
list
100 CLS
110 PRINT "18 geteilt durch 6 ergibt";
120 PRINT 18/6
Ready
110 PRINT" _
```

An dieser Stelle erfolgt die erste Änderung der Zeile. Geben Sie nun die Zahl '24' ein. Damit bei nochmaligem Drücken der COPY-Taste nicht die '18' mit übernommen wird, bewegen Sie den COPY-CURSOR auf das Leerzeichen nach der '18' (SHIFT nicht vergessen!)

```
Ready
list
100 CLS
110 PRINT "18 geteilt durch 6 ergibt";
120 PRINT 18/6
Ready
110 PRINT "24_
```

Nun kopieren Sie den restlichen Teil der Zeile mit der COPY-Taste, drücken RETURN, und die Änderung der Zeile ist abgeschlossen. Damit das anschließende Ergebnis auch stimmt, müssen wir noch die Zeile 120 ändern. Verfahren Sie hierzu wie bei der Zeile 110. Sicher, es ist nicht ganz einfach, die Methode mit dem COPY-Cursor zu beherrschen, doch auch hier bringt's die Übung. Ich selbst hatte meine Probleme, diese außergewöhnliche Editierung zu beherrschen, doch nun möchte ich nicht mehr darauf verzichten.

Wenden wir uns nun der zweiten Methode zur Programmänderung zu. Der Befehl 'EDIT' ruft eine bestimmte Zeile zur Änderung auf den Bildschirm. Geben Sie nun den Befehl 'EDIT 110' ein (RETURN nicht vergessen!):

```
Ready  
edit 110  
110 PRINT "24 geteilt durch 6 ergibt";
```

Die Zeile 110 wird angezeigt und der Cursor befindet sich am Anfang dieser Zeile. Bewegen Sie nun den Cursor zu dem ersten zu ändernden Zeichen. In diesem Beispiel wollen wir aus der '24' wieder eine '18' machen. Wenn Sie nun die Zahl 24 eingeben, werden Sie folgendes beobachten:

```
Ready  
edit 110  
110 PRINT "1824 geteilt durch 6 ergibt";
```

Die Zahl '24' wurde nicht mit der Zahl '18' überschrieben, sondern es wurde eingefügt. Der EDIT-Befehl arbeitet grundsätzlich im Einfügemodus. Die Zahl '24' muß also nun noch gelöscht werden. Dazu verwenden Sie die Taste 'CLR', die bekanntlich das

unter dem Cursor befindliche Zeichen löscht. Drücken Sie diese Taste nun zweimal:

```
Ready
edit 110
110 PRINT "18_geteilt durch 6 ergibt";
```

Wenn alle Änderungen abgeschlossen sind, drücken Sie RETURN. Überzeugen Sie sich davon, daß diese Zeile auch wirklich geändert wurde. Geben Sie dazu den Befehl 'LIST' ein.

Sie haben nun zwei Methoden zur Programmänderung kennengelernt. Entscheiden Sie selbst, welche Sie in Zukunft verwenden werden.

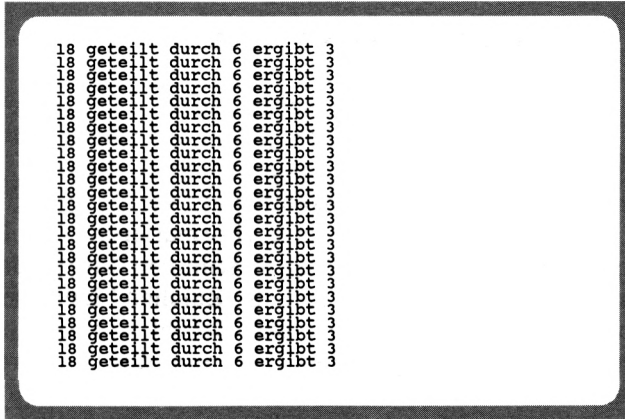
VERZWEIGUNG

Programme werden nur selten wie im letzten Beispiel stur von vorne nach hinten abgearbeitet. Oft werden, meist zu bestimmten Bedingungen, Zeilen übersprungen und an anderer Stelle fortgesetzt. Hier ist ein Sprung erforderlich.

DER BEFEHL 'GOTO' SPRINGT ZU EINER ANGEGBENEN ZEILE

In BASIC gibt es den Befehl 'GOTO', der eine beliebige Zeilennummer anspringt.

Lassen Sie uns dies gleich in die Praxis umsetzen. Bekanntlich endet das Programm nach der Zeile 120. Was geschieht, wenn wir danach, z.B. in Zeile 130 wieder in die Zeile 110 springen? Ich behaupte, es wird ununterbrochen "18 GETEILT DURCH 6 ERGIBT 3" ausgegeben. Doch sehen Sie selbst. Geben Sie die Zeile '130 GOTO 110' ein und starten das Programm mit 'RUN'.

A screenshot of a BASIC program's output, enclosed in a black rectangular frame. The output consists of 20 lines of text, each line containing the string "18 geteilt durch 6 ergibt 3". The text is left-aligned and appears to be printed in a monospaced font. The lines are stacked vertically, creating a continuous stream of the same message.

Wir haben mit der Zeile 130 eine sogenannte Endlosschleife erstellt, da das Programm ständig die Zeilen 110 bis 130 durchläuft. Wenn wir nun nicht eingreifen, läuft das Programm noch stundenlang so weiter.

DIE TASTE 'ESC' UNTERBRICHT DEN PROGRAMMABLAUF

Drücken Sie nun die Taste 'ESC', so hält das Programm an. Eine beliebige andere Taste läßt das Programm wieder fortfahren. Wollen Sie das Programm abbrechen, so drücken Sie die Taste 'ESC' zweimal. Der Rechner meldet 'BREAK IN ...', und zeigt somit an, in welcher Zeile das Programm angehalten wurde. In welcher Zeile das Programm nun gestoppt wird, ist rein zufällig.

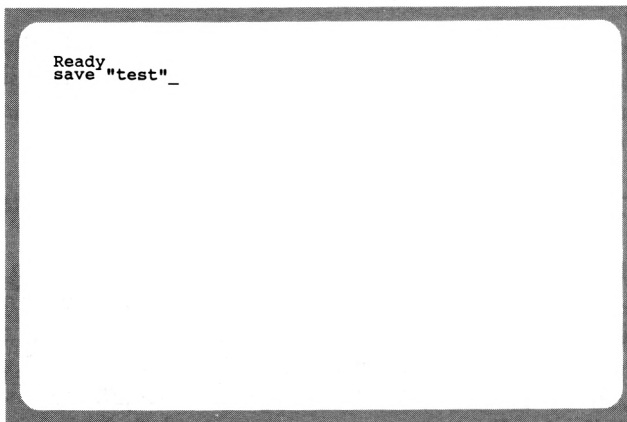
Soll der Programmablauf fortgesetzt werden, so ist auch hierfür ein Befehl vorgesehen. Fortsetzen heißt auf englisch "continue". Da BASIC-Befehle der englischen Sprache angelehnt sind, heißt der entsprechende Befehl 'CONT'.

DER BEFEHL 'CONT' SETZT DEN PROGRAMMABLAUF NACH EINER UNTERBRECHUNG FORT

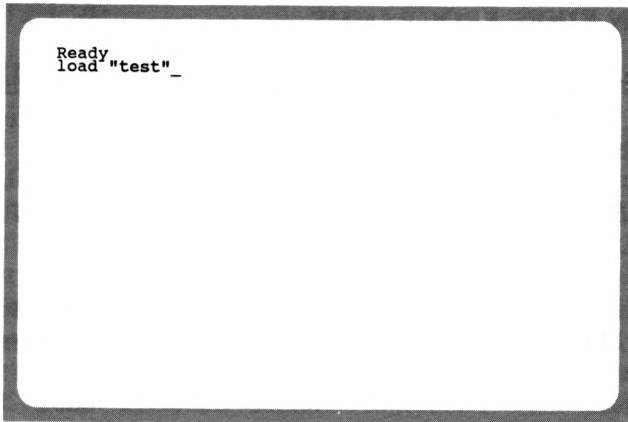
Geben Sie nun diesen Befehl ein. Das Programm befindet sich wieder in der Endlosschleife und kann wiederum mit der Taste 'ESC' unterbrochen oder abgebrochen werden.

SPEICHERN UND LADEN VON PROGRAMMEN

Wenn Sie den Rechner nun ausschalten, so ist das Programm verloren. Da Sie sicherlich nicht nach jedem Einschalten des Geräts das gewünschte Programm eintippen möchten, können die Programme auf dem eingebauten Diskettenlaufwerk abgelegt werden. Speichern wir nun unser Programm (Formatieren von Disketten ist in Kapitel 8 beschrieben). Nun bestimmen Sie einen Namen für das Programm, der maximal 8 Zeichen lang sein darf. Nehmen wir den Namen "TEST". Der Befehl zum Speichern des Programms lautet 'SAVE'. Geben Sie nun den entsprechenden Befehl ein:



Wenn Sie das Programm nun wieder laden und starten möchten, so benutzen Sie den Befehl LOAD.



Sollte Ihnen der Name des Programms entfallen sein, so können Sie den Inhalt der Diskette mit dem Befehl **CAT** auf dem Bildschirm anzeigen lassen.

Das Programm wird mit **RUN** gestartet. Dem Befehl **RUN** kann noch eine Zeilennummer angefügt werden. Hier ein Beispiel:

Befehl	Funktion
RUN	startet ein bereits im Speicher befindliches Programm
RUN 100	startet ein bereits im Speicher befindliches Programm ab der Zeile 100

Das Programm wird also nicht wie gewohnt bei der ersten Zeile, sondern bei dieser hinter dem Befehl angegebenen Zeile gestartet.

So, das war zunächst alles, was Sie an Informationen zum Speichern und Laden mit dem Diskettenlaufwerk benötigen. Im Kapitel 'DAS DISKETTENLAUFWERK' wird an späterer Stelle auf weitere Möglichkeiten eingegangen, die dieses Laufwerk bietet.

LÖSCHEN EINES PROGRAMMS

Wie bereits bekannt, wird das Programm beim Ausschalten des Rechners gelöscht. Doch dies ist sicherlich nicht die einzige Möglichkeit.

DER BEFEHL 'NEW' LÖSCHT DAS IM SPEICHER BEFINDLICHE PROGRAMM

Geben Sie nun den Befehl 'NEW' ein, so wird das Programm gelöscht. Man sollte natürlich das zu löschende Programm vorher abspeichern.

Für alle skeptischen Leser: Wenn Sie nach dem 'NEW' den Befehl 'LIST' eingeben, so werden Sie feststellen, daß das Programm wirklich gelöscht wurde. Es werden keine Zeilen mehr angezeigt. Auch der Befehl 'RUN' ist nun wirkungslos.

Sie haben nun die ersten Grundlagen zur Erstellung eines Programms erworben, was für Sie der Grundstein zum Erarbeiten der anschließenden, etwas anspruchsvolleren Kapitel sein wird. Wenn Sie in diesem Kapitel etwas überfordert wurden, so empfiehlt es sich, dieses Kapitel nochmals durchzuarbeiten.

KAPITEL 4: PROGRAMMIERHILFEN

Das umfangreiche BASIC Ihres CPC6128, auf das Sie wirklich stolz sein können, war Grund für dieses Kapitel. Nicht jeder Rechner weist in der Standardversion Befehle vor, die das Erstellen und Bearbeiten von Programmen um ein erhebliches Maß erleichtern. Da gibt es Befehle zum gezielten Löschen von Zeilen, zur automatischen Zeilennummerierung, um nur zwei der im folgenden beschriebenen Programmierhilfen zu nennen. Jeder, der sich beim Erstellen seiner Programme vom Komfort des BASIC 1.0 verwöhnen lassen möchte, sollte dieses Kapitel nicht ungelesen lassen.

AUTOMATISCHE ZEILENNUMERIERUNG

Beim Erstellen von Programmen ist die Schrittweite meist 10. Das Eingeben der Zeilennummer am Anfang jeder Zeile kann einem mit der Zeit ziemlich auf dem Wecker fallen. Dies haben auch die Schöpfer des in Ihrem Rechner enthaltenen BASICs erkannt und einen komfortablen Befehl zur automatischen Zeilennummerierung entwickelt. Schauen wir uns zunächst die Beschreibung dieses Befehls an:

Problem: **Automatische Zeilennummerierung**

Befehl: **AUTO *start,increment***

Parameter: *start* - **Anfangszeile**
 increment - **Schrittweite**

Beispiel: **AUTO 100,10**
 Beginnend bei der Zeile 100 werden alle
 folgenden Zeilen mit der Schrittweite 10
 vorgegeben (100, 110, 120, 130, 140)

Bemerkung: Die automatische Numerierung kann mit der Taste 'ESC' abgebrochen werden. Ist eine einzugebene Zeile bereits vorhanden, so wird dies mit einem Stern (*) hinter der Zeilennummer signalisiert. Mit RETURN bleibt diese Zeile erhalten.

Hier sehen Sie erstmalig eine genaue Beschreibung eines Befehls, wie sie auch im weiteren Verlauf des Buches bei jedem neuen Befehl zu finden ist. Es taucht ein Fremdwort auf, das Ihnen vielleicht nicht viel sagt: Parameter. Als Parameter bezeichnet man alles, was man dem Befehl übergibt, also hinten anhängt. So muß z.B. bei dem bereits bekannten Befehl 'LOAD' ein Parameter angegeben werden, nämlich der Name des Programms.

Doch zurück zum Befehl AUTO. Angenommen, Sie möchten die ersten Zeilen eines Programms eingeben, das bei der Zeile 10 beginnen und eine Schrittweite von 5 haben soll. Üblich ist, daß vor jeder einzugebenen Zeile die Zeilennummer angegeben wird. Die erste Zeile erhält somit die Nummer 10, die zweite die Nummer 15 und so weiter. Diese Arbeit nimmt Ihnen wie gesagt der Befehl AUTO ab. Probieren Sie nun diesen Befehl aus und geben den folgenden Befehl ein:

AUTO 10,5

Sofort wird die erste Zeile (10) vorgegeben. Der Rechner wartet nun auf die Eingabe dieser Zeile. Geben Sie hier z.B. den Befehl 'PRINT "Zeile 10"' ein und beenden die Zeile mit RETURN. Nun wird die zweite Zeile (15) ausgegeben, deren Eingabe wiederum erwartet wird. Geben Sie hier den Befehl 'PRINT "Zeile 15"' ein. Anschließend beenden Sie die Eingabe der Zeilen mit der Taste 'ESC'. Die Meldung 'READY' erscheint und signalisiert, daß die automatische Zeilennumerierung nicht mehr aktiv ist.

Geben Sie nun nochmals den Befehl 'AUTO 10,5' ein, um eine Besonderheit des Befehls kennenzulernen. Der Rechner reagiert wie folgt:

10*

Was bedeutet nun der Stern? Wenn Sie die Beschreibung des Befehls nicht nur überflogen haben, so werden Sie sicher die Lösung kennen. Der Stern signalisiert, daß diese Zeile bereits vorhanden ist. Dies ist auch wichtig, sonst könnte man sein Programm unabsichtlich überschreiben. Wollen Sie, daß die Zeile unverändert bleibt, so drücken Sie nur die Taste 'RETURN'. Wenn die Zeile jedoch geändert werden soll, so geben Sie diese neue Zeile ein und drücken wieder 'RETURN'. Probieren Sie dies einmal aus und übernehmen die Zeile 10 zunächst mit 'RETURN' und geben bei der Zeile 15 den Befehl 'PRINT "neue Zeile 15"' ein. Danach beenden Sie den Befehl AUTO mit der Taste 'ESC' und schauen sich das Ergebnis mit dem Befehl 'LIST' an.

Sie haben soeben eine Möglichkeit kennengelernt, die Eingabe Ihrer Programme zu vereinfachen. Nutzen Sie dies auch bei der Eingabe der Programmabschnitte in diesem Buch aus.

UMNUMERIERUNG VON ZEILEN

Wie Sie bereits wissen, hat die Schrittweite der Zeilennummern eines Programms keinen Einfluß auf den Programmablauf. Ob Sie die Zeilennummern um jeweils 1,5 oder 10 erhöhen, liegt allein in Ihrem Ermessen. Sie sollten jedoch beachten, daß eine zu kleine Schrittweite das Einfügen von Zeilen erschwert. Lassen Sie uns das Problem an einem praktischen Beispiel verdeutlichen: Geben Sie zunächst das folgende Programm ein:

```

1 CLS
2 PRINT "Wir multiplizieren nun zwei Zahlen:"
3 PRINT "Die erste Zahl ist 15"
4 PRINT "Die zweite Zahl ist 12"
5 PRINT "Das Ergebnis ist";15*12
    
```

Wie Sie sehen, ist dieses Programm mit der Schrittweite von jeweils 1 numeriert worden. Das Programm erzeugt das folgende Ergebnis:

```
Wir multiplizieren nun zwei Zahlen:
Die erste Zahl ist 15
Die zweite Zahl ist 12
Das Ergebnis ist 180
```

Um diese Ausgabe optisch zu verbessern, soll die erste Zeile unterstrichen werden. Die Ausgabe auf dem Bildschirm soll demnach wie folgt aussehen:

```
Wir multiplizieren nun zwei Zahlen:
-----
Die erste Zahl ist 15
Die zweite Zahl ist 12
Das Ergebnis ist 180
```

Dazu muß hinter der Zeile 2 des Programms eine Zeile eingesetzt werden. Hier ist es erforderlich, die Zeilen hinter der Zeile 2 um eins zu erhöhen, um der neuen Zeile die Nummer 3 geben zu können. Das vollständige Programm ist dann dieses:

```
1 CLS
2 PRINT "Wir multiplizieren nun zwei Zahlen:"
3 PRINT "-----:"
4 PRINT "Die erste Zahl ist 15"
5 PRINT "Die zweite Zahl ist 12"
6 PRINT "Das Ergebnis ist";15*12
```

Wie würden Sie nun dieses Problem angehen? Sicher würden Sie die Zeilen 3 bis 5 erneut mit der neuen Zeilennummer eingeben. Doch dies ist nicht im Sinne des Erfinders. Ihr komfortables BASIC hält dazu einen Befehl für Sie bereit, dessen Beschreibung wir uns zunächst ansehen wollen:

- Problem:** Umnumerieren von Zeilen
- Befehl:** RENUM *neu, alt, increment*
- Parameter:** *neu* - neue Anfangszeile
alt - alte Anfangszeile
increment - neue Schrittweite
- Beispiel:** RENUM 100,10,5
Ab der Zeilennummer 10 des Programms wird mit der ersten Zeilennummer 100 und der Schrittweite 5 neu numeriert.
- Bemerkung:** Die letzte Zeile des Programms darf nach der neuen Numerierung die Nummer 65535 nicht überschreiten, sonst wird der Befehl RENUM nicht ausgeführt und die Fehlermeldung "Improper argument" ausgegeben

Dieser Befehl ist etwas komplizierter als die bisher beschriebenen. Doch dies soll uns nicht entmutigen. Sie sehen, daß diesem Befehl drei Parameter übergeben werden müssen. Da ist zunächst die Zeilennummer, die als erste zur Neunumerierung verwendet wird. Danach folgt die Zeilennummer, ab der neu numeriert werden soll. Sie müssen also nicht grundsätzlich das gesamte Programm neu numerieren. Der dritte Parameter gibt schließlich die neue Schrittweite an.

Wie muß nun der Befehl RENUM aufgebaut sein, um in unserem Beispiel die Zeilen 3 bis 5 umzuwandeln in 4 bis 6? Bauen wir dazu den Befehl Schritt für Schritt auf. Zunächst muß die erste neue Zeilennummer angegeben werden. Dies ist die Nummer 4. Die erste alte Zeilennummer stellt den zweiten Parameter dar und ist 3. Die neue Schrittweite soll weiterhin 1 betragen. Der gesamte Befehl lautet dann:

RENUM 4, 3, 1

Dieser Befehl führt zum folgenden Erfolg, den Sie wieder mit LIST begutachten können:

```
1 CLS
2 PRINT "Wir multiplizieren nun zwei Zahlen:"
4 PRINT "Die erste Zahl ist 15"
5 PRINT "Die zweite Zahl ist 12"
6 PRINT "Das Ergebnis ist";15*12
```

Es wurde mit einem Befehl das gesamte Programm so gestaltet, daß nun Platz für die Zeile 3 ist, die wir nun eingeben können:

```
3 PRINT "....."
```

Um derartige Probleme in Zukunft zu vermeiden, arbeiten wir mit der Schrittweite 10. Wie muß der Befehl 'RENUM' aufgebaut sein, um unser Beispielprogramm ab der Zeile 100 mit der Schrittweite 10 neu zu nummerieren? Dies dürfte für Sie nun kein Problem mehr sein. Geben Sie dazu den folgenden Befehl ein:

```
RENUM 100,1,10
```

Der Befehl RENUM numeriert nicht nur die Zeilen, sondern führt auch etwas ganz wichtiges aus, er ändert die Sprungbefehle auf die Zeilen mit um. Was heißt das? Probieren wir dies aus. Geben Sie die letzte Zeile des folgenden Programms zusätzlich ein:

```
100 CLS
110 PRINT "Wir multiplizieren nun zwei Zahlen:"
120 PRINT "....."
130 PRINT "Die erste Zahl ist 15"
140 PRINT "Die zweite Zahl ist 12"
150 PRINT "Das Ergebnis ist";15*12
160 GOTO 110
```

Wir haben mit dem letzten Befehl eine Endlosschleife erzeugt. Wenn wir nun das Programm neu numerieren, so erhält die Zeile 110 eine andere Nummer. Der GOTO-Befehl darf aber nicht mehr in die Zeile 110 springen, da diese nach der neuen Nummerierung sicher nicht mehr vorhanden ist. Um dies zu verhindern, ändert der Befehl RENUM nicht nur die Nummern am Anfang jeder Zeile, sondern auch die Sprungadresse auf diese Zeilen. Dies ist sehr wichtig, da der Befehl sonst nicht einzusetzen wäre. Überzeugen Sie sich davon, indem Sie den Befehl 'RENUM 10,100,5' eingeben. Wenn Sie nun das Programm mit 'LIST' ausgeben, bemerken Sie, daß die Sprungadresse hinter dem GOTO-Befehl ebenfalls sinngemäß geändert wurde.

Sie beherrschen nun einen Befehl, der Ihnen in Zukunft oft hilfreich zur Seite stehen wird.

GEZIELTES LÖSCHEN VON ZEILEN

Sie haben bereits den Befehl 'NEW' kennengelernt, der das im Speicher befindliche Programm löscht. Doch was, wenn nicht das gesamte Programm, sondern nur eine oder mehrere Zeilen gelöscht werden sollen? Das Löschen einer einzelnen Zeile bereitet keinerlei Schwierigkeiten. Sie geben einfach die Zeilennummer und anschließend RETURN ein. Um dies zu testen, löschen wir das alte Programm und geben ein neues ein. Verwenden Sie dazu den Befehl AUTO:

```
10 PRINT  
20 PRINT  
30 PRINT  
40 PRINT  
50 PRINT  
60 PRINT  
70 PRINT  
80 PRINT  
90 PRINT
```

Dieses Programm gibt zwar nur 9 Leerzeilen aus, was für unsere folgenden Übungen jedoch vollkommen ausreicht. Löschen Sie nun die Zeile 50, indem Sie diese Nummer gefolgt von der Taste 'RETURN' eingeben. Wenn Sie anschließend das Programm listen, so sehen Sie den Erfolg. Die Zeile 50 existiert nicht mehr. Was aber, wenn mehrere Zeilen, z.B. 20 bis 40 gelöscht werden sollen? Sicher, Sie können die Zeilen 20, 30 und 40 einzeln löschen, doch dies ist bei größeren Zeilenbereichen recht umständlich. Ein weiterer Befehl erleichtert uns diese Arbeit:

Problem: gezieltes Löschen von Zeilenbereichen

Befehl: DELETE *anfang-ende*

Parameter: *anfang* - erste zu löschende Zeile
ende - letzte zu löschende Zeile

Beispiel: DELETE 40-130
 löscht die Zeilen 40 bis 130

Nun haben Sie die Lösung: Um die Zeilen 20 bis 40 unseres Beispielprogramms zu löschen, geben Sie den folgenden Befehl ein:

DELETE 20-40

Es gibt mehrere Möglichkeiten, die beiden Parameter anzugeben. Sehen Sie dazu die folgende Tabelle:

DELETE 10	löscht die Zeile 10
DELETE 100-150	löscht die Zeilen 100 bis 150
DELETE 150-	löscht alle Zeilen ab einschließlich 150
DELETE -100	löscht alle Zeilen bis einschließlich 100

Die Möglichkeit, den Zeilenbereich derartig zu variieren, ist ebenfalls für den Befehl LIST gegeben. Auch hier verdeutlicht die folgende Aufstellung, was gemeint ist:

- LIST 10 listet die Zeile 10
- LIST 100-150 listet die Zeilen 100 bis 150
- LIST 150- listet alle Zeilen ab einschließlich 150
- LIST -100 listet alle Zeilen bis einschließlich 100

FUNKTIONSTASTEN

Nun werden Sie eine der komfortabelsten Programmierhilfen kennenlernen. Oft ist es ärgerlich, häufig benutzte Befehle immer wieder eingeben zu müssen. Ihr CPC6128 bietet Ihnen die Möglichkeit, häufig benutzte Zeichenfolgen auf Tastendruck zu erzeugen. Dazu vorgesehen sind die 12 Tasten Ihres Ziffernblocks:

7	8	9
4	5	6
1	2	3
0	.	ENTER

Auf jede dieser Tasten können Sie nun eine Folge von maximal 32 Zeichen legen, insgesamt jedoch nicht mehr als 120 Zeichen. Schauen wir uns dazu die Beschreibung des entsprechenden Befehl an:

- Problem:** Belegung der Funktionstasten
- Befehl:** KEY *n*, *string*
- Parameter:** *n* - Nummer der Funktionstaste
string - Zeichenkette (max. 32 Zeichen)
- Beispiel:** KEY 0,"LIST"
 Legt den Befehl LIST auf die Funktionstaste 0.
- Bemerkung:** Die Gesamtzahl der gespeicherten Zeichen darf 120 nicht überschreiten.

Ist der Befehl nicht äußerst interessant? Lassen Sie uns ihn gleich einsetzen. Auf dem Zifferntastenblock gibt es 13 Funktionstasten mit den Nummern 0 bis 12. Sicher werden Sie sich fragen, wie 13 Funktionstasten mit 12 Tasten realisiert werden. Dies ist möglich, da die Taste 'ENTER' des Tastenblocks zwei Funktionen hat. Eine durch normale Betätigung und eine zusammen mit der Taste 'CTRL'. Das folgende Diagramm zeigt Ihnen, welche Nummern den Tasten zugewiesen sind:

7	8	9
4	5	6
1	2	3
0	10	11/12

Die Taste '.' hat also die Nummer 10, die Taste 'ENTER' die Nummern 11 und 12. Lassen Sie uns nun einige Tasten mit sinnvollen Befehlen belegen. Der Befehl 'LIST' wird oft benötigt. Wir legen ihn auf die Funktionstaste 0. Der folgende Befehl führt dies aus:

KEY 0,"LIST"

Vergessen Sie nie das Leerzeichen zwischen dem Befehl und der Nummer! Ihr Rechner nimmt es Ihnen sonst übel (Syntax error). Drücken Sie nun die Funktionstaste '0', so erscheint der Befehl 'LIST' auf dem Bildschirm. Doch dieser Befehl wird noch nicht ausgeführt. Dazu müssen Sie die Taste 'RETURN' drücken. Doch auch das 'RETURN' können Sie mit der Funktionstaste auslösen. Dazu müssen Sie den internen Code für das 'RETURN' mit einem speziellen Befehl, den wir später noch kennenlernen, anhängen. Dieser Code ist 13. Der Befehl sieht dann wie folgt aus:

KEY 0, "LIST"+CHR\$(13)

Dieser Befehl bringt viel neues, das Sie nicht verwirren soll. Wichtig ist zunächst, was dieser Befehl bewirkt. Drücken Sie die Funktionstaste 0 und Sie werden sehen, daß der Befehl unmittelbar ausgeführt wird.

Es gibt jedoch auch Befehle, denen man kein 'RETURN' anhängen sollte. Einer dieser Befehle ist 'NEW'. Stellen Sie sich vor, dieser Befehl liegt mit angehängtem 'RETURN' auf einer der Funktionstasten. Wenn Sie dann diese Funktionstaste versehentlich berühren, so wird Ihr Programm gelöscht. Hängen Sie also nur solchen Befehlen ein 'RETURN' an, die durch Fehlbedienung nichts anrichten können.

Sie haben bereits den Code für 'RETURN' kennengelernt. In Verbindung mit den Funktionstasten ist ein weiterer Code wichtig, der des Anführungszeichens. Versuchen Sie einmal, den bereits bekannten Befehl 'RUN' auf die Funktionstaste 1 zu legen! Sie bekommen bestimmt graue Haare! Wo liegt das Problem? Der Befehl 'KEY 1,"RUN"' führt zum 'Syntax error'. Der Rechner erkennt das zweite Anführungszeichen als Ende der Zeichenkette an und weiß mit dem dritten gar nichts anzufangen. Wir müssen also hier den Code für das Anführungszeichen (34) verwenden! Der korrekte Befehl lautet dann:

KEY 1, "RUN"+CHR\$(34)

Wenn nun auch noch ein 'RETURN' angehängt werden soll, erscheint der Befehl recht kompliziert:

