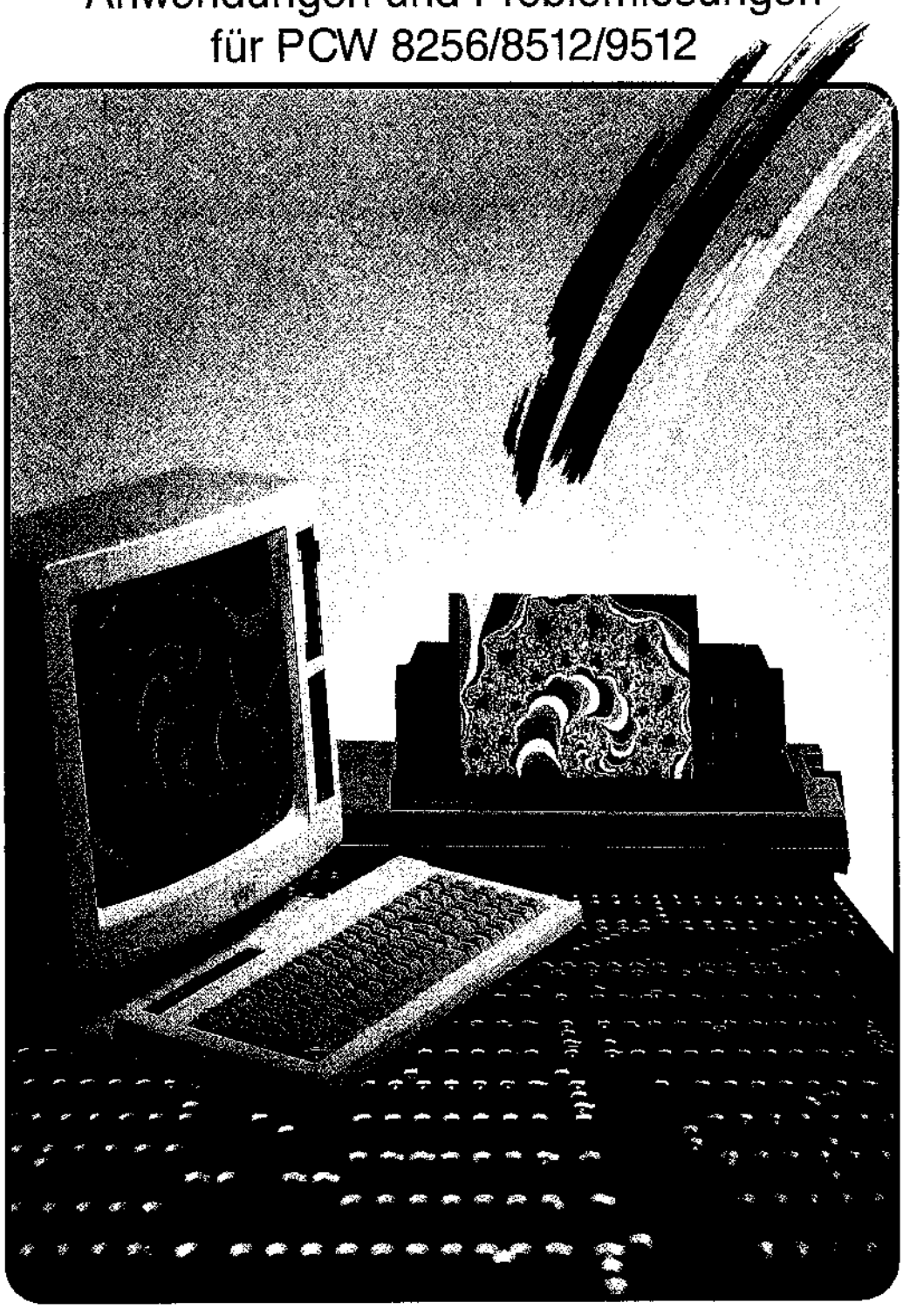
JOYCE

JOYCE - mehr als ein Textsystem

Anwendungen und Problemlösungen
für PCW 8256/8512/9512



DMV
Daten- und
Medienverlag

Mit großem
Hardwareteil

Hinweis des Verlages:

Die in diesem Buch veröffentlichten Programme, Verfahren, Schaltungen etc. werden ohne Rücksicht auf eventuell bestehende Patente abgedruckt. Sie sind ausschließlich für Amateur- und Lehrzwecke bestimmt. Eine gewerbliche Nutzung ist nicht gestattet.
Alle Angaben und Programme dieses Buches werden vom Autor und vom Verlag sorgfältig überprüft. Dennoch lassen sich Fehler nie ganz ausschließen. Der Verlag kann daher weder die Gewähr für Fehlerlosigkeit noch die juristische Verantwortung oder irgendeine Haftung für eventuell auftretende Folgen übernehmen.

DMV-Verlag
Fuldaer Straße 6 · 3440 Eschwege · Tel. (0 56 51) 80 09-0
ISBN 3-926177-02-0 **Printed in Germany**

Im Jahre 1985 von der Firma Schneider auf den Markt gebracht, später von der deutschen Tochter des englischen Herstellers Amstrad weiter vertrieben, nahmen die PCWs - im Sprachgebrauch stets 'JOYCE' genannt - von Anfang an eine Sonderstellung unter den Computern ein. Mangels Farb- und Musikausstattung zum Spielen fast untauglich, durch die mitgelieferte Software und das geschlossene Hardwarekonzept schon von Seiten der Hersteller zur Textverarbeitung verdammt, ließen die PCWs ihre Besitzer über die mannigfaltigen Anwendungsmöglichkeiten stets im unklaren. Leser der in diesem Verlag herausgegebenen Monatszeitschrift 'PC International' konnten mitverfolgen, wie findige PCW-Besitzer die Geheimnisse ihres Computers nach und nach aufdeckten - mit der Zeit ergab sich eine stattliche Sammlung von Unterlagen zu den von Vertreiber, Hersteller und Händlern gern totgeschwiegenen Fähigkeiten der PCWs, die, wie bekannt ist, über Textverarbeitung weit hinausgehen. Die Autoren des vorliegenden Buches - in der 'JOYCE-Szene' keine Unbekannten - haben die bislang verfügbaren Informationen in mühevoller Kleinarbeit gefiltert und um wesentliche Teile ergänzt. Wenngleich es nicht den Anspruch erhebt, der Weisheit letzter Schluß zu sein, steht Ihnen mit diesem Buch doch die meines Erachtens vollständigste Zusammenstellung an Informationen über die 'unbekannte Seite' der PCWs zur Verfügung.

Für den Herausgeber
Eschwege, im April 1989
Michael Ebbrecht

Inhaltsverzeichnis

Vorwort

Seit Erscheinen der PCW auf dem deutschen Markt gab es auf dem Computersektor enorme Weiterentwicklungen, die für Preisstürze der "veralteten" Modelle verantwortlich waren. Wenn dennoch, wie die Verkaufsinserate verraten, so viele der über 100000 "PCW'ler" allein in der Bundesrepublik ihrem Joyce treu blieben, so liegen dafür handfeste Gründe vor.

Zunächst besticht der PCW durch die Einheit von Drucker, Tastatur und Monitor, die optimal aufeinander abgestimmt sind und einfachste Handhabung des mitgelieferten LocoScript garantieren. Dieses Textverarbeitungssystem war es auch, daß die meisten Anwender zu ihrer Kaufentscheidung bewog. Benutzerfreundliche Menueführungen gestatteten es, sofort nach dem Einschalten loszuschreiben, 90 Zeichen und 32 Zeilen (Auflösung von 720x256 Pixel!) am Monitor dargestellt zu bekommen, was längst nicht Standard ist, und ohne umständliche Anpassungen des Druckers gleich auszudrucken. Der PCW wurde als Textsystem angeboten und als solches gekauft. Wieviel jedoch der Joyce tatsächlich darüber hinaus leisten kann und daß er ein leistungsfähiger Computer ist, dem eine riesige internationale Bibliothek von CP/M-Software zur Verfügung steht, blieb vielen Anwendern verborgen.
Der PCW-Anwender braucht nicht einmal allzu hohe Zugeständnisse an die Geschwindigkeit seines Computers zu machen, da neuere Rechner meist auf ältere Software angewiesen sind, die für sie angepaßt wurde, und die deshalb die erreichbare Verarbeitungsgeschwindigkeit des Computers nicht ausnutzt.
Eines jedoch gibt es an diesem System zu kritisieren. Die mitgelieferten Handbücher erklären zum einen die integrierte Software höchst ungenau und unvollständig, zum anderen finden die Hardware und ihre Ausbaumöglichkeiten gar keine Erwähnung. Diesem Manko soll das vorliegende Buch entgegentreten und mithelfen, den Joyce vor den Augen seiner Anwender zu dem zu machen, was er de facto ist:

ein vollwertiger Computer.

Zu diesem Buch

Es gab in jüngster Vergangenheit bereits einige Publikationen, die dem Joycebesitzer eine wertvolle Hilfe sein können. Hier sind:

- die Artikel zum Joyce der Monatsschrift Amstrad/PC-International,
- das Buch "Praktische Textverarbeitung mit Joyce" von Jürgen Siebert,
- das CP/M Plus Anwender-Handbuch CPC 6128/Joyce von Jürgen Hückstädt,
- und das große Logo-Buch zu CPC und Joyce von Gerhard Sauer

zu nennen.
Andere Schriften zum Joyce sind weit weniger wertvoll und geben dem Anwender kaum die Hilfe und die Tips, die er von ihnen erwarten könnte.
Da gerade zum Betriebssystem CP/M und zur Textverarbeitung derart gute Publikationen vorliegen, die man nur zur Anschaffung empfehlen kann, möchten wir diese Themen (mit Ausnahme von MAIL232) nicht noch einmal aufgreifen und weiter ausbauen. Außerdem erschienen für den Joyce erweiterte LocoScript-Versionen und ein vollkommen neues Textverarbeitungssystem von Arnor (Prowort), das durch Schnelligkeit, integrierte Korrekturfunktionen und vieles mehr derart überzeugt, daß man kaum noch guten Gewissens das ein oder andere System durch Erwähnung bevorzugen kann.
Mit dem Buch von Gerhard Sauer liegt zwar eine gute Dokumentation zu "Logo" vor, doch scheint sich diese Sprache bei den Joyceanwendern nicht in dem Maße durchgesetzt zu haben, daß sich für sie die Anschaffung des Buches zu lohnen schien. Einsatz könnte diese Sprache zum Beispiel bei der Erstellung von Briefköpfen, kleineren Diagrammen oder schnellen Kalkulationen zwischendurch finden. Derartigen Anwendungen angemessen wurde ein entsprechend kurzes Kapitel dieses Buches auch der leider verkannten Sprache Logo gewidmet.
In erster Linie werden wir uns also um die Vertiefung und

Verfeinerung der Kenntnisse der Sprache BASIC und ihrer Jetsamverwaltung bemühen, wobei auch dem fortgeschrittenen Anwender wertvolle Tips an die Hand gegeben werden können.
Ebenfalls wichtig erschien uns, über Hardwareerweiterungen wie Scanner oder Akustikkoppler für den Joyce zu berichten. Nützliche Tips und Tricks für Basteleien an vorhandener Hardware dürfen in diesem Zusammenhang natürlich nicht fehlen.
Alles in allem hoffen wir mit vorliegendem Buch, dem Anwender ein erweitertes Einsatzspektrum seines Joyce zu eröffnen und ein effektiveres Arbeiten und Experimentieren mit diesem Computer zu ermöglichen.
Begleitend zu diesem Buch ist über den DMV-Verlag eine Diskette erhältlich, die sämtliche Listings (auch viele der kleinen Programme zum BASIC-Teil) "ready to run" beinhaltet. Darüber hinaus befinden sich auf ihr auch Programme, deren Listings im Buch zu viel Platz beansprucht hätten und für Erklärungen über Programmierung nicht mehr benötigt wurden. Dennoch handelt es sich um wertvolle Programme, die dem Anwender zusätzliche Informationen liefern können und sogar - wie das Programm "Generjet" - bei einer Programmierung an sich behilflich sein können. (Ein sogenanntes "tool").