```
KEY 1, "RUN"+CHR$(34)+CHR$(13)
```

Sie müssen sich lediglich die beiden Codes 13 und 34 merken!

Mit der Zeit werden Sie eine Standardbelegung Ihrer Funktionstasten entworfen haben, die Sie nicht immer wieder nach dem Einschalten des Rechners belegen müssen. Sie können die Sammlung der KEY-Befehle in einem Programm festhalten, daß Sie jederzeit erreichbar auf einer Kassette abspeichern. Dieses Programm könnte dann in etwa so aussehen:

```
10 KEY 0, "LIST"+CHR$(13)
20 KEY 1, "PRINT "
30 KEY 2, "NEW"
40 KEY 3, "RENUM "
50 KEY 4, "AUTO "
60 KEY 5, "DELETE "
```

Entwickeln Sie nach oder während der Erarbeitung des folgenden Kapitels Ihr persönliches Funktionstastenprogramm!

Noch ein wichtiger Hinweis zum Schluß: Der Dreifingergriff (SHIFT/CTRL/ESC), der den Rechner in den Einschaltzustand versetzt, löscht alle Funktionstasten!

KAPITEL 5:

BASIC EINFÜHRUNG

Dieses Kapitel soll eine Einführung in die Programmiersprache BASIC darstellen. Dabei sollen nicht alle BASIC-Befehle stur nacheinander beschrieben werden, sondern es wird eine Adressenverwaltung Schritt für Schritt aufgebaut, in der die Befehle dann zum gegebenen Zeitpunkt beschrieben und eingesetzt werden. Der Leser dieses Buches soll durch dieses Kapitel nicht zum perfekten Programmierer werden, sondern einen intensiven Einblick in die praxisnahe Programmierung erhalten. Dieser Einblick bietet ihm dann beste Voraussetzungen, sich durch weitere Fachbücher weiterzubilden.

PROBLEMBESCHREIBUNG ZUR ADRESSENVERWALTUNG

Eine Adressenverwaltung ist eines der beliebtesten Programme auf dem Heimcomputer. Die breite Einsatzfähigkeit trägt wesentlich dazu bei. Es gibt kaum jemand, der dieses Programm nicht einzusetzen weiß. Zwar ist eine Adressenverwaltung mit nur wenig Adressen per Heimcomputer nicht unbedingt effektiver als die herkömmliche Methode mit dem Adressenbüchlein, aber erheblich eindrucksvoller. Wenn sich jedoch sehr viele Adressen ansammeln, so sind sie in einem Heimcomputer bestens aufgehoben.

Welche Möglichkeiten sollte ein solches Programm bieten? Nun, auf jeden Fall müssen Adressen ein- und ausgegeben werden.

Bevor wir weitere Ansprüche an das Programm stellen, muß noch etwas klargestellt werden. In einem Programm, das Daten verwaltet, unterscheidet man zwischen zwei grundsätzlichen Bestandteilen:

- 1. Das Programm**
- 2. Die Daten**

Die Daten sind nicht Bestandteil des Programms. Dies wäre zwar möglich, aber diese Daten können dann nur vom Programmierer und nicht vom Anwender verwaltet werden. Eine Änderung der Daten hätte gleichzeitig eine Änderung des Programms zur Folge, was wir sofort wieder vegessen können.

Das Programm befindet sich also auf dem externen Speichermedium (Diskette) und wird bei Bedarf in den Rechner eingeladen. Aber was ist nun mit den Daten? Sie fallen erstmalig bei der Ersterfassung an. Wohin dann damit?

Wir müssen eine Datei organisieren. Eine Datei ist eine Ansammlung von Daten auf einem externen Speichermedium. Es gibt zwar mehrere Dateiorganisationsformen, wir beschäftigen uns aber nur mit der einfachsten Methode, der sequentiellen Datei. Sequentiell bedeutet, daß die Daten hintereinander angeordnet sind. Die zwei wichtigen Bestandteile unserer Adressenverwaltung sind nun

- 1. Das Programm**
- 2. Die Datei**

DATEIORGANISATION

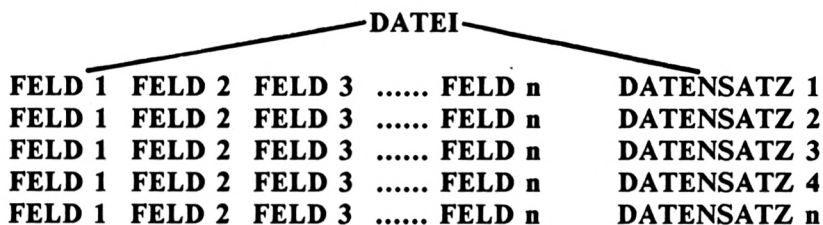
Wenn ich jetzt fragen würde, welcher Teil zuerst organisiert werden muß, werden viele antworten, "das Programm". Die Antwort ist aber falsch. Zuerst muß man sich im klaren sein, was wie wo abgespeichert wird. Über das "wie" sind wir uns im klaren, sequentiell soll gespeichert werden.

Wir wollen unsere Daten auf dem eingebauten Diskettenlaufwerk abspeichern.

Bleibt nun noch die Frage, was gespeichert werden soll. "Adressen natürlich" werden viele spontan reagieren, doch sind wir uns überhaupt im Klaren, was alles zu einer Adresse gehören soll? Natürlich nicht. Sammeln wir doch einmal alles, woraus sich eine Adresse zusammensetzen könnte:

Anrede	Postleitzahl
Vorname	Ort
Name	Telefonnummer
Strasse	Bemerkung

Diese einzelnen Bestandteile einer Adresse nennt man DATENFELDER. Man spricht so z.B. vom Feld "Anrede". Die Gesamtheit der Datenfelder ergibt den DATENSATZ. Jeder Datensatz besteht also in unserem Beispiel aus diesen 8 Feldern. Alle Datensätze zusammen ergeben schließlich die Datei. Sehen wir uns diese Struktur einmal in folgendem Bild an:



Die Hierarchie ist also DATEI - DATENSATZ- DATENFELD

RECHNERINTERNE SPEICHERUNG DER DATEN

Mit einer sequentiellen Datei läßt sich zwar sehr bequem arbeiten, jedoch gibt es hier auch Nachteile gegenüber anderen Organisationsformen. Stellen Sie sich vor, die gesamte Datei mit den Adressen befindet sich auf einer Kassette. Sie möchten nun eine ganz bestimmte Adresse aus dieser Datei haben. Hier tritt dann das Problem auf. Sie können nicht aus einer Datei nur einen Datensatz lesen. Die Floppystation ist nicht in der Lage, aus einer sequentiellen Datei einzelne Datensatz zu lesen. Wie kann man nun auf die Adressen zugreifen?

Um auf einen Datensatz zuzugreifen, muß die gesamte Datei in den Rechner eingelesen werden. Eine sequentielle Datei darf demnach nicht größer als der vorhandene Speicherplatz des Rechners sein. Ein Vorteil stellt sich jedoch wieder heraus. Wenn die Datei einmal eingelesen ist, so kann im Rechner blitzschnell auf die Datensätze zugegriffen werden. Für den gesamten Programmablauf gilt also folgende Regelung:

- 1. Datei laden**
- 2. Datensätze lesen, ändern, löschen**
- 3. Datei speichern**

Nach dem Start des Programms muß also zuerst die Adressendatei komplett eingelesen werden. Danach können die Datensätze verarbeitet werden. Vor Beendigung des Programms muß die Datei jedoch wieder gespeichert werden, sofern sie geändert wurde. Wurden die Datensätze jedoch nicht geändert und auch keine gelöscht oder hinzugefügt, so stimmt die Datei noch mit der ursprünglich eingeladenen Datei überein. Sie braucht also nicht wieder gespeichert werden.

VARIABLEN

Wie bekannt, werden die Datensätze im Rechner gespeichert. Die Frage ist nur, wo und wie werden sie gespeichert?

RECHNERINTERNE DATEN WERDEN IN VARIABLEN GESPEICHERT

Variablen sind also das Zauberwort. Variablen sind Speicherbereiche im CPC6128, die mit einem Namen versehen werden. Durch Angabe dieses Namens kann auf diese Speicherbereiche zugegriffen werden. Wir haben schon zwei verschiedene Datentypen kennengelernt. Da gibt es numerische Daten (Zahlen) und alphanumerische Daten (Strings, Text). Stringvariablen werden durch das Zeichen '\$' hinter der Variablenbezeichnung gekennzeichnet. Woraus setzen sich nun die Bezeichnungen der Variablen zusammen? Nun, zunächst umfaßt eine Variablenbezeichnung höchstens 40 Zeichen. Das erste Zeichen MUSS ein Buchstabe sein, weitere Zeichen dürfen Zahlen oder Buchstaben sein. Es gibt wenige Ausnahmen. BASIC-Befehle oder Funktionen dürfen nicht verwendet werden:

z.B. PRINT, PI, KEY

Beispiele numerische Variablen:

ZAEHLER
PLZ
TELEFON
BETRAG

Beispiele Stringvariablen:

ANREDE\$
ARTIKELBEZEICHNUNG\$
KONTOBEZEICHNUNG\$
X\$

Bei der Auswahl der Variablen sollten Sie sinnvolle Bezeichnungen verwenden. So wird die Variable zum Speichern des Vornamens z.B. "VORNAMES" genannt. Den Einkaufspreis würde man "EINKAUFSPR" nennen, usw. Sinnvolle Variablennamen tragen wesentlich zur Übersichtlichkeit des Programms bei.

VARIABLENVERARBEITUNG

Wenden wir uns nun wieder nach langer Zeit dem Rechner zu. Wir werden nun Variablen einrichten, verarbeiten und ausgeben.

Die Speicherung von numerischen Daten in Variablen z.B. ist eigentlich ein Kinderspiel. Nehmen wir an, die Zahl 45.12 soll in der Variablen EINKAUFSPR gespeichert werden. Der entsprechende Befehl ist dann

```
EINKAUFSPR=45.12
```

Ist doch einfach, oder? Soll diese Variable ausgegeben werden, so lautet der Befehl

```
PRINT EINKAUFSPR
```

Geben Sie beide Befehle nun einmal in den Rechner ein. Soll die Variable EINKAUFSPR wieder gelöscht werden, so geben Sie sinngemäß den Befehl

```
EINKAUFSPR=0
```

ein. Die Variablen können aber auch für Rechenoperationen benutzt werden. Geben wir einmal den doppelten Wert von EINKAUFSPR aus. Wenn Sie die Variable gelöscht haben, so geben Sie bitte erneut 'EINKAUFSPR=45.12' ein.

```
PRINT EINKAUFSPR*2
```

Der Rechner verdoppelt EINKAUFSPR und gibt das Ergebnis (90.24) aus. Die Variable selbst behält den Wert 45.12. Was aber ist zu tun, wenn die Variable EINKAUFSPR verdoppelt werden

soll? Da die Variable einen neuen Wert erhalten soll, beginnt der entsprechende Befehl mit 'EINKAUFSPR='. Der gesamte Befehl zur Verdoppelung ist:

```
EINKAUFSPR=EINKAUFSPR*2
```

Es wird immer erst der Wert rechts vom Gleichheitszeichen errechnet und dann in die Variable links vom Gleichheitszeichen abgelegt.

Stringvariablen werden auf ähnliche Art und Weise eingerichtet. Speichern wir nun den String "FRANKFURT" in die Stringvariable STADT\$. Der Befehl dazu lautet

```
STADT$="FRANKFURT"
```

Auch Stringvariablen werden mit dem Befehl PRINT auf dem Bildschirm ausgegeben. Sehen Sie selbst:

```
PRINT STADT$
```

Was geschieht? Natürlich, der in STADT\$ gespeicherte String wird ausgegeben. Soll diese Stringvariable wieder gelöscht werden, so muß ein sogenannter Leerstring in STADT\$ gespeichert werden.

```
STADT$=""
```

Ein Leerstring besteht also aus zwei unmittelbar aufeinanderfolgenden Anführungszeichen. Selbstverständlich kann mit diesen Strings nicht gerechnet werden. Auch nicht, wenn eine Zahl als String abgelegt wird. Das heißt, wenn Sie z.B. X\$="123" eingeben würden, so können Sie trotzdem nicht ohne weiteres mit X\$ rechnen.

Strings können jedoch verkettet werden. Die folgende Befehlsfolge bestätigt es:

```
STADT1$="DUESSELDORF"  
STADT2$="FRANKFURT"  
STADT$=STADT1$+STADT2$  
PRINT STADT$
```

Zuerst wurden die beiden Städte in STADT1\$ und STADT2\$ gespeichert. Danach wurde ein dritter String (STADT\$) gebildet, der beide Strings aufgenommen hat. Der vierte Befehl hat den String STADT\$ dann ausgegeben.

Es wäre sinnvoll, zwischen den beiden Städten ein Leerzeichen zu setzen. Dazu müsste der dritte Befehl entsprechend geändert werden:

```
STADT$=STADT1$+" "+STADT2$
```

Es wurde also ein Leerzeichen zwischen beide Strings gesetzt.

Wir haben jetzt die zwei wichtigsten Variablentypen kennengelernt. Die numerische Variable und die Stringvariable, die durch ein zusätzliches '\$' gekennzeichnet ist. In solchen Variablen können auch Adressen gespeichert werden. Dies ist jedoch nicht sinnvoll. Zur Speicherung von mehreren gleichartig aufgebauten Datensätzen im Rechner werden andere Variablen benötigt.

TABELLEN

Tabellen werden in der Datenverarbeitung oft eingesetzt. Wenn Sie mehrere Daten einer Gruppe im Rechner speichern wollen, so ist es sehr umständlich, jedem Datenelement dieser Gruppe einen Variablennamen zu geben. In diesem Fall erhalten alle Datenelemente den gleichen Variablennamen, der jedoch zusätzlich gekennzeichnet ist. Nehmen wir das Beispiel einer Gruppe von fünf Städtenamen:

**BONN
PARIS
LONDON
ROM
MADRID**

Um diese fünf Städte im Rechner zu speichern, wären nach der bisherigen Methode 5 Variablen erforderlich. Wenn wir diese Datengruppe jedoch in einer Tabelle (auch Array genannt) speichern würden, so wäre ein Variablenname ausreichend. Da die Datenelemente jedoch unterschieden werden müssen, werden sie gekennzeichnet. Dazu wird hinter der Variablen eine in Klammern gesetzte Zahl angegeben. Diese Zahl bezeichnet man als Index. Das erste Datenelement erhält so den Index 1, das zweite den Index 2, usw. Der Index fängt zwar bei 0 und nicht bei 1 an, jedoch geht dadurch die Übersicht verloren, wenn das erste Datenelement den Index 0 erhält. Man kann diesen Index 0 ignorieren. Dies vergeudet zwar Speicherplatz, doch werden gleichzeitig Fehlerquellen vermieden.

Eine Tabellen- oder Arrayvariable ist z.B. **ADRESSE\$(1)**. Auch **XY(212)** ist eine numerische Arrayvariable. Der Index ist also beliebig groß anzusetzen. Es gibt hier nur zwei Dinge zu beachten:

1. **Übersteigt der Index die Zahl 10, so muß für diese Tabelle Speicherplatz reserviert werden.**
2. **Die Tabelle darf nicht so groß definiert werden, daß sie den Speicher überschreitet.**

Es werden also Tabellen bis zum Index 10 selbständig vom Rechner verwaltet. Doch wie werden größere Arrays definiert? Hierfür gibt es einen speziellen BASIC-Befehl, den Befehl 'DIM'.

- Problem:** Dimensionierung von Arrays
- Befehl:** DIM *v1* (*n1,n2,..*)
- Parameter:** *v1* - Variablenname des Arrays
n1,n2... - maximaler Index der einzelnen Dimensionen
- Beispiel:** DIM D\$(20)
 Es soll ein Array mit der Bezeichnung D\$ bis zum Index 20 eingerichtet werden.
- Bemerkung:** Jedes Array darf im Programm nur EINMAL dimensioniert werden, sonst Fehlermeldung "Array already dimensioned"

Wenn der maximale Index z.B. 5 sein soll, so ist keine DIM-Anweisung notwendig, da bis zum Index 10 der Rechner dies selbst durchführt. Die Sache hat nur einen kleinen Haken: Wird nur bis zum Index 5 gearbeitet, so wird auch Platz für den Index 6-10 reserviert, der eigentlich nicht benötigt wird. Hier kann man Abhilfe schaffen, indem eine DIM-Anweisung mit dem Index 5 eingesetzt wird. Der Rechner reserviert dann auch nur Platz bis zum 5. Index.

Doch nun zurück zu unseren fünf Städtenamen, die in einer Tabelle abgelegt werden sollen. Wenn die Variable STADT\$ verwendet wird, werden die Städte mit den folgenden Zuweisungen gespeichert.

```
10 STADT$(1)="BONN"
20 STADT$(2)="PARIS"
30 STADT$(3)="LONDON"
40 STADT$(4)="ROM"
50 STADT$(5)="MADRID"
```

Man könnte hier auch zuvor die Anweisung 'DIM STADT\$(5)' geben, wenn der Index 6 bis 10 nicht benötigt wird.

Wird ein Index benutzt, der über die reservierte Tabelle hinaus-schießt, so wird die Fehlermeldung "Subscript out of range" gegeben. Den Index bezeichnet man auch als Subskript.

In einem weiteren Beispiel sollen nun die fünf Länder der zuvor gespeicherten Hauptstädte ebenfalls in einer Tabelle abgelegt werden. Die Tabelle erhält den Namen LAND\$. Folgende Zuweisungen müssen gemacht werden:

```
60 LAND$(1)="BRD"  
70 LAND$(2)="FRANKREICH"  
80 LAND$(3)="ENGLAND"  
90 LAND$(4)="ITALIEN"  
100 LAND$(5)="SPANIEN"
```

Nun haben wir zwei Tabellen erstellt, deren Index in Zusammen-hang stehen. Das heißt, STADT\$(1) enthält die Hauptstadt von LAND\$(1). Allgemein kann gesagt werden, STADT\$(X) ent-hält die Hauptstadt von LAND\$(X).

Der besondere Reiz an Tabellen ist, daß als Index nicht nur eine Zahl, sondern auch Variablen oder beliebige Ausdrücke eingesetzt werden können. Eine Arrayvariable mit der Bezeichnung 'DS\$(X-2*I+13)' ist somit durchaus möglich. Der Vorteil dieser Indizierung ist unverkennbar.

Diese beiden soeben erstellten Tabellen bezeichnet man als ein-dimensional, da sie nur einen Index besitzen. Nun können Tabellen aber beliebig viele Indizes erhalten. Anstatt der beiden zuvor aufgebauten Tabellen kann auch eine zweidimensionale Tabelle erstellt werden. Beachten Sie zunächst die entsprechende Befehlsfolge:

```
10 DIM TABELLE$(5,1)  
20 TABELLE$(1,0)="BONN"  
30 TABELLE$(2,0)="PARIS"  
40 TABELLE$(3,0)="LONDON"  
50 TABELLE$(4,0)="ROM"  
60 TABELLE$(5,0)="MADRID"  
70 TABELLE$(1,1)="BRD"
```

```

80 TABELLE$(2,1)="FRANKREICH"
90 TABELLE$(3,1)="ENGLAND"
100 TABELLE$(4,1)="ITALIEN"
110 TABELLE$(5,1)="SPANIEN"

```

Die DIM-Anweisung in Zeile 10 ist zwar nicht erforderlich, da der Rechner ohnehin 'DIM D\$(10,10)' ausführen würde, jedoch würden mit dieser Anweisung 11*11 (121) Plätze reserviert. Wir benötigen jedoch nur 10 Tabellenplätze. Bei zwei- und mehrdimensionalen Tabellen sollte man die Größe der Tabellen möglichst genau bestimmen, da sonst viel Speicherplatz vergeudet wird.

Wir haben nun eine zweidimensionale Tabelle, in der der zweite Index die Datengruppe (Stadt oder Land) identifiziert und der erste Index den Tabellenplatz. Man kann diese Tabelle auch als Matrix darstellen:

	0	1
1	BONN	BRD
2	PARIS	FRANKREICH
3	LONDON	ENGLAND
4	ROM	ITALIEN
5	MADRID	SPANIEN

Kommen wir nun wieder zu unserer Adressenverwaltung zurück. Eine solche zweidimensionale Tabelle ist ideal für die Aufnahme der sequentiellen Adressendatei. Der erste Index numeriert die Datensätze, der zweite Index numeriert die Datenfelder innerhalb der Datensätze. Da unsere Datensätze 8 Felder enthalten, ist der zweite Index maximal 8. Der erste Index, also die Anzahl der

Adressen, ist wahlweise. Legen wir uns hier auf maximal 200 Adressen fest. Der Name der Tabelle soll ADRESSE\$ sein. Die DIM-Anweisung zum Einrichten dieser Tabelle ist dann

DIM ADRESSE\$(200,8)

Wie die Tabelle verwaltet wird, erfahren Sie an späterer Stelle.

DATENEINGABE ÜBER TASTATUR

In den bisherigen Übungen haben wir nur Daten verarbeitet und ausgegeben. Interessant wird es erst, wenn dem Programm Daten über die Tastatur übergeben werden. Der Befehl dazu heißt INPUT. Doch INPUT alleine hat keine Wirkung. Dem Befehl folgen zwei Parameter: ein String, der vor dem eigentlichen INPUT ausgegeben wird und eine Variable, in der die eingegebenen Daten gespeichert werden sollen. Wichtig: Der Befehl INPUT kann nicht im Direktmodus, sondern nur im Programm eingesetzt werden. Doch beachten Sie zunächst die Befehlsbeschreibung:

Problem: Dateneingabe über Tastatur

Befehl: INPUT *string*;*vl*

Parameter: *string* - beliebige Zeichenkette
vl - Variable, in der die Eingabe gespeichert wird.

Beispiel: INPUT "ZAHL";X
Es wird der String "ZAHL" ausgegeben und anschließend ein numerischer Wert von der Tastatur eingelesen und in der Variablen X gespeichert.

Bemerkung: Der Befehl kann nicht im Direktmodus eingesetzt werden.

Geben Sie nun den Befehl NEW ein, der ein zuvor im Speicher befindliches Programm löscht. Die folgenden BASIC-Zeilen demonstrieren den Einsatz des Befehls INPUT:

```
10 PRINT "KREISBERECHNUNG"  
20 PRINT "-----"  
30 INPUT "DURCHMESSER ";DURCHM  
40 PRINT "KREISFLAECHE:";(DURCHM/2)^2*PI
```

Starten Sie dieses Programm mit dem Befehl RUN. In Zeile 30 wird der Durchmesser des zu berechnenden Kreises abgefragt. Die Eingabe wird stets mit RETURN abgeschlossen. Hinter dem String nach dem INPUT muß ein Semikolon angegeben werden. Der String kann aber auch weggelassen werden. Das folgende Programm bestätigt es:

```
10 PRINT "KREISBERECHNUNG"  
20 PRINT "-----"  
30 INPUT DURCHM  
40 PRINT "KREISFLAECHE:";(DURCHM/2)^2*PI
```

Sie brauchen natürlich nicht das gesamte Programm neu einzugeben, sondern nur die Zeile 30 entsprechend zu ändern. Sie bemerken, daß der String vor dem INPUT nur dokumentierende Aufgaben hat. Die Dokumentation des INPUT kann aber auch ein PRINT in der Zeile zuvor übernehmen:

```
10 PRINT "KREISBERECHNUNG"  
20 PRINT "-----"  
25 PRINT "DURCHMESSER ";  
30 INPUT DURCHM  
40 PRINT "KREISFLAECHE:";(DURCHM/2)^2*PI
```

Geben Sie zusätzlich die Zeile 25 ein und starten nochmals das Programm. Es läuft genauso wie die erste Version ab. Wichtig ist das Semikolon hinter dem PRINT in Zeile 25. Lassen Sie es weg, so erfolgt der INPUT-Befehl in der Zeile darunter. Probieren Sie es aus.

Auch Strings können über die Tastatur mit dem Befehl INPUT eingelesen werden. Auch hier zunächst ein Beispiel. Löschen Sie zuvor das alte Programm mit NEW.

```
10 PRINT "WIE HEISSEN SIE?"
20 INPUT NAMES$
30 PRINT "GUTEN TAG ";NAMES$; ", WIE GEHT ES IHNEN?"
```

Aus der Zeile 10 und 20 kann auch eine Zeile gebildet werden. Der String vor dem INPUT wird dann beim INPUT angegeben.

```
10 INPUT "WIE HEISSEN SIE?";NAMES$
20 PRINT "GUTEN TAG ";NAMES$; ", WIE GEHT ES IHNEN?"
```

Diese Beispiele demonstrieren alles, was der Befehl INPUT zu bieten hat. Beachten Sie, daß das Programm auch weiterläuft, wenn nichts eingegeben, also nur RETURN gedrückt wird. Die Variablen nehmen dann den Wert 0 oder bei Strings ""(Leerstring) an.

SCHLEIFEN

Lassen Sie uns zunächst eine Übung machen, die dann später zur Schleifensteuerung führt. Es sollen fünf Namen über die Tastatur eingelesen werden. Diese Namen sollen in dem Array NAMES\$ gespeichert werden. Mit den bisher beschriebenen Befehlen würden wir folgendes Programm aufbauen:

```
10 PRINT"EINGABE DER NAMEN"
20 PRINT"-----"
30 INPUT"NAME ";NAMES$(1)
40 INPUT"NAME ";NAMES$(2)
50 INPUT"NAME ";NAMES$(3)
60 INPUT"NAME ";NAMES$(4)
70 INPUT"NAME ";NAMES$(5)
```

Mit diesem Programm werden jetzt nacheinander die fünf Namen eingelesen. Doch ist dies nicht sehr umständlich? Stellen Sie sich vor, es sollen 100 Namen eingelesen werden!

In der Praxis werden Sie nie einen solchen Programmierstil sehen. Man bedient sich dazu einer Schleife. Doch wie wird nun eine Schleife aufgebaut? Schauen Sie sich zunächst die Beschreibung des entsprechenden Befehls an:

Problem: Schleifensteuerung

Befehl: FOR $v1 = n1$ TO $n2$ STEP $n3$

Parameter:

$v1$ -	numerische Variable
$n1$ -	Anfangswert von $v1$
$n2$ -	Endwert von $v1$
$n3$ -	Inkrement, Wert um den $v1$ nach jedem Schleifendurchlauf erhöht wird

Beispiel: FOR A=1 TO 20 STEP 1
Die Variable A wird nach jedem Durchlauf um 1 erhöht, bis sie den Wert 20 erhält (20 Schleifendurchläufe)

Bemerkung: Ist das Inkrement 1, so kann der Parameter STEP weggelassen werden. Der Befehl aus dem Beispiel könnte demnach auch so eingegeben werden:
FOR A=1 TO 20

Jede Schleife beginnt mit einer FOR-Anweisung. Alle anschließenden Befehle gehören dann zu dieser Schleife und werden demnach mehrmals ausgeführt. Doch wie wird das Ende der Schleife gekennzeichnet? Dafür gibt es eine weitere Anweisung, die die Aufgabe hat, die Schleifenvariable auf den Endwert zu prüfen. Ist der Endwert noch nicht erreicht, so wird die Schleifenvariable erhöht und die Schleife nochmals durchlaufen. Entspricht die Schleifenvariable jedoch dem Endwert, so wird hinter dem Befehl NEXT mit dem Programmablauf fortgefahren.

Problem:	Schleifenende festlegen
Befehl:	NEXT <i>v</i>1
Parameter:	<i>v</i>1 - Schleifenvariable
Beispiel:	NEXT A Kennzeichnet das Ende der im vorherigen Beispiel enthaltenen Schleife.
Bemerkung:	Wird die Variable <i>v</i>1 nicht angegeben, bezieht sich der Befehl NEXT auf die zuletzt einggerichtete Schleife.

Mit diesen beiden Befehlen können wir nun beliebige Schleifen programmieren. Wenden wir uns wieder der anfangs beschriebenen Dateneingabe mit INPUT zu. Hier sollten fünf Namen in ein Array mit dem Index 1 bis 5 eingelesen werden. Wenn wir nun eine Schleife einsetzen, so benötigen wir nur eine INPUT-Anweisung, in der der Index als Variable, nämlich der Schleifenvariablen eingesetzt wird. Die Schleife soll von 1 bis 5 laufen. Wie sieht nun der entsprechende FOR-Befehl mit allen Parametern aus?

FOR I=1 TO 5

Das gesamte, geänderte Programm ist folgendes:

```
10 PRINT "EINGABE DER NAMEN"  
20 PRINT "-----"  
30 FOR I=1 TO 5  
40 INPUT "NAME";NAMES(I)  
50 NEXT I
```

Eine sehr elegante Lösung zum Einlesen von 5 Strings, oder? Interessant ist der genaue Ablauf nach dem Starten dieses Programms. Zuerst werden die beiden PRINT-Befehle ausgeführt. Nun beginnt die Schleife mit der Variablen I, dem Anfangswert 1, dem Endwert 5 und dem Inkrement (Erhöhungswert) 1. Erinnern Sie sich, wird kein STEP-Parameter angegeben, so wird mit dem Inkrement 1 gearbeitet. Zu Anfang der Schleife erhält I

den Wert 1. In Zeile 40 wird also die Arrayvariable mit dem Index 1 eingelesen. Anschließend prüft der Befehl NEXT, ob das Schleifenende (Variable I größer 5) erreicht ist. Da dies nicht der Fall ist, wird I um das Inkrement 1 erhöht und das Programm hinter dem Befehl FOR, also in Zeile 40 fortgesetzt. Die Variable I enthält nun den Wert 2. Der folgende INPUT liest also nun den nächsten Namen in dem Array mit dem Index 2 ein. So geht es nun weiter, bis der Befehl NEXT die Schleife abschließt. Hat die Variable I den Wert 4, so wird sie nochmals um 1 erhöht und die Schleife mit dem Wert I=5 durchlaufen. Anschließend wird die Variable I vom Befehl NEXT auf den Endwert (5) überprüft. Dies ist der Fall und das Programm wird hinter dem NEXT fortgesetzt, also beendet.

Wir können das Programm nun dahingehend erweitern, daß nach Eingabe der Namen alle nochmals ausgegeben werden. Dazu muß eine weitere Schleife angehängt werden:

```
60 FOR I=1 TO 5
70 PRINT "NAME ";NAMES(I)
80 NEXT I
```

Es ist also alles nicht so kompliziert, wie Sie es vielleicht zu Anfang angenommen haben.

Was kann man noch mit einer Schleife anfangen? Nun, z.B. können Sie eine Warteschleife aufbauen, die den Programmablauf verzögert. Hier bauen Sie nur eine Schleife mit FOR auf und schließen Sie unmittelbar dahinter mit NEXT ab. Nun ist es wichtig, wie oft so eine Schleife durchlaufen werden muß, um z.B. eine Warteschleife von 1 Sekunde zu erzeugen. Es kann davon ausgegangen werden, daß eine derartige Schleife 1000 mal in der Sekunde durchlaufen wird. Eine Warteschleife von ca. 3 Sekunden wird dann wie folgt aufgebaut:

```
FOR X=1 TO 3000:NEXT
```

Erinnern Sie sich: In einer Zeile können mehrere Befehle eingesetzt werden, wenn diese durch einen Doppelpunkt getrennt werden. Lassen Sie uns nun in unserem Programm eine Warteschleife zwischen der Dateneingabe und der Datenausgabe einfügen. Eine Zeilennummer, die dazwischen liegt, ist z.B. die Zeile 55. Wie sieht diese Zeile nun aus, wenn wir ca. 2 Sekunden verzögern möchten?