Die im Buch veröffentlichten Platinenlayouts brauchen nicht selbst geätzt zu werden. Sie können über die Fa.:

Joyce-Platinenservice, Roesoll 36, 2305 Heikendorf

bezogen werden. Den Platinen wurde eine ausführliche Anleitung mit auf den Weg gegeben, so daß ein einigermaßen geschickter Bastler sie selbst bestücken kann. Dadurch, daß wir die Layouts selbst entwickelten und der Anwender sie selbst bestückt, konnten die Platinen recht preiswert auf den Markt gebracht werden (DM 49,-). Neben dem Sprachsynthesizer oder der computergesteuerten Lichterkette, die beide ein Novum für den Joyce darstellen, bietet sich mit der Schnittstelle eine echte Alternative zur üblichen Hardwareerweiterung.

Die Sprachen des Joyce

In den folgenden Kapiteln wird in erster Linie auf die Sprachen Logo und BASIC eingegangen. Dies liegt nahe, da sie jedem Anwender auf den Systemdisketten zur Verfügung stehen. Wer in diesen Sprachen zu programmieren weiß, Kreativität und bestimmte geistige Leistungsfähigkeit an ihnen erprobt hat, dem fällt es auch leicht, auf andere Sprachen umzulernen.
Daß die Sprache Logo z.B. nicht als "Exotensprache" im luftleeren Raum steht, beweist die Tatsache, daß es als Abkömmling von LISP zu sehen ist. LISP entstand in den frühen sechziger Jahren am Massachusets Institut of Technology speziell um die Möglichkeiten künstlicher Intelligenz zu studieren. Seymore Papert (ein Schüler Jean Piagets), der an diesem Institut arbeitete, entwickelte dann in langjähriger Arbeit die Sprache Logo, die man als ausgefeilten Dialekt von LISP sehen kann.
Dennoch sollen zu Ende des Kapitels noch einige Bemerkungen zu anderen Sprachen fallen. Eine beachtliche Zahl von Joyce-Programmierern arbeitet mit Turbo-Pascal. Anwender, die sich über diese Sprache noch nicht informieren konnten, werden hier ein kleines Review vorfinden, das kurz Vor- und Nachteile dieser Sprache aufzeigt.
Daß die Sprache C, die auf vielen Großrechenanlagen und Rechnern neueren Typs in zunehmendem Maße eingesetzt wird, auch für den Joyce erhältlich ist, war vielen Joyce-Usern unbekannt. Welche immensen Vorteile hinter dem Systemangebot C-Compiler - zudem noch aus deutschen Landen - stecken, wird zu Ende des Kapitels Erwähnung finden.
Forth, als Public Domain Software für den Joyce erhältlich, scheint aus dem Rennen der Gunst der Programmierer geschlagen zu sein. Vor dem Einstieg in die Arbeit mit Logo steht an dieser Stelle also nur der Hinweis, daß der Joycer selbst darauf nicht zu verzichten braucht.
Zur Schreibweise: Sind Tasten der Konsole gemeint, so werden sie in eckige Klammern und in große Lettern gesetzt. Variablen und Listingauszüge finden sich in Fett- bzw. Kleindruck.

Logo

Allgemein:

In der Zeit, in der der Joyce auf dem Markt ist, hat sich herausgestellt, daß sich die meisten seiner Anwender auf die Benutzung des mitgelieferten LocoScript beschränken, oder auf zugekaufte Programmsysteme zurückgreifen. Weitaus seltener versuchen sie sich darin, selbst zu programmieren. Dabei liegt gerade mit Logo eine Sprache vor, die für Einsteiger besonders geeignet scheint, und zum Experimentieren einlädt.
Diese Eigenschaft der Sprache hat ihr zu Unrecht ihren schlechten Ruf als "Kindersprache" eingebracht. Dabei sind Sprachen wie BASIC oder Pascal nicht so ohne weiteres - zumindest nicht ohne elementare Kenntnisse über die GSX-Schnittstelle - dazu zu bewegen, Grafik auf den Schirm zu bringen. Für den Logo-Anwender ist dies allerdings kaum ein Problem, wurde Logo doch gerade durch seine Turtle-Grafik bekannt.
Wer also in erster Linie seine Sekretärin Joyce zum Schreiben nutzt, sollte doch ruhig einmal probieren, wie man sie auch zum Zeichnen kleinerer Grafiken oder zum Errechnen und Aufzeichnen von Diagrammen bewegen kann. Bestimmt wird dies auch der erste Schritt zu komplexen Programmstrukturen werden, die von dreidimensionaler Grafik bis hin zur Listenverarbeitungen reichen können.

Die Initialisierung:

Die Sprache Logo hat bei den nicht professionellen Joyce-Anwendern auch immer ein wenig im Schatten gestanden, weil die Dokumentation in Form des Handbuches kaum ihren Namen verdient. Viele Befehle, die Logo kennt und ausführen kann, werden überhaupt nicht erklärt, und Befehle wie po, ss oder **setsplit** werden gleich mehrfach erörtert.
Die meisten Handbücher sagen nicht einmal, wie Logo denn zu

starten sei, was an dieser Stelle geschehen soll.
Für den Anwender, der mal schnell zwischendurch in Logo zeichnen oder rechnen möchte, empfiehlt es sich, eine Startdiskette anzufertigen, die nur ins Laufwerk geschoben werden muß und Logo automatisch startet. Am zweckmäßigsten kann dies mit Hilfe von LocoScript bewerkstelligt werden. Zunächst werden die Files:

J14GCPM.EMS
LOGO.COM
KEYS.DRL
SUBMIT.COM
LANGUAGE.COM
SETKEYS.COM

von den Seiten 2 und 4 der Systemdisketten in eine Gruppe von Laufwerk M: in LocoScript geladen. Danach wird mit "E" unter LocoScript eine Datei erstellt, die nichts weiter enthält, als den genau so geschriebenen Text:

```
LANGUAGE 0
SETKEYS KEYS.DRL
LOGO.COM
```

Diese Datei muß jetzt als ASCII-Datei (eigener Menuepunkt unter LocoScript) gespeichert werden. Dabei fragt Joyce nach dem Namen, den diese Datei erhalten soll. Er wird mit "PROFILE.SUB" angegeben und ebenso wie die vorher in M: gespeicherten Dateien in die Gruppe 0 des Laufwerks A:, welches mit einer neu formatierten Diskette bestückt wurde, abgespeichert.
So befinden sich die Dateien nun auf einer neuen Startdiskette in Laufwerk A:. Nach dem gleichzeitigen Drücken von [SHIFT]-[EXTRA]-[EXIT] müßte der Monitor dunkel werden, das Laufwerk gutwillig brummend seine Arbeit aufnehmen und das Betriebssystem booten. Das Betriebssystem arbeitet dann die PROFILE.SUB-Datei der Reihe nach ab, schaltet auf den amerikanischen Zeichensatz um, damit z.B. die Ä's als eckige

Klammern auf dem Monitor erscheinen, ändert die Tastaturbelegung, damit z.B. Drücken von [F1] pausing unter Logo verursacht und ruft dann Logo selbst auf. Am Ende des Prozesses steht uns Logo mit seinem ?-Prompt zur Verfügung.
Wer wirklich professionell mit Logo arbeiten möchte, sollte auch die Standardeinstellung des Druckers ändern, damit in den Listings ebenfalls eckige Klammern und dergleichen erscheinen. Diese Änderung geschieht am zweckmäßigsten mit dem Programm SETLST.COM von Seite 2 der Systemdisketten. SETLST.COM wird ebenfalls auf die Startdiskette kopiert. Unter LocoScript wird wieder ein ASCII-File mit dem Namen "DRUCKER.DRL" erstellt, der nur die Zeile:

```
↑'ESC'R↑'0'
```

enthält. Auch diese Datei wird auf der Startdiskette abgespeichert. In die POFILE.SUB-Datei braucht nur noch die Zeile:

```
SETLST DRUCKER.DRL
```

vor der Zeile LOGO.COM eingefügt werden, so daß bei einem Neustart der Drucker automatisch richtig eingestellt wird.
Auf der Startdiskette selbst ist jetzt noch genügend Platz, um etliche Prozeduren, die den Wortschatz der Sprache erweitern sollen, zu speichern. Einer effektiven Arbeit mit Logo steht jetzt nichts mehr im Wege.

L o g o a l s R e c h n e r

Für viele Aufgabenbereiche kann es durchaus sinnvoll erscheinen, Logo auf dem Joyce zu booten und schnell einige Rechnungen durchzuführen. Mal abgesehen von der Leistungsfähigkeit des Systems, an die nur Taschenrechner der gehobenen Klasse heranreichen, kann der Drucker sämtliche Aktionen des Rechners protokollieren. Dazu wird lediglich hinter das ?-Prompt von Logo

```
copyon
```

eingegeben, und nachdem [RETURN] gedrückt wurde, werden sowohl Aufgaben als auch Lösungen oder Ergebnisse zu Papier gebracht. Fast könnte man meinen, Joyce kopiere eine Registrierkasse.
Wird ein Protokoll nicht mehr gewünscht, reicht die Eingabe von

```
copyoff
```

um den Drucker seine Arbeit einstellen zu lassen. Ein weiterer Vorteil des Systems besteht darin, daß man häufig gebrauchte Konstanten wie Kreiszahl pi, den Mehrwertsteuersatz oder den Skontorabatt einfach eingeben und abrufen kann und daß häufig durchzuführende Rechnungen in Prozeduren festgelegt werden können, die dann schnell zur Hand sind. Eine solche Festlegung kann in Logo so aussehen:

```
make "Mwst 0.14
```

Bei diesem Befehl gilt es zu beachten, daß die Leerstellen und die Anführungszeichen gesetzt werden, um keine Fehlermeldung folgen zu lassen. Außerdem sollte man sich merken, ob Mwst oder MWST oder mWst definiert wurden. Das sind für Logo drei verschiedene Vokabeln! Nachdem nun die Mwst für Logo festgelegt wurde, kann man auch damit rechnen. Ein simples Beispiel würde so aussehen:

```
50 * :Mwst
```


Hier wird Mwst jetzt mit einem vorangestellten Doppelpunkt gekennzeichnet. Die unterschiedlichen Notationen in Listings bedeuten:
-Das einleitende Anführungszeichen zeigt Logo, daß es sich bei dem Folgenden um einen Platzhalter oder ein freies Fach für etwas handelt.
-Der einleitende Doppelpunkt zeigt Logo, daß hier der Wert, für den der Platzhalter steht, eingesetzt werden muß.
-Keine Bezeichnung läßt Logo vermuten, daß eine Prozedur oder Kommando gemeint ist. Logoeigene Befehle werden immer klein geschrieben.
Hier nun die möglichen Rechenoperationen:

Form	Ausgabe	Erklärung
2.1 + 3.4 oder:		Sollen + oder - als Vorzeichen dienen, so
+ 2.1 3.4	5.5	müssen sie ohne Leerstelle vor die betreffande Zahl gesetzt werden.
(†Dezimalpunkte !)		(Subtraktion entsprechend)
3 * 2 oder: * 2 3	6	Multiplikation
8 / 2 oder: / 8 2	4	Division
Komplex:		
(1.4 + 6.6) / (4 - 1 * 2)	4	Punktrechnung vor Strichrechnung !
(+ 4 2 3 9 2)	20	Mehr als 2 Elemente müssen in runde Klammern gesetzt werden.
sin 30	0.5	Sinuswert für Winkel 30 Grad (Angaben nur in Gradmaß)
cos 30	0.866025388240814	Cosinuswert für Winkel 30 Grad. (In allen Logorechnungen sind nur die ersten 7 Stellen hinter dem Punkt verläßlich !!!)
arctan 0.5	26.565051177178	Arctanwert von 0.5 (Gradmaß)
round 2.2	2	Rundet auf ganze Zahl auf oder ab.
int 2.3	2	Gibt den ganzzahligen Wert einer Zahl an
quotient 10 4	2	Ganzzahliger Wert des Bruchs 10/4
remainder 10 4	2	Der Rest bei der Ganzzahldivision 10/4
1 + random 49	z.B. 16	Ermittelt eine Zufallszahl, die wegen 1+ größer als 0 und kleiner als 50 ist.Die Zufallsreihen sind nach jedem Neustart gleich. Sie können mit "rerandom" wieder zurückgesetzt werden.

Basierend auf diesen Grundfähigkeiten von Logo lassen sich alle anfallenden Probleme lösen. Was Logo nicht kann, läßt sich als Prozedur definieren und in seinen Wortschatz einbauen. Dies läßt sich mit dem Editor von Logo einfach bewerkstelligen. Nach Eingabe von ed geht Logo in einen

anderen Modus über, und man kann das Geschriebene wie in einem Textsystem löschen, mit [DEL] korrigieren, oder mit den Cursortasten darin herumfahren.
Ein einfaches Programm müßte jetzt so eingetippt werden:

```
to kreisumfang :r
make "pi 3.1415926
op 2 * :pi * :r
end
```

Die erste Zeile gibt Logo an, wie diese Prozedur in Zukunft zu heißen hat. Wenn man später nicht so viel tippen will, könnte sie auch kr heißen. Mit dem **:r** signalisiert man Logo, daß in Zusammenhang mit der Prozedur "Kreisumfang" eine Eingabe - in diesem Fall der Radius - zu machen sein wird, die Logo dann auch unbedingt beim Aufruf erwartet.
Die zweite Zeile müßte hier nicht stehen. Hätte man Logo außerhalb dieser Prozedur den Wert für **pi** gegeben, wie vorhin beim Beispiel **Mwst**, könnte er ihn sich auch für viele andere Aufgaben aus seinem Arbeitsspeicher holen.
Die dritte Zeile beinhaltet die eigentliche Rechnung, deren Ergebnis als **op** (Output) an den Monitor gegeben wird. Die vierte Zeile signalisiert Logo, daß die Prozedur hier beendet ist. Sollte man sich bei Eingabe dieser Prozedur sehr vertippt haben und man möchte lieber noch mal von Vorne anfangen, so genügt ein Druck auf die [STOP]-Taste. Ist alles richtig eingetippt, wird die [EXIT]-Taste gedrückt, Logo verläßt den Editier-Modus und meldet: kreisumfang defined. (Der Umgang mit [STOP]- und [EXIT]-taste wird in älteren Handbüchern verwechselt!)
Nach der Eingabe von:

```
kreisumfang 5
```

wird Logo den Umfang eines Kreises mit dem Radius 5 berechnen und das Ergebnis auf den Schirm bringen. Hier wird eigentlich noch nicht ersichtlich, inwieweit eine Arbeitserleichterung gegeben ist, es sei denn, jemand müßte am laufenden Band Kreisumfänge errechnen. Ziel vorliegenden Buches ist es nun nicht, einen Kurs über Programmieren in Logo zu liefern. Es sollen nur die Möglichkeiten kurz

aufgezeigt werden. Ein schönes Beispiel für eine komplexere Struktur liefert die Errechnung der Quadratwurzel. Aus unverständlichen Gründen hat Logo sie nicht in seinem Grundwortschatz, und so muß sie als Prozedur extra definiert werden. Nach dem Aufruf von **ed** gibt man ein:

```
to sqrt :x
op wurz (:x + 1) * 0.5
end
to wurz :a
local "b make "b ((:x /:a + :a) * 0.5)
if :a - :b < 1.e-03 [op :b]
op wurz :b
end
```

Nach dem Drücken von [EXIT] erscheint: Sqrt defined wurz defined. Hier wurden zwei Prozeduren definiert, die aber beim Abspeichern auf Diskette mit:

```
save "sqrt
```

beide gesichert würden, und beim Laden von Diskette in den Arbeitsspeicher mit:

```
load "sqrt
```

beide im Wortschatz zur Verfügung stünden. Logo lädt und sichert zusammengehörige Prozeduren gemeinsam. Die erste Prozedur nun liefert die Hälfte des um eins vermehrten Radikanden durch **op** (Output) an die zweite Prozedur, wobei dieser Rest den Wert **:a** erhält. In der zweiten Prozedur wird jetzt der Radikant durch **:a** geteilt, um **:a** vermehrt und um die Hälfte verkleinert. Was dabei herauskommt muß Logo jetzt als Wert **:b** sehen. Als nächstes schaut Logo in der if-Zeile, ob die Differenz zwischen **:a** und **:b** schon kleiner als 1.e-03 ist. Trifft dies zu, kann Logo die erste in eckigen Klammern stehende Befehlsliste ausführen, also den Wert von **:b** als Ergebnis ausgeben. Trifft es nicht zu, geht alles noch mal bei **wurz** von vorne los, wobei **:b** der neue Anfangswert von **wurz** ist.
Local "b in der zweiten Prozedur zeigt Logo, daß der Wert, der im Folgenden ausgerechnet und durch **make** der Variablen **"b** zugeordnet wird, nur für diesen einen Durchlauf der

Prozedur Gültigkeit hat und bei jedem weiteren Durchlauf neu zugemessen werden kann.
Bei seiner Rechnerei läßt sich Logo auch in die Karten schauen. Tippt man hinter das ?-Prompt

```
trace
```

ein, so zeigt Logo die Rechenergebnisse aller Zwischenschritte. Noch genauer kann man die Sache verfolgen, wenn

```
watch
```

eingegeben wird. Danach läßt sich Logo mit einer neuen Wertzuweisung solange Zeit, bis [RETURN] gedrückt wurde. Außerdem wird der Rechenschritt, bei dem er sich gerade befindet, genau aufgezeichnet. Durch

```
notrace und nowatch
```

hinter dem Prompt, wird dieser Zustand wieder abgeschaltet.
Eigentlich müßte man für den Fall, daß jemand eine Null oder negative Zahl radizieren will, eine Sicherung in die Wurzelfunktion einbauen. Dies kann auch wieder durch eine **if**-Zeile geschehen. In der zweiten Zeile der **sqrt**-Prozedur könnte dann stehen:

```
if :x = 0 [op 0] if :x < 0 [op "neg.]
```

Es lohnt sich schon tatsächlich, eine Wurzelfunktion im Arbeitsspeicher Logo's zu haben, zumal sie auch für den eigentlichen Arbeitseinsatz von Logo, der Grafikprogrammierung, gut zu gebrauchen ist. An späterer Stelle wird sie noch einmal auftauchen, wenn demonstriert werden soll, wie Logo nach dem Satz des Pythagoras Dreiecke berechnet und konstruiert.
Sollte man sich bei der Eingabe der Wurzelfunktion ohne es zu merken vertippt haben, so wird Logo nach dem Aufruf der Prozedur eine Fehlermeldung auf den Monitor bringen. Tippt man nun gleich den Befehl **ed** ohne weitere Spezifikation hinter das Prompt, so geht Logo mit der fehlerhaften Prozedur in den Editiermodus über, und läßt den Cursor an

der fehlerhaften Stelle blinken.
Dies mag als Einblick in die Rechenfähigkeiten Logo's genügen und als Überleitung zur Beschäftigung mit der Grafik dienen.

L o g o a l s G r a f i k e r

Vorweg:

Anwender, die nur ein wenig mit Logo herumspielten und einige kleine Grafiken auf den Schirm zaubern konnten, hatten, wenn sie nicht tiefer in die Materie eingestiegen waren, zwei Probleme:
Zum einen schweigt sich das Handbuch darüber aus, wie man eine Hardcopy des Bildschirms zu Papier bringen kann, - nämlich mit gleichzeitigem Drücken von:

[EXTRA]/[PTR]

und wenn sich das durch Mund- zu Mundpropaganda herumgesprochen hatte, so bekam man - wenn es nicht gelang, durch Druck auf [PTR] rechtzeitig den Vorgang abzubrechen - zusätzlich zu seiner Grafik noch die Meldung aufs Papier gedruckt, daß A:das aktuelle Laufwerk sei. Dieser Mißstand läßt sich dadurch beheben, daß hinter das Logo-Prompt eingetippt wird:

```
type word char 27 "0
```

Nach Druck auf [RETURN] verschwindet diese lästige Anzeige. Wiederherstellen läßt sie sich durch Eintippen der 1 anstelle der 0. Mit dem Befehl **type word char 27 "**(+Ziffer oder Zahl) läßt sich noch mehr anstellen, wie im Abschnitt über Beschriftung von Grafik ersichtlich wird. Neugierigen sei an dieser Stelle zunächst empfohlen, sich die Escape-Folgen in diesem Buch anzusehen. Unter Logo wird nur für ESC

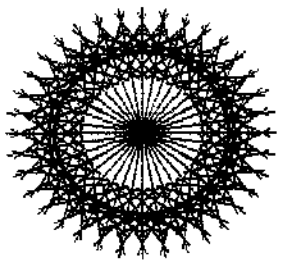
das **type word char 27 "** eingesetzt. Nach diesen einleitenden Bemerkungen aber nun zur eigentlichen Grafikprogrammierung.