```
55 FOR I=1 TO 2000:NEXT I
```

Lassen Sie jetzt das Programm mit LIST auf dem Bildschirm anzeigen. Sie werden sehen, daß alle nachträglich eingegebenen Zeilen ordnungsgemäß eingesetzt wurden. Starten Sie das Programm nun mit RUN, dann werden Sie die Auswirkung der Zeile 55 bemerken.

Bisher haben wir nur mit dem Inkrement 1 gearbeitet, daß ja bekanntlich auch jeden anderen Wert annehmen kann. Sollen die Namen im vorherigen Beispiel nicht von 1 bis 5, sondern umgekehrt von 5 bis 1 eingelesen werden, so ist dies durchaus möglich. Der Anfangswert der Schleife ist dann 5, der Endwert 1 und das Inkrement, also der STEP-Parameter -1. Wie müssen die Zeilen 30 und 60 abgeändert werden? Sicher haben Sie die Lösung längst parat:

```
30 FOR I=5 TO 1 STEP -1  
40 FOR I=5 TO 1 STEP -1
```

Lassen Sie uns noch ein paar Übungen zum Befehl FOR machen. Es sollen die FOR-Befehle zu folgenden Schleifen aufgebaut werden:

	Schleifen- variable	Anfangs- wert	End- wert	Inkrement
A)	INDEX	10	18	1
B)	ZAHL	30	15.5	-0.5
C)	SCHLEIFE	590	1800	0.25
D)	ZEIT	1	10	0.3

Die Lösungen:

- A) FOR INDEX=10 TO 18 STEP 1
oder
FOR INDEX=10 TO 18
- B) FOR ZAHL=30 TO 15.5 STEP -0.5
- C) FOR SCHLEIFE=590 TO 1800 STEP 0.25
- D) FOR ZEIT=1 TO 10 STEP 0.3

Noch eine Anmerkung zum Abschluß des Kapitels. Sollten Sie mit FOR Schleifen aufbauen, die logisch falsch sind (z.B. FOR I=10 TO 0), so wird die Schleife grundsätzlich einmal durchlaufen!

ERSTE REAKTIONEN DES PROGRAMMS

Nachdem wir so weit in BASIC fortgeschritten sind, wird es Zeit, mit dem Adressenprogramm zu beginnen. Wie sollte ein solches Programm anfangen? Wenn Sie schon einmal vergleichbare Programme bedient haben, wissen Sie sicher, was ein solches Programm nach dem Starten auf dem Bildschirm zeigt. Es meldet sich meist mit einem Programmkopf. Diesen Kopf, der nur aus wenigen PRINT-Anweisungen besteht, wollen wir zunächst erstellen. Anweisungen, die Ihnen gänzlich unbekannt sind, sollten Sie nicht daran hindern, trotzdem den Anfang des Programms einzugeben und zu starten. Diese Anweisungen werden danach beschrieben.

```

100 REM =====
110 REM   Programmkopf
120 REM =====
130 CLS
140 PRINT STRING$(40,"=")
150 LOCATE 12,2
160 PRINT "ADRESSENVERWALTUNG"
170 PRINT STRING$(40,"=")

```

Nachdem Sie diesen Teil des Programms eingegeben und gestartet haben, wollen Sie natürlich wissen, was die neuen Befehle bedeuten. Fangen wir bei den Zeilen 100-120 an. Hier wird dieser Abschnitt des Programms dokumentiert. Damit der Rechner den dokumentierenden Teil nicht als Befehl interpretiert, werden diese Zeilen mit REM gekennzeichnet. REM bedeutet REMARKS, also Anmerkungen. Solche REM-Zeilen können Sie an beliebigen Stellen im Programm einfügen. Die Kennzeichnung der verschiedenen Programmabschnitte trägt zur Übersichtlichkeit des Programms bei.

Der nächste neue Befehl ist eigentlich kein Befehl, sondern eine Funktion, genauer eine Stringfunktion. Wie Sie wissen, bezeichnet man eine Zeichenkette in der Fachsprache als String. Mit der Funktion STRING\$ kann ein bis max. 255 vervielfachtes Zeichen ausgegeben werden. Schauen wir uns wieder zunächst die Beschreibung an:

Problem: Erzeugen von Zeichenketten

Funktion: STRING\$(n,"z")

Parameter: n - Anzahl der Zeichen (0-255)
"z" - zu vervielfachendes Zeichen

Beispiel: PRINT STRING\$(80,":")
Es werden 80 Doppelpunkte ausgegeben

Diese Funktion können Sie in Verbindung mit dem PRINT-Befehl ausprobieren. Da dieser Befehl recht einfach ist, kommen wir gleich zum nächsten.

Der Befehl PRINT gibt Daten auf dem Bildschirm aus. An welcher Stelle des Bildschirms ausgegeben wird, hängt von dem letzten PRINT- bzw. CLS-Befehl ab. Um die Ausgabe mit PRINT an eine beliebige Stelle des Bildschirms einzuleiten, enthält Ihr BASIC den folgenden Befehl:

Problem: Positionieren des Cursors

Befehl: LOCATE *sp,z*

Parameter: *sp* - Spalte
z - Zeile

Beispiel: LOCATE 5,10
setzt den Cursor auf die 5. Spalte der 10. Zeile

Nach jedem PRINT-Befehl wird die Position des Cursors verändert. Diesen Cursor sieht man jedoch nicht auf dem Bildschirm, da die Ausgabe von Programm gesteuert wird. Möchte man also die Position, an der die Ausgabe des PRINT-Befehls erfolgen soll, selbst bestimmen, so setzt man vor den PRINT-Befehl den Befehl LOCATE. Mit diesem Befehl gibt man, wie oben beschrieben, die Spalte und die Zeile der Position an. Ein Beispiel soll hier Klarheit schaffen. Der Text 'HALLO' soll in der Mitte des Bildschirms ausgegeben werden. Dazu werden die Befehle LOCATE und anschließend PRINT benötigt. Die Befehlsfolge sieht dann so aus:

```
LOCATE 18,13:PRINT "HALLO"
```

Erinnern Sie sich: In einer Befehlszeile können mehrere Befehle untergebracht werden, wenn diese durch einen Doppelpunkt getrennt werden.

Doch nun wieder zu unserem Programm.

UNTERPROGRAMME

Im weiteren Verlauf der Adressenverwaltung werden wir die BASIC-Zeilen zur Ausgabe des Programmkopfes noch öfter benötigen. Damit wir nicht immer wieder diese Zeilen eingeben müssen, was das Programm auch unnötig verlängern würde, machen wir daraus ein Unterprogramm (auch Routine genannt). Doch wie wird ein Unterprogramm gekennzeichnet und wie führt man es aus? Zwei Fragen, die wie immer sofort geklärt werden.

Sie haben bereits im vorherigen Kapitel den Befehl GOTO kennengelernt, der es ermöglicht, das Programm in irgendeiner Zeile fortzusetzen. Da nach Ablauf eines Unterprogrammes jedoch wieder zurück verzweigt werden muß, und der Rechner nicht wissen kann, von wo mit GOTO aufgerufen wurde, ist der Befehl GOTO hier fehl am Platze. Zum Aufruf von Routinen gibt es einen ähnlichen Befehl, der nun zunächst beschrieben wird.

Problem: Aufruf von Unterprogrammen (Routinen)

Befehl: GOSUB *zn*

Parameter: *zn* - Zeilennummer, ab der die Routine beginnt

Beispiel: GOSUB 100
ruft ein Unterprogramm bei Zeile 100 auf

Bemerkung:

- Die mit GOSUB aufgerufenen Unterprogramme müssen mit dem Befehl RETURN abgeschlossen werden.
- Stößt das Programm während des Ablaufs auf ein RETURN, ohne daß ein GOSUB-Befehl ausgeführt wurde, so führt dies zur Fehlermeldung "Unexpected RETURN"

Dies sieht eigentlich recht einfach aus. Ist es auch, wenn man nur einige Male damit gearbeitet hat. Wir machen nun die Zeilen 100-170 zu einem Unterprogramm, das von beliebiger Stelle im

späteren Programm aufgerufen werden kann. Wie Sie der Befehlsbeschreibung entnommen haben, müssen wir dazu dieses Unterprogramm mit dem Befehl RETURN abschließen. Dazu hängen wir eine Zeile an:

```
180 RETURN
```

Nun dürfen wir dieses Programm jedoch nicht mehr starten. Machen Sie es doch, so erscheint die Fehlermeldung "Unexpected RETURN in 180". Dies ist auch klar, denn wir steigen mit RUN einfach in dieses Unterprogramm ein, das nur mit GOSUB aufgerufen werden darf. Wie machen wir nun weiter?

Beachten Sie zunächst, wie die Zeilennummern des zukünftigen Programms organisiert werden.

- 10-99** **Hier sollen alle Vorabinformationen für das Programm untergebracht werden. Z.B. Einrichten von Tabellen, Zuweisung von Variablen.**
- 100-999** **In diesen Zeilen werden alle Routinen untergebracht.**
- 1000-** **Ab der Zeile 1000 beginnt das eigentliche Programm, das Hauptprogramm.**

Wir müssen also das Unterprogramm überspringen, um ins Hauptprogramm zu gelangen. Dazu geben wir die Zeile 99 ein:

```
99 GOTO 1000
```

In Zeile 1000 beginnt nun das Hauptprogramm. Dies kennzeichnen wir zunächst mit REM-Zeilen und setzen anschließend den Befehl GOSUB 100 ein:

```
1000 REM =====
1010 REM     Hauptprogramm
1020 REM =====
1030 GOSUB 100
```


Nun ist das Programm wieder lauffähig. Es wird zuerst das Unterprogramm zum Anzeigen des Programmkopfes ausgeführt.

DAS MENÜ

Schauen wir zunächst, welche Fortschritte unser Programm gemacht hat:

```
99 GOTO 1000
100 REM =====
110 REM      Programmkopf
120 REM =====
130 CLS
140 PRINT STRING$(40,"=")
150 LOCATE 12,2
160 PRINT"ADRESSENVERWALTUNG"
170 PRINT STRING$(40,"=")
180 RETURN
1000 REM =====
1010 REM      Hauptprogramm
1020 REM =====
1030 GOSUB 100
```

Das Programm meldet sich nun nach dem Starten mit seinem Namen. Doch dies ist sicherlich nicht alles, sondern lediglich ein kleiner Anfang. Da wir mit diesem Programm mehrere Möglichkeiten haben möchten, die Adressen zu bearbeiten, müssen wir diese auch anzeigen. Überlegen wir, was das Programm alles können soll. Zunächst soll es die Daten auf ein externes Speichermedium speichern und von dort wieder laden können. Wir benötigen dazu die beiden folgenden Auswahlmöglichkeiten:

- 1- DATEI LADEN
- 2- DATEI SPEICHERN

Weiterhin müssen wir natürlich Adressen eingeben. Dazu benötigen wir die Funktion 3:

-3- ADRESSEN EINGEBEN

Die eingegebenen Daten müssen auch zu ändern sein. Der folgende Punkt 4 soll dies ermöglichen:

-4- ADRESSEN AENDERN

Wenn wir vom Onkel Karl ab sofort nichts mehr wissen möchten, sollten wir ihn aus der Adreßdatei nehmen. Wir benötigen also eine Funktion "löschen":

-5- ADRESSEN LOESCHEN

Nun kommt das wichtigste: Alle Daten nützen uns wenig, wenn wir diese nicht ausgeben, also abrufen können. Mit folgendem Teilprogramm sollen Adressen ausgegeben werden:

-6- ADRESSEN AUSGEBEN

Kein Programm sollte mit dem Netzschalter des Rechners beendet werden. Wir schaffen deshalb mit dem letzten Punkt den Programmausgang:

-7- PROGRAMM BEENDEN

Nachdem wir uns über alle Programmfunktionen im Klaren sind, erweitern wir unser Programm so, daß diese Funktionen als Menü ausgegeben werden. Die folgenden Anweisungen werden dazu hinzugefügt:

```

1040 LOCATE 1,7
1050 PRINT"   PROGRAMMFUNKTIONEN:"
1060 PRINT"   -----"
1070 PRINT
1080 PRINT"   -1- Datei laden"
1090 PRINT"   -2- Datei speichern"
1100 PRINT"   -3- Adressen eingeben"

```

1110 PRINT" -4- Adressen aendern"
1120 PRINT" -5- Adressen loeschen"
1130 PRINT" -6- Adressen ausgeben"
1140 PRINT" -7- Programm beenden"

Wenn Sie das Programm nun noch einmal starten, werden Sie erkennen, daß es langsam einen professionellen Charakter bekommt. Nun muß der Anwender des Programms entscheiden, welche Programmfunktionen er benötigt. Wir lesen dazu eine numerische Variable ein, die den Wert 1 bis 7 entsprechend der ausgewählten Funktion enthalten soll. Vorher wird noch ein PRINT mit dem Inhalt "AUSWAHL" ausgegeben.

```
1150 LOCATE 10,18:PRINT"Auswahl ";  
1160 INPUT funktion
```

DIE ABFRAGE MIT IF

Was aber, wenn der Anwender eine ungültige Zahl eingibt, z.B. 9? Dies sollte vom Programm unterbunden werden. Die Frage ist nur wie? Wir müssen feststellen, welcher Wert eingegeben wurde, also die Variable F auf einen ungültigen Wert abfragen. Die Anweisung dazu wird nun beschrieben:

Problem: Abfrage auf Bedingungen

Befehl: IF bedingung THEN anweisung

Parameter: bedingung - Vergleichoperation
(z.B. A=10 oder B>12)
anweisung - Befehle, die ausgeführt
werden, wenn die Bedingung
wahr ist.

Beispiel: IF F=0 THEN GOTO 1000
Wenn die Variable F den Wert 0 hat, wird zur Zeile 1000 verzweigt, ansonsten wird in der Zeile dahinter das Programm fortgesetzt.

Bemerkung: Folgt dem THEN ein GOTO, so kann dieses GOTO ignoriert werden. Das letzte Beispiel könnte auch so eingegeben werden:
IF F=0 THEN 1000

Eine Bedingung kann nicht nur den Vergleichsoperator '=' beinhalten, sondern auch die Operatoren '>' (größer als) und '<' (kleiner als). Auch Kombinationen der drei sind erlaubt. Alles in allem können folgende Vergleichoperatoren eingesetzt werden:

Symbol	Bedeutung
=	gleich
>	größer als
<	kleiner als
>=, =>	größer als oder gleich
<=, =<	kleiner als oder gleich
<>, ><	ungleich

Der IF-Befehl prüft also erst, ob die Bedingung wahr ist. Ist dies der Fall, so werden die Anweisungen hinter dem THEN ausgeführt. Trifft die Bedingung nicht zu, so wird das Programm hinter der IF-Zeile fortgesetzt.

Doch dies ist nicht alles, was eine IF-Abfrage zu leisten vermag. Es können auch mehrere Bedingungen logisch miteinander verknüpft werden. Dies hört sich komplizierter an, als es eigentlich ist. Nehmen wir hier ein Beispiel zu Hilfe:

```
IF ANTW=1 OR ANTW>12 THEN 1000
```

Zwei Vergleiche, die miteinander verknüpft wurden. OR heist ODER und somit kann der Befehl wie folgt interpretiert werden: Wenn ANTW gleich 1 oder ANTW größer als 12, dann verzweige nach 1000. Zum Sprung nach Zeile 1000 muß also entweder ANTW=1 oder ANTW>12 oder beides zutreffen. Nur wenn beide Bedingungen verneint werden, wird das Programm hinter der IF-Zeile fortgesetzt. Die OR-Verknüpfung ist eine der möglichen Verknüpfungen. Ein weiteres Beispiel:

```
IF ANTW=1 AND TEST>12 THEN 1000
```

Dies ist eine UND-Verknüpfung, die nur dann erfüllt ist, wenn ANTW gleich 1 und TEST größer als 12 ist.

Kommen wir zu unserem Problem zurück. Wir wollten abfragen, ob der Anwender eine Zahl eingegeben hat, die nicht in dem Bereich von 1-7 liegt. Wie muß die entsprechende IF-Anweisung codiert werden? Schauen Sie sich die Lösung an:

```
IF funktion=0 OR funktion>7 THEN ...
```

Ja, was dann? Geben wir einfach eine Fehlermeldung in der letzten Bildschirmzeile aus und gehen zurück zur Zeile 1000. Zur Ausgabe der Fehlermeldung benötigen wir ein weiteres Unterprogramm, das die Art des Fehlers einem String entnimmt und diesen in Zeile 24 ausgibt. Sehen wir uns dieses Programm zunächst einmal an:

```
200 REM =====  
210 REM   FEHLERMELDUNG  
220 REM =====  
230 LOCATE 1,24  
240 PRINT fehler$  
250 PRINT CHR$(7);  
260 FOR I=1 to 1000:NEXT I  
270 LOCATE 1,24  
280 PRINT STRING$(39," ")  
290 RETURN
```

Zunächst wird in Zeile 230 der Cursor in die letzte Zeile positioniert. Die Zeile 240 gibt den String 'fehler\$', der vom aufrufenden Programm gesetzt wurde, aus. Bevor also diese Routine vom Hauptprogramm aufgerufen wird, muß in der Stringvariablen 'fehler\$' die auszugebende Fehlermeldung gespeichert werden.

ASCII-CODE

Nun zu der Funktion in Zeile 250, die wir bereits bei den Funktionstasten eingesetzt haben. Erinnern Sie sich, wir haben den Befehlen ein RETURN -CHR\$(13)- angehängt, damit sie automatisch ausgeführt werden. Doch gehen wir nun näher auf diese Funktion ein. Jedes Zeichen wird im Rechner codiert. Es erhält einen Wert zwischen 0 und 255. Diese Codierung ist genormt nach der amerikanischen ASCII-Norm. Die CPC6128-Zeichen weichen geringfügig von diesem Code ab. Doch nicht nur Zeichen werden mitcodiert. Auch spezielle Funktion, wie in unserem Programm das Ausgeben eines kurzen Tones, können mit einem ASCII-Code ausgegeben werden. Zum Codieren der Zeichen nach dem ASCII-Code gibt es zwei Funktionen, die zunächst beschrieben werden.

Problem: Umwandlung eines ASCII-Codes in ein Zeichen

Funktion: CHR\$(n)

Parameter: n - ASCII-Code (0-255)

Beispiel: FOR I=32 TO 255: PRINT CHR\$(I):NEXT I
zeigt alle mit dem CPC6128 darstellbare Zeichen

Bemerkung: Wird ein ASCII-Code über 255 angegeben, so folgt die Fehlermeldung "Improper argument"

Problem: Umwandlung eines Zeichens in den ASCII-Code

Funktion: ASC("z")

Parameter: "z" - ein Zeichen, oder eine Stringvariable, die ein Zeichen enthält

Beispiel: **PRINT ASC("A")**
 zeigt den ASCII-Code des Buchstabens "A" (65)
 auf dem Bildschirm an.

Bemerkung: - Ist die Stringvariable leer, so folgt die
 Fehlermeldung "Improper argument"
 - Ist die Stringvariable länger als ein Zeichen,
 so wird der ASCII-Code des ersten Zeichens
 ermittelt.

Dies ist eine Menge an Informationen, die erst einmal verdaut werden muß. Wie gesagt, arbeitet der Rechner intern nur mit den ASCII-Codes der Zeichen. Demnach versteht er auch, wenn Sie diesen ASCII-Code und nicht das Zeichen selbst eingeben.

Wenn Sie wissen möchten, welchen ASCII-Code ein bestimmtes Zeichen hat, so ermitteln Sie dies mit der Funktion ASC("z"). Sie geben dazu im Direktmodus nur den Befehl PRINT ASC("z") - für z setzen Sie das gewünschte Zeichen - ein. Der CPC6128 antwortet dann prompt mit dem entsprechenden Code.

In Zeile 250 wird also mit dem Befehl 'PRINT CHR\$(7)' ein Ton erzeugt, der akustisch auf einen Fehler hinweisen soll. Übrigens ist auch dieser Code durch ASCII genormt und bei den meisten Rechnern zu finden.

Doch weiter in unserer Fehlerroutine. Dem akustischen Signal folgt eine Warteschleife, die eine Verzögerung von ca. 1 Sekunde zum Lesen der Meldung erzeugt. Anschließend wird die Fehlermeldung in Zeile 280 wieder gelöscht, indem die Zeile einfach mit 39 Leerzeichen überschrieben wird. In der Zeile 290 erfolgt schließlich der Rücksprung zum Hauptprogramm.

Wir können nun unsere IF-Abfrage hinter dem INPUT vervollständigen:

```
1170 IF funktion=0 OR funktion>7 THEN fehler$=  
"ungueltiger Wert":GOSUB 200:GOTO 1150
```

Geben Sie diese Zeile nun ein. Beachten Sie, daß die Aufteilung in zwei Textzeilen nicht der Aufteilung auf Ihrem Bildschirm entspricht. Wenn die eingebene Zahl nun 0 oder größer als 7 ist, liegt eine Fehlbedienung des Anwenders vor, auf den wir ihn in der Routine 200 hinweisen. Dazu wird zunächst die Fehlermeldung im String untergebracht, dann die Fehleroutine aufgerufen und anschließend zum erneuten INPUT verzweigt.

Starten Sie nun das Programm und geben Sie einen falschen Wert ein. Das Programm stellt die Falscheingabe fest und gibt die Fehlermeldung aus.

BERECHNETES GOTO

Wenn ein gültiger Wert eingegeben wurde, müssen wir nun die nötigen Schritte einleiten. Das heißt, es muß zum entsprechenden Programmteil verzweigt werden.

Um abhängig von der Variablen 'funktion' im Programm zu verzweigen, gibt es einen Befehl, der dies auf leichteste Art und Weise ermöglicht. Schauen Sie sich die Beschreibung dieses Befehls an:

Problem: Berechneter Sprung

Befehl: ON *v1* GOTO *n1,n2,n3,...*

Parameter: *v1* - numerische Variable
n1,n2,n3,... - Zeilennummern, zu denen abhängig vom Inhalt der Variablen F gesprungen werden soll

Beispiel: **ON F GOTO 100,200,300,400**
Ist F=1, dann wird zur Zeile 100 verzweigt, ist
F=2, dann wird zur Zeile 200 verzweigt, usw.
Ist F größer als 4, so wird nicht verzweigt,
sondern in der nächsten Zeile weitergemacht.

Bemerkung: **Ist F keine ganze Zahl (z.B. 3.45), so wird**
immer die Zahl vor dem Komma benutzt.

Ein idealer Befehl also für unseren Zweck. Wir bestimmen erst, bei welchen Zeilennummern die Programmteile begonnen werden sollen. Unser nächster Befehl soll wie folgt eingegeben werden:

```
1180 ON funktion GOTO 5000,10000,15000,  
20000,25000,30000,35000
```

In welchen Zeilen nun die Programmteile untergebracht werden, ist im Folgenden ersichtlich:

'funktion'	Zeile	Programmteil
1	5000	DATEI LADEN
2	10000	DATEI SPEICHERN
3	15000	ADRESSEN EINGEBEN
4	20000	ADRESSEN AENDERN
5	25000	ADRESSEN LOESCHEN
6	30000	ADRESSEN AUSGEBEN
7	35000	PROGRAMM BEENDEN

Wir werden nun einen Programmteil nach dem anderen fertigstellen. Sinngemäß fangen wir mit der Funktion 3, ADRESSEN EINGEBEN an. Doch zunächst müssen wir noch die Tabelle zur Aufnahme der Adressen dimensionieren. Fügen Sie dazu die Zeile 10 ein:

```
10 DIM adressen$(200,7):REM Adressentabelle
```

Wir haben Platz für 200 Adressen reserviert. Das dürfte in der Regel vollkommen ausreichen. Wer hat schon mehr als 200 Bekannte? Für den kommerziellen Einsatz ist das Programm nur

beschränkt einsatzfähig. Eine Firma, die ihre 2000 Kunden verwalten möchte, sollte sich nach einem anderen geeigneten Programm umsehen.

Lassen Sie uns vor den einzelnen Programmteilen eine Routine schreiben, die für jedes Programm einen schönen Kopf ausgibt. Diese Routine soll vorher auch den Programmkopf ausgeben. Jeder Programmteil hat einen Namen, der auch im Menü angegeben wurde. Diese Namen legen wir in einer Tabelle ab, deren Index der Variablen 'funktion' entspricht. Was heißt das? In der Übersicht auf der letzten Seite sehen Sie, daß z.B. der Programmteil 'ADRESSEN EINGEBEN' die Nummer 3 hat. Diese Nummer ist dann der Index für den Tabellenplatz. Der Tabelle geben wir den Namen 'funktion\$'. Somit enthält z.B. die Variable 'funktion\$(3)' z.B. den Text 'ADRESSEN EINGEBEN'. Das Einrichten dieser Tabelle zählt zu den Vorbereitungen und kommt somit in den Zeilenbereich 0-99. Die folgenden Zeilen erstellen diese Tabelle:

```
20 funktion$(1)=" Datei laden  "
21 funktion$(2)=" Datei speichern "
22 funktion$(3)="Adressen eingeben"
23 funktion$(4)="Adressen aendern "
24 funktion$(5)="Adressen loeschen"
25 funktion$(6)="Adressen ausgeben"
26 funktion$(7)="Programm beenden "
```

Beachten Sie dabei, daß alle Strings die gleiche Länge erhalten (zweites Anführungszeichen untereinander). In dem Unterprogramm ab Zeile 300 wollen wir nun den Programmkopf und anschließend die vom Anwender ausgewählte Funktion anzeigen:

```
300 REM =====
310 REM Koepfe Programmteile
320 REM =====
330 GOSUB 100
340 LOCATE 10,5:PRINT STRING$(19,"**")
```

```
350 LOCATE 10,6:PRINT """+funktion$(funktion)+"""
360 LOCATE 10,7:PRINT STRING$(19,"""")
370 RETURN
```

Hier sehen Sie, daß auch ein Unterprogramm ein anderes Unterprogramm aufrufen kann. Es wird zunächst der allgemeine Programmkopf ausgegeben. Danach wird die erste Zeile des Kästchens, das den Namen der Funktion umhüllen soll, erzeugt. Die nächste Zeile sieht komplizierter aus, als sie eigentlich ist. Zunächst wird der Cursor auf die 10. Spalte der 6. Zeile positioniert. Dort wird zunächst ein Sternchen, dann der Name der Funktion ausgegeben. Dieser Name wird der Tabelle 'funktion\$' mit dem Index 'funktion' entnommen. An diesen String wird dann ein weiteres Sternchen angehängt. Die Zeile 360 erzeugt die untere Zeile des Rahmens. Die letzte Zeile muß nicht mehr beschrieben werden.

Doch wann wird diese Routine aufgerufen. Logischerweise kann sie erst ausgeführt werden, wenn der Anwender bereits eine Funktion ausgewählt hat, deren Nummer in der Variablen 'funktion' enthalten ist. Also hinter der IF-Abfrage. Fügen wir nun die Zeile 1175 ein:

```
1175 GOSUB 300
```

Wir müssen diese Zeile nehmen, da nach der Zeile 1180 in einen anderen Teil des Programms verzweigt wird.

ADRESSEN EINGEBEN

Nun wird es ernst. Der erste anspruchsvolle Programmteil muß erstellt werden. Doch fangen wir erst einmal mit dem einfachsten an:

```
15000 REM =====
15010 REM ADRESSEN EINGEBEN
15020 REM =====
```

So weit, so gut. Jetzt wird es kompliziert (nicht den Mut verlieren). Wir müssen uns irgendwie merken, wieviel Datensätze eingegeben wurden. Dazu erhöhen wir in diesem Programteil stets einen Zeiger um eins, der die Anzahl der eingegebenen Adressen erhält. Nennen wir ihn sinngemäß 'zeiger'. Die nächste Zahl soll diesen Zeiger um eins erhöhen:

```
15030 zeiger=zeiger+1
```

Wenn z.B. 'zeiger' ungleich 0 ist, so wurden vorher bereits Daten geladen oder eingegeben, die dann hier erweitert werden. Somit erweitern wir die Adressendatei automatisch.

Nun müßten wir 7 INPUT-Zeilen eingeben, um die 7 Datenfelder einer Adresse einzulesen. Doch dies geht einfacher. Zunächst speichern wir ab Zeile 30 alle Feldnamen in dem Array (Tabelle) 'feld\$(1)' bis 'feld\$(7)'.

```
30 feld$(1)="ANREDE      "
31 feld$(2)="VORNAME    "
32 feld$(3)="NAME       "
33 feld$(4)="STRASSE    "
34 feld$(5)="PLZ/ORT    "
35 feld$(6)="TELEFON    "
36 feld$(7)="BEMERKUNG  "
```

Diese Feldnamen werden zu Anfang des Programms im Array gespeichert und können im gesamten Programmablauf eingesetzt werden. Wozu wir sie jetzt benötigen, werden Sie im folgenden Verlauf des Programteils feststellen.

```
15040 PRINT
15050 FOR index=1 to 7
15060 PRINT feld$(index);
15070 INPUT adressen$(zeiger,index)
15080 NEXT index
```

Starten Sie das Programm, nachdem Sie diese Zeilen eingegeben haben, und wählen die Funktion 3 aus. Was nun geschieht, bewirkt allein diese ausgetüftelte Schleife. Ein gesamter Adressensatz wird mit 4 Befehlen eingelesen. Innerhalb der Schleife wird zunächst der Name des Feldes ausgegeben. Die Variable 'index' wird als Index auf das Array der Feldnamen benutzt. Gleich hinter dem Feldnamen (Semikolon!) wird dann der Feldinhalt eingelesen. Hier wird zweimal indiziert, da die Adressen bekanntlich in einem zweidimensionalen Array gespeichert werden. Der erste Index ergibt sich aus der Variablen 'zeiger', die nach jeder Eingabe einer Adresse um 1 erhöht wird. Als zweiter Index wird wiederum die Variable 'index' eingesetzt. Es werden also alle 7 Felder nacheinander über die Tastatur eingegeben.

Nachdem wir die Felder eingelesen haben, müssen wir dem Anwender die Möglichkeit geben, die Eingabe zu wiederholen. Er kann sich irgendwo verschrieben haben und muß dann korrigieren.

```
15090 LOCATE 1,18
15110 PRINT "Daten richtig eingegeben (j/n)";
15120 INPUT antwort$
15130 IF antwort$="j"THEN 15150
15140 IF antwort$="n" THEN zeiger=zeiger-1:
GOSUB 300:GOTO 15000
15145 fehler$="'j' oder 'n' druecken!":GOSUB 300:
GOTO 15090
```

Wenn die Daten richtig eingegeben wurden, so wird das Programm fortgesetzt. Sind die Daten nicht richtig eingegeben worden, so wird der Programmteil nochmals gestartet. Vorher wird jedoch der Satzähler um eins vermindert, da sonst in Zeile 15030 ein Datensatz weiter geschaltet wird. Wir wollen den letzten Satz jedoch nochmal eingeben. Wird keine richtige Antwort gegeben, so wird eine Fehlermeldung ausgegeben und anschließend neu eingelesen.

Das Programm wird nun mit einer weiteren Frage fortgesetzt:

```

15150 LOCATE 1,20
15160 PRINT "Weitere Eingaben (j/n)";
15170 INPUT antwort$
15180 IF antwort$="j" THEN GOSUB 300:GOTO 15000
15190 IF antwort$="n" THEN GOTO 1000
15200 fehler$=" 'j' oder 'n' druecken!":GOSUB 200:GOTO 15150

```

Hier muß sich der Anwender entscheiden, ob er weitere Adressen eingeben möchte oder nicht. Wenn ja, dann wird der Programmteil erneut gestartet. Wenn nein, so wird zurück ins Menü verzweigt. Eine falsche Eingabe führt wieder zur Fehlermeldung und erneuten Abfrage.

Damit haben wir den ersten Programmteil abgeschlossen. Man kann nun beliebig viele Daten eingeben, aber noch nicht viel damit anfangen. Zum Schluß folgt nochmals der gesamte Programmteil "ADRESSEN EINGEBEN!", den Sie mit Ihrem nun vergleichen können:

```

15000 REM =====
15010 REM Adressen eingeben
15020 REM =====
15030 zeiger=zeiger+1
15040 PRINT
15050 FOR index=1 TO 7
15060 PRINT feld$(index);
15070 INPUT adressen$(zeiger,index)
15080 NEXT index
15090 LOCATE 1,18
15100 PRINT "Daten richtig eingegeben (j/n)";
15110 INPUT antwort$
15120 IF antwort$="j" THEN 15150
15130 IF antwort$="n" THEN zeiger=zeiger-1:
GOSUB 300:GOTO 15000
15140 fehler$=" 'j' oder 'n' druecken!":
GOSUB 200:GOTO 15090
15150 LOCATE 1,20
15160 PRINT "Weitere Eingaben (j/n)";

```

```
15170 INPUT antwort$
15180 IF antwort$="j" THEN GOSUB 300:GOTO 15000
15190 IF antwort$="n" THEN 1000
15200 fehler$="'j' oder 'n' druecken!":
GOSUB 200:GOTO 15150
```

ADRESSEN ÄNDERN

Nun möchten wir dem Anwender unseres Programms die Möglichkeit geben, einzelne Felder eines bestimmten Datensatzes zu ändern. Wir benutzen dazu die Funktionstasten. Mit der Taste 'v' soll vorwärts, und mit der Taste 'r' rückwärts "geblättert" werden. Soll eine Adresse geändert werden, so wird die Taste 'a' gedrückt. Mit der Taste 'RETURN' wird zurück ins Menü verzweigt, die Änderung also abgeschlossen.

Das hört sich alles gut an, muß aber erst programmiert werden. Doch fangen wir zunächst mit den ersten Zeilen an:

```
20000 REM =====
20010 REM   Adressen aendern
20020 REM =====
```

Diese Zeilen bedürfen keines Kommentars. Nun aber wird es kritisch. Wie gehen wir dieses Problem an? Wir wählen uns zunächst eine Variable aus, die den ersten Index des Arrays (Tabelle) der Adressen darstellen soll. Dies ist die Variable 'zaehler', die den Anfangswert 1 erhält.

```
20030 zaehler=1
```

Nun können wir die erste Adresse in der Tabelle ausgeben. Sie besteht bekanntlich aus 7 Feldern, deren Namen in einem Array gespeichert sind. Wir codieren wieder eine Schleife, die die Nummer des Feldes, den Namen und den Inhalt ausgibt.

```

20040 LOCATE 1,10
20050 FOR index=1 TO 7
20060 PRINT index;feld$(index);adressen$(zaehler,index)
20070 NEXT index

```

Es werden also 7 Zeilen ausgegeben, die jeweils die Nummer des Feldes, dessen Bezeichnung und den Inhalt enthält. Der Inhalt des Feldes ergibt sich aus dem Index 'zaehler', der bekanntlich auf den Datensatz zeigt, und dem eigentlichen Feldindex ('index').

Nachdem die erste Adresse ausgegeben wurde, muß der Anwender weiter entscheiden. Er hat vier Möglichkeiten:

Taste 'v'	vorwärts blättern (nächste Adresse)
Taste 'r'	rückwärts blättern (letzte Adresse)
Taste 'a'	angezeigte Adresse ändern
Taste 'RETURN'	zurück ins Menü

Nun müssen wir solange die Tastatur abfragen, bis irgendeine Taste gedrückt ist. Der Befehl dazu ist INKEY\$.

Problem: Lesen eines Zeichens von Tastatur

Befehl: s=INKEY\$

Parameter: s - Stringvariable, in der die gedrückte Taste gespeichert werden soll

Beispiel: X\$=INKEY\$
 Testet, ob eine Taste gedrückt wurde und speichert diese Taste in X\$. Danach wird der Programmablauf sofort fortgesetzt.

Bemerkung: Ist keine Taste gedrückt, so wird die Stringvariable gelöscht.

Wollen Sie das Programm nur dann fortsetzen, wenn eine Taste gedrückt ist, so ist z.B. folgende Zeile notwendig (diese Zeile bitte nicht eingeben):

```
10 X$=INKEY$:IF X$="" THEN 10
```

Es wird also immer wieder die Zeile 10 ausgeführt, bis eine Taste gedrückt ist. So eine Anweisung folgt auch in unserem Programm.

```
20080 taste$=INKEY$:IF taste$="" THEN 20080
```

Wenn diese Zeile durchlaufen wurde, hat der Benutzer eine Taste gedrückt. Wir müssen nun abfragen, ob er eine der Tasten **v**, **r**, **a** oder **RETURN** gedrückt hat und entsprechend reagieren. Zunächst fragen wir die Taste **RETURN** ab, die bekanntlich den ASCII-Code 13 hat:

```
20090 IF taste$=chr$(13) THEN 1000
```

Wir verzweigen also zum Menü, wenn die Taste **RETURN** gedrückt wurde. Wird die Taste **'v'** gedrückt, so soll der nächste Datensatz ausgegeben werden. Wir erhöhen dazu den Index **'zaehler'** um 1, aber nur wenn **'zaehler'** nicht schon den letzten Satz erreicht hat (**zaehler=zeiger**). Die folgende Zeile erfüllt diese Aufgabe:

```
20100 IF taste$="v" AND zaehler<zeiger THEN  
zaehler=zaehler+1:GOSUB 300:GOTO 20040
```

Nur wenn die Taste **'v'** gedrückt wurde **UND** die letzte Adresse noch nicht erreicht ist, wird der Zaehler auf die nächste Adresse gesetzt und dies wieder ausgegeben.

Bei der Taste **'r'** ist es ähnlich, nur muß **'zaehler'** um 1 vermindert werden, wenn **'zaehler'** nicht schon 1 ist, weil dann die erste Adresse vorliegt.

```
20110 taste$="r" AND zaehler>1 THEN zaehler=zaehler-1:  
GOSUB 300:GOTO 20040
```

Bei Zeile 20040 wird die Adresse mit dem Index 'zaehler' ausgegeben. Dieser Zähler wurde hier um eins vermindert.

Nun müssen wir noch abfragen, ob die Taste 'a' gedrückt wurde. Mit dieser Taste wird die Änderung eingeleitet.

```
20120 IF taste$="a" THEN 20140
20130 fehler$="Eingabefehler!":GOSUB 200:GOTO 20080
```

Die Zeile 20130 wird erreicht, wenn die Tasten 'v' oder 'r' die Datei über- oder unterschreiten wollten oder wenn keine gültige Taste gedrückt wurde. Dann werden die Tasten einfach ignoriert, eine Fehlermeldung ausgegeben und zur Zeile 20080 verzweigt. In dieser Zeile wird erneut die Tastatur abgefragt.

Nun fehlt nur noch die eigentliche Änderung des Datensatzes ab Zeile 20140. Dort soll zuerst nach der Nummer und anschließend nach dem neuen Inhalt des zu ändernden Feldes gefragt werden.

```
20140 LOCATE 1,18
20150 INPUT "Feldnummer (1-7): ";nummer
20160 IF nummer>1 AND nummer<8 THEN 20180
20170 fehler$="Nummer 1-7 eingeben!":GOSUB 200:GOTO 20140
20180 INPUT "Neuer Inhalt: ";adressen$(zaehler,nummer)
20190 GOSUB 300:GOTO 20040
```

Dies war schon der Rest unseres Programmteils. Zunächst wird die Nummer des zu ändernden Feldes in 'nummer' eingelesen. Danach wird getestet, ob 'nummer' einen gültigen Wert enthält. Ist dies der Fall, so wird ab Zeile 20180 der neue Inhalt dieses Feldes eingelesen und der neu entstandene Adressensatz ab der Zeile 20040 ausgegeben. Bei Eingabe einer ungültigen Zahl wird in Zeile 20170 eine Fehlermeldung ausgegeben. Dieser Programmteil kann nur über die Taste 'RETURN' verlassen werden. Nun folgt auch hier die Auflistung des kompletten Teils "ADRESSEN AENDERN":

```
20000 REM =====
20010 REM   Adressen ändern
20020 REM =====
20030 zaehler=1
20040 LOCATE 1,10
20050 FOR index=1 TO 7
20060 PRINT index;feld$(index);adressen$(zaehler,index)
20070 NEXT index
20080 taste$=INKEY$:IF taste$="" THEN 20080
20090 IF taste$=CHR$(13) THEN 1000
20100 IF taste$="v" AND zaehler<zeiger THEN zaehler=zaehler+1:
GOSUB 300:GOTO 20040
20110 IF taste$="r" AND zaehler>1 THEN zaehler=zaehler-1:
GOSUB 300:GOTO 20040
20120 IF taste$="a" THEN 20140
20130 fehler$="Eingabefehler!":GOSUB 200:GOTO 20080
20140 LOCATE 1,18
20150 INPUT "Feldnummer (1-7)";nummer
20160 IF nummer>0 AND nummer<8 THEN 20180
20170 fehler$="Nummer 1-7 eingeben!":GOSUB 200:GOTO 20140
20180 INPUT "Neuer Inhalt: ";adressen$(zaehler,nummer)
20190 GOSUB 300:GOTO 20040
```

ADRESSEN LÖSCHEN

Beim Löschen der Adresse wenden wir eine andere Technik als beim Ändern an. Sie sollen ja möglichst vielseitig programmieren lernen. Wir müssen den Vor- und Zunamen der zu löschenden Adresse angeben. Ist er nicht bekannt, so kann man ihn mit dem "blättern" in "ADRESSEN AENDERN" ermitteln.

Doch bevor dieser Teil des Programms beginnt, soll geprüft werden, ob überhaupt Daten im Rechner gespeichert sind. Zur Ausgabe der Fehlermeldung bauen wir eine Routine ab Zeile 500 auf. Schauen wir uns zunächst die komplette Routine an:

```

500 REM =====
510 REM     Keine Daten!
520 REM =====
530 fehler$="Keine Daten im Rechner!"
540 GOSUB 200
550 RETURN

```

Diese Routine ist zwar kurz, trotzdem sinnvoll. Wir sollten die Überprüfung auch im Programmteil "ADRESSEN AENDERN" machen. Wenn keine Adressen vorhanden sind, kann man auch keine ändern! Fügen Sie dazu die folgende Zeile ein:

```
20030 IF zeiger=0 THEN GOSUB 500:GOTO 1000
```

Es wird geprüft, ob Daten im Rechner sind. Die Variable 'zeiger' enthält immer den letzten Datensatz und somit die Anzahl der im Rechner gespeicherten Adressen. Ist diese Variable 0, so sind keine Daten eingegeben und auch keine geladen worden. Wir geben dann die Fehlermeldung "Keine Daten im Rechner!" mit der Routine ab Zeile 500 aus und verzweigen zurück zum Menü.

Fangen wir nun mit unserem Programmteil an. Die ersten Zeilen sind erfahrungsgemäß immer die leichtesten.

```
25000 REM =====
25010 REM  ADRESSEN LOESCHEN
25020 REM =====
25030 IF zeiger=0 THEN GOSUB 500:GOTO 1000
25040 LOCATE 1,10
25050 INPUT "Vorname: ";vorname$
25060 INPUT "Name   : ";name$
```

Wir prüfen zunächst, ob Daten im Rechner vorhanden sind. Wenn ja, lesen wir Vorname und Name der zu löschenden Adresse ein. Diese beiden Eingaben speichern wir in 'vorname\$' und 'name\$'. Nun müssen wir die gesamte Adresstabelle nach diesen Namen durchsuchen. Eine FOR...NEXT-Schleife kann hier nicht eingesetzt werden, da diese mit Sicherheit vorzeitig verlassen wird, nämlich dann, wenn die Adresse gefunden wurde. FOR...NEXT-Schleifen sollte man nicht vorzeitig abbrechen. Der Grund dafür ist folgender: Die Adresse des Schleifenanfangs wird im Rechner abgelegt. Erst nach ordnungsgemäßem Ablauf der Schleife wird diese Adresse wieder gelöscht. Wird die Schleife nun mehrmals vorzeitig verlassen, so füllt sich dieser Speicherbereich. Dies kann dazu führen, daß das Programm aussteigt.

Wir bauen also eine selbst gesteuerte Schleife auf.

```
25070 schleife=1
25080 IF adressen$(schleife,2)=vorname$ AND
adressen$(schleife,3)=name$ THEN 25110
25090 IF schleife<zeiger THEN schleife=schleife+1:
GOTO 25080
25100 fehler$="Adresse nicht vorhanden!":GOSUB 200:GOTO 1000
```

Der Anfangswert der Schleifenvariablen 'schleife' wird zunächst auf 1 gesetzt. Innerhalb der Schleife wird geprüft, ob die mit 'schleife' indizierte Adresse mit der gesuchten übereinstimmt. Der zweite Index 2 und 3 entspricht dem Vornamen und dem Namen, der mit 'vorname\$' und 'name\$' verglichen wird. Stimmt der Name überein, so wird das Program in Zeile 25110 fortgesetzt. In der Zeile 25090 wird die Schleifenvariable um 1 erhöht, sofern sie noch nicht das Ende der Adresstabelle erreicht hat.

Wird die Schleife komplett durchlaufen, also nicht durch Auffinden des Namens abgebrochen, so ist der Name nicht gefunden worden. Hier geben wir eine Meldung aus und verzweigen zurück ins Menü.

Wir müssen das Programm nun an der Stelle fortsetzen, zu der beim Auffinden des gesuchten Namens verzweigt wird. Dies ist die Zeile 25110. Hier soll die komplette zu löschende Adresse nochmals angezeigt werden, damit der Anwender dann entscheiden kann, ob sie wirklich gelöscht werden soll.

```
25110 GOSUB 300:LOCATE 1,10
25120 FOR index=1 TO 7
25130 PRINT feld$(index);adressen$(schleife,index)
25140 NEXT index
25150 LOCATE 1,18
25160 INPUT "Adresse loeschen (j/n) ";antwort$
25170 IF antwort$="j" THEN 25200
25180 IF antwort$="n" THEN 1000
25190 fehler$="'j' oder 'n' druecken!":GOSUB 200:GOTO 25150
```

Hier wird zunächst ein neuer Bildschirm aufgebaut und der Cursor positioniert (Zeile 25110). Danach wird mit der Schleife in Zeile 25120 bis 25140 der Adressensatz ausgegeben. Nun fragen wir nach, ob diese Adresse wirklich gelöscht werden soll. Soll diese Adresse nicht gelöscht werden (unentschlossene Leute gibt es überall), so wird zurück ins Menü verzweigt. Soll die Adresse doch gelöscht werden, so geschieht dies ab Zeile 25200. Werfen wir zunächst einen Blick in die letzten Zeilen dieses Programmteils.

```
25200 FOR i1=schleife TO zeiger-1
25210 FOR i2=1 TO 7
25220 adressen$(i1,i2)=adressen$(i1+1,i2)
25230 NEXT i2
25240 NEXT i1
25250 zeiger=zeiger-1
25260 GOTO 1000
```

Zum Löschen einer Adresse reicht es nicht, den entsprechenden Tabellenplatz zu löschen. Würden wir dies tun, so bleibt die Größe der Tabelle trotz Löschen einer Adresse erhalten. Dies ist nicht im Sinne des Erfinders. Alle Adressen, die sich hinter der zu löschenden Adresse befinden, müssen eine Stelle nach vorne rücken. Wenn z.B. die Adresse mit dem Index 5 gelöscht wird und 7 Adressen im Rechner gespeichert sind, so rückt die 6. Adresse auf die 5., die somit gelöscht wird. Die letzte, also 7. Adresse rückt auf die 6. Wenn wir nun die Anzahl der Adressen auf 6, also um eins vermindern. So ist die Adresse mit dem Index 5 "echt" gelöscht worden.

Bei der Programmierung dieses Prinzips taucht zum ersten Mal eine verschachtelte Schleife auf. Innerhalb der Schleife mit der Variablen 'i1' befindet sich die Schleife 'i2'. Die Schleife 'i1' durchläuft alle Datensätze ab dem zu löschenden Datensatz. Die Felder werden alle einzeln nachgerückt. Sind alle Felder aufgerückt, so kann der nächste Datensatz bearbeitet werden. Die äußere Schleife läuft nur bis 'zeiger-1', da der letzte Datensatz in Zeile 25220 mit i1+1 indiziert wird. Wenn Sie die entsprechenden Zeilen studieren, so erkennen Sie die Technik dieses "Aufrückens".

Nun haben wir auch diesen Teil unseres zur Zeit recht umfangreichen Programms fertiggestellt. Es folgt nun wie immer die komplette Auflistung dieses Abschnittes:

```
25000 REM =====
25010 REM Adressen loeschen
25020 REM =====
25030 IF zeiger=0 THEN GOSUB 500:GOTO 1000
25040 LOCATE 1,10
25050 INPUT "Vorname: ";vorname$
25060 INPUT "Name : ";name$
25070 schleife=1
25080 IF adressen$(schleife,2)=vorname$ AND
adressen$(schleife,3)=name$ THEN 25110
25090 IF schleife<zeiger THEN schleife=schleife+1:GOTO 25080
25100 fehler$="Adresse nicht vorhanden!":GOSUB 200:GOTO 1000
25110 GOSUB 300:LOCATE 1,10
```

```
25120 FOR index=1 TO 7
25130 PRINT feld$(index);adressen$(schleife,index)
25140 NEXT index
25150 LOCATE 1,18
25160 INPUT "Adresse loeschen (j/n) ";antwort$
25170 IF antwort$="j" THEN 25200
25180 IF antwort$="n" THEN 1000
25190 fehler$="'j' oder 'n' druecken!":GOSUB 200:GOTO 25150
25200 FOR i1=schleife TO zeiger-1
25210 FOR i2=1 TO 7
25220 adressen$(i1,i2)=adressen$(i1+1,i2)
25230 NEXT i2
25240 NEXT i1
25250 zeiger=zeiger-1
25260 GOTO 1000
```

ADRESSEN AUSGEBEN

Dieser Programmabschnitt gehört sicherlich nicht zu den leichtesten. Hier wollen wir zum ersten Mal auf dem Drucker ausgeben. Doch auch Bildschirmausgabe soll ermöglicht werden. Nach Auswahl, ob denn nun auf dem Bildschirm oder dem Drucker ausgegeben werden soll, geben Sie die Suchbegriffe ein. Doch schreiben wir zunächst die ersten Zeilen.

```
30000 REM =====
30010 REM Adressen ausgeben
30020 REM =====
30030 if zeiger=0 THEN GOSUB 500:GOTO 1000
30040 LOCATE 1,10
30050 INPUT "Drucker oder Bildschirm (d/b) ";antwort$
30060 IF antwort$="d" THEN geraet=8:GOTO 30090
30070 IF antwort$="b" THEN geraet=0:GOTO 30090
30080 fehler$="'d' oder 'b' druecken!":GOSUB 200:GOTO 30040
```


Hier wird wieder geprüft, ob Daten im Rechner sind (Zeile 30030). Danach wird gefragt, ob auf Drucker oder Bildschirm ausgegeben werden soll. Die Antwort ('d' oder 'b') wird in der Stringvariablen 'antwort\$' festgehalten. Falls der Drucker ausgewählt wurde, wird die Variable 'geraet' auf 8 gesetzt. Dies wird für einen neuen PRINT-Befehl benötigt, den wir zunächst beschreiben:

Problem: Ausgabe auf verschiedene Geräte

Befehl: PRINT #g,.....

Parameter:

g -	Nummer des Geräts (0-9)
0 -	Bildschirm
1 bis 7 -	Bildschirmfenster (in diesem Buch nicht beschrieben)
8 -	Drucker
9 -	Diskette

Beispiel: PRINT #8,"ADRESSENLISTE"
Gibt den Text "ADRESSENLISTE" auf dem Drucker aus.

Bemerkung: Der Befehlszusatz '#g' kann auch bei den bereits bekannten Befehlen LIST und INPUT verwendet werden.

Wir können also mit PRINT auch auf dem Drucker oder auf das Diskettenlaufwerk Daten ausgeben. Wir haben in der Variablen 'geraet' diese entsprechende Nummer abgelegt. Wenn wir nun zur Ausgabe immer 'PRINT #geraet,....' angeben, so wird abhängig von der Variablen entweder auf dem Drucker oder auf dem Bildschirm ausgegeben.

Doch warum kann die Gerätenummer auch bei den Befehlen LIST und INPUT angegeben werden. Beim Befehl LIST ist die Ausgabe des Listings auf dem Drucker interessant. Zu beachten ist jedoch, daß der Zusatz '#8' HINTER dem Zeilenbereich angegeben wird. Zwei Beispiele:

LIST #8 gibt das gesamte Programm auf dem Drucker aus

LIST 1000-,#8 gibt die Zeilen ab einschließlich 1000 auf dem Drucker aus

Der Zusatz '#9' beim Befehl INPUT liest Daten von der Diskette, doch dazu kommen wir später. Nun wieder zu unserem Programm.

Wir haben also bisher die Gerätenummer zur Ausgabe bestimmt. Da wir nicht immer grundsätzlich alle Adressen ausgeben möchten, wollen wir für jedes Feld ein Suchkriterium bestimmen. Schauen Sie sich zunächst die folgenden Zeilen an:

```
30090 GOSUB 300:LOCATE 1,10
30100 PRINT "Suchbegriffe:"
30110 PRINT "-----"
30120 PRINT
30130 FOR index=1 TO 7
30140 such$(index)=" "
30150 PRINT feld$(index);
30160 INPUT such$(index)
30170 NEXT index
```

Die Zeile 30090 erstellt nun einen neuen Bildschirm, bevor die Zeilen 30100-30110 signalisieren, daß nun die Suchbegriffe eingegeben werden müssen. Die Eingabe erfolgt wieder in einer Schleife. Doch zuvor müssen die Suchbegriffe, die im Array 'such\$' gespeichert werden, gelöscht werden. Sie können noch Daten aus vorherigen Suchen enthalten. Die Begriffe werden dann in der Zeile 30160 eingelesen. Wenn z.B. alle Adressen aus Düsseldorf gesucht werden sollen, so geben Sie in den ersten 4 Feldern nur RETURN ein. Das 5. Feld, daß die Postleitzahl und den Ort enthält, wird mit "4000 Duesseldorf" gefüllt und anschließend mit RETURN abgeschlossen. Die restlichen 2 Felder werden wiederum mit RETURN ignoriert. Wollen Sie aber alle Herren aus Düsseldorf suchen, so geben Sie zusätzlich bei der Anrede "Herr" ein. Sie können also nach mehreren Begriffen gleichzeitig suchen.

Nach der Eingabe der Suchbegriffe soll die Suche beginnen.

```
30180 FOR i1=1 TO zeiger
30190 gefunden=0
30200 FOR i2=1 TO 7
30210 IF such$(i2)="" OR such$(i2)=adressen$(i1,i2)
THEN gefunden=gefunden+1
30220 NEXT i2
```

Wir benötigen wieder eine zweifach verschachtelte Schleife, die innerhalb der Datensätze alle Felder durchsucht. Nach dem Anfang der äußeren Schleife (Zeile 30180) wird in Zeile 30190 eine Variable 'gefunden' auf 0 gesetzt. In dieser Variablen sollen alle Felder gezählt werden, für die entweder kein Suchbegriff eingegeben wurde, oder für die ein Suchbegriff eingegeben wurde, der mit dem Inhalt des Feldes übereinstimmt. Wenn für ein Feld kein Suchbegriff eingegeben, also nur RETURN gedrückt wurde, so enthält die Variable 'such(i2)' nichts (such(i2)=""). Dieser Vergleich und das Erhöhen der Variablen 'gefunden' geschieht in der Zeile 30210. Wenn nach Ablauf der inneren Schleife (i2) der Erfolgszähler 'gefunden' 7 beträgt, entspricht die Adresse den Suchkriterien und kann ausgegeben werden. Die folgenden Zeilen leiten diese Ausgabe ein:

```
30230 IF gefunden<>7 THEN 30300
30240 IF geraet=0 THEN GOSUB 300
30250 PRINT #geraet
```

Hier wird die Ausgabe des gefundenen Datensatzes vorbereitet. Nur wenn der Datensatz nicht den Suchbegriffen entspricht ('gefunden' ungleich 7) wird die gesamte Ausgabe übersprungen. Wenn der Bildschirm ausgewählt wurde, wird ein neuer Kopf erzeugt. Anschließend wird auf das ausgewählte Gerät eine Leerzeile ausgegeben. Es folgen nun die restlichen Zeilen dieses Programmabschnittes, die anschließend wieder dokumentiert werden.