Die freie Turtlebewegung

Die Überschrift macht hier schon kenntlich, daß der Zeichencursor sich auch unter anderen Grundbedingungen als frei auf der Fläche des Monitors bewegen kann. Gemeint ist hier ein Koordinatensystem, das Vorgaben für die Turtlebewegung geben kann. Dazu aber später mehr. Hier interressieren erst einmal die Befehle, die dem Grafikcursor an seiner Position relativ zu seiner Stellung Anweisungen geben.
In der kurzen erklärenden Einleitung des Handbuches werden die grundlegenden Formen von Eingaben z.B. mit **repeat** gezeigt. Prinzipielle Formen der Eingabe brauchen deshalb nicht mehr erklärt werden. Interessant wäre es aber einmal, eine Verschachtelung von Repeat-Anweisungen zu zeigen. Dies könnte etwa so aussehen:
(Die Eingabe erfolgt in einer Zeile, wird die Zeile für die Monitordarstellung zu lang, macht Logo automatisch einen Zeilenumbruch, der durch ein Ausrufungszeichen zu Ende der Zeile kenntlich gemacht wird, darum braucht man sich nicht zu kümmern)

```
cs ct fs ht repeat 36 [fd 150 repeat 10 [fd 100 bk 100 rt 36] bk 150 rt 10]
```

Nach Betätigen von [RETURN] entsteht folgendes Gebilde:



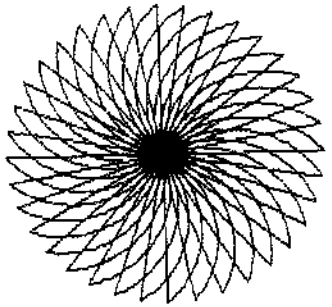
Nachdem der Schirm mit **cs** (clear screen) und **ct** (clear text) gesäubert wurde, wurde dem Grafikcursor mit **fs** (full screen) der vollständige Monitor zum Malen zur Verfügung gestellt. Mit **ht** (hide turtle) wurde der Cursor unsichtbar gemacht, was ein schnelleres Zeichnen zur Folge hat. Danach muß der Cursor die zwei Wiederholungsanweisungen durcharbeiten. In der ersten werden Strahlen der Länge 150 in 36 Schritten zu 10 Grad gelegt, was eine volle 360 Grad Drehung ausmacht, in der zweiten Prozedur werden um die Spitzen der Strahlen noch einmal Strahlen der Länge 100 gelegt. Hier ist die Anzahl allerdings geringer, es sind nur 10, die jeweils 36 Grad auseinander liegen.
Durch Variation der Längen und Gradeinteilungen lassen sich so interessante geometrische Gebilde erzeugen. Die bisherigen Anweisungen reichen auch aus, um die Turtle auf Kurvenbahnen zu schicken. So würde die Eingabe von:

```
repeat 50 [fd 4 rt 2]
```

eine Linie erzeugen, die 50 Mal nach jeweils 4 Schritten eine Rechtsdrehung vornimmt. Abstände und Winkel sind dabei so klein, daß der Eindruck einer Kurve entsteht. Auch diese Anweisung läßt sich in komplexere Prozeduren einbauen. So würde:

```
to t :a
repeat 50 [ fd 4 rt 2 ] home rt :a
local "b make "b (:a + 10) if :b > 360 [stop]
t :b
end
```

obige Prozedur - aufgerufen mit: **cs ct fs t 10** - die Rosette auf der folgenden Seite aus Kreissegmenten produzieren. Der Befehl **home** läßt die Turtle wieder in ihre Ausgangsposition zurückkehren, wobei sie eine Linie hinter sich herzieht und ihre Spitze wieder nach oben zeigt. Damit das nächste Rosettensegment nun um 10 Grad gegenüber dem vorherigen versetzt ist, muß sie nach dem home-Befehl eine Drehung nach rechts machen, die nach jedem Segment um 10 Grad größer ist als die vorherige. Deshalb wird **b** nach jedem Durchgang um 10 vergrößert und **t** wieder mit dem neuen **b**-Wert

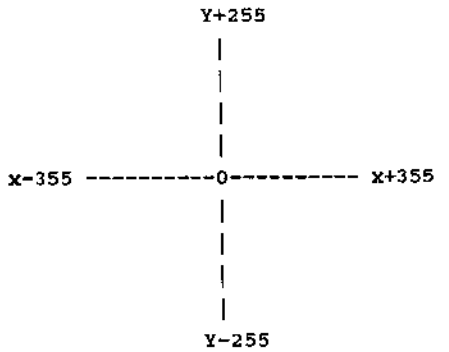


aufgerufen. Logo setzt den errechneten Wert von **:b** automatisch im folgenden Durchgang als **:a** ein. Damit dieser Prozeß nicht ad infinitum läuft, wurde mit **if** wieder ein Abbruch definiert. Sobald bei 360 Grad der Kreis geschlossen ist, stoppt Logo die gesamte Prozedur.
Mit Verknüpfung von Vorwärtsschritten und Rechtsdrehungen läßt sich nun eine Prozedur definieren, die nach Eingabe des Radius einen Kreis zeichnet. Um diese Prozedur anwenderfreundlicher zu gestalten, wäre es von Vorteil, den Cursor nach dem Zeichnen eines Kreises wieder auf seine Ausgangsposition zu dirigieren. An diesem Teil der Prozedur kann man sehr schön die Arbeit mit dem Koordinatensystem erklären, womit auch gleich die Überleitung zum nächsten Kapitel gegeben wäre.

Die Turtlebewegung im Koordinatengitter

Für die Positionierung des Grafikcursors mit Hilfe von Koordinatenpunkten stehen eine x-Achse mit 710 und eine y-Achse mit 510 Skalenpunkten zur Verfügung. Diese Werte beziehen sich auf den mit **fs** eingerichteten vollen Grafikschirm, wobei tatsächlich jeweils einige Grafikpunkte pro Koordinate mehr zur Verfügung stehen, aber bei der Arbeit mit Extremwerten bestünde Gefahr, den Cursor aus dem sichtbaren Bereich zu steuern. Der Nullpunkt der Skalen

liegt in der Bildschirmmitte, wie die erläuternde Grafik zeigt.
Logo kennt nun einige Befehle, die mit diesen Koordinaten arbeiten, wobei der zuerst genannte Wert sich immer auf die x-Achse bezieht.



setpos [-355 255]
würde den Grafikcursor in die linke obere Ecke steuern,
dot [0 0]
in die Mitte einen Punkt malen,
setx 10
würde die Turtle auf der x-Achse auf den Punkt 10 setzen,
sety 10
diese entsprechend auf die y-Achse.
seth towards [100 100]
würde die Spitze des Grafikcursors auf diesen Punkt ausrichten. (seth 90 bewegt die Turtle nicht im Koordinatensystem, sondern bewegt die Turtle wie unter dem Befehl rt, nur daß die Gradzahlen hier wie Himmelsrichtungen zu sehen sind: 0 = Nord, 90 = Ost, 180 = Süd, 270 = West etc.

Man kann allerdings auch Fragen stellen, die mit oder über Koordinaten beantwortet werden. So antwortet Logo auf **tf**

(turtle facts) mit einer Liste, von der die beiden ersten Zahlen die Koordinatenpunkte wiedergeben, auf denen sich der Cursor befindet. (Der Rest bezeichnet der Reihe nach: Richtung, in die der Zeichenstift zeigt (s. **seth**), Zustand des Zeichenstifts **pu** = abgehoben, **pd** = Zeichenstift unten, **pe** = Zeichenstift radiert, **px** = radiert oder zeichnet im Gegensatz zum Untergrund. Die vorletzte Zahl gibt die Farbe des Zeichenstifts an (0 oder 1) und die letzte, ob die Turtle sichtbar ist = true oder nicht = false).
Befände sich auf den Koordinaten 100 100 ein gesetzter Punkt, so würde

```
dotc [100 100]
```

mit der Ausgabe 1 antworten; wäre diese Stelle nicht belegt, so würde 0 ausgegeben. Dotc fragt also nach, ob ein Punkt gesetzt ist, oder nicht.
Um für die Kreisprozedur den Ausgangspunkt festzulegen, zu dem der Grafikcursor nach verrichteter Arbeit wieder zurückkehren soll, fehlt noch eine Vokabel. Wie besprochen liefert die Frage nach den Turtle-facts mit **tf** eine Liste, von der aber für **setpos** oder **setx/sety** nur die ersten zwei Werte benötigt werden. Um diese beiden Werte aus der Liste "herauszufiltern", können **piece** (wird z.B. im Handbuch nicht erklärt) und **item** eingesetzt werden.

```
piece 2 4 tf
```

würde aus der Liste der Turtle-facts das zweite bis vierte Element ausgeben. **Piece** ist also in der Lage, mehrere nebeneinanderliegende Elemente einer Liste auszuwerfen. **Item** ist nur in der Lage, ein einzelnes Element auszugeben. So würde

```
item 2 tf
```

genau das zweite Element dieser Liste zur Ausgabe haben. Die Belegung einer Variablen mit der derzeitigen Cursorposition könnte also so aussehen:

```
local "a make "a (piece 1 2 tf)
```


Will man später den Cursor auf diese Position setzen, so reicht:

```
setpos :a
```

um ihn dorthin zu dirigieren.
Arbeitet man mit Variablen für die Koordinatenpunkte, so muß aus ihnen vor der Verwendung in setpos eine Liste erstellt werden. Dies kann mit **se** bewerkstelligt werden. Als Befehl sähe das so aus :

```
setpos se :x :y
```

Zurück jedoch zur Kreisprozedur. Als einzige Eingabe soll die Prozedur "Kreis die Eingabe :r, den Radius, benötigen. Nach Feststellung und Merken des Standortes durch **make** und **piece/tf** muß die Turtle den so festgelegten Kreismittelpunkt auf gerader Linie um den Radius :r verlassen, eine Rechtsdrehung machen und dann eine Strecke wandern, die sich aus dem Umfang - also (2*pi*r) errechnet. Diese Strecke wird z.B. 100 mal durch eine Rechtsdrehung unterbrochen. Da bei den Drehungen insgesamt 360 Grad herauskommen müssen, muß jede einzelne der 100 Drehungen 3.6 betragen. Die Turtle hat folglich 100 mal den hundertsten Teil des Umfangs zu laufen und sich zwischendurch immer um 3.6 nach rechts zu drehen. Wenn sie fertig ist, kann sie sich, durch **setpos** dorthin gelenkt, wieder auf dem Kreismittelpunkt ausruhen. Als Prozedur sieht das folgendermaßen aus:

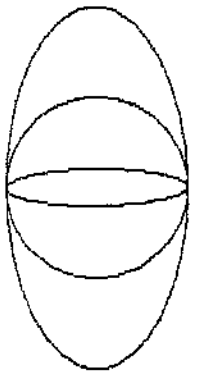
```
to Kreis :r
(local "a "b)
make "pi 3.14159265
make "a (piece 1 2 tf)
pu fd :r rt 92 pd
make "b 2 * :pi * :r / 100
repeat 100 [fd :b rt 3.6]
pu setpos :a lt 92 pd
end
```

Über die Kreisprozedur lassen sich auch Ellipsen konstruieren. Gibt man Logo mit **sf** die Frage nach den screen-facts, den Bildschirmzuständen, ein, so erhält man eine Liste mit 5 Elementen. Daraus kann man der Reihe nach

die Hintergrundfarbe des Grafikfensters, ob **ts**, **ss** oder **fs** aktiviert sind, die Fensteraufteilung, seine Aufteilung durch **fence**, **window** oder **wrap** und als letztes das Achsenverhältnis erfahren. Standartmäßig ist dieses Verhältnis mit 0.46875 ausgelegt. Es kann aber mit **setscrunch** geändert werden. Wie **setscrunch** genau arbeitet, läßt sich am besten feststellen, wenn folgender Befehl eingegeben wird:

```
cs ct fs ht setscrunch 1 Kreis 100 setscrunch 0.5 Kreis 100 setscrunch 0.1 Kreis 100
```

Damit werden der Reihe nachverschiedene Ellipsen erzeugt. Wird **setscrunch** mit 0.49625 eingestellt (Verhältnis 1 zu 1), entsprechen bei einer Hardcopy sowohl auf der X- wie auf der Y-Achse 5.7 Logoeinheiten genau einem Millimeter auf dem Papier!
Im Kapitel über das Rechnen mit Logo wurde eine Dreiecksprozedur angekündigt, die mit Hilfe der Wurzelfunktion im rechtwinkligen Dreieck nach dem Satz des Pythagoras die fehlende Seite ausrechnet und zeichnet. Es wäre nützlich, wenn Logo das Ergebnis seiner Rechnung unter die Zeichnung stellen würde. Die Realisierung dieses Problems führt uns in das nächste Kapitel, wo erläutert wird, wie Schrift in Grafiken gesetzt wird.



Die "setscrunch-Ellipsen"

Beschriftung von Grafik

In diesem Kapitel sollen sämtliche Fähigkeiten des Systems zum Tragen kommen. Es soll also ein Programm entwickelt werden, daß nach gewissen Vorgaben etwas ausrechnet, eine kleine Grafik dazu auf den Schirm bringt und das Ergebnis unter diese Grafik stellt. Dafür bietet sich die Dreiecksberechnung an. Das Problem gestaltet sich nicht zu komplex und bietet genügend Raum für Erklärungen.
Sind zwei Seiten im rechtenwinkeligen Dreieck bekannt, so läßt sich nach a↑2+b↑2=c↑2 die fehlende Seite berechnen. Bekannt seien in unserem Fall die Seiten a und b. Es wird also wieder die bereits erstellte Wurzelfunktion benötigt, denn c ist ja dann:

```
make "c sqrt (:a * :a + :b * :b)
```

Das Prozedere soll nun sein, daß der Grafikcursor von einem Punkt, den er sich merkt, die Strecke **:a** läuft, eine Rechtsdrehung um 90 Grad vollzieht, die Strecke **:b** läuft und dann die Spitze seines Cursors mit **seth towards** auf den gemerkten Ausgangspunkt dreht. Er braucht dann nur noch mit der errechneten Strecke **:c** zu laufen, um das Dreieck zu vollenden. Als Prozedur sähe das dann so aus:

```
to Dreieckab :a :b
cs ct fs ht
(local "h "c)
make "h (piece 1 2 tf)
make "c sqrt ((:a *:a)+ (:b*:b))
rt 45 fd :a rt 90 fd :b seth towards :h fd :c
```

(Eine weitere elegante Lösung bei Dreieckskonstruktionen, auf die ich hier aber nicht weiter eingehen möchte, bietet sich darin, daß man Halbreise schlagen lassen und mit **dotc** und **if**-Bedingung Schnittpunkte abfragen kann, die dann als unbekannte Ecke des Dreiecks angesprochen werden können.)
Jetzt bleibt in unserem Beispiel oben noch das Problem, daß der Textcursor unter das gezeichnete Dreieck gesteuert werden muß, um die Länge von **:c** auszudrucken. Ebenso wie bei

setpos wird der Textcursor über Koordinatenpunkte gesteuert und als Eingabe zu dem Befehl:

```
setcursor [:x :y]
```

wird eine Liste erwartet. Hier jedoch bezeichnet der Wert von x eine Spalte (0-88) und y eine Zeile (0-29). **setcursor [0 0] type [hier]** würde in der linken oberen Ecke das Wort "hier" erscheinen lassen, **setcursor [88 29] type [hier]** würde das Wort in der rechten unteren Ecke des Monitors sichtbar machen. Um also den Textcursor nach den Turtlekoordinaten auszurichten, müßte man eine Umrechnungstabelle für die verschiedenen Koordinatensysteme anlegen. Da jedoch die Dreiecksprozedur mit **cs** und **ct** gestartet wird, landet der Grafikcursor wieder in der Bildschirmmitte. Würden wir also den Textcursor mit **setcursor [45 20]** dirigieren, so käme er auf jeden Fall unter dem gezeichneten Dreieck zu stehen. Hinter dem Befehl zur Plazierung des Cursor käme dann zu stehen, was er dort machen soll, nämlich **c=** und den Wert von **c** auf den Schirm zu drucken, also:

```
type [c =\ ] type :c
```

(Wegen \ wird der nachfolgende Zwischenraum gedruckt.Bei richtiger Tastaturbelegung liegt der Backlash auf Taste [F3])
Ein Problem bleibt noch offen. Die Dreiecksprozedur wurde mit **fs** gestartet, um auch größeren Dreiecken auf dem Grafikmonitor Platz zu bieten und um bei einer Hardcopy auf dem Drucker den Textcursor nicht zu sehen. Soll jetzt der Textcursor etwas auf den Schirm bringen, muß kurzzeitig mit **ts** (text screen) auf diesen Status umgeschaltet werden. Nachdem der Textcursor seine Arbeit verrichtet hat, kann wieder auf **fs** geschaltet werden. Die vollständige Zeile, die am Ende der Dreiecksprozedur eingesetzt werden muß, sieht also so aus:

```
ts setcursor [45 20] type [c=\ ] type :c fs
```


Zum Abschluß der kleinen Einführung möchte ich auf verschiedene Arten der Beschriftung von Grafiken, die schon nicht mehr so sehr nach mathematisch definierten Gebilden aussehen und in denen sogar Zufallszahlen eine Rolle spielen, eingehen.
Zunächst macht es immer einen guten Eindruck, wenn Grafiken umrandet sind. Eine Hardcopy sieht dann gleich viel gewinnender aus. Den Rahmen können wir mit **setpos** anfertigen. Zwei Vierecke werden ineinander gelegt und der Zwischenraum mit dem Befehl:

```
fill
```

aufgefüllt. Dieser Befehl ist mit Vorsicht zu genießen. Ist ein Zwischenraum nicht vollständig geschlossen, wird der gesamte Monitor ausgefüllt, und dieser Vorgang läßt sich nur durch brutales Ausschalten des Computers unterbrechen. Außerdem darf die Turtle nicht auf einem belegten Bildschirmpunkt stehen. Sie muß mit **pu** in die zu füllende Fläche gesetzt und danach mit **pd fill** in Aktion gesetzt werden. Nun zum Rahmen:

```
to Rahmen
pu setpos [-355 255] pd rt 90
repeat 2 [fd 710 rt 90 fd 510 rt 90]
pu setpos [-350 250] pd
repeat 2 [fd 700 rt 90 fd 500 rt 90]
pu setpos [-352 252] pd fill pu home pd
end
```

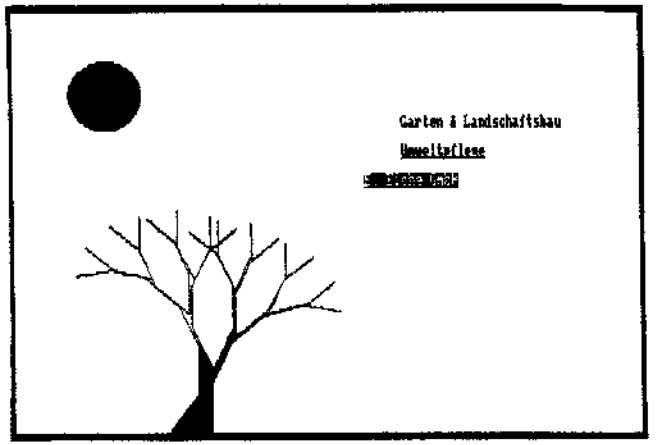
Nun zu dem, was in den Rahmen hinein soll,: ein Baum. Bei unserem Baum kann man in Grenzen mit Zufallszahlen arbeiten, wobei nach dem fill-Befehl des ganzen Bildes recht unterschiedliche Exemplare zustande kommen können. Für den Baum werden zwei Werte für Geraden **g** und Drehungen **d**, und ein Wert als Zähler für die Verschachtelungstiefe **v** gebraucht. Bei den Aufrufen der Baumprozedur werden die Strecken kontinuierlich gekürzt (Faktor 0.9), so daß der Eindruck von Ästen entsteht. Gelaufene Strecken werden genauso zurückgegangen, wobei durch die random-Zahl leichte Verschiebungen auftreten, die sich nach unten hin verbreitern:

```
to Baum :g :d :v
if :v = 0 [stop] fd :g rt :d/2
Baum :g * 0.9 :d :v - 1 lt :d fd random 3
Baum :g * 0.9 :d :v - 1 rt :d/2 fd random 3 bk :g
end
```