```
30260 FOR index=1 TO 7
30270 PRINT #geraet, feld$(index);adressen$(i1,index)
30280 NEXT index
30290 PRINT #geraet
30295 IF geraet=0 THEN INPUT "Druecken Sie RETURN ";antwort$
30300 NEXT i1
30310 GOSUB 300:LOCATE 1,18
30320 PRINT "*** DATEIENDE ***"
30330 INPUT "Druecken Sie RETURN ";antwort$
30340 GOTO 1000
```

In diesen Zeilen dreht sich fast alles um die Ausgabe des gefundenen Datensatzes. Die Zeilen 30260 bis 30280 bilden eine Schleife, in der die 7 Felder der Adresse mit ihrer Bezeichnung auf das ausgewählte Gerät ausgegeben werden.

Wenn Bildschirmausgabe ausgewählt wurde, wird erst nach Drücken von RETURN mit der Suche fortgefahren (Zeile 30295). Weitersuchen heißt hier, die Schleife 'i1' mit dem Befehl 'NEXT i1' um eins zu erhöhen. Ist die Druckerausgabe aktuell, so wird die Zeile zum Betätigen von RETURN übersprungen. Es werden also alle Adressen hintereinander ausgedruckt.

Hinter dem Ende der Schleife 'i1' ist die Suche beendet. Es wird durch die Meldung "*** DATEIENDE ***" signalisiert. Wenn der Anwender nun RETURN drückt, wird der Programmabschnitt beendet, also zurück zum Menü verzweigt.

Dies war ein weiterer Meilenstein zu unserer Adressenverwaltung. Nun müssen nur noch die Teile zum Laden und Speichern der Datei auf Diskette geschrieben werden.

Mit dem folgenden Listing können Sie nochmals Ihren Programmteil "ADRESSEN AUSGEBEN" kontrollieren:

```
30000 REM =====
30010 REM Adressen ausgeben
30020 REM =====
30030 IF zeiger=0 THEN GOSUB 500:GOTO 1000
30040 LOCATE 1,10
30050 INPUT "Drucker oder Bildschirm (d/b) ";antwort$
30060 IF antwort$="d" THEN geraet=8:GOTO 30090
30070 IF antwort$="b" THEN geraet=0:GOTO 30090
30080 fehler$="'d' oder 'b' druecken!":GOSUB 200:GOTO 30040
30090 GOSUB 300:LOCATE 1,10
30100 PRINT"Suchbegriffe:"
30110 PRINT"-----"
30120 PRINT
30130 FOR index=1 TO 7
30140 such$(index)=""
30150 PRINT feld$(index);
30160 INPUT such$(index)
30170 NEXT index
30180 FOR i1=1 TO zeiger
30190 gefunden=0
30200 FOR i2=1 TO 7
30210 IF such$(i2)="" OR such$(i2)=adressen$(i1,i2) THEN
gefunden=gefunden+1
30220 NEXT i2
30230 IF gefunden<>7 THEN 30300
30240 IF geraet=0 THEN GOSUB 300
30250 PRINT #geraet
30260 FOR index=1 TO 7
30270 PRINT #geraet, feld$(index);adressen$(i1,index)
30280 NEXT index
30290 PRINT #geraet
30295 IF geraet=0 THEN INPUT "Druecken Sie RETURN";antwort$
30300 NEXT i1
30310 GOSUB 300:LOCATE 1,18
30320 PRINT "*** DATEIENDE ***"
30330 INPUT "Druecken Sie RETURN ";antwort$
30340 GOTO 1000
```

DATEI SICHERN

Wir nähern uns dem Abschluß der Adressenverwaltung. Die eingegebenen Daten gehen mit dem Abschalten des Rechners natürlich verloren. Wir wollen unsere Adressen auf dem Diskettenlaufwerk speichern. Was muß nun zuerst beachtet werden? Beginnen wir diesen Teil wie üblich mit dem Bildschirmkopf und der Abfrage, ob Daten im Rechner gespeichert sind.

```
10000 REM =====
10020 REM   DATEI SPEICHERN
10030 REM =====
10040 IF zeiger=0 THEN GOSUB 500:GOTO 1000
```

Diese Zeilen sind uns nicht mehr unbekannt. Bevor wir die Daten auf der Diskette speichern können, muß diese erst eingelegt werden. Die folgenden Zeilen fordern dazu auf.

```
10050 LOCATE 1,12
10060 PRINT "Bitte Datendiskette einlegen! "
10070 PRINT
10080 INPUT "Druecken sie danach RETURN ";antwort$
```

Auch diese Zeilen dürften Ihnen keine Verständnisschwierigkeiten bereiten. Doch nun geht es los. Um die Daten mit dem Befehl 'PRINT #9,...' auf die Diskette speichern zu können, müssen wir das Diskettenlaufwerk vorbereiten. Man sagt dazu, "die Datei muß geöffnet werden". Dazu gibt es den folgenden Befehl:

Problem:	Öffnen einer Datei zum Schreiben
Befehl:	OPENOUT "name"
Parameter:	<i>name</i> - Name der Datei
Beispiel:	OPENOUT "Adressen" Öffnet die Datei "Adressen" zum Schreiben

Mit diesem Befehl öffnen wir also unsere Datei, bevor die Daten übermittelt werden.

Da wir für unsere Datei immer den Namen **Adressen** benutzen, müssen wir ein evtl. vorhandene Datei unter diesen Namen löschen. Das Löschen und das Öffnen der Datei übernehmen die folgenden Zeilen:

```
10090 a$="Adressen": |ERA,aa$
10100 OPENOUT "Adressen"
```

Das Zeichen **'** erreichen Sie mit **SHIFT @**.

Die Daten können jetzt gespeichert werden. Das erste, was wir auf die Diskette schreiben, ist die Anzahl der zu übermittelnden Datensätze, die bekanntlich in **'zeiger'** gespeichert ist. Danach senden wir die einzelnen Felder der Adressentabelle mit Hilfe einer verschachtelten Schleife, die Ihnen nicht mehr unbekannt ist. Zuerst Feld 1 bis 7 des ersten Datensatzes, dann Feld 1 bis 7 des zweiten, usw.:

```
10110 PRINT #9,zeiger
10120 FOR i1=1 TO zeiger
10130 FOR i2=1 TO 7
10140 PRINT #9,adressen$(i1,i2)
10150 NEXT i2
10160 NEXT i1
```

Nach der Speicherung der Daten muß die Datei wieder geschlossen werden. Der Befehl dazu heißt sinngemäß **'CLOSEOUT'**. Wir setzen ihn in die nächste Zeile:

10170 CLOSEOUT

Nach Speicherung der Daten soll dies noch einmal gemeldet werden. Die folgenden Zeilen sind dafür zuständig.

```
10180 GOSUB 300
10190 PRINT "Die Adressen sind gespeichert!"
10200 FOR i=1 TO 2000:NEXT i1
10210 GOTO 1000
```

Zunächst wird ein neuer Bildschirm erzeugt. Nach dem Hinweis folgt eine Warteschleife von ca. 2 Sekunden. Danach wird ins Menü verzweigt.

So einfach ist das Speichern von Daten auf dem Diskettenlaufwerk. Doch nun wollen wir diese Daten wieder zurückholen.

DATEI LADEN

Das Laden der Adressen verläuft fast auf die gleiche Weise wie das Speichern. Hier wird nicht geprüft, ob Daten im Rechner vorhanden sind. Der Anwender sollte also neu eingegebene Adressen erst abspeichern, bevor er eine Adressendatei lädt.

```
5000 REM =====
5010 REM   Datei laden
5020 REM =====
5030 LOCATE 1,12
5040 PRINT"Bitte Diskette einlegen!"
5050 PRINT
5060 INPUT"Druecken Sie danach RETURN ";antwort$
5070 PRINT
```

Nachdem alle Vorbereitungen getroffen sind, kann die Datei wieder geöffnet werden. Diesmal aber zum Lesen. Schauen Sie sich auch hier zunächst die Beschreibung an:

Problem:	Öffnen einer Datei zum Lesen
Befehl:	OPENIN "name"
Parameter:	<i>name</i> - Name der Datei
Beispiel:	OPENIN "Adressen" Öffnet die Datei "Adressen" zum Lesen

Setzen wir diesen Befehl nun in unser Programm ein:

```
5080 OPENIN "Adressen"  
5090 INPUT #9,zeiger  
5100 FOR i1=1 TO zeiger  
5110 FOR i2=1 TO 7  
5120 INPUT #9,adressen$(i1,i2)  
5130 NEXT i2  
5140 NEXT i1  
5150 CLOSEIN
```

Hier wird die Datei "Adressen" auf der Diskette zum Lesen geöffnet. Der Lesezugriff wird durch den Befehl 'OPENIN' gekennzeichnet. Danach wird die Anzahl der in dieser Datei gespeicherten Daten eingelesen. Diese Variable wird als Endwert für die Schleife 'i1' benutzt. Es werden also soviel Datensätze eingelesen, wie zuvor gespeichert wurden. Der Kanal wird in Zeile 5180 mit dem Befehl 'CLOSEIN' wieder geschlossen.

```
5160 GOSUB 300:PRINT "Datei ist geladen"  
5170 FOR i=1 TO 2000:NEXT i  
5180 GOTO 1000
```

Die letzten Zeilen dieses Abschnitts geben wieder eine Meldung aus und verzweigen nach der Warteschleife zurück ins Menü.

PROGRAMM BEENDEN

Wie zu Anfang des Kapitels bereits erwähnt, sollte ein Programm nicht mit Hilfe des Netzschalters ausgeschaltet werden. Wie ein Programm beendet werden sollte, wird nun demonstriert. Blicken wir zunächst wieder auf die ersten Zeilen.

```
35000 REM =====
35010 REM   Programm beenden
35020 REM =====
35030 IF zeiger=0 THEN 35150
```

Die ersten Zeilen werden Ihnen sicher schon langweilig. Doch die Zeile 35030 sieht ungewohnt aus. Wenn 'zeiger' 0 ist, also wenn keine Daten im Rechner gespeichert sind, wird das Programm ohne weiteres sofort beendet. Dies geschieht in Zeile 35150.

```
35040 LOCATE 1,12
35050 PRINT"Sind alle Daten gesichert (j/n) ";
35060 INPUT antwort$
35070 IF antwort$="n" THEN 1000
35080 IF antwort$="j" THEN 35100
35090 fehler$=" 'j' oder 'n' druecken!":GOSUB 200:GOTO 35040
```

Langsam erkennt man den Sinn eines Programmteils "PROGRAMM BEENDEN". Es wird also festgestellt, ob alle Daten gesichert wurden. Immerhin kann man diesen Programmabschnitt versehentlich aufgerufen haben. Wenn Sie nun mit nein antworten, kommen Sie wieder in das Menü zurück. Wenn Sie die Adressen jedoch abgespeichert haben, kann Sie nichts daran hindern, das Programm zu beenden.

```
35100 GOSUB 300:LOCATE 1,12
35110 PRINT"Das Programm kann mit 'GOTO 1000'wieder"
35120 PRINT"gestartet werden, ohne dass Daten ver-"
35130 PRINT"loren gehen!!"
35140 PRINT
35150 END
```

Nun folgt vor Beendigung des Programms ein wichtiger Hinweis: Nach dem Verlassen des Programms kann man es mit dem Befehl 'GOTO 1000' wieder ohne Datenverlust starten. Der Befehl RUN löscht vor dem Start sämtliche Variablen, ist also nicht zu empfehlen.

Nun haben wir es geschafft. Sie besitzen nun eine bis aufs Letzte dokumentierte Adressenverwaltung, bei deren Erstellung Sie sicher viel gelernt haben. Das, was Sie nun gelernt haben, können Sie entweder zur Verwirklichung eigener Ideen oder zur Durchführung individueller Programmänderungen, speziell bei dieser Adressenverwaltung, verwenden. Falls Sie einige Abschnitte nicht sofort verstanden haben, arbeiten Sie diese ruhig nochmals durch. Möchten Sie BASIC bis ins letzte Detail lernen, so empfehle ich Ihnen das Buch "Das BASIC Trainingsbuch zum CPC", das sich an dieses Buch anlehnt.

KAPITEL 6: FARBE UND GRAFIK

Für viele Käufer der zahlreich angebotenen Homecomputer ist die Farbdarstellung und vor allem die Grafikfähigkeit des Rechners ein entscheidendes Kaufkriterium. Der CPC bietet in seiner Preisklasse unvergleichbare Farbenpracht und hervorragende Grafikeigenschaften. So verfügt der Besitzer des CPC6128 über 27 Farben, von denen je nach Auflösung der Grafik 2, 4 oder 16 Farben ausgewählt werden können. Die Auflösung der Grafik reicht bis zu 640 x 200 Bildpunkten, was für einen Rechner dieser Preisklasse geradezu sensationell ist.

DIE GRAFIKBETRIEBSARTEN

Der CPC6128 arbeitet mit drei verschiedenen Grafikauflösungen, die mit dem bereits bekannten Befehl 'MODE' ausgewählt werden. Es besteht somit ein unmittelbarer Zusammenhang zwischen Zeichendichte und Grafikauflösung. Die folgende Tabelle soll hier einen ersten Überblick verschaffen.

Befehl	Zeichen/ Zeile	Auflösung Grafik	Auflösung Zeichen	Farben
MODE 0	20	160 x 200	32 x 8	16
MODE 1	40	320 x 200	16 x 8	4
MODE 2	80	640 x 200	8 x 8	2

Der Zusammenhang zwischen Zeichendichte und Grafikauflösung wird anhand dieser Übersicht deutlich. Sie erkennen auch, daß sich ein Zeichen im MODE 2 aus 8 mal 8 Punkten zusammensetzt. Im MODE 1 verdoppelt sich diese Matrix auf 16 x 8 Punkten. Die meisten vergleichbaren Rechner stellen einzelne Zeichen in einer Matrix aus 8 x 8 Punkten dar und das bei 40 Zeichen pro Zeile. Der CPC6128 bietet also in jeder Hinsicht eine doppelt so hohe Auflösung wie vergleichbare Rechner.

Jeweils 8 Bildpunkte werden in einer Speicherstelle festgehalten. Der gesamte Bildschirm setzt sich aus 640 x 200, also 128.000 Punkten zusammen, deren Speicherung 16.000 Speicherstellen erfordert. Von Ihrem Gesamtspeicher (64 KByte) gehen also ca. 16 KByte zur Grafikspeicherung drauf. Bei den anderen Auflösungsarten verringert sich dieser Aufwand nicht, da zusätzlich zu dem Punkt die entsprechende Farbe festgehalten werden muß. Auch normaler Text wird im Grafikspeicher festgehalten. Viele andere Rechner vereinigen Text und Grafik nicht. Da gibt es jeweils einen eigenen Speicherbereich für Text und Grafik.

DIE BILDSCHIRMFARBEN

Nach dem Einschalten des Rechners erkennen Sie die Standardfarben der Betriebsart **MODE 1**, blau und hellgelb, vorausgesetzt Sie haben einen Farbmonitor oder -fernseher angeschlossen. Dies sind nur zwei der 27 möglichen Farben. In der folgenden Tabelle sind alle Farben zu ersehen.

Nummer	Farbe	Nummer	Farbe
0	schwarz	13	weiß
1	blau	14	pastellblau
2	hellblau	15	orange
3	rot	16	rosa
4	magenta	17	pastellmagenta
5	hellviolett	18	hellgrün
6	hellrot	19	seegrün
7	purpur	20	helltürkis
8	hellmagenta	21	limonengrün
9	grün	22	pastellgrün
10	türkis	23	pastelltürkis
11	himmelblau	24	hellgelb
12	gelb	25	pastellgelb
		26	hellweiß

Bei einer derart großen Farbenpracht ist die Bezeichnung der Farben nicht einfach. Da tauchen Farben wie hellweiß auf, die

außer in Waschmittelwerbungen eigentlich unsinnig sind. Aber so kleinlich wollen wir nicht sein.

Jede Farbe wird mit einer eindeutigen Nummer bestimmt, nicht etwa mit der Bezeichnung. Die Nummern der Standardfarben sind somit 1 für blau und 24 für hellgelb.

DIE RAHMENFARBE

Nicht nur die Hintergrund- und die Zeichenfarbe kann bestimmt werden, auch der Rahmen kann in einer individuellen Farbe dargestellt werden. Der Rahmen ist der Bereich des Bildschirms, den man mit dem Cursor nicht erreichen kann. Nach dem Einschalten des Rechners ist der Rahmen nicht zu erkennen, da er in der gleichen Farbe wie der Hintergrund dargestellt ist. Mit einem Befehl kann man diese Farbe jedoch ändern. Wie immer folgt nun zunächst die Beschreibung dieses Befehls.

Problem: Einstellung der Rahmenfarbe

Befehl: `BORDER farb1, farb2`

Parameter: *farb1* - Farbnummer
farb2 - wird nur angegeben, wenn die Farbe ständig zwischen 'farb1' und 'farb2' wechseln soll

Beispiel: `BORDER 0`
 ändert die aktuelle Rahmenfarbe in schwarz
`BORDER 0,13`
 läßt den Rahmen zwischen schwarz und weiß blinken

Bemerkung: Es kann in jeder Betriebsart (MODE 0 bis MODE 2) jede der 27 Farben angegeben werden!

Nun kennen Sie bereits den ersten Befehl, um die 27 Farben zu begutachten. Warten wir also nicht länger und ändern die

Rahmenfarbe in die Zeichenfarbe (Nummer 24) um. Der entsprechende Befehl ist

BORDER 24

Sie erkennen den Rahmen, der bisher immer verborgen war. Diesen Rahmen können Sie nun in allen beliebigen Farben erstrahlen lassen. Da wir aber schreibfaul sind, lassen wir ein Programm für uns arbeiten. Dieses Programm soll alle Farben nacheinander anzeigen. Das folgende Programm übernimmt diese Aufgabe:

```
10 FOR farbe=0 TO 26
20 BORDER farbe
30 IF inkey$="" THEN 30
40 NEXT farbe
```

Was, dieses Programm sagt Ihnen nichts? Dann haben Sie sicher das Kapitel der BASIC-Einführung überschlagen und wollten dieses angenehmere Kapitel vorziehen. Das hat sicherlich keinen Zweck, da hier immer wieder kleinere Programme auftauchen, die Ihnen die Funktion der neuen Grafikbefehle verdeutlichen sollen.

Doch nun wieder zu dem Programm. In einer Schleife erzeugen wir die einzelnen Farbnummern, die dann in der Zeile 20 zugewiesen werden. Wie alle anderen Parameter können Sie auch diese Farbnummer als Variable angeben. Die Zeile 30 wartet auf einen Tastendruck Ihrerseits, damit die nächste Farbe bestimmt wird. So können Sie auf dem schnellsten Wege alle Farben Ihres CPC6128 begutachten. Wenn Sie nur einen Grünmonitor besitzen, werden Sie nur verschiedene Grünabstufungen feststellen. Dies soll Sie aber nicht daran hindern, trotzdem die Grafikmöglichkeiten zu erforschen.

Der Befehlsbeschreibung haben Sie entnommen, daß der Bildschirmrahmen blinken kann. Dazu werden zwei Farbnummern hinter dem Befehl mit einem Komma getrennt angegeben. Der Rahmen wechselt dann ständig zwischen diesen beiden Farben. Probieren wir dies nun aus, indem wir die zwei Farben mit dem

größten Kontrast verwenden, schwarz und hellweiß. Der Befehl dazu ist

BORDER 0,26

Es ist sicher nicht empfehlenswert, mit einem derartig blinkenden Rahmen weiterzuarbeiten, wenn Sie Ihre Augen schonen möchten. Schalten wir dieses Blinken nun wieder ab.

BORDER 1

Eine Anwendungsmöglichkeit für das Blinken des Rahmens ist die Fehlermeldung in einem Programm. Der Anwender wird so optisch eindrucksvoll auf einen Fehler hingewiesen.

EINSTELLEN DER BLINKGESCHWINDIGKEIT

Der CPC6128 läßt kaum Wünsche offen. So kann auch die Blinkgeschwindigkeit des Rahmens und des Bildschirms eingestellt werden. Dazu holen wir folgenden Befehl aus der Trickkiste:

Problem: Einstellen der Blinkgeschwindigkeit

Befehl: **SPEED INK *time1,time2***

Parameter: *time1* - Zeit für die 1. Farbe
time2 - Zeit für die 2. Farbe (jeweils
in Einheiten zu 0.02 Sek.)

Beispiel: **SPEED INK 50,100**
Hält die erste Farbe eine Sekunde und die zweite Farbe zwei Sekunden

Bemerkung: Für '*time1*' und '*time2*' kann ein Wert zwischen 1 und 255 angegeben werden. Dies entspricht einer Verzögerung von 0.02 bis 5.1 Sekunden.

Mit diesem Befehl bestimmen Sie also den Blinkrythmus des Rahmens. Auch diesem Befehl folgt nun ein Beispielprogramm:


```
10 INPUT "1. Farbe (0-26):    ";farbe1
20 INPUT "Verzoegerung (1-255):";verz1
30 INPUT "2. Farbe (0-26):    ";farbe2
40 INPUT "Verzoegerung (1-255) ";verz2
50 BORDER farbe1,farbe2
60 SPEED INK verz1,verz2
```

Hier werden beide Farben und deren Verzögerung bestimmt. Die anschließenden Befehle lassen dann den Rahmen blinken.

ÄNDERN DER ZEICHENFARBE

Die Standardfarbe der Zeichen ist wie bereits bekannt hellgelb (Nummer 24). Von dieser Farbe können Sie jedoch abweichen. Der folgende Befehl unterstützt Sie dabei:

Problem: Ändern der Zeichenfarbe

Befehl: PEN #*n*,*farbreg*

Parameter: #*n* - Nummer des Bildschirm-
 fensters (kann ignoriert
 werden)
 farbreg - Nummer des Farbregisters
 (0-15)

Beispiel: PEN 1
 Ändert die Zeichenfarbe in die Farbe, die das
 Farbregister 1 enthält

Bemerkung: Farbregister werden mit dem Befehl INK
 geändert

Sicher sind Sie von den Farbregeistern verwirrt worden. Wie Sie bereits wissen, können nicht alle 27 Farben gleichzeitig verwendet werden. Je nach Betriebsart (Befehl MODE) können 2, 4 oder 16 der 27 Farben eingesetzt werden. Im 40-Zeichenmodus können Sie zwischen 4 Farben auswählen. Diese vier Farben werden vom Befehl INK bereitgestellt. Schauen wir uns auch diesen Befehl zunächst an.

Problem: Bestimmen der Auswahlfarben

Befehl: INK *farbreg, farb1, farb2*

Parameter: *farbreg* - Farbregeister (0-15)
farb1 - Farbnummer (0-26)
farb2 - wird angegeben, wenn zwischen den Farben 'farb1' und 'farb2' ständig gewechselt werden soll (blinken)

Beispiel: INK 1,3
 Setzt die Farbe rot in das Farbregeister 1
 INK 3,1,9
 läßt die Farbe des Registers 2 zwischen blau und grün blinken

Da wir maximal 16 Farben (MODE 0) auswählen können, gibt es auch 16 Farbregeister mit den Nummern 0 bis 15. Wollen wir also beispielsweise die Zeichenfarbe ändern in rot, so müssen wir die Nummer 3 für rot in ein Farbregeister setzen. Nehmen wir das Register 1, so lautet der Befehl

INK 1, 3

Nun kann der Befehl 'PEN' die Zeichenfarbe ändern, indem dieses Farbregeister angegeben wird.

PEN 1

Dieser Befehl sagt nun: "Ändere die Zeichen in die Farbe, die im Farbregister 1 enthalten ist". In dieses Register haben wir zuvor die rote Farbe gesetzt. Die Farbregister sind nach dem Einschalten des CPC bereits mit Farben belegt. Welche Farben dies sind, entnehmen Sie der folgenden Tabelle:

Farbregister	MODE 0	MODE 1	MODE 2
0	1	1	1
1	24	24	24
2	20	20	1
3	6	6	24
4	26	1	1
5	0	24	24
6	2	20	1
7	8	6	24
8	10	1	1
9	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	1,24	20	1
15	16,11	6	24

Da die Farbauswahl bei **MODE 1** und **MODE 2** begrenzt ist, wiederholen sich dementsprechend die Farben. Mit dem Farbregister 0 bei **MODE 1** z.B. werden gleichzeitig die Farbregister 4, 8 und 12 angesprochen. Dadurch können auch nicht mehr als vier Farben in **MODE 1** und zwei Farben in **MODE 2** ausgewählt werden.

Der Befehl INK bewirkt auch, daß alle Zeichen, die mit der zuvor im Farbregister enthaltenen Farbe ausgegeben wurden, gleichzeitig mit geändert werden. Ein Beispiel: Die Standardzeichenfarbe nach dem Einschalten wird dem Farbregister 1 entnommen. Wenn Sie nach dem Einschalten nun die Farbe im Register 1 ändern, so ändert sich gleichzeitig der Text, der in dieser Farbe dargestellt ist. Geben Sie z.B. den folgenden Befehl ein:

INK 1,13

Alle Zeichen auf dem Bildschirm werden sofort weiß. Schreiben Sie jedoch Zeichen mit der Farbe aus Topf 2 auf dem Bildschirm, so hat eine Änderung der Farbe in Register 1 keine Auswirkung.

Die standardmäßige Zuweisung für den Bildschirmhintergrund ist das Farbregister 0. Wenn Sie also die Zeichenfarbe aus dem Register 0 wählen, so sind Zeichen- und Hintergrundfarbe identisch. Die Folge ist, daß die Zeichen nicht mehr erkannt werden können. Probieren Sie dies einmal aus:

PEN 0

Nun verschwindet der Cursor. Sie müssen dies nun rückgängig machen, indem Sie den folgenden Befehl "blind" eingeben:

PEN 1

Das Farbregister 1 ist also standardmäßig für die Zeichenfarbe zuständig. Da ein Register immer für den Hintergrund reserviert ist, können wir im MODE 1 drei verschiedene Schriftfarben verwenden. Das folgende Programm gibt drei Zeilen in verschiedenen Farben aus:

```
10 MODE 1
20 INK 1,12:INK 2,9:INK 3,13
30 PEN 1:PRINT "gelb"
40 PEN 2:PRINT "gruen"
50 PEN 3:PRINT "weiss"
```

Wir verwenden für die drei Schriftfarben die Farbreister 1 bis 3. Hier setzen wir in Zeile 20 die Farben gelb, grün und weiß in die Register 1 bis 3. Anschließend werden die drei Zeilen mit jeweils einer anderen Farbe ausgegeben.

In einem Farbreister können aber auch zwei Farben untergebracht werden, was dann ein Blinken der Farben verursacht. Versetzen Sie zunächst den Rechner in den Einschaltzustand (SHIFT-CTRL-ESC). Nun ändern wir die Zeichenfarbe (Register 1) um in schwarz/weiß.

INK 1,0,13

Sofort blinken alle Zeichen auf dem Bildschirm. Auch die Zeichen, die Sie nun eingeben, blinken, da sie dem Farbreister 1 entsprechen. Um die folgende Schrift unabhängig von der bereits auf dem Bildschirm enthaltenen Schrift zu machen, ist der folgende Befehl notwendig:

PEN 2

Von nun an kommt die Zeichenfarbe aus dem Register 2. Das Blinken können Sie abschalten, indem Sie die Farbe 24 ins Register 1 setzen.

INK 1,24

Es folgt wieder ein kleines Demonstrationsprogramm:

```
10 MODE 1
20 INK 2,1,26
30 INK 3,26,1
40 PEN 2:PRINT "HIN UND HER"
50 PEN 3:PRINT "HIN UND HER"
60 PEN 1
```

ÄNDERN DER HINTERGRUNDFARBE

Nun haben wir bereits die Rahmen- und die Zeichenfarbe geändert. Ein weiterer Befehl kann auch die Farbe des Hintergrundes ändern:

- Problem:** Ändern der Hintergrundfarbe
- Befehl:** **PAPER #n, *farbreg***
- Parameter:** #n - Nummer des Bildschirm-
fensters (kann ignoriert
werden)
- farbreg* - Farbregister (0-15)
- Beispiel:** **PAPER 2**
 Ändert die Hintergrundfarbe um in die Farbe
 aus Farbregister 2
- Bemerkung:** **Um den gesamten Hintergrund zu ändern, muß**
 dem Befehl PAPER ein CLS folgen.

Der Befehl PAPER ähnelt dem Befehl PEN insoweit, als bei beiden ein Register angegeben werden muß, das die gewünschte Farbe enthält. Die Standardzuweisung für den Hintergrund ist Farbregister 0. Auch hier dürfen wir nicht das Farbregister 1 angeben, da dieses die Farbe der Zeichen enthält. Sie würden dann wieder die Schrift unsichtbar machen.

Doch setzen wir nun einmal den Befehl PAPER ein. Wir wollen den Hintergrund hellrot erscheinen lassen. Diese Farbe ist standardmäßig im Register 3 enthalten. Geben Sie also den folgenden Befehl ein:

PAPER 3

Nun erstrahlt aber nicht etwa der gesamte Hintergrund des Bildschirms hellrot, sondern nur der Hintergrund der nachfolgenden Zeichen. Um den gesamten Hintergrund zu ändern, müssen Sie nach dem Befehl PAPER noch den Bildschirm löschen (CLS).

Auch der Hintergrund kann zum Blinken gebracht werden, wenn beim PAPER-Befehl ein Farbregister angegeben wird, das zwei Farben enthält. Lassen Sie uns nun den Bildschirmhintergrund zwischen schwarz und weiß blinken, nachdem Sie den Rechner wieder in den Einschaltzustand versetzt haben (SHIFT-CTRL-ESC). Die folgenden Befehle müssen dazu nacheinander eingegeben werden:

```
INK 3,0,13
PAPER 3
CLS
```

Die Frequenz dieses Blinkens können Sie mit dem bereits erwähnten Befehl 'SPEED INK' ändern. Wollen Sie z.B. jede Sekunde die Farbe wechseln lassen, so geben Sie den Befehl