Wird "Baum" mit dieser Prozedur aufgerufen, so bleibt nach Fertigstellung eine Ecke des Stammes offen. Sie wird später im Zusammenhang des ganzen Bildes mit **setpos** geschlossen. Die Prozedur "Bild" setzt nun einen Rahmen, rückt den Baum an die richtige Stelle, ruft die (hoffentlich noch im Arbeitsspeicher vorhandene - sonst neuzuschreibende) Kreisprozedur auf und beschriftet die entstandene Grafik mit **setcursor**. Hier kommen auch wieder die weiter oben beschriebenen Escape- oder **type word char 27 "** Folgen zur Geltung. (p/q = invers an/aus; r/u = Unterstreichen an/aus). Was dort nun geschrieben steht, bleibt der Phantasie überlassen. Dieser Text vielleicht mal als Beispiel für einen Briefkopf:

```
to Bild :g :d :v
type word char 27 "0
cs ct fs ht
Rahmen
pu setpos [-130 -250] pd Baum :g :d :v
setpos [-180 -250] pu setpos [-155 -245] pd fill
pu setpos [-250 150] pd Kreis 40 fill
ts setcursor [55 8] type [Garten & Landschaftsbau]
setcursor [55 10] type word char 27 "r type [Umweltpflege] type word char 27 "u
setcursor [55 12] type word char 27 "p type [E. Eiche GmbH] type word char 27 "q
fs
end
```

Theoretisch könnte man in dem entstehenden Bild mehrere Faktoren dem Zufallsgenerator Logo's überlassen. So etwa den Stand der Sonne. Allerdings müßten dann in die Prozedur noch Sicherungen eingebaut werden, damit es keine Überschneidungen gibt. Aber dies alles nur zur Anregung. Wurde alles richtig eingetippt, müßte nach dem Aufruf mit **Bild 60 50 5** (andere Zahlen ausprobieren!) die Grafik entstehen, die auf der folgenden Seite abgebildet ist.
Nach der Grafik nun noch einige Tips für Anwender, die sich nicht gern beim Programmieren den Kopf zerbrechen, und doch gern Computergrafik auf dem Papier hätten.



Man kann den Zeichencursor so lange etwas auf dem Papier verrichten lassen, bis ein Tastendruck ihn unterbricht. Diese Unterbrechung kann mit der Bedingung:

```
if keyp [stop]
```

erreicht werden. Dieses **keyp** fragt ab, ob eine Taste gedrückt wurde. Wenn die Meldung darüber positiv, also "true" ist, tritt die zweite Bedingung in Kraft und die Prozedur wird abgebrochen. Sieht eine Prozedur dann folgendermaßen aus:

```
to v
fd 2 if keyp [stop] v
end
```

läuft nach dem Aufruf mit **v** der Grafikcursor so lange vorwärts, bis eine Taste gedrückt wird. Da immer nur zwei Schritte gemacht werden, wird die Linie auch nicht zu schnell gezogen. So hat der Anwender reichlich Zeit, z.B. die Leertaste zu drücken um die Bewegung aufzuhalten und andere Prozeduren aufzurufen, die ähnlich erstellt werden. So etwa für den Rückwärtsschritt oder die Rechtsdrehung:

```
to r
rt 2 if keyp [stop] r
end
```

Unterteilt man vorher den Bildschirm mit **setsplit 2**, kann man trotz großen Grafikbereichs sogar noch seine Befehle auf dem Schirm sehen. Hat man sich an seine eigenen Befehle gewöhnt, wird auch das nicht mehr nötig sein und man kann mit der Turtle umgehen, als hätte man eine "Maus" zur Verfügung.
Noch besser wäre es, die Sache umzudrehen, nämlich die Turtle so lange in eine Richtung laufen zu lassen, wie eine bestimmte Taste gedrückt bleibt.
Mit **rc** erwartet Logo eine Eingabe von Tastatur, die mit **if**-Bedingung abgefragt werden kann.

```
if rc = "f [fd 5]
```

Nur dann, wenn die Taste [F] gedrückt wird, läuft der Cursor 5 Schritte nach vorn. Dies in eine Prozedur eingebaut, die sich selbst aufruft, würde, solange [F] gedrückt bleibt, die turtle nach vorn bewegen. Es ist nun keine Schwierigkeit, die entsprechenden Schritte für andere Richtungen einzubauen:

```
to z
setsplit 2
local "a make "a rc
if :a = "v [fd 5] (also vorwärts,z=zurück,r=rechtsherum etc)
if :a = "z [bk 5]
if :a = "r [rt 5]
if :a = "l [lt 5]
z
end
```

Nach dem Aufruf mit **cs ct z** , braucht der Anwender nur den Finger auf der Taste [V] zu halten, um die Turtle nach vorn zu bewegen. Ein längerer Wechsel zwischen [V] und [R] würde die Turtle in eine Rechtskurve zwingen. Da **setsplit 2** eingeschaltet wurde, stört es nicht, wenn mit der Stoptaste die Prozedur angehalten wird, um mal eben zwischen **pd** und **pe** etc. umzuschalten. Der Wiederaufruf mit "z gestaltet sich problemlos.
Wem auch dieser Tip keine Anregung geben kann, der sollte Logo wenigstens zum Nichtstun animieren. Mit:

```
repeat 975 []
```


wäre Logo etwa 3 Sekunden lang damit beschäftigt, nichts 975 mal zu wiederholen. Dies 19500 Mal zu versuchen, würde ihn somit etwa eine Minute lang beschäftigen, ohne daß etwas dabei herauskäme. Da Logo bei Werten über 32767 meint, daß ihm die Zahl zu groß sei, (Meldung: **number too big**) muß man schon **repeat**-Anweisungen verschachteln, will man ihn für längere Zeit zum Warten verdammen. Etwa fünf Minuten Wartezeit sähen dann so aus:

```
repeat 5 [ repeat 19500 [] ]
```

Vollkommen boshafte und militante Nicht-Logo-Anwender sind somit wenigstens in die Lage gesetzt, Logo und Joyce als Eieruhr zu mißbrauchen. Der Befehl:

```
type char 7
```

läßt Logo den Monitor einmal piepsen. Und so würde nach der Anweisung:

```
repeat 5 [ repeat 19500 [] ] repeat 50 [type char 7]
```

Logo den Joyce veranlassen, nach einer Frist von 5 Minuten einen ordentlichen Alarm zu veranstalten. Für ein Nickerchen läßt sich die Zeit selbstredend beliebig ausdehnen. Als Ausrede (und dem Leser als Anregung) für die Pieperei kann die Konstruktion eines Telefongebührenzählers vorgeschoben werden.
Eine komplette, mit Eingabeabfrage gestaltete Eieruhr könnte so aussehen:

```
to Eieruhr
pr [In wieviel Minuten moechten Sie von mir benachrichtigt werden ?]
local "a make "a rq
pr [Wie oft darf ich piepsen, um Sie zu alamieren ?]
local "b make "b rq
pr [Wenn sie jetzt eine Taste druecken, laeuft die Zeit]
label "warten
if not keyp [go "warten] ct cs
(pr [Ich bin jetzt damit beschaeftigt,\ ] :a [\ Minuten auf die Uhr zu achten])
repeat :a [repeat 19500 []] (pr [Die Zeit von\ ] :a [\ Minuten ist jetzt abgelaufen !!])
repeat :b [type char 7]
end
```

Das Vorherige setzt den Anwender jetzt auch in die Lage, eine "Analogo"-Uhr zu programmieren. Der Sekundenzeiger könnte mit Vor- und Rücklauf unter **px** Status seine Bahn ziehen, getaktet über eine **repeat[]**-Anweisung. Nach einer Drehung von 360 Grad könnte er den Minutenzeiger um 6 Grad nach rechts drehen, und den Stundenzeiger nach jeweils 12 * 360 Grad um 6 Grad weiterrücken.
Das Zifferblatt müßte nach dem Zeichnen mit **savepic** abgespeichert werden und vom Programm mit **loadpic** aufgerufen werden.
Wer sich bei der Ausarbeitung eines derartigen Programms - natürlich noch mit Weckfunktion - nicht den Kopf zerbrechen möchte, kann auf die Diskette zum Buch zurückgreifen. Es wird mit **load "wecker** in den Arbeitsspeicher geholt und durch Aufruf mit: **wecker** aktiviert. Nach der Initialisierung löscht dieses Programm einige seiner Teile, die nicht mehr benötigt werden, aus dem Arbeitsspeicher. Deshalb muß nach einem Abbruch des Programms vor dem Neustart die Prozedur immer wieder neu geladen werden.

Zuletzt

Das Bisherige mag einen kleinen Einstieg in die Welt der Turtle (und des Programmierens allgemein) gegeben haben.
An dieser Stelle soll jedoch explizit darauf hingewiesen sein, daß es sich wirklich nur um ein vorsichtiges Herantasten an die tatsächlichen Fähigkeiten dieser Sprache handelt. Der Bereich der Wort und Listenverarbeitung z.B. wurde in dieser Einführung überhaupt nicht angesprochen. Wollte man sämtliche Möglichkeiten, die die Sprache Logo bietet, erörtern, wäre das Resultat ein eigenständiges Werk und würde den Rahmen des Gesamtkonzepts dieses Buches sprengen.
Die Übersicht auf den nächsten Seiten über den kompletten Befehlssatz kann dem Anwender eine kleine Hilfe sein, sich weitere Bereiche selbst zu erarbeiten. Dort wurde eine ganze Reihe von nützlichen und erforderlichen Befehlen mit einem

Asterix (*) versehen. Diese Befehle sind in den meisten Schneiderhandbüchern zum Joyce nicht aufgeführt.
Wer durch die Anregungen animiert noch tiefer in die Materie Logo einsteigen will, dem sei das bereits im Vorwort erwähnte große Logo Buch zu CPC und Joyce von Sauer empfohlen. Es erschien bei Data Becker und ist über den DMV-Verlag zum Preis von 39,- DM zu bekommen. In diesem Buch wird auch gerade die Listenverarbeitung mit Logo als ein Schwerpunkt behandelt.

```
to Logo-Kapitel
if Inhalt = nicht Gedächtnis [noch einmal lesen]
end
```

Übersicht zum Logo-Befehlssatz

(Befehle mit Asterisk (*) sind im Schneider-Systemhandbuch nicht erklärt)

Befehl	Form	Erklärung
and	and "a<"b "c>"b	Gibt "true" aus, wenn die zwei gesetzten Ausdrücke beide wahr sind. Sonst ist die Ausgabe "false". In diesem Beispiel ist die Ausgabe "true", da die Reihe des Alphabets als aufsteigend gesehen wird. "And" darf nur in Präfix-Notation verwendet werden.
arctan	arctan 1	Gibt den Arcus-Tangens aus. Die Ausgabe bei arctan 1 wäre: 45. Die Angaben erfolgen in Grad.
ascii	ascii "g	Hat die Ausgabe des Ascii-Wertes eines Zeichens zur Folge. Hier wäre die Ausgabe 103. Von Listen wird immer nur das erste Element in Asciicode übersetzt.
bf	bf "Wort	But-first, also bis auf das erste Element einer Liste (in Klammern) oder eines Wortes, werden die anderen Elemente als Liste ausgegeben. In diesem Fall: ort
bk	bk 10; bk 5-2	Back dirigiert die Turtle zurück. Im ersten Beispiel um 10 Schritte. Die Schrittangabe kann auch in Form eines zu berechnenden Ausdrucks oder einer Variablen gegeben werden.
bl	bl "Wort	But-last, also bis auf das letzte Element einer Liste oder Wortes, werden alle Elemente ausgegeben. Hier: Wor (s. bf)
bye	bye	Logo wird verlassen und auf CP/M-Ebene zurückgekehrt. Die Prozeduren im Arbeitsspeicher gehen verloren.
catch*	catch "A [Prozed 2]	Mit throw und catch werden Sprünge über Prozeduren hinaus definiert. Throw "Anfang würde in diesem Beispiel einen Sprung zu der mit catch "A bezeichneten Stelle bewirken. Die zweite Angabe zu catch beinhaltet eine Liste mit ausführbaren Kommandos. Hier etwa eine Prozed 2 genannte Prozedur aufzurufen. Die unterschiedlichen Prozeduren, in denen throw und catch zu finden sind, müssen in einem übergeordneten Zusammenhang stehen. catch "error: Tritt während eines Programmablaufs ein Fehler auf, so kann dieser durch das eingefügte catch "error dahingehend abgefangen werden, daß Logo bei einem Eingabefehler automatisch den Rücksprung in die darauf folgende Zeile vornimmt.
changef		Dieser Befehl ist in Logo zwar vorgesehen, führt beim Joyce jedoch nur zu einer Fehlermeldung.
char	char 103	Gibt das Zeichen des Ascii-Wertes aus. Hier wäre die Ausgabe "g
clean	clean	Bewirkt das Löschen des Grafikfensters.
co	co	(Continue) Nach Unterbrechung eines Programmablaufs durch pausing [F1] wird fortgefahren.
.contents	contents	Listet den Wortschatz des Arbeitsspeichers auf. Dabei sind auch Primitive wie paddle oder tones zu finden, die nicht auf dem Joyce implementiert sind.
copyoff	copyoff	Schaltet den Drucker aus.
copyon	copyon	Schaltet den Drucker ein. Sämtliche Aktionen, die über den Monitor laufen, werden protokolliert. Grafik kann so nicht erfasst werden.
cos	cos 60	Ergibt den Cosinus in Winkelgrad. Ausgabe wäre hier 0.5
count	count "Wort	Zählt die Buchstaben eines Wortes oder die Elemente einer Liste. Im zweiten Fall wäre count [a b c d e] die Ausgabe 5, im ersten Fall 4.

cs	cs	(Clear screen) Wie bei clean wird der Grafikschirm gelöscht. Zusätzlich wird der Grafikcursor in seine Ausgangsstellung gebracht, wobei die Spitze des Cursors nach oben weist. (s.a. seth 0)
ct	ct	(Clear Text) Der Textbereich wird gelöscht und der Textcursor in die obere linke Ecke seines Bereichs gesetzt.
cursor	cursor	Hat die genaue Stellung des Textcursors zur Ausgabe. Das erste Element der Liste bezeichnet die Spalte, das zweite die Zeile.
defaultd*	defaultd	Hat das Standardlaufwerk, mit dem Logo gerade arbeitet, zur Ausgabe.
define*	define "test :a	Kann innerhalb eines Programms ein neues Programm definieren, wobei die Meldung in diesem Beispiel: "test defined" unterbleiben würde. Hier würde jetzt in Test und unter dem Namen test die Prozedur a ablaufen.
dir	dir dir "m:	Ohne Laufwerksangabe werden die Logo-Files im Standart-laufwerk als Liste ausgegeben. Ansonsten wird auf das angesprochene Laufwerk zurückgegriffen.
dirpic	dirpic	Liefert die gespeicherten Bild-Dateien. (s. dir)
dot	dot [100 100]	Zeichnet einen Punkt an die durch Koordinaten bezeichnete Stelle. Koordinateneingabe muß eine Liste sein.
dotc*	dotc [100 100]	Wurde an der bezeichneten Koordinate ein Punkt gesetzt, hat dotc die Ausgabe 1. Wurde an diesem Koordinatenpunkt kein Zeichen vorgefunden, ist die Ausgabe 0.
ed	ed "test	Bereitet die Prozedur test zum Editieren vor und versetzt in den Editiermodus. Ed ohne Eingabe geht ohne weitere Angaben in den Editiermodus. Wird nach einer Fehlerausgabe ed eingetippt, wird die fehlerhafte Prozedur zum Editieren vorbereitet und der Cursor an die Stelle gerückt, in der der Fehler auftrat.
edall	edall	Bereitet alle im Arbeitsspeicher befindlichen Prozeduren zum Editieren vor und geht in den Editiermodus über.
edf*	edf "test	Lädt den File test von Diskette, bereitet ihn zum Editieren vor und geht in den Ed-Modus. Nach erfolgter Korrektur und Beendigung mit exit wird der korrigierte File wieder auf Diskette gespeichert.
emptyp*	emptyp :test	Primitive, die mit p enden, sind Prüfwörter. In diesem Fall wird festgestellt, ob der Inhalt von test leer ist. Die Ausgabe ist true oder false. emptyp [] ergibt true.
end	end	Bezeichnet das Ende einer Prozedur.
equalp*	equalp "a "a	Prüft wie das =Zeichen ob zwei Dinge gleich sind. Ausgabe ist true oder false. Die Umkehrung ist not equalp.
er	er "Nwat	Löscht die Festlegung oder Prozedur Nwat.
erall	erall	Löscht den Inhalt des gesamten Arbeitsspeichers.
erasefile	erasefile "test	Löscht den File test ohne weitere Bestätigung von der Diskette.
erasepic*	erasepic "test	Löscht die Bilddatei test von Diskette.
ern	ern "a	Löscht den globalen Namen a.
error	error	Hat eine Liste zur Ausgabe, in der ein aufgetretener Fehler genannt wird. (s.a.catch)
fd	fd 10	RÜckt den Grafikcursor hier um 10 Schritt nach vorn, wobei je nach Penileinstellung Linien markiert werden. fd 0 malt an der Cursorposition einen Punkt.
fence	fence	Begrenzt das Grafikfeld, so daß beim Versuch dieses Feld zu überschreiten die Meldung: "Turtle out of bounds" ausgegeben wird.

fill*	fill	Füllt Flächen, in die der Grafikcursor unter pu hineingesetzt wird. Vor Aktivierung von fill wird der Status pd gesetzt. Fill kann während der Ausführung nur durch Abschalten des Computers gestopt werden. Sind Flächen nicht eindeutig eingegrenzt, wird der gesamte Monitor aufgefüllt!
first	pr first "test	Hätte in diesem Beispiel t zur Ausgabe. First gibt das erste Element eines Wortes oder einer Liste aus.
fput	show fput "a [b c]	Fput braucht zwei Eingaben, aus denen es eine Liste erzeugt. Die zweite Eingabe muß in Form einer Liste vorliegen. An diese Liste wird das erste Eingabeelement vorne angehängt. Hier wäre die Ausgabe: [a b c]
fs	fs	(Full screen) Stellt den gesamten Monitor als Grafikbereich zur Verfügung. Wird jedoch nach fs nicht cs (clear screen) eingegeben, bleibt wegen eines Systemfehlers in der Bildschirmmitte immer ein zweiter Grafikcursor stehen.
glist*	glist "ausmaß	Logo unterscheidet zwischen einem Ding, seiner Eigenschaft und dem Wert der Eigenschaft. Eigenschaften werden mit pprop (s.u.) zugewiesen. Hätte vor der Abfrage mit glist die Zuweisung pprop "test "ausmaß "groß stattgefunden, hätte also der Gegenstand test die Eigenschaft eines Ausmaßes, welches groß wäre, so würde die Abfrage in unserem Beispiel [test] ergeben. Glist stellt also fest, zu welchen Dingen eine bestimmte Eigenschaft gehört. (s.a. pps pprop remprop plist gprop)
go	go "vorne	Wurde innerhalb einer Prozedur mit label "vorne eine Markierung gesetzt, so wird nach einer Zeile mit go "vorne wieder in die label-Zeile gesprungen.
gprop*	gprop "test "ausmaß	(s. Beispiel glist) gprop gibt hier den Wert der Eigenschaft ausmaß des Dinges test aus. Ausgabe wäre hier also groß. Im Zusammenhang mit den Eigenschaftslisten tauchen Bezeichnungen auf, die groß geschrieben werden und denen ein Punkt vorangestellt wird. Diese Bezeichnungen sind in erster Linie für den Logo-Interpreter von Bedeutung. So zeigt: [.PRM und eine Zahl], daß die Eigenschaft zum Grundwortschatz von Logo gehört. Zur Probe: plist "fd hat zur Ausgabe: [.PRM 7020]. So zeigt .APV, daß es sich bei der Ausgabe um den Wert einer globalen Variablen handelt. Wurde mit make "test 10 definiert, so ergiebt: gprop "test ".APV die Antwort 10.
home	home	Dirigiert den Grafikcursor in seine Ausgangsstellung, wobei die Spitze nach oben zeigt.
ht	ht	(Hide turtle) Macht den Grafikcursor unsichtbar. Dadurch wird die Geschwindigkeit bei der Abarbeitung von Prozeduren wesentlich erhöht.
if	if :a<:b [stop]	If stellt eine Bedingung. In diesem Fall die Bedingung, daß a kleiner als b ist. Trifft diese Bedingung zu,(if arbeitet mit true oder false) wird die darauf folgende Liste zur Ausführung gebracht. In diesem Fall würde eine gesamte Prozedur gestopt. Trifft die Bedingung nicht zu, wird zur nächsten Anweisung übergegangen. If keyp [stop] würde dann eine Prozedur abbrechen, wenn die Tastatur betätigt wird.
int	int 3.141	Gibt den ganzzahligen Wert einer Zahl aus. Hier wäre die Ausgabe 3
item	item 2 :tf	Mit item wird aus einer Liste ein Element herausgefiltert. Hier wäre es aus der Liste der Turtle-facts das zweite Element.