```
SPEED INK 50,50
```

ein. Erinnern Sie sich: Die Werte in diesem Befehl stellen Einheiten von 0.02 Sekunden dar ($50 \times 0.02 = 1$ Sekunde).

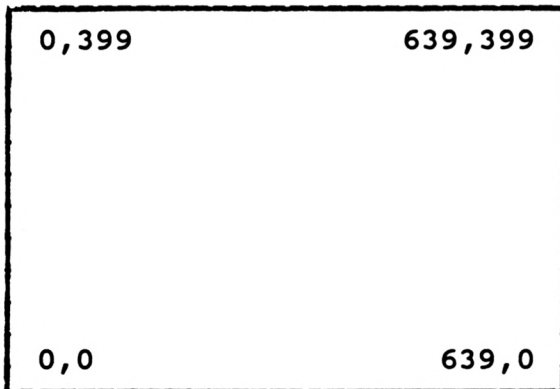
Zum Abschluß wieder ein kleines, eindrucksvolles Programmbeispiel:

```
10 MODE 1
20 INK 2,0,13
30 INK 3,13,0
40 SPEED INK 50,50
50 PEN 2
60 PAPER 3
70 CLS
80 PRINT "POSITIV UND NEGATIV"
```

HOCHAUFLÖSENDE GRAFIK

Wie Ihnen bereits bekannt ist, können Sie zwischen drei verschiedenen Grafikauflösungen mit unterschiedlicher Farbenanzahl wählen. Zum Zeichnen von Figuren gibt es einige Befehle, die ausschließlich am Beispiel der hochauflösenden Grafik vorgestellt werden.

Erinnern Sie sich: Die hochauflösende Grafik besteht aus 640 x 200 Bildpunkten und zwei Farben, einer Hintergrund- und einer Zeichenfarbe. Damit jeder Punkt angesprochen werden kann, muß man sich den Bildschirm als Koordinatensystem vorstellen. Ähnlich wie beim bereits bekannten Befehl LOCATE, der den Cursor an eine beliebige Position bringen kann, wird auch bei den Grafikbefehlen eine Position bestimmt, nämlich mit zwei Werten. Der erste Wert bestimmt die Position auf der waagerechten Achse (X-Achse) des Bildschirms von links nach rechts orientiert. Der zweite Wert entspricht der Position auf der senkrechten Achse (Y-Achse) von unten nach oben zählend. Da das Auflösungsverhältnis von 640 x 200 Punkten nicht dem des Bildschirms entspricht, erstreckt sich der Y-Wert nicht von 0 - 199, sondern von 0 - 399, wobei zwei Y-Werte die gleiche Position bestimmen (0 und 1, 2 und 3, ...). Die vier Eckpunkte haben folgende Koordinaten:



Wenn Sie sich mit diesem System vertraut gemacht haben, können wir uns mit den ersten Befehlen zur Erzeugung der Grafik beschäftigen.

ZEICHNEN VON PUNKTEN

Problem:	Setzen eines Punktes	
Befehl:	PLOT x,y	
Parameter:	x -	X-Achsen-Position (0-639)
	y -	Y-Achsen-Position (0-399)
Beispiel:	PLOT 0,0 Setzt einen Grafikpunkt in die untere, linke Ecke des Bildschirms	

Die Grafikbefehle sind allgemein gut verständlich, wenn man das Koordinatensystem beherrscht. Das Setzen einzelner Punkte wird aber langweilig, wenn man nicht weitere Befehle einsetzt. So ist es sehr interessant, Punkte innerhalb von Schleifen zu erzeugen. Sehr einfach ist es z.B., Linien zu erzeugen. Zwar gibt es dafür einen speziellen Befehl, doch zur Übung des Befehls PLOT ist diese Anwendung trotzdem interessant.

Versuchen Sie nun einmal, eine senkrechte Linie am äußersten linken Bildschirmrand von unten nach oben zu zeichnen. Das folgende Programm erfüllt diese Aufgabe:

```
10 MODE 2
20 FOR Y=0 TO 399
30 PLOT 0,Y
40 NEXT Y
```

Dieses Programm ist recht leicht zu verstehen. Lassen Sie uns den Schwierigkeitsgrad ein wenig erhöhen. Schreiben Sie nun ein Programm, das einen Rahmen um den gesamten Bildschirm erzeugt. Hier die Lösung:

```

10 MODE 2
20 FOR Y=0 TO 399
30 PLOT 0,Y:PLOT 639,Y
40 NEXT Y
50 FOR X=0 TO 639
60 PLOT X,0:PLOT X,639
70 NEXT X

```

Sie haben doch nicht etwa vier Schleifen benötigt? Die beiden parallelen Linien können in einer Schleife gleichzeitig gezogen werden! Doch dies sind kleine Tricks, die die Programmiererfahrung bald mit sich bringt.

Das Zeichnen der senkrechten Linien kann erheblich beschleunigt werden. Erinnern Sie sich, jeweils zwei aufeinanderfolgende Y-Werte bestimmen den gleichen Punkt! Wir können die Y-Schleife also halbieren. Dazu erhöhen wir das Schleifeninkrement (STEP-Parameter) auf 2. Ändern Sie dazu die folgende Zeile:

```
20 FOR Y=0 TO 399 STEP 2
```

Beachten Sie nach dem erneuten Start des Programms die Beschleunigung. Denken Sie also immer daran, ob nicht das Zeichnen der Y-Achse beschleunigt werden kann.

ZEICHNEN EINES KREISES

Da der CPC6128 keinen Befehl zum Zeichnen eines Kreises bereithält, müssen wir uns anders helfen. Jeder einzelne Punkt muß berechnet und gesetzt werden. Dabei benutzen wir die folgenden Formeln:

$$\begin{aligned}
 x &= r * \cos(a) \\
 y &= r * \sin(a)
 \end{aligned}$$

Für 'r' muß der Radius und für 'a' der Winkel eingesetzt werden. In Verbindung mit einer Schleife, die den Winkel von 1 bis 360 Grad hochzählt, kann jeder einzelne Punkt des Kreises erzeugt werden. Beachten Sie zunächst das folgende Programm:

```
10 MODE 2
20 x=320:y=200:r=100
30 DEG
40 FOR a=1 TO 360
50 PLOT x+r*COS(a),y+r*SIN(a)
60 NEXT a
```

In der Zeile 20 wird der Mittelpunkt und der Radius des gewünschten Kreises bestimmt. Der Befehl DEG besagt, daß alle folgenden Winkelberechnungen in Neugrad durchzuführen sind. Der alternative Befehl dazu ist RAD und rechnet im Bogenmaß (Altgrad). Dabei gilt '90 Altgrad = 100 Neugrad'. Die folgende Zeile stellt den Schleifenanfang dar. Hier soll ein ganzer Kreis (360 Grad) erzeugt werden. Wenn Sie die Anfangs- und Endwerte ändern, werden Kreisabschnitte erzeugt. In der Zeile 50 ist die eigentliche Formel untergebracht. Zu der errechneten Punktposition muß noch der Mittelpunkt (x und y) addiert werden. Anschließend folgt das Ende der Schleife.

Mit dieser Routine können Sie nun Kreise erzeugen, auch wenn Sie den mathematischen Zusammenhang nicht verstanden haben. Wenn Sie auch noch Ellipsen erzeugen möchten, so kann ich Ihnen hier nicht weiterhelfen. Vielleicht holen Sie Ihre alten Schulbücher aus dem Schrank und geben sich selbst etwas Nachhilfe in Geometrie. Doch Spass beiseite - für viele schöne Grafiken sind mathematische Algorithmen unverzichtbar.

ZEICHNEN VON GERADEN

Problem: Zeichnen einer Geraden

Befehl: DRAW x, y

Beispiel: DRAW 639,0
Zeichnet eine Gerade zwischen der Position des Grafikcursors und der angegebenen x/y - Position

Die einzige Frage, die dieser Befehl aufkommen läßt, ist der Grafikcursor. Zu Anfang, also nach dem Einschalten des Rechners, steht dieser Cursor auf '0,0'. Jeder anschließende Grafikbefehl aktualisiert diesen Cursor. Setzt man also einen Punkt auf '100,200', so steht auch der Grafikcursor auf diesem Punkt. Ein anschließender DRAW-Befehl zeichnet die Gerade von diesem Punkt ausgehend. Der letzte Punkt dieser Geraden ist wiederum die neue Position des Grafikcursors.

Es besteht jedoch die Möglichkeit, den Grafikcursor an eine beliebige andere Stelle zu setzen. Der folgende Befehl hat diese Aufgabe:

Problem: Setzen des Grafikcursors

Befehl: MOVE x,y

Beispiel: MOVE 200,250
setzt den Grafikcursor auf die Position '200,250'

Sicher haben Sie bemerkt, daß in den letzten beiden Befehlsbeschreibungen die Parameter nicht erwähnt wurden. Dies ist reine Absicht, da wohl mittlerweile jeder weiß, was mit ' x,y ' gemeint ist.

Zeichnen wir nun eine Senkrechte am äußersten linken Bildschirmrand, wie wir es zuvor mit dem Befehl PLOT realisiert haben. Dazu benötigen wir nur zwei Befehle, nach dem Einschalten sogar nur einen:

```
10 MODE 2
20 MOVE 0,0
30 DRAW 0,399
```

Der Befehl 'MOVE 0,0' muß nach dem Einschalten des Rechners nicht eingegeben werden, da dann der Grafikkursor ohnehin auf dieser Position steht.

Nun soll auch der Rahmen, der beim Befehl PLOT erzeugt worden ist, mit DRAW gezeichnet werden.:

```
10 MODE 2
20 MOVE 0,0: REM Ecke unten links
30 DRAW 0,399: REM zur Ecke oben links
40 DRAW 639,399: REM zur Ecke oben rechts
50 DRAW 0,639: REM zur Ecke unten rechts
```

Dieses Programm erklärt sich selbst. Das Verbinden einzelner Punkte ist nun kein Problem mehr. Interessant wird das Zeichnen von Linien in Verbindung mit Schleifen. Überzeugen Sie sich selbst:

```
10 MODE 2
20 FOR i=0 TO 399 STEP 10
30 MOVE 399,i
40 DRAW 0,399-i
50 NEXT i
```

Ein einfaches Program mit großer Wirkung. Weil's so schön war, folgt noch ein weiteres:

```
10 MODE 2
20 MOVE 0,0
30 FOR i=0 TO 399 STEP 10
40 DRAW 399,i
50 DRAW 399-i,399
60 DRAW 0,399-i
70 DRAW i,0
80 NEXT i
```

Ist es nicht bewundernswert, wenn derartige Bilder auf dem Bildschirm entstehen, noch dazu wenn sie aus Ihrer eigenen Feder stammen? Vielleicht versuchen Sie sich auch einmal an derartigen Fantasiegrafiken. Es macht sehr viel Spaß, auch wenn es nicht auf Anhieb klappt. Im Vertrauen, ich habe mir an diesem Bild auch den Kopf zerbrochen. Die Schwierigkeit ist, die Bilder, die man sich vorstellt, in das Koordinatensystem umzurechnen, also den Algorithmus zu bilden. Das artet zu echten Denksportaufgaben aus.

RELATIVES ZEICHNEN

Zum Abschluß dieses Kapitels lernen Sie eine Variante der bisher beschriebenen Befehle kennen. Bei den Befehlen PLOT, MOVE war der Grafikkursor bisher nicht relevant, da wir immer einen absoluten Punkt bestimmt haben. Fügen wir jedoch den Befehlen den Buchstaben 'R' an, so hat dieser eine andere Wirkung. Die Befehle sehen dann wie folgt aus:

```
PLOTR x,y
DRAWR x,y
MOVER x,y
```

Bei diesen Befehlen wird keine absolute x/y-Koordinate angegeben, sondern ein relativer Wert. Kurzum wird zur Ermittlung der endgültigen Koordinate zur X-Position des Grafikcursors der mit dem Befehl angegebene relative X-Wert addiert. Das gleiche gilt für die Y-Koordinate. Dazu ein Beispiel: Der Grafikcursor befindet sich an der Position '200,100'. Der Befehl 'PLOTTR 50,50' setzt nun einen Punkt an die Position '250,150'.

Das folgende Beispiel soll hier etwas mehr Licht hineinbringen. Es soll ein Quadrat von 10 mal 10 Punkten gezeichnet werden. Die Koordinaten der oberen linken Ecke befinden sich in den Variablen 'x' und 'y'.

```
10 MODE 2
20 x=200:y=100
30 MOVE x,y
40 DRAWR 10,0: REM 10 nach rechts
50 DRAWR 0,10: REM 10 nach oben
60 DRAWR -10,0: REM 10 nach links
70 DRAWR 0,-10: REM 10 nach unten
```

Sie haben bemerkt, daß auch negative Zahlen angegeben werden können. Dies ist notwendig, um auch Positionen links und unterhalb der Grafikposition relativ anzusprechen.

Dies ist der Abschluß der Grafikkapitel für Einsteiger. Weitere Grafikbefehle finden Sie in Ihrem Handbuch oder in weiteren DATA-BECKER-BÜCHERN zum CPC6128.

KAPITEL 7: TONERZEUGUNG

Ein Computer eignet sich sehr gut zur Erzeugung von Tönen. Der Prozessor, das eigentliche Arbeitspferd im Computer, wird von einem Taktgenerator angesteuert. Dieser Generator erzeugt je nach Rechner ca. 1-4 MHz (Millionen Takte pro Sekunde). Der einfachste Weg ist, diesen Takt auf einen Lautsprecher zu legen. Da die dabei entstehenden Töne vom menschlichen Gehör nicht mehr wahrgenommen werden können, muß die Taktfrequenz erheblich tiefer sein. Dies wird erreicht, indem die Taktfrequenz vor Ausgabe auf den Lautsprecher geteilt wird. Die dabei entstehenden Frequenzen werden dann als Töne wahrgenommen.

Dies war die erste Generation der Tonerzeugung mit dem Computer. Da diese Töne, vergleichbar mit den Wecktönen einer Armbanduhr, nicht zufriedenstellend waren, wurde die Tonerzeugung in Heimcomputern ständig weiterentwickelt. Das Resultat ist, daß viele Homecomputer mit sogenannten Synthesizern ausgerüstet sind, die auf einem Chip untergebracht sind. Die damit erzeugten Töne sind denen der "natürlichen" Musikinstrumente verblüffend ähnlich.

Den tonerzeugenden Chip im CPC6128 kann man nicht als Synthesizer im eigentlichen Sinn bezeichnen. Ihm fehlen wichtige Eigenschaften wie verschiedene Wellenformen (Sinus, Rechteck etc.) und die klangbeeinflussenden Filter. Auch ist das eigentliche Prinzip des Synthesizers, die Frequenz (Höhe) der Töne von einer Spannung abhängig zu machen, nicht gegeben. Der Tongenerator im CPC6128 macht die Frequenz von einem Teilungsfaktor abhängig.

Der Tongenerator des CPC6128 verfügt über drei Stimmen. Es können also drei Töne gleichzeitig erklingen. Weiterhin ist ein Rauschgenerator vorhanden, mit dem neben den Tönen auch Geräusche erzeugt werden können.

Alles, was dem Tongenerator entlockt wird, erklingt über den im CPC eingebauten Lautsprecher. Der Ton eines eventuell angeschlossenen Fernsehers muß deshalb ausgeschaltet werden. Die Lautstärke des eingebauten Lautsprechers kann rechts am CPC eingestellt werden.

Wer seine eigenen Kompositionen jedoch in Stereo genießen möchte, dem bietet sich die Möglichkeit, den Tonausgang des CPC einer Stereoanlage zuzuführen. Dazu befindet sich hinten links am CPC eine Ausgangsbuchse (I/O). Hier muß ein Kabel angefertigt werden, falls dies nicht im Fachhandel erhältlich ist. Der Ausgang des CPC muß mit dem Eingang Ihres Verstärkers verbunden werden, an dem normalerweise ein Kasstendeck oder ein Tuner angeschlossen wird. Auch die oft vorhandene AUX-Buchse kann verwendet werden.

Im folgenden Kapitel lernen Sie die wichtigsten BASIC-Befehle zur Tonerzeugung kennen. Die gesamte, komplexe Tonerzeugung kann an dieser Stelle nicht erläutert werden. Aufbauende Informationen finden Sie in Ihrem Handbuch oder in anderen DATA BECKER BÜCHERN zum CPC.

DER SOUND-BEFEHL

Problem: Bestimmen der Toneigenschaften

Befehl: SOUND *a, b, c, d, e, f, g*

Parameter:

- a* - Kanalstatus (in diesem Kapitel immer 1)
- b* - Teiler, bestimmt die Frequenz (0-4095)
Frequenz = 125000/Teiler
- c* - Dauer des Tons (-32768 bis +32767, in Einheiten zu 0.01 Sekunden)
- d* - Lautstärke (0-15)
- e* - Hüllkurve für die Lautstärke (0-15)
- f* - Hüllkurve für den Ton (0-15)
- g* - Rauschcharakter (0-15)

Beispiel: **SOUND 1,1000,100,7,0,0,0**
 Erzeugt einen Ton von 125 Herz 1 Sekunde
 lang mit der Lautstärke 7

Da dieser Befehl sehr umfangreich ist, werden wir nicht alle Parameter einsetzen. Vor allem die Programmierung der Hüllkurven würde den Rahmen dieses Buches sprengen. Die Kenntnisse, die Sie in diesem Kapitel erwerben, bieten jedoch eine gute Grundlage zur Vertiefung mit Hilfe Ihres Handbuchs.

DIE TONHÖHE

Die für uns interessanten Parameter sollen zunächst näher beschrieben werden. Der erste Parameter 'a' gibt einen Kanalstatus an, der u.a. gleichzeitiges Spielen der drei Stimmen ermöglicht. In unseren folgenden Beispielen verwenden wir nur eine Stimme, und zwar die Stimme 1.

Der nachfolgende Parameter 'b' bestimmt die Tonhöhe. Hier wird ein Teiler angegeben, durch den die Grundfrequenz von 125000 Herz geteilt wird. Da der Teiler Werte zwischen 0 und 4095 annehmen kann, sind Frequenzen zwischen 30 und 125000 Herz möglich. Das folgende Programm läßt einen Ton von 30000 auf 300 Herz abschwellen. Sie können sich dann die verschiedenen Frequenzen besser vorstellen.

```
10 FOR teiler=4.1 TO 416 STEP 0.1
20 PRINT INT(125000/teiler)
30 SOUND 1,teiler,1,7,0,0,0
40 NEXT teiler
```

Die Funktion 'INT' schneidet von dem Wert in den nachfolgenden Klammern die Nachkommastellen ab, da diese in unserem Beispiel nicht benötigt werden. Sie sehen, daß wie bei allen Befehlen auch beim Befehl SOUND alle Parameter mit Variablen besetzt werden können.

Die Veränderung der Tonhöhe in einer Schleife ermöglicht zahlreiche Effekte. So kann z.B. leicht eine Sirene simuliert werden:

```
10 FOR teiler=120 TO 200
20 SOUND 1,teiler,2,7,0,0,0
30 NEXT teiler
40 FOR teiler=200 TO 120 STEP -1
50 SOUND 1,teiler,2,7,0,0,0
60 NEXT teiler
70 GOTO 10
```

Diese Sirene, die an amerikanische Polizeiwagen erinnert, kann mit der Taste 'ESC' abgebrochen werden (zweimal drücken). Für alle, die noch ein wenig mit der Sirene spielen möchten, bietet sich die folgende Variante des Programms an:

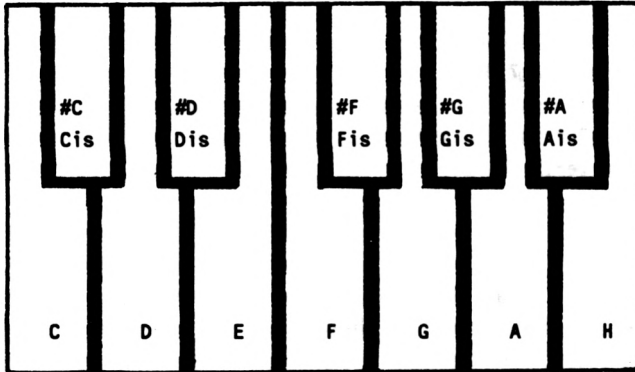
```
5 INPUT "Anfangsteiler";a
6 INPUT "Endteiler   ";b
7 INPUT "Tempo (1-10) ";t
10 FOR teiler=a TO b
20 SOUND 1,teiler,t,7,0,0,0
30 NEXT teiler
40 FOR teiler=b TO a STEP -1
50 SOUND 1,teiler,t,7,0,0,0
60 NEXT teiler
70 GOTO 10
```

Hier werden die wichtigsten Eigenschaften der gewünschten Sirene zunächst in Variablen festgehalten, die im dann folgenden Programm eingesetzt werden. Auch dieses Programm beenden Sie mit 'ESC'.

MUSIK AUS FREQUENZEN

Musikinstrumente erzeugen die Töne nicht stufenlos. Erst eine bestimmte Abstufung der Töne sind für das menschliche Gehör angenehm. Zum weiteren Verständnis müssen einige Grundlagen erläutert werden. In Eine sogenannte Tonleiter besteht aus 12

Tönen, wie Sie der folgenden Übersicht einer Klaviatur entnehmen können.



Eine Klaviatur besteht aus mehreren solcher Abschnitte. Diese Tonfolgen nennt man Oktaven. Die erste Oktave umfaßt die tiefen Töne, die letzte Oktave die höchsten Töne.

In welche Stufen sind die einzelnen Töne nun geteilt? Die Berechnung der Frequenzen der einzelnen Töne geht international vom Kammerton 'A' der mittleren Oktave aus. Dieser Ton hat 440 Herz. Lassen wir diesen Ton nun einmal erklingen. Welcher Teiler muß für diese Frequenz eingesetzt werden? Erinnern Sie sich: Der Teiler wird mit der Formel 'TEILER=125000/FREQUENZ' ermittelt. Bei 440 Herz ergibt dies 284.090909. Der Teiler muß jedoch stets ganzzahlig, also ohne Nachkommastellen, sein. Dadurch ergibt sich eine geringfügige Frequenzabweichung, die jedoch nur den Spezialisten auffallen wird. Erzeugen Sie nun das mittlere 'A'.

SOUND 1,284,100,7,0,0,0

Wie werden nun die anderen Töne aus dieser Grundfrequenz berechnet? Die entsprechende Formel ist wie folgt aufgebaut:

$$\text{FREQUENZ} = 440 * (2^{\uparrow}(\text{OKTAVE} + (\text{N} - 10) / 12))$$

OKTAVE - von -3 bis 4

N - Position des Tons in der Tonleiter ('C'=1, 'A'=12)

Der mathematische Ursprung der Formel soll an dieser Stelle nebensächlich sein. Wichtig ist es, daß Sie diese Formel einzusetzen wissen. Wenn z.B. der Ton 'C' der untersten Oktave erzeugt werden soll, so errechnet sich die Frequenz wie folgt:

$$440 * (2^{\uparrow}(-3 + (1 - 10) / 12)) = 32.7031957 \text{ Herz}$$

Zur Erzeugung dieses Tons muß noch der Teiler ermittelt werden. Die Formel dazu kennen Sie bereits.

$$125000 / 32.7031957 = 3822.25643$$

Der Teiler 3822 also erzeugt das 'C' der untersten Oktave. Auf diese Weise läßt sich wieder ein Programm zum Spielen der Tonleiter erzeugen:

```
10 INPUT "Oktave (-3 bis 4)";oktave
20 FOR ton=1 TO 12
30 frequenz=440*(2^(oktave+(ton-10)/12))
40 teiler=INT(125000/frequenz)
50 SOUND 1,teiler,20,7,0,0,0
60 NEXT ton
```

Es wird immer interessanter, mit dem Tongenerator zu experimentieren. Nun werden wir diesem Kapitel die Krone aufsetzen. Das folgende Programm "KLAVIER FÜR EINSTEIGER" verwandelt Ihren CPC in ein Musikinstrument.

```
10 MODE 0
20 DIM ton(255)
30 ton$="Q2W3ER5T6Y7UI900P"
40 FOR i=1 TO 17
50   index=ASC(MID$(ton$,i,1))
60   ton(index)=i
70 NEXT i
80 CLS
90 PRINT "      KLAVIER FUER"
100 PRINT "     EINSTEIGER"
110 LOCATE 1,5
120 PRINT " 2 3   5 6 7   9 0"
130 PRINT
140 PRINT "Q W E R T Y U I O P"
150 a$=INKEY$:IF a$="" THEN 150
160   IF ton(asc(a$))=0 THEN 150
170   ton=ton(asc(a$))
180   frequenz=440*(2^(ton/12))
190   teiler=INT(125000/frequenz)
200   SOUND 1,teiler,30,7,0,0,0
210 GOTO 150
```

Ein kleines Programm mit großer Wirkung! Vielleicht nehmen Sie noch einige Verbesserungen vor, nachdem Sie die Arbeitsweise durchschaut haben.

RAUSCHEN

Bei der Erklärung des SOUND-Befehls haben Sie bereits von einem Parameter gehört, mit dem Rauschen erzeugt werden kann. Dieser steht an letzter Stelle und enthält den Wert 1 bis 15, wenn das Rauschen aktiviert werden soll. Mit einem SOUND-Befehl können also sowohl Töne als auch Rauschen erzeugt werden, je nach Wert der Parameter 'b' und 'g'. Die folgende Übersicht soll hier Klarheit schaffen:

SOUND a,b,c,d,e,f,g			
Parameter		Resultat	
'b'	'g'	Ton	Rauschen
0	0	nein	nein
1-4095	0	ja	nein
0	1-15	nein	ja
1-4095	1-15	ja	ja

Sie sehen, daß Töne und Rauschen gleichzeitig erzeugt werden können und zwar mit einem SOUND-Befehl. Das Rauschen kann durch Veränderung des Parameters zwischen 1 (helles Rauschen) und 15 (dumpfes Rauschen) eingestellt werden. Das folgende Programm macht dies deutlich:

```
10 FOR r=1 TO 15
20   SOUND 1,0,20,7,0,0,1
30 NEXT r
```

Das unterschiedliche Rauschen ist unüberhörbar. Der Ton wurde ausgeschaltet, indem der zweite Parameter auf Null gesetzt wurde.

Mit Hilfe des Rauschens können nun verschiedene Effekte erzeugt werden. Geräusche wie z.B. Wind, Schüsse, Explosionen werden mit einem Rauschgenerator erzeugt. Das folgende Programm soll eine Lokomotive nachahmen:

```
10 SOUND 1,0,10,7,0,0,15
20 SOUND 1,0,5,7,0,0,0
30 GOTO 10
```

Diese Lokomotive können Sie mit der Taste 'ESC' anhalten. Dem Rauschen (0.1 Sek.) folgt eine Pause (0.05 Sekunden). Erinnern Sie sich: Die Tondauer wird mit dem dritten Parameter in Einheiten von 0.01 Sekunden bestimmt.

Das letzte Beispiel soll einen Schuss erzeugen. Dazu wird in einer Schleife die Lautstärke vermindert.

```
10 a$=INKEY$:IF a$="" THEN 10
20 FOR v=7 TO 1 STEP -1
30   SOUND 1,0,10,v,0,0,15
40 NEXT v
50 GOTO 10
```

Die Schüsse werden durch einen Tastendruck ausgelöst. Das Programm kann mit der Taste 'ESC' abgebrochen werden. Vielleicht versuchen Sie sich auch einmal als Geräuschemacher. Die dafür erforderlichen BASIC-Befehle sind für Sie nun kein Problem mehr.

KAPITEL 8:

DAS DISKETTENLAUFWERK

Wie Sie sicher nicht übersehen haben, ist Ihr CPC6128 mit einem eingebauten Diskettenlaufwerk versehen. Dieses Gerät soll Ihre Programme und Daten aufzeichnen. Das einfache Speichern und Laden von Programmen haben Sie bereits gelernt. Im Adressenprogramm haben Sie auch schon Daten auf dem Diskettenlaufwerk gesichert. Dies alles soll in diesem Kapitel nicht mehr behandelt werden. Andere besondere Fähigkeiten werden hier erläutert und mit Beispielen demonstriert.

DIE DISKETTEN

Die von SCHNEIDER gelieferten Diskettenlaufwerke unterscheiden sich in einem wesentlichen Punkt von den herkömmlichen Laufwerken, der Diskettengröße. Nach jahrelangem Einsatz von 5 1/4 Zoll-Disketten scheinen die 3 1/2 Zoll-Disketten einen neuen Standard auf dem Computermarkt zu setzen. Computer der neuen Generation (AMIGA, 520ST, MacIntosh) sind mit diesem Diskettenformat ausgerüstet. SCHNEIDER-Laufwerke hingegen haben ein Diskettenformat, das bei keinem anderen Gerät zu finden ist: 3 Zoll-Format.

Die Disketten des CPC können haben einen Vorteil gegenüber dem 3 1/2 Zoll-Disketten; sie können von beiden Seiten bespielt, also umgedreht werden. Somit haben Sie auf einer Diskette 2 mal 180 KByte Speicherkapazität zur Verfügung. Die Diskette ist physikalisch eingeteilt in 40 Spuren. Jede dieser Spuren ist wiederum in 9 Sektoren aufgeteilt. Auf jedem dieser 360 Sektoren können 512 Byte gespeichert werden. Daraus resultiert die Gesamtkapazität von 180 KByte. Doch keine Angst, diese physikalische Struktur hat für Sie keine Bedeutung. Das Betriebssystem, genauer das DOS (Disk-Operating-System) nimmt Ihnen die Arbeit ab und organisiert den Diskettenaufbau.

DAS FORMATIEREN

Bevor Sie die Diskette als Datenspeicher einsetzen können, müssen sie formatiert werden. Beim Formatieren werden die Sektoren auf der Diskette zur Aufnahme der Daten angelegt. Zunächst eine exakte Vorgehensweise zum Formatieren:

1. **Legen Sie die Seite A der mitgelieferten Systemdiskette (CP/M) in das Laufwerk.**
2. **Rufen Sie das CP/M-Betriebssystem mit dem Befehl |CPM auf (Das Zeichen | erreichen Sie mit SHIFT und @).**
3. **Laden Sie das Programm zum Formatieren von Diskette durch Eingabe von DISCKIT3.**
4. **Drücken Sie die Funktionstaste f7 zur Wahl der Formatierung.**
5. **Drücken Sie die Funktionstaste f6 zur Wahl des Datenformates.**
6. **Legen Sie die zu Formatierende Diskette ein.**
7. **Drücken Sie die Y-Taste.**
8. **Drücken Sie nach Beendigung des Formatierens irgendeine Taste.**
9. **Legen Sie wieder die Systemdiskette ein.**
10. **Verlassen Sie das Hilfsprogramm mit Funktionstaste f0.**
11. **Versetzen Sie den Rechner in den Einschaltzustand (SHIFT-CONTROL-ESC)**

Dies ist eine ziemlich umfangreiche Prozedur, die jedoch unerlässlich ist. Nach erfolgreichem Ablauf besitzen Sie eine formatierte Diskettenseite, die Sie zur Datenspeicherung nutzen können.

DATEINAMEN

Bei der Vergabe von Namen für Programme und Dateien auf der Diskette müssen Sie folgendes beachten:

Programmnamen dürfen nur 8 Zeichen umfassen.

Weitere drei Zeichen stehen zur Verfügung, wenn Sie den Dateinamen von diesen drei Zeichen mit einem Punkt trennen.

Normal abgespeicherte BASIC-Programme erhalten automatisch den Zusatz **.BAS**, wenn Sie keinen eigenen Namenszusatz angeben. Ein Beispiel:

SAVE "TEST"	Das Programm wird unter dem Namen TEST.BAS auf der Diskette geführt.
SAVE "TEST.DEM"	Das Programm wird unter dem Namen TEST.DEM auf der Diskette geführt.

DAS INHALTSVERZEICHNIS

Geben Sie einige BASIC-Zeilen ein und speichern Sie dies als Programm mit dem Namen **TESTCAT**.

SAVE "TESTCAT"

Das Inhaltsverzeichnis der Diskette wird bei Bedarf mit dem Befehl **CAT** auf dem Bildschirm ausgegeben.

CAT

Geben Sie diesen Befehl nun ein, so zeigt sich das Verzeichnis wie folgt:

```
Drive A: user 0

TESTCAT .BAS 1K

177K free
```

Sie erkennen, daß Ihr Programm nicht größer als 1 KByte ist und daß Sie noch Platz für 177 KByte auf der Diskette haben.

Zum Laden des Programms geben Sie den folgenden Befehl ein:

```
LOAD "TESTCAT"
      oder
LOAD "TESTCAT.BAS"
```

Im folgenden werden weitere Möglichkeiten zur Programmverwaltung beschrieben.

PROGRAMME ALS ASCII-DATEIEN

Im bisherigen Verlauf des Buches tauchten die Begriffe "Datei" und "Programm" auf, die stets auseinandergehalten werden müssen. Wo liegt nun der Unterschied? Programme liegen sowohl im Speicher als auch auf der Diskette oder anderen Speichermedien in komprimierter Form vor. So werden z.B. Befehle codiert abgelegt, da sonst z.B. der Befehl "PRINT" fünf Speicherstellen belegen würde. Um dies zu vermeiden, besitzt jeder Befehl einen Code (0-255), der nur eine Speicherstelle belegt. Dies ist z.B. mit den Zeichen des CPC zu vergleichen, die intern ebenfalls codiert werden ('ASC' und 'CHR\$').

Der CPC bietet eine Möglichkeit an, Programme in "ausgeschriebener" Form, also in Klartext auf die Diskette zu speichern. Zu diesem Befehl folgt nun zunächst die Beschreibung:

- Problem:** Speichern von Programmen in Klartext
- Befehl:** SAVE "name",A
- Parameter:** name- Programmname
- Beispiel:** SAVE "Adressen",A
Speichert das Programm mit dem Namen "Adressen" in Klartext
- Bemerkung:** Das abgespeicherte Programm kann als Datei, eingelezen werden.
nicht jedoch als lauffähiges Programm

Sicher haben Sie keine Vorstellung davon, wozu diese Speicher- methode genutzt werden kann. Erst beim Einsatz einer Textver- arbeitung werden Sie diesen Vorteil zu schätzen wissen. Als ASCII-Datei abgespeicherte Programme können von einer Textverarbeitung eingelezen und verarbeitet werden. Interessant ist dies, wenn Programmlistings in Texte übernommen werden sollen, wie es auch in diesem Buch praktiziert wurde.

Mit dem folgenden Beispiel können Sie sich davon überzeugen, daß diese Programme als Datei eingelezen werden können. Geben Sie dazu das folgende Demonstrationsprogramm ein:

```
10 REM =====  
20 REM PROGRAMM ALS DATEI  
30 REM =====  
40 PRINT "Das Programm ist beendet"
```

Speichern Sie dieses Programm mit folgendem Befehl auf eine Diskette:

```
SAVE "DEMO.ASC",A
```

Nun wollen wir dieses Programm wie eine Datei einlesen und auf dem Bildschirm anzeigen. Dazu ist das folgende Programm erforderlich:

```

10 OPENIN "DEMO.ASC"
20 FOR i=1 TO 4
20 INPUT #9,a$
30 PRINT a$
40 NEXT i
50 CLOSEOUT

```

Wenn Sie dieses Programm eingeben und starten, werden Sie verblüfft feststellen, daß das zuvor gespeicherte Demo-Programm nun auf dem Bildschirm angezeigt wird.

GESCHÜTZTE PROGRAMME

Vielleicht haben Sie schon einmal versucht, handelsübliche Programme in den Rechner zu laden und anschließend aufzulisten. Wenn Ihnen das nicht gelungen ist, so liegt das daran, daß diese Programme geschützt wurden. Auch Sie können Ihre Programme auf Wunsch derartig schützen. Die folgende Beschreibung des entsprechenden Befehls zeigt Ihnen wie:

Problem: Schützen von Diskettenprogrammen

Befehl: SAVE "name",P

Parameter: name - Programmname

Beispiel: SAVE "secret",P
Speichert das Programm "secret" so auf die Diskette, daß es nicht mehr aufgelistet werden kann.

Bemerkung: Es muß eine Version des geschützten Programms normal gespeichert werden, da der Schutz nicht mehr rückgängig gemacht werden kann.

Diese Beschreibung sagt fast alles aus, was Sie zum Speichern von geschützten Programmen benötigen. Lassen Sie uns diesen Befehl

gleich ausprobieren. Geben Sie dazu die folgenden BASIC-Zeilen ein:

```
10 REM =====
20 REM    secret
30 REM =====
40 PRINT "Geben Sie den Code ein:";
50 INPUT a$
60 IF a$<>"xy12" THEN NEW
70 PRINT "o.k."
```

Dieses Programm speichern Sie mit dem Befehl

SAVE "secret.p",P

Nun laden Sie das Programm wieder mit dem Befehl

LOAD "secret.p"

Starten Sie das Programm mit 'RUN'. Was, das Programm startet nicht? Keine Angst, hier ist nichts schief gelaufen, denn geschützte Programme können nicht mit 'LOAD' und 'RUN', sondern nur mit 'RUN"name"' gestartet werden. Geben Sie also den korrekten Befehl ein:

RUN "secret.p"

Dies ist eine Möglichkeit des Ladens eines Programm mit gleichzeitigem Starten.

Nachdem das Programm geladen wurde, wird es automatisch gestartet. Kann der Code vielleicht doch geknackt werden? Vielleicht versuchen Sie, das Programm mit der Taste 'ESC' abzubrechen? Drücken Sie diese Taste dazu zweimal. Siehe da, das Programm wurde tatsächlich abgebrochen. Doch freuen Sie sich nicht zu früh, denn wenn Sie nun den Befehl 'LIST' eingeben, geschieht nichts, da kein Programm mehr im Rechner ist. Es wurde beim Unterbrechen mit 'ESC' gelöscht. Es gibt also keinen Weg, dieses Programm zu knacken. Dies soll nicht bedeuten, daß erfahrene Kenner des CPC ebenfalls erfolglos sein

werden. Mit den entsprechenden Maschinenkenntnissen läßt sich jeder noch so gute Code knacken. Fassen wir zum Abschluß noch einmal zusammen:

Der Zusatz ',P' hinter dem Befehl SAVE schützt das Programm gegen unbefugtes Listen.

Das Programm kann nur mit 'RUN"name"' geladen und gestartet werden.

Wird das Programm mit 'ESC' gestoppt, so wird der Speicher gelöscht.

Der Schutz kann nicht mehr rückgängig gemacht werden.

SICHERN VON SPEICHERAUSZÜGEN

Der gesamte Speicher des CPC beträgt 2 mal 64 KByte, also 2 mal 65535 Bytes (Zeichen). Sollen einzelne Bereiche dieses Speichers abgespeichert werden, so ist folgender Befehl erforderlich:

Problem: Speichern von Speicherausügen

Befehl: SAVE "name",B,von,bis,start

Parameter:	<i>name</i> -	Programmname
	<i>von</i> -	Anfangsadresse (0-65535)
	<i>bis</i> -	Endadresse
	<i>start</i> -	Startadresse (bei Maschinenprogrammen)

Beispiel: SAVE "routine.BIN",49152,65535
Speichert den gesamten Bildschirminhalt

Bemerkung: Der Speicherausschnitt muß mit dem Befehl RUN"name" geladen werden.

Dieser Befehl ist vor allem zur Speicherung von Maschinenprogrammen interessant. Dies ist jedoch erfahrenen Programmierern vorbehalten, die die Geschwindigkeit des BASIC nicht zufriedenstellt.

Weiterhin kann der Befehl auch zur Speicherung von erzeugten Grafiken benutzt werden. Dazu wird der gesamte Bereich der Bildschirmspeichers (49152 bis 65535) auf einer Diskette abgelegt. Interessant wird die Abspeicherung des Grafikbildschirms, wenn eine mathematisch aufwendige Funktionsdarstellung auf Diskette festgehalten werden soll, deren Ausführung mehr Zeit als die Speicherung benötigt. Dies ist vor allem bei dreidimensionalen Funktionen der Fall.

PROGRAMMVERKÜPFUNG

Jeder Programmierer entwickelt im Laufe der Zeit ausgeklügelte Unterprogramme, die in vielen Programmen einzusetzen sind. Eine umfangreiche Unterprogramm - Bibliothek verkürzt die Neuentwicklung von Programmen erheblich. Wie aber werden die Unterprogramme zu einem Hauptprogramm verknüpft? Der umfangreiche Sprachschatz des CPC6128 bietet auch hier eine optimale Lösung an:

Problem: Verknüpfen von Programmen

Befehl: MERGE "name"

Parameter: name - Programmname

Beispiel: MERGE "UPRO1"
Lädt das Programm "UPRO1.BAS" zu dem im Speicher bereits befindlichen Programm

Bemerkung: Bei gleichen Zeilennummern werden die Zeilen des
speicherresidenten Programms gelöscht.

Die Funktion dieses Befehls wird an folgender Übersicht deutlich:

Progr. im Speicher	Progr. auf Disk.	Ergebnis
10 REM P1	5 REM P2	5 REM P2
20 REM P1	15 REM P2	10 REM P1
		15 REM P2
		20 REM P1
10 REM P1	5 REM P2	5 REM P2
20 REM P1	10 REM P2	10 REM P2
		20 REM P1

Hier wird deutlich, daß die Zeilen beider Programme so sortiert werden, daß das resultierende Programm den Vorschriften der Zeilennummerierung entspricht. Konkurrieren jedoch zwei Zeilennummern, so wird die Zeilennummer des im Speicher befindlichen Programms gelöscht. Dies ist auch verständlich, denn jede Zeilennummer darf nur einmal im Programm enthalten sein.

Sollten die Zeilennummern der beiden zu verknüpfenden Programme den Voraussetzungen für eine Verknüpfung nicht entsprechen, so müssen sie vorher angepasst werden. Dazu kann man z.B. den bereits behandelten Befehl **RENUMBER** verwenden.

Abschließend nun ein Beispiel zur Anwendung des Befehls **MERGE**. Geben Sie zunächst das nachzuladende Programm ein:

```
10 REM Programm 2
25 REM Programm 2
30 REM Programm 2
45 REM Programm 2
```

Dieses Programm soll später nachgeladen werden und wird deshalb mit dem Befehl

SAVE "PROG2"

auf eine Diskette abgespeichert. Nun löschen wir den Speicher (NEW) und geben das Programm ein, zu dem das vorherige geladen werden soll:

```
10 REM Programm 1
20 REM Programm 1
30 REM Programm 1
40 REM Programm 1
50 REM Programm 1
```

Nun werden beide Programme verknüpft, indem den Befehl

MERGE "PROG2"

eingeben. Der Rechner verhält sich nun wie beim normalen Laden des Programms. Der anschließende Befehl 'LIST' jedoch wird Ihnen verdeutlichen, was tatsächlich geschehen ist:

```
10 REM Programm 2
20 REM Programm 1
25 REM Programm 2
30 REM Programm 2
40 REM Programm 1
45 REM Programm 2
50 REM Programm 1
```

Die beiden konkurrierenden Zeilen 10 und 30 wurden durch die Zeilen des nachgeladenen Programms ersetzt. Alle anderen Zeilen wurden korrekt eingeordnet. Der Befehl MERGE wird Ihnen bei der Erstellung neuer Programme mit alten Unterprogrammen sehr hilfreich sein.

LOESCHEN VON DATEIEN

Nicht mehr benötigte Programme sollten nicht ewig den kostbaren Speicherplatz der Diskette vergeuden. Das DOS des CPC, das sogenannte AMSDOS hält einen Befehl zum Löschen von Dateien auf Diskette bereit. Alle AMSDOS-Befehle beginnen mit dem Zeichen '|'. Dieses Zeichen erreichen Sie, wenn Sie die Taste SHIFT gedrückt halten und die Taste '@' betätigen. Doch nun zum Befehl:

Problem: Löschen von Dateien

Befehl: |ERA,"name"

Parameter: name - Dateiname

Beispiel: |ERA,"secret.p"
löscht das Programm "secret.p"

Probieren Sie den Befehl aus und löschen Sie das MERGE-Programm aus dem letzten Beispiel:

|ERA,"prog2.bas"

Beachten Sie, daß der Zusatz .BAS stets mit angegeben werden muß

UMBENENNEN VON DATEIEN

Sollten Sie einmal den Wunsch verspüren, einer Datei einen anderen Namen zu geben, so ziehen Sie den folgenden AMSDOS-Befehl ein:

Problem: Umbenennen von Dateien

Befehl: |REN,"*nameneu*","*namealt*"

Parameter: *nameneu* - neuer Dateiname
namealt alter Dateiname

Beispiel: |REN,"**DEMO.BAS**","**DEMO01.BAS**"
nennt das Programm DEMO01.BAS um in
DEMO.BAS"

Auch hier muß der Namenszusatz immer mit angegeben werden.

Sicherlich stellt das AMSDOS und vor allem das Betriebssystem CP/M mehr Befehle zur Verfügung, als in diesem Kapitel beschrieben wurden. Zu viel des Guten verwirrt jedoch erfahrungsgemäß den Leser. Aus diesem Grund beschränkt sich dieses Buch auf die wichtigsten Befehle zur Verwaltung von Programmen auf der Diskette.

KAPITEL 9: NOCH MEHR BEFEHLE

Wenn Sie dieses Kapitel des Buches erfolgreich erreicht haben, so werden Sie es sicher nicht erwarten können, weitere Befehle des umfangreichen CPC-BASICs kennenzulernen. Dem soll nichts im Wege stehen. Sie werden nun mit Befehlen konfrontiert, deren Komplexität sich in Grenzen hält. Zu allen Befehlen finden Sie wie gewohnt eine ausführliche Beschreibung sowie erklärende Beispielprogramme.

JOYSTICKABFRAGE

Wenn Sie Ihren CPC ausnahmsweise von hinten betrachten, werden Sie auf der rechten Seite einen Anschluß bemerken, der an fast jedem Homecomputer zu finden ist. Hier werden die sogenannten Joysticks angeschlossen, die hauptsächlich dazu dienen, Spiele optimal steuern zu können.

Da die Abfrage der Stellung eines Joysticks beim CPC recht einfach ist, werden wir den entsprechenden Befehl nun kennenlernen.

Problem: Abfrage des Joysticks

Funktion: JOY (*n*)

Parameter: *n* - Nummer des Joysticks
(0 oder 1)

Beispiel: PRINT JOY(0)
Zeigt den Code der Joystickbetätigung auf dem Bildschirm.

Bemerkung: Es gelten folgende Codes:

5	1	9	
			Feuerknopf 1 =32
4		8	
			Feuerknopf 2 =16
6	2	10	

Sie haben bemerkt, daß zwei Joysticks abgefragt werden können, obwohl nur ein Anschluß vorhanden ist. Es können nur zwei Joysticks angeschlossen werden, wenn es sich um Original - Joysticks von SCHNEIDER oder Joysticks anderer Firmen, die für den CPC bestimmt sind, handelt. Reicht Ihnen jedoch ein Joystick aus, so können Sie jeden handelsüblichen Joystick (z.B. für ATARI, COMMODORE64) verwenden. Dieser wird dann als 'JOY(0)' angesprochen.

Schreiben wir nun zunächst ein Programm, das die Betätigung des Joysticks optisch am Bildschirm zu erkennen gibt.

```
10 PRINT JOY(0)
20 GOTO 10
```

Wenn Sie dieses Programm eingeben und starten, so werden Sie die Codes jeder Betätigung des Joysticks erkennen. Auf diese Werte kann das entsprechende Programm dann reagieren.

Mit der Funktion 'JOY (n)' läßt sich ein eindrucksvolles Programm zum Zeichnen auf dem Bildschirm schreiben. Der Feuerknopf soll zum Löschen des Bildschirms benutzt werden.

```
10 MODE 2
20 CLS
30 x=320:y=200
40 PLOT x,y
50 a=JOY(0)
60 IF a=4 AND x>0 THEN x=x+1:GOTO 40
70 IF a=8 AND x<639 THEN x=x-1:GOTO 40
80 IF a=1 AND y<399 THEN y=y+1:GOTO 40
90 IF a=2 AND y>0 THEN y=y-1:GOTO 40
100 IF a=16 THEN CLS
110 GOTO 40
```

Dieses Programm ist recht einfach gestaltet. Sicher werden Sie keine Probleme haben, die Arbeitsweise zu verstehen und kleine Verbesserungen vorzunehmen. So wäre z.B. eine Erweiterung auf diagonales Zeichnen sehr interessant. Ein Tip: Sie benötigen dazu vier weitere IF-Abfragen, die jeweils beide Variablen ('x' und 'y') auf einen gültigen Wert überprüfen und aktualisieren müssen.

ZUFALLSZAHLEN

Fast jeder BASIC-Anfänger schreibt seine ersten Spiele mit Hilfe von Zufallszahlen. Ohne Zufallszahlen wäre die Programmierung von Glücksspielen unmöglich. Der CPC erzeugt mit einer Funktion Zufallszahlen zwischen 0.000000001 und 0.999999999:

- Problem:** Erzeugen von Zufallszahlen
- Funktion:** RND (1)
- Beispiel:** PRINT RND(1)
Erzeugt eine Zufallszahl und gibt diese auf dem Bildschirm aus.
- Bemerkung:** Die erzeugte Zahl liegt zwischen 0.000000001 und 0.999999999

Ein Befehl, der leicht anzuwenden ist. Schwieriger wird es, wenn die Zahl innerhalb eines anderen Bereichs liegen soll. Dazu muß der Zufallswert aufbereitet werden. Soll z.B. eine Zahl zwischen 0 und 999 ermittelt werden, so muß die Zufallszahl wie im folgenden Programm aufbereitet werden:

```
10 REM *****  
20 REM    WUERFEL  
30 REM *****  
40 zahl=RND(1)  
50 PRINT INT(zahl*1000)
```

Die Zufallszahl wird also mit 1000 multipliziert. Der Bereich der möglichen Zufallszahlen liegt dann zwischen 0.000001 und 999.999999. Da die Nachkommastellen nicht benötigt werden, werden sie mit der Funktion INT "abgeschnitten".

Eine weitere Schwierigkeit ist, wenn die gewünschte Zahl nicht bei Null, sondern bei einer anderen Zahl beginnen soll. Mit folgender Formel kann jeder beliebige Bereich von Zufallszahlen erzeugt werden:

<p>Umrechnen von Zufallszahlen</p> $\text{INT} (\text{RND}(1) * ((B+1)-A))+A$ <p>A - erste gültige Zahl B - letzte gültige Zahl</p>

Sollen beispielsweise Zahlen zwischen 1 und 6 ermittelt werden, so wird die erforderliche Formel wie folgt aufgebaut:

```
ergibt INT(RND(1) * ((6-1)+1))+1  
ergibt INT(RND(1) * (5+1))+1  
ergibt INT(RND(1) * 6)+1
```

Das abschließende Programm soll Lottozahlen (6 aus 49) erzeugen:

```
10 REM *****
20 REM   Lottozahlen
30 REM *****
40 PRINT "Wieviele Reihen ";reihen
50 FOR i1=1 TO reihen
60   FOR i2=1 TO 6
70     PRINT INT(RND(1)*49)+1;" - ";
80   NEXT i2
90 PRINT
100 NEXT i1
```

Auch dieses Programm werden Sie schnell durchschauen. Es hat aber einen Haken: In einer Reihe können zwei oder mehrere gleiche Zahlen auftauchen. Dies zu vermeiden, soll nun Ihre Aufgabe sein. Ein Tip: Speichern Sie alle in einer Reihe bereits gezogenen Zahlen in einem Array (Tabelle), das Sie dann mit der aktuellen Zahl vergleichen. Nur wenn die ermittelte Zahl nicht im Array enthalten ist, wird sie akzeptiert. Vor der nächsten Reihe muß dieses Array aber wieder gelöscht werden. Viel Spaß!

LEFT\$

Die im folgenden behandelten Funktionen zur Verarbeitung von Strings werden Ihnen ganz neue Möglichkeiten eröffnen. So können z.B. einzelne Teile von Strings zur weiteren Verarbeitung ermittelt werden oder Strings in ein anderes String eingesetzt werden. Es lohnt sich sicherlich, diese Befehle einzustudieren. Jede Funktion beginnt zunächst mit der Beschreibung.

Problem: **Linken Teilstring ermitteln**

Funktion: **LEFT\$(string,anz)**

Parameter: *string* - **Stringvariable oder String in
Anführungszeichen**
anz - **Anzahl der Zeichen, die von links
ausgehend übernommen werden
sollen**

Beispiel: **PRINT LEFT\$("ABCDEFGH",3)**
**Gibt die linken drei Zeichen des Strings (ABC)
auf dem Bildschirm aus.**

Bemerkung: **Ist die Anzahl der Zeichen größer als der
String, so wird der gesamte String übernommen**

Diese interessante Funktion kann also Strings von links ausgehend "abschneiden". Das folgende Programm sagt mehr als viele Worte:

```
10 INPUT "Geben Sie Ihren Namen ein: ";name$
20 anz=1
30 teil$=left$(name$,anz)
40 PRINT teil$
50 IF teil$=name$ THEN 80
60 anz=anz+1
70 GOTO 30
80 PRINT "Ende"
```

Da dieser Befehl einfach zu handhaben ist, wollen wir gleich zum nächsten Befehl übergehen:

RIGHTS**Problem:** Rechten Teilstring ermitteln**Funktion:** RIGHTS(*string*,*anz*)**Parameter:** *string* - Stringvariable oder String in
Anführungszeichen
anz - Anzahl der Zeichen, die von rechts
ausgehend übernommen werden
sollen**Beispiel:** PRINT RIGHTS("ABCDEFG",3)
Gibt die rechten drei Zeichen des Strings
(EFG) auf dem Bildschirm aus.**Bemerkung:** Geht die Anzahl der Zeichen über die Länge
des Strings hinaus, so wird der gesamte String
übernommen.

Diese Funktion gleicht der vorher beschriebenen und braucht auch nicht weiter beschrieben werden. Vielleicht ändern Sie das Programm auf der letzten Seite, indem Sie die Funktion 'LEFTS' gegen 'RIGHTS' ersetzen. Der Unterschied der beiden Funktionen wird dann deutlich.

Nun haben wir den linken und den rechten Ausschnitt eines Strings ermittelt. Eine weitere Funktion kann sogar Ausschnitte inmitten eines Strings bilden:

MID\$

Problem: Stringausschnitt ermitteln

Funktion: MID\$(*string*,*pos*,*anz*)

Parameter: *string* - Stringvariable oder String in Anführungszeichen
pos - Position im String, ab der der Ausschnitt beginnen soll
anz - Anzahl der Zeichen, von der Position ausgehend übernommen werden sollen

Beispiel: PRINT MID\$("ABCDEFG",2,4)
Gibt ab der Position zwei vier Zeichen des Strings (BCDE) auf dem Bildschirm aus

Bemerkung: Ist die Anzahl der Zeichen größer als der Rest des Strings, wird der gesamte Rest übernommen.

Nachdem wir bisher Strings von links oder rechts beginnend abgeschnitten haben, bietet diese Funktion nun die Möglichkeit, Zeichenfolgen mitten im String zu ermitteln. Dazu benötigt diese Funktion einen Parameter mehr, als die beiden vorherigen, denn neben der Anzahl der Zeichen muß auch die Anfangsposition übergeben werden.

Diese Funktion wird oft eingesetzt, um die Zeichen eines Strings einzeln auszugeben. So kann z.B. eine Zeichenfolge senkrecht auf dem Bildschirm ausgegeben werden. Das folgende Programm demonstriert dies:

```
10 a$="WAAGERECHT"  
20 FOR p=1 TO 10: REM Laenge des Strings  
30 PRINT MID$(a$,p,1)  
40 NEXT p
```

Die Anfangsposition wird in einer Schleife jeweils um eins nach rechts verschoben. Die Länge des Stringabschnittes ist konstant eins, da nur jeweils ein Zeichen ausgegeben werden soll. Was aber, wenn die Länge des Strings nicht bekannt ist? Auch dieses Problem ist gelöst, nachdem Sie die folgende Funktion kennengelernt haben:

ERMITTELN DER STRINGLÄNGE

Problem: Ermitteln der Stringlänge

Funktion: LEN(*string*)

Parameter: *string* - Stringvariable oder String in Anführungszeichen

Beispiel: A\$="ABCDEFGG":PRINT LEN(A\$)
Gibt die Länge des Strings 'A\$' (7) auf dem Bildschirm aus.

Bemerkung: Ist der String leer, so ergibt die Funktion den Wert 0.

Setzen wir diese Funktion nun in das Programm auf der vorherigen Seite ein, so wird ein beliebiger String senkrecht ausgegeben:

```
10 INPUT "String: ";a$
20 FOR p=1 TO LEN(a$)
30   PRINT MID$(a$,p,1)
40 NEXT p
```

Diese Lösung ist eleganter als die letzte, da die Länge des Strings nun nicht bekannt sein muß.

Nachdem Sie die wichtigsten Stringfunktionen kennengelernt haben, folgt nun ein kleines, eindrucksvolles Programm:

```
10 REM *****
20 REM  Laufschrift
30 REM *****
40 CLS
50 INPUT "Text:      ";text$
60 INPUT "Laenge:   ";laenge
70 INPUT "Tempo (1-7):";tempo
80 l=LEN(text$)
90 text$=STRING$(l,"-")+text$+STRING$(l,"-")
100 FOR i=1 TO l
110  LOCATE 12,20
120  PRINT MID$(text$,i,laenge)
130  FOR wait=1 TO tempo*50:NEXT wait
140 NEXT i
150 GOTO 100
```

Der Text, der in Laufschrift gezeigt werden soll, wird in die Variable 'text\$' eingelesen. Danach wird die Länge des Ausschnittes der Laufschrift in 'laenge' festgehalten. Nachdem ein Tempofaktor in 'tempo' gespeichert wird, erhält die Variable 'l' die Länge des Textes. Damit der Text mit führenden und abschließenden Zeichen getrennt wird, ist die Zeile 90 eingefügt worden. Hier wird die bereits in der Adressenverwaltung benutzte Funktion 'STRING\$' zur Erzeugung von Ketten eines Zeichens eingesetzt. Die folgende Schleife läuft bis zur Länge des eingegebenen Textes und gibt jeweils den Teilstring aus, der sich aus der Position und der anfangs bestimmten Länge ergibt. Die Warteschleife in Zeile 130 läuft je nach Wert des Tempofaktors zwischen 50 und 350. Diese Schleife wird immer wieder aufgerufen, bis das Programm mit der Taste 'ESC' abgebrochen wird.

INSTR

Oft ist es erforderlich, einen String nach einem bestimmten Inhalt zu durchsuchen, vor allem bei der Datenverwaltung. Es folgt zunächst die Beschreibung der Funktion, die dies ermöglicht:

Problem: Suchen eines Strings in einem anderen

Funktion: `INSTR(pos,string1,string2)`

Parameter:

- `pos` - Position, ab der gesucht werden soll
- `string1` - String, in dem gesucht werden soll
- `string2` - String, nach dem gesucht werden soll

Beispiel: `PRINT INSTR (3,"ABCDEFGHIJ","FGH")`
Durchsucht den zuerst genannten String ab der Position 2 nach dem zuletzt genannten String und gibt dessen Position (6) auf dem Bildschirm aus.

Bemerkung: Wird der gesuchte String nicht gefunden, so wird der Wert 0 ausgegeben. Die Angabe der Position kann entfallen, wenn der gesamte String durchsucht werden soll.

Diese komfortable Funktion unterstützt das Suchen gewisser Daten in einer Datei. Bisher konnten wir nur Strings an genau bestimmten Positionen überprüfen, mit 'INSTR' jedoch wird der gesamte String nach einer Zeichenfolge durchsucht. Ein Programm soll das verdeutlichen:


```
5 CLS
10 INPUT "TEXT:          ";text$
20 INPUT "SUCHBEGRIFF: ";such$
30 p=INSTR(text$,such$)
40 IF p=0 THEN 70
50 PRINT "Der Suchbegriff befindet sich an der";p;". Stelle im Text"
60 GOTO 80
70 PRINT "Suchbegriff nicht gefunden"
80 PRINT "Suche beendet"
```

Hier geben Sie zunächst einen Text und anschließend einen Suchbegriff ein. Die INSTR-Funktion in Zeile 30 speichert das Ergebnis der Suche in 'pos'. Hier wurde keine Anfangsposition eingesetzt, da der gesamte String 'text\$' durchsucht werden soll. Nur wenn das Ergebnis der Suche, also die Variable 'pos' ungleich Null ist, ist der Suchbegriff gefunden worden. Die Position, an der die Suche erfolgreich war, wird dann ausgegeben.

Abschließend nun eine erweiterte Form des Programms, die es ermöglicht, den gefundenen Suchbegriff durch einen neuen Begriff zu ersetzen:

```
5 CLS
10 INPUT "TEXT:          ";text$
20 INPUT "suchen nach:  ";such$
30 INPUT "ersetzen durch:";ers$
40 p=INSTR(text$,such$)
50 IF p=0 THEN 90
60 text$=LEFT$(text$,p-1)+ers$+MID$(text$,p+LEN(such$),
LEN(text$)-len(such$)-(p-1))
70 PRINT "Neuer Text:   ";text$
80 GOTO 100
90 PRINT "Suchbegriff nicht gefunden!"
100 PRINT "Suche beendet"
```

Die Zeile 60 wird Sie vielleicht verwirren, da hier eine Menge Stringfunktionen untergebracht wurden. Dies ist auch erforderlich, da der neue String aus folgenden Teilen zusammgebaut wird:

1. **Der Teil bis vor der Position des gefunden Datensatzes (p-1)**
2. **Der neue Begriff ('ers\$')**
3. **Der Rest, angefangen bei der Position hinter dem Suchbegriff (p+LEN(such\$)) bis zum Ende des Textes (LEN(text\$)-len(such\$)-(p-1))**

Eine derartige Routine zum Suchen und Ersetzen finden Sie in fast jeder Textverarbeitung. Mit dem CPC kann sie auf einfachstem Weg realisiert werden.

ANHANG 1:

RESERVIERTE WORTE

Die Namen von Variablen sind zwar frei wählbar, jedoch nur mit Einschränkungen. Alle Worte, die im Sprachschatz des BASIC enthalten sind, dürfen nicht verwendet werden. Wird dies nicht beachtet, so erinnert der CPC Sie mit einem "Syntax error" an das, was Sie stets beachten sollten. Sollten Sie diese Fehlermeldung in einer BASIC-Zeile erscheinen, wo Sie es sich ganz und gar nicht erklären können, so achten Sie zuerst auf die Variablenbezeichnungen. Oft sind Worte aus der folgenden Liste als Variablennamen benutzt worden, was derartige Folgen hat.

ABS	DATA	ERROR
AFTER	DEF	EVERY
AND	DEFINT	EXP
ASC	DEFREAL	
ATN	DEFSTR	FIX
AUTO	DEG	FN
	DELETE	FOR
BIN\$	DEG	FRE
BORDER	DELETE	
	DI	GOSUB
CALL	DIM	GOTO
CAT	DRAW	
CHAIN	DRAWR	HEX\$
CHR\$		HIMEM
CINT	EDIT	
CLEAR	EI	IF
CLG	ELSE	INK
CLOSEIN	END	INKEY
CLOSEOUT	ENT	INKEY\$
CLS	ENV	INP
CONT	EOF	INPUT
COS	ERASE	INSTR

CREAL	ERL	INT
	ERR	
JOY	OUT	STR\$
	PAPER	STRING\$
KEY	PEEK	SWAP
	PEN	SYMBOL
LEFT\$	PI	
LEN	PLOT	TAB
LET	PLOTR	TAG
LINE	POKE	TAGOFF
LIST	POS	TAN
LOAD	PRINT	TEST
LOCATE		TESTR
LOG	RAD	THEN
LOG10	RANDOMIZE	TIME
LOWERS\$	READ	TO
	RELEASE	TROFF
MAX	REM	TRON
MEMORY	REMAIN	
MERGE	RENUM	UNT
MID\$	RESTORE	UPPERS\$
MIN	RESUME	USING
MOD	RETURN	
MODE	RIGHT\$	VAL
MOVE	RND	VPOS
MOVER	ROUND	
	RUN	WAIT
		WEND
NEXT	SAVE	WHILE
NEW	SGN	WIDTH
NOT	SIN	WINDOW
	SOUND	WRITE
ON	SPACES\$	
ON BREAK	SPC	XOR
ON ERROR GOTO	SPEED	XPOS
ON SQ	SQ	
OPENIN	SQR	YPOS
OPENOUT	STEP	
OR	STOP	ZONE
ORIGIN		

ANHANG 2:

FEHLERMELDUNGEN

So seltsam es auch klingt: Fehlermeldungen sind für einen Computer genauso wichtig wie die Befehle. Je mehr verschiedene Fehlermeldungen ein Rechner vorweisen kann, desto einfacher ist es für den Anwender, seine Fehler zu lokalisieren. Stellen Sie sich vor, Ihr Rechner würde auf jeden Fehler mit "Syntax error" reagieren! Sicher würden Sie dann Ihre Fehler nur sehr mühsam feststellen. Da der CPC sehr viele Fehlermeldungen ausgeben kann, werden diese nun beschrieben. Hier können Sie immer dann hereinschauen, wenn eine Ihnen noch nicht bekannte Fehlermeldung auftritt.

Die Nummer vor der jeweiligen Fehlermeldung wird für das Abfangen des Fehlers innerhalb eines Programms (ON ERROR GOTO) benötigt. Dieser Befehl jedoch wurde in diesem Buch nicht behandelt, sodaß Sie diese Nummer erst später zu nutzen wissen werden.

1 Unexpected NEXT

Es ist ein NEXT im Programm enthalten, dem keine FOR-Schleife vorangegangen ist, oder die Schleifenvariable beim NEXT stimmt nicht mit der beim FOR-Befehl überein.

2 Syntax error

Diese Fehlermeldung dürfte Ihnen nicht unbekannt sein. Sie wird immer dann gegeben, wenn der CPC die von Ihnen eingegebene Zeile nicht versteht.

3 Unexpected RETURN

Der Programmablauf wurde an einem RETURN beendet, da kein GOSUB-Befehl vorangegangen ist.

4 DATA exhausted

Es sollen mehr Daten mit READ eingelesen werden, als in DATA-Zeilen angegeben wurden.

5 Improper argument

Dies ist eine allgemeine Fehlermeldung, die immer dann auftritt, wenn der Wert einer Funktion oder ein Parameter ungültig ist.

6 Overflow

Das Ergebnis einer Berechnung liegt über der größten speicherbaren Zahl oder es die Zuweisung für eine Integervariable ist nicht als 16-Bit Integerzahl zu codieren.

7 Memory full

Das Programm oder die Variablen haben eine Größe angenommen, die über den zur Verfügung stehenden Speicherplatz hinausgeht. Der MEMORY-Befehl verursacht diese Fehlermeldung, wenn versucht wird, den BASIC-Anfang an einen ungültigen Bereich zu legen.

8 Line does not exist

Eine angegebene Zeilennummer ist nicht im Programm enthalten.

9 Subscript out of range

Der Index eines Arrays ist zu klein, oder größer als mit DIM reserviert.

10 Array already dimensioned

Ein Array in der DIM-Anweisung wurde bereits dimensioniert.

11 Division by zero

Es wurde eine mathematisch nicht erlaubte Division durch Null ausgeführt.

12 Invalid direct command

Es wurde eine Anweisung im Direkt-Modus eingegeben, die nur in Programmen auftreten darf (z.B. INPUT).

13 Type mismatch

Ein numerischer Wert wurde angegeben, wo ein String erwartet wird, oder umgekehrt.

14 String space full

Es existiert kein weiterer Speicherplatz zum Einrichten von Strings.

15 String too long

Die Grenze eines Strings von 255 Zeichen wurde z.B. durch Stringaddition überschritten.

16 String expression too long

Zur Ermittlung von umfangreichen Stringausdrücken werden Strings zwischengespeichert. Wenn dieser Speicher überschritten wird, erscheint diese Fehlermeldung.

17 Cannot CONTINUE

Das Programm kann mit dem Befehl CONT nicht fortgesetzt werden.

18 Unknown user function

Eine Funktion (FN) wurde nicht mit DEF FN bestimmt.

19 RESUME missing

Das Programm wurde in einer ON ERROR GOTO Routine abgebrochen.

20 Unexpected RESUME

Ein RESUME kann nicht ausgeführt werden, da keine ON ERROR GOTO Routine angesprochen wurde.

21 Direct command found

Beim Laden eines Programms von der Kassette wurde eine Zeile ohne Zeilennummer gefunden.

22 Operand missing

Ein erforderlicher Parameter wurde nicht angegeben.

23 Line too long

Eine BASIC-Zeile überschreitet die maximale Größe (255 Zeichen).

24 EOF met

Es wurde versucht, eine Datei über das Ende hinaus zu lesen.

25 File type error

Der Lesezugriff auf ein Programm oder eine Datei ist falsch. So kann z.B. keine Datei mit LOAD geladen werden.

26 NEXT missing

Es wurde eine FOR-Schleife ohne entsprechendes NEXT programmiert.

27 File already open

Eine Datei, die noch nicht geschlossen ist, wurde versucht zu öffnen.

28 Unknown command

Ein Befehl wurde eingegeben, der nicht im Befehlssatz enthalten ist.

29 WEND missing

Eine WHILE-Schleife wurde ohne entsprechendes WEND eingesetzt.

30 Unexpected WEND

Das Programm trifft auf einen WEND, dem keine entsprechende WHILE-Schleife vorangegangen ist.

ANHANG 3: STICHWORTREGISTER

:	57
;	54
,	55
?	51
\$	93
Addition.....	43
Adressenverwaltung	89
AMSTRAD.....	7
AND - Verknüpfung.....	117
Arrays	96
ASC.....	118
ASCII - Code	6, 87, 118, 178
ASCII - Dateien.....	178
Auflösung, Monitor-	28
AUTO.....	74
Auto-Repeat.....	10
BASIC	
Definition	40, 62
speicher.....	7
bedingte Verzweigung (IF).....	115
berechneter Sprung	120
Betriebssystem.....	7
Bildschirmeditor.....	30
Blinkgeschwindigkeit	152
BORDER	150
BREAK	72
Buchstaben.....	17
CAPS LOCK	20
CAT	183
CHR\$.....	87, 118
CLOSEOUT.....	144
CLR	25

CLS	15
CONT	72
COPY-Taste	29, 68
CTRL.....	8, 73
Cursor	
Grafik-	163
Kopier-.....	29
Positionierung.....	110
Übungen	8
Cursortasten.....	9
Datei	90
Datenfelder	91
Datensatz	91
DEL	21
DELETE	84
Dezimalpunkt	46
DIM.....	97
Dateneingabe	101
Direkt-Modus.....	41, 63
Diskettenlaufwerk	90
Division	43
DRAW	163
DRAWR.....	166
Drei-Finger-Griff.....	8
Druckausgabe	137
EDIT	68
Editor.....	12, 29, 68
Einfügemodus.....	14
Einschaltmeldung	7
ENTER-Taste	13, 39
ESC	8, 67, 72
Exponent.....	47, 52
Farben	
Anzahl.....	148
Auswahl	154
Code.....	149
Hintergrund-	158

Rahmen-.....	150
Standard-.....	155
Zeichen-.....	153
Farbmonitor.....	28
Fehlermeldung.....	117
Felder.....	91
Fließkommazahlen	47
FOR	103
Funktionstasten	85
Gerade zeichnen.....	163
Gleitkommazahlen.....	47
GOSUB	111
GOTO	71
Grafik	
Auflösung	148
-cursor	164
hochauflösende.....	160
Gross/Kleinschrift	19
Grünmonitor.....	28
Grundrechenarten	42
Hauptspeicher.....	7
IF.....	115
Index.....	97
INK.....	154
INKEY\$.....	128
INPUT	101
INPUT #	145
INSTR.....	197
JOY	188
Joystickabfrage.....	188
KEY.....	85
Klammerrechnung.....	50
Kopiercursor	29, 68

Kreis	
Berechnung.....	53, 101
zeichnen.....	162
laden	
Programm.....	71
Datei.....	144
Leertaste.....	20, 44, 138
LEFT\$.....	192
LEN.....	196
Linien zeichnen.....	163
LIST.....	67
LOAD.....	74
LOCATE.....	110
LOCOMOTIVE SOFTWARE.....	7
Löschen, Bildschirm.....	15
MENUE.....	113
MERGE.....	186
MID\$.....	195
MODE.....	26, 148
MOVE.....	164
MOVER.....	166
Multiplikation.....	43
NEW.....	83
NEXT.....	105
ON GOTO.....	120
OPENOUT.....	143
OR.....	117
PAPER #.....	158
Parameter.....	42
PEN #.....	153
PI.....	51
PLOT.....	161
PLOTR.....	166
Potenzierung.....	51

PRINT	
Abkürzung.....	51
Übungen	41
PRINT #.....	137
Programm	
Änderung.....	67
Definition	62
Fortsetzung nach Unterbrechung.....	72
Löschen.....	75
Name.....	73
Numerierung	63, 77, 79
Schrittweite.....	64
Schutz	180
Start.....	66
Unterbrechung	72
Verknüpfung	186
Zeilen löschen	83
Programmierhilfen	77
Punkt zeichnen.....	161
Rauschen	174
READY-Meldung.....	7
rechnen	42
REM	109
RENUM	81
RETURN.....	111
RIGHT\$.....	194
RND.....	190
Routinen	111
RUN	66, 75
SAVE	71
Schleifen	103
Schutz von Programmen	180
scrollen.....	67
sequentielle Speicherung.....	90
SHIFT	8, 19
SOUND.....	169
SPEED INK.....	152
Speicherauszüge sichern	182

speichern	
Programm	73
Daten	142
STEP	104
STRING\$	109
Strings	
Definition	49
Funktionen	192
Variablen	93
Subtraktion	43
Syntax error	15, 40
Synthesizer	168
Tabellen	96
Tastatur	5
Abfrage	128
Tasten	5
Töne	
Eigenschaften	169
Erzeugung	168
Höhe	170
Unterprogramme	111
Variablen	49, 93
Vergleiche	115
Vergleichsoperatoren	116
Verzweigung	71
Warteschleife	106
Zeichendichte, einstellbare	26
Zeilennummer	43, 63
automatische Numerierung	77
Umnummerierung	79
Ziffernblock	85
Zufallszahlen	190

Deutsche CPC Software aus bester Hand

DATAMAT

Deutschlands meistgekaufte Dateiverwaltung

bietet einiges, was in dieser Preisklasse bisher unvorstellbar schien:

- menügesteuertes Diskettenprogramm, dadurch extrem einfach zu bedienen
- für jede Art von Daten
- völlig frei gestaltbare Eingabemaske
- 80 Zeichen pro Zeile
- Hardcopy
- 50 Felder pro Datensatz
- 512 Zeichen pro Datensatz
- bis zu 4000 Datensätze pro Datei je nach Umfang
- 27 Farben für Rand, Hintergrund und Buchstaben
- Schnittstelle zu TEXTOMAT
- Benutzung von Rechenfeldern
- Anzeige des Disketteninhaltes
- läuft mit ein oder zwei Floppys
- komplett in Maschinensprache, dadurch extrem schnell
- deutscher/amerikanischer Zeichensatz
- fast jeder Drucker ist anschließbar
- duplizieren der Datendiskette
- gute Benutzerführung
- Hauptprogramm komplett im Speicher – kein lästiges Nachladen
- deutsches Handbuch mit Übungslexikon
Sie können:
 - jeden Datensatz in wenigen Sekunden suchen
 - nach beliebigen Feldern selektieren
 - nach allen Feldern, auf-oder absteigend sortieren
 - Listen in völlig freiem Format drucken
 - Etiketten drucken
 - Komplett nur DM 148,-*
Für CPC 464, 664 und 6128
Die richtige Version wird automatisch geladen

TEXTOMAT

Deutschlands meistgekaufte Textverarbeitung

bietet Profilleistung zum Hobbypreis! TEXTOMAT in Stichworten:

- Diskettenprogramm durchgehend menügesteuert
- deutscher/amerikanischer Zeichensatz
- Rechenfunktionen für alle Grundrechenarten
- über 17000 Zeichen pro Text im Speicher
- beliebig lange Texte durch Verknüpfung
- 80 Zeichen pro Zeile
- läuft mit ein oder zwei Floppys
- 27 Farben für Rahmen-Hintergrund-Bildschirmfarbe
- es können Trennvorschläge gemacht werden
- Wordwrap
- Tabulatoren
- Seitennumerierung
- Proportionalischrift auf entsprechendem Drucker
- Zuweisungstabelle für ASCII-Code
- frei definierbare Steuerzeichen, z. B. für Indices, Schriftarten, Unterstreichen, Formate
- umfangreiche Formularanpassungen

TEXTOMAT PLUS

neues Textverarbeitungsprogramm der Superlative

Erheblich erweiterte, leistungsstärkere TEXTOMAT-Version.

Bietet alle Möglichkeiten von TEXTOMAT und **zusätzlich**:

- + ergonomische, schreibmaschinenähnliche Texteingabe arbeitet grundsätzlich im 80 Zeichenmodus
 - + 2 dynamisch verwaltete Textbereiche im Speicher. Zwischen beiden Texten kann beliebig hin- und hergeschaltet sowie kopiert werden. Wahlweise Menüsteuerung oder schnelle Direktwahl der Funktionen. 10 Floskel Tasten für häufig wiederkehrende Worte oder Redewendungen. Sehr komfortable Cursorsteuerung (vor/zurück – Zeichen/Wort/ Satz/Absatz)
 - + Trennvorschläge nach deutscher Grammatik
 - + Kopf- und Fußzeilen während des Textes ändernbar
 - + bedingter Seitenwechsel
 - + BASIC Programme können eingelesen, editiert und abgespeichert werden, dabei automatisch ASCII Um- und Rückwandlung
 - + Suchen und Ersetzen mit vielen Optionen und Joker (vor/rückwärts – Klein/Großschreibung – ganze Wörter)
 - + komplettes Terminalprogramm zum problemlosen Senden und Empfangen von Texten sowohl zum Halb- als auch Voll-duplexbetrieb
- TEXTOMAT PLUS für CPC 6128 kostet DM 198,-*

- Blockoperationen, 'Suchen und Ersetzen'
- Serienbriefformstellung mit DATAMAT
- formatierte Ausgabe auf dem Bildschirm
- Anpassung an fast jeden Drucker
- ausführliches Handbuch mit Übungslektion
- Komplett nur DM 148,-*
Für CPC 464, 664 und 6128
Die richtige Version wird automatisch geladen

Profi-Painter CPC

PROFI PAINTER, ein sensationelles Programm zum Malen, Entwerfen und Zeichnen auf CPC Computern. Den berühmten Vorbildern der 32-bit Welt steht PROFIPAIINTER kaum nach und übertrifft diese sogar in manchen Punkten. Im einzelnen:

1. Allgemeine Zeichenfunktionen

DAS LINEAL zeichnet Geraden – DAS RECHTECK stellt Rechtecke dar (lassen sich mit Muster oder Farbe ausfüllen) – DAS AUSGEFÜLLTE RECHTECK MIT RUNDEN ECKEN – DER KREIS zeichnet auch Ellipsen (können mit Muster oder Farbe ausgefüllt werden) – DAS POLYGON ist auch ausfüllbar – DER FARBEIMER füllt beliebige Flächen mit Muster oder Farbe – DAS RADIERGUMMI.

Die aktuelle Strichstärke der Funktionen ist jederzeit über ein Menü veränderbar. Es stehen vier Strichstärken (Dicke 0–3) zur Verfügung.

2. Freihand-Zeichenfunktionen

DER PINSEL eignet sich zum Malen auf dem Bildschirm. Verschiedene Pinselformen stehen diesbezüglich zur Verfügung. Mit dem BLEISTIFT kann der Benutzer aus „freier Hand“ Linien auf dem Bildschirm zeichnen. DIE SPRÜHDOSE ermöglicht das Sprühen einer Farbe oder eines Musters auf dem Bildschirm. DAS LASSO ermöglicht das Einrahmen und Verschieben beliebiger Bildausschnitte.

3. Sonstige weitere Funktionen

24 zweifarbige Muster stehen zur Verfügung. Eigene Muster können mit dem Mustereditor entworfen werden. So lassen sich alle 24 Muster undefinieren. Diese werden mit dem betreffenden Dokument abgespeichert, welches ca. eine DIN A 4 Seite umfaßt. Das Dokument läßt sich in alle Richtungen verschieben. Mit dem Auswahlviereck können beliebige Bildstellen zur weiteren Bearbeitung markiert werden.

Weitere Bearbeitungsschritte sind: – verschieben – ausschneiden – in die Zwischenablage kopieren – löschen – ausfüllen – invertieren – Konturen ziehen – horizontal drehen – vertikal drehen – rotieren.

4. Hilfen

Das gesamte Dokument kann mit einem imaginären Raster unterlegt werden. Im Vergrößerungsmodus ist es möglich, jeden Punkt einzeln zu manipulieren. Die Farben können jederzeit geändert werden.

Dokumente können als Hardcopy auf Schneider- und EPSON-Druckern ausgegeben werden.

5. Texteditor

Texte lassen sich problemlos in die Grafik integrieren. Dazu stehen mehrere Zeichensätze in jeweils drei Größen zur Verfügung. Jeder läßt sich zusätzlich fett, kursiv, unterstrichen oder Outline darstellen.

6. Steuerung und Windowing

Nahezu alle Funktionen lassen sich mit Joystick steuern: nur auf das entsprechende Symbol oder Window zeigen und den Feuerknopf betätigen.

Das professionelle deutsche Spitzenprogramm, komplett mit ausführlichem Handbuch, für CPC 464, 664 oder 6128.

PROFI PAINTER CPC DM 198,-*

Lieferbar ab ca. November.

Budget-Manager

Der Budget-Manager ist die universelle Buchführung sowohl für private Zwecke als auch zur Planung, Überwachung und Abwicklung von Budgets jeglicher Art.

In Stichworten:

430 Budgetsätze – 335 Kontensätze – Budget- und Kontenpläne sind per Programm zu erstellen – volle Menuesteuerung erleichtert die Arbeit – Einzelanzeige von Konten, Geldfälligkeiten, Amortisation, Zinsen, Tilgung bis zur kleinen Privatbilanz – Tabellen- und Graphikausgabe auf dem Bildschirm und als Hardcopy auf dem Drucker möglich – zweites Laufwerk wird unterstützt. Für CPC 464 und 664.

BUDGET-MANAGER DM 148,-*

Profimat CPC

Zur Programmierung in Maschinensprache benötigt man einen Assembler. Doch Assembler ist nicht gleich Assembler. Deshalb gibt es PROFIMAT nun auch für die SCHNEIDER-Rechner. Durch den integrierten Editor wird das Arbeiten mit PROFIMAT zum Vergnügen. Verketteten von Quelltexten für besonders lange Assemblerprogramme ist selbstverständlich möglich. PROFIMAT für den SCHNEIDER ist aber mehr als nur ein Assembler, er ist gleichzeitig auch Monitor.

Der absolute Clou dieses Assemblers ist die Möglichkeit, die frisch assemblierten Programme im TRACE-Modus (Einzelschritt-) laufen zu lassen und so jede Änderung an den CPU-Registern verfolgen zu können. PROFIMAT ist frei verschiebbar und kann somit nie in Konflikt mit Ihren eigenen Maschinenprogrammen kommen. Einfache Handhabung durch den komfortablen Editor auch für Anfänger garantiert. Selbstverständlich „beherrscht“ der Assembler auch die sogenannten Pseudo-Ops, die bedingtes Assemblieren möglich machen.

PROFIMAT CPC DM 99,-*

Lieferbar ab ca. November für SCHNEIDER CPC 464, 664 und 6128

Mathemat CPC

MATHEMAT, das unentbehrliche Hilfsmittel für Schule, Beruf und Studium, ist nun auch für die SCHNEIDER-Rechner erhältlich.

MATHEMAT beschäftigt sich mit der Geometrie und Algebra sowie mit der Kurvendiskussion. Mit MATHEMAT können Sie beliebige Funktionen ableiten, integrieren und zeichnen lassen. Die Grafiken werden mit einer Skalierung versehen, so daß die Ausdrücke auch in Schule und Studium verwendet werden können. MATHEMAT optimiert selbständig die errechneten Ableitungen, löst Klammern auf etc.

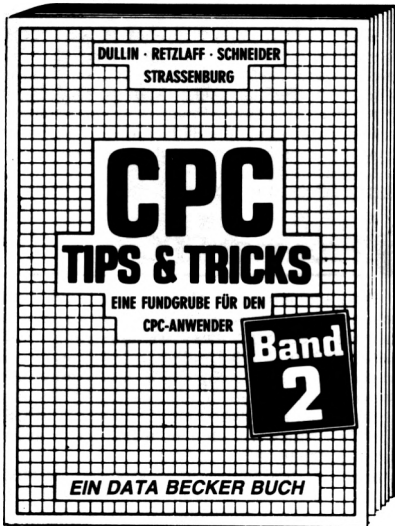
Weitere Programmteile sind der Taschenrechner und der Teil Geometrie/Algebra, in dem Sie Flächen- und Körperberechnungen durchführen können. Eine Hardcopy vom Bildschirm ist jederzeit möglich. Jeder an den SCHNEIDER CPC anschließbare Drucker kann einfach angepaßt werden.

MATHEMAT CPC DM 99,-*

Lieferbar ab ca. November für SCHNEIDER CPC 464, 664 und 6128

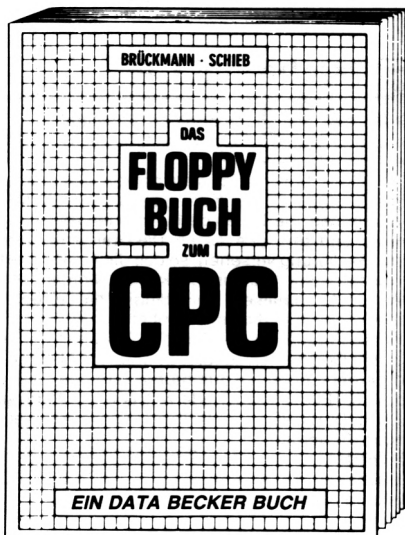
* unverbindliche Preisempfehlung

Alle Programme auf 3" Disketten



Der 2. Band CPC Tips & Tricks ist für alle CPC Besitzer interessant. Ob sie nun einen 464, 664 oder 6128 besitzen! Aus dem Inhalt: Menuegenerator, Maskengenerator, BASIC-Befehlsweiterungen, Programmierhilfen wie Dump, BASIC-Zeile von BASIC aus erzeugen, wichtige Systemroutinen und deren Nutzung, Beschleunigung von Programmen u.v.m. Wer noch mehr über seinen CPC wissen will, der kommt an diesem Buch nicht vorbei!

Dullin/Straßenburg/Retzlaff
CPC Tips & Tricks Band II
über 250 Seiten, DM 39,-
Erscheint im November
ISBN 3-89011-131-9



Alles über Floppyprogrammierung vom Einsteiger bis zum Profi. Natürlich mit ausführlichem ROM-Listing, einer äußerst komfortablen Dateiverwaltung, einem hilfreichen Disk-Monitor und einem ausgesprochen nützlichen Disk-Manager. Dazu eine Fundgrube verschiedener Programme und Hilfsroutinen, die das Buch für jeden Floppy-Anwender zur Pflichtlektüre machen!

Brückmann/Schieb
Das Floppy-Buch zum CPC
250 Seiten, DM 49,-
ISBN 3-89011-093-2

DAS STEHT DRIN:

Das Buch CPC 6128 FÜR EINSTEIGER sollte das erste Buch zum Schneider CPC 6128 sein. Es ist eine leichtverständliche Einführung in Handhabung, Einsatz und Programmierung des SCHNEIDER CPC 6128, die keinerlei Vorkenntnisse voraussetzt.

Aus dem Inhalt:

- Die Bedienung der Tastatur
- Der erste Befehl
- Das erste Programm
- BASIC-Einführung Schritt für Schritt mit Erstellung einer kompletten Adressenverwaltung
- GRAFIK-Befehle mit Beispielprogramm
- SOUND-Befehle mit eindrucksvollen Demonstrationsprogrammen (z.B. Orgel auf der Tastatur)
- Das Diskettenlaufwerk und seine Bedienung
- Weitere nützliche Befehle (z.B. Joystickabfrage, Zufallszahlen)

UND GESCHRIEBEN HAT DIESES BUCH:

Norbert Szczepanowski, Datenverarbeitungs-Kaufmann ist Bestsellerautor bei DATA BECKER. Er hat mehrjährige Erfahrung in der Programmierung zahlreicher Rechner.

ISBN 3-89011-143-2

СРО

СРО

СРО

СРО

СРО

СРО

СРО

СРО

СРО

СРО

СРО

СРО

СРО

СРО

СРО

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.