keyp*	keyp	Hier handelt es sich um ein Prüfwort, welches true ausgibt, wenn eine Taste gedrückt wird. Es ist besonders gut für von außen steuerbare Abbruchbedingungen von Prozeduren geeignet. (s. if)
label	label "vorne	Label markiert hier eine Sprungadresse für go "vorne. Die Sprünge finden innerhalb einer Prozedur statt.
last	pr last "vorne	Last hat im Gegensatz zu first das letzte Element einer Liste oder eines Wortes zur Ausgabe. In diesem Fall wäre es "e.
lc	lc "MwSt	Gibt das eingegebene Wort lediglich in kleinen Buchstaben wieder. Hier must.
list	(list "a "b "c)	Die angeführten Elemente werden zu einer Liste zusammengefasst und als solche ausgegeben. Hier [a b c]
listp*	listp :a	Untersucht wird hier, ob es sich bei dem Wert von a um eine Liste handelt. Wie bei den anderen Prüfwörtern ist die Ausgabe true oder false.
load	load "test	Eine Datei mit dem Namen test wird nach diesem Befehl in den Arbeitsspeicher geladen.
loadpic	loadpic "strip	Ein mit savepic abgespeichertes Bild wird mit loadpic wieder in den Arbeitsspeicher geladen. Hier handelt es sich um ein Bild mit dem Namen strip.
local	local "a	Die Variable a wird zur lokalen Größe erklärt, die ihren Wert nur für einen Durchlauf der Prozedur erhält, in der sie auftritt.
lput	show lput "a [b c]	Es handelt sich um die gegenteilige Funktion von fput. Hier wird an das Ende einer Liste ein Element angehängt und mit dem Resultat eine neue Liste eröffnet. Hier [b c a]
lt	lt 90	Dreht den Grafikcursor um 90 Grad nach links.
make	make "a 100	Make weist einer Variablen einen Wert zu. Hier erhält a den Wert 100.
memberp*	memberp "o "Wort	Auch hier verrät das p am Ende von memberp das Prüfwort. Abgefragt wird, ob das zuerst genannte Element in dem folgenden Wort oder der Liste enthalten ist. Zutreffend wird true ausgegeben, sonst false.
namep*	namep "a	Es wird nachgeprüft, ob in diesem Beispiel a im Arbeitsspeicher belegt ist. Ausgabe ist true oder false.
nodes	nodes	Gibt den zur Verfügung stehenden Platz im Arbeitsspeicher aus. Vor der Abfrage sollte der Speicher mit recycle behandelt werden.
not	not 2 < 1	Gibt true aus, wenn eine Aussage - wie in diesem Fall - falsch ist.
notrace*	notrace	Schaltet den mit trace erreichten Überwachungszustand ab.
nowatch*	nowatch	Schaltet den mit watch erreichten Überwachungszustand ab.
numberp*	numberp :a	Das p am Ende verrät hier das Prüfwort. Es wird nachgeprüft ob es sich beim Wert von a um eine Zahl handelt. Die Ausgabe ist true oder false. Mit numberp können Tastatureingaben, die z.B. nach rc eine Zahl erwarten, mit if-Bedingung abgefangen werden, falls sich der Anwender vertan hat und z.B. einen Buchstaben eingetippt hat.
op	op proz :a	Der Wert von a wird mit op (output) ausgegeben. In diesem Fall würde op den Wert von a an eine andere Prozedur mit dem Namen proz übergeben, die dann mit a weiterarbeitet.
or	or (1<2) (2<1)	Wenn mindestens eine der zwei Bedingungen, die hinter or stehen, wahr ist, ist die Ausgabe true. Trifft keine der Aussagen zu, ist die Ausgabe false.
pause	pause	Mit pause wird eine Prozedur unterbrochen (nicht abgebrochen). Die Prozedur kann in diesem Zustand auch verändert werden und anschließend mit co fortgesetzt werden. Bei richtiger Tastaturbelegung bewirkt [F1] auch ein pausing.

pd	pd	(Pen down) Der Zeichencursor zieht bei Bewegung eine Linie hinter sich her.
pe	pe	(Pen eraser) Der Zeichencursor löscht Linien, über die er hinweggeführt wird.
piece*	piece 2 4 tf	Aus der Liste z.B. der tf (turtle facts) würde in diesem Fall das 2.bis 4.Element aussortiert und als Liste ausgegeben.
plist*	plist "a	Wurde mit make "a 1 gesetzt, so ergibt die Frage nach a mit plist : [.APV 1] (vergl. glist). Mit plist wird also die Eigenschaft eines Dinges erfragt. .APV = associated property value ist die Bezeichnung für Eigenschaft.
po	po "test	Bringt den Programmtext der Prozedur "test" aus dem Arbeitsspeicher in das Textfenster.
poall	poall ^x	Bringt alle Programmtexte der im Arbeitsspeicher befindlichen Prozeduren ins Textfenster. Ist dieses vollgeschrieben, wartet Logo mit weiteren Eingaben, bis die Returntaste gedrückt wurde.
pons	pons	(print out names) Mit pons werden alle Namen und ihre zugehörigen Werte ausgegeben.
pops	pops	(print out procedures) Alle Prozedurtexte und Werte der Namen werden ausgegeben.
pots	pots	(print out titles) Bringt die selbstdefinierten Prozedurnamen nebst den erwarteten Eingaben.
pprop*	pprop "test "groß pprop "test "gut	(s. glist, plist) Weist einem Objekt eine Eigenschaft zu, die mit plist wieder abgefragt werden kann. plist "test ergäbe [gut groß]
pps*	pps	Die Abfrage mit pps ergibt alle Namen des Arbeitsspeichers mit den ihnen zugewiesenen Eigenschaften.
pr	pr 100 - 50	(Print) Das Ergebnis wird als Ausgabe an den Monitor gegeben.
pu	pu	(pen up) In diesem Zustand bewirkt die Bewegung des Grafikcursors nichts auf dem Monitor.
px*	px	(pen xor) Trifft der Grafikcursor bei seiner Bewegung auf gesetzte Punkte, werden diese gelöscht. Sind keine Punkte vorhanden, werden welche gesetzt.
quotient	quotient 10 4	Als Ausgabe hat quotient das ganzzahlige Ergebnis aus einem Bruche. Hier 10/4, also 2
random	random 6	Ermittelt wird hier eine Zufallszahl. Die Ausgabe bewegt sich in diesem Fall zwischen 0 und 6.
rc	rc	(read character) Nach rc erwartet Logo eine Eingabe von Tastatur, die nicht auf den Monitor ausgegeben wird, die kein Return erwartet und sofort programmintern verwertet wird.
recycle	recycle	Der Arbeitsspeicher wird von unnötig belegten nodes, die Logo für die Organisation gebraucht hat, gereinigt. Durch recycle werden oft erhebliche Speicherplatzkapazitäten frei. Es sollte deshalb vor größeren Operationen durchgeführt werden.
remainder	remainder 10 4	Hier wird der Rest der Ganzzahldivision ausgegeben. In diesem Fall: 2.
remprop*	remprop "test "a	Mit remprop wird die Eigenschaft eines Objektes wieder entfernt. Hier wird dem Objekt "test die Eigenschaft "a genommen.
repeat	repeat 9 [fd 5]	Es werden zwei Eingaben erwartet. Die erste sagt repeat, wie oft es etwas tun soll, die zweite Eingabe - eine Liste - sagt, was es tun soll. Repeat 19500 [] würde bewirken, daß die leere Liste 19500 Mal durchgeführt würde, was weiter nichts bewirken würde, als das der Computer für ca. eine Minute in einen Wartezustand versetzt würde.
rerandom*	rerandom	Bei jedem Aufruf mit random wird aus der Liste des Zufallsgenerators eine vorherbestimmte Zahl geholt. Diese Liste wird mit rerandom wieder auf ihren ersten Wert gesetzt.

rl	rl	(read list) Es wird eine Liste als Eingabe erwartet, die erst nach Drücken von Return verarbeitet wird. Das Ergebnis von rl kann gut weiterverarbeitet werden. count rl würde z.B. die Elemente einer Eingabe zählen.
round	round 3.14	Es wird auf ganze Zahlwerte gerundet. Ausgabe wäre hier 3.
rq	rq	Ähnlich wie rl, nur daß hier als Eingabe nur ein Element erwartet wird. Ebenfalls erst nach Return.
rt	rt 90	Dreht den Zeichencursor ausgehend von seiner momentanen Stellung (hier um 90 Grad) nach rechts.
run	run [a b c]	Run führt eine Befehlsliste aus. Wären a b c Prozeduren, würden sie aktiviert.
save	save "test	Sichert die im Arbeitsspeicher befindlichen und selbstgeschriebenen Prozeduren unter dem Namen Test auf Diskette des aktuellen Laufwerks.
savepic	savepic "test	Sichert eine Grafik als Bilddatei (extension .pic) unter dem Namen test auf dem Standartlaufwerk. Der Vorteil der Bildspeicherung besteht darin, daß die Grafiken mit loadpic "name wesentlich schneller auf den Monitor zu bringen sind. Außerdem sind Hardwareerweiterungen (scanner) und Software (desktop-publisher) auf dem Markt, die mit diesen pic-Dateien kompatibel sind.
se	se :a :b	Mit se können verschiedene Elemente zu einer Liste zusammengefasst werden. Will man z.B. bei setpos mit den Werten von :a und :b arbeiten, müssen diese, da setpos eine Liste mit zwei Elementen erwartet, mit se zusammengefasst werden.
setcursor	setcursor [45 15]	Hier wird der Textcursor in Spalte 45 und Zeile 15 (Bildschirmmitte) gesetzt.
setd*	setd "m:	Mit setd kann das Standartlaufwerk gewählt werden. In dieser Eingabe würde M: als solches angesprochen werden.
seth	seth 90	Von einer absoluten Stellung ausgehend (0=oben) wird die Turtle um z:B. 90 Grad gedreht. Nach seth 90 würde sie immer nach rechts weisen.
setpc*	setpc 0	Setpc kennt nur zwei Stellungen, 0 oder 1. Der Effekt bei der Grafikprogrammierung ist derselbe wie bei pe (0) oder pd (1).
setpos	setpos [0 0]	Setzt den Grafikcursor auf die durch die Koordinatenpunkte bestimmte Stelle, hier in den Bildmittelpunkt. Als Eingabe wird eine Liste erwartet.
setscrunch	setscrunch 0.5	Damit wird das Achsenverhältnis der Koordinaten geändert. In dem Beispiel wurde von der Standarteinstellung abgewichen.. Maximal kann das Verhältnis im Maßstab 1:10/10:1 (0.46875 Standard) geändert werden.
setsplit	setsplit 2	Damit wird der Monitor in einen Grafik und einen Textbereich geteilt. Die Zahl hinter setsplit gibt die Größe des Textschirms in Zeilen an.
setx	setx 10	Die Turtle wird parallel zur x-Achse des Koordinatensystems bewegt und auf den Punkt 10 dieser Achse gesetzt.
sety	sety 10	Entsprechend setx. Nach Absolvierung dieser Beispiele stände die Turtle auf dem Punkt, der auch durch setpos [10 10] erreicht worden wäre.
sf	sf	(screen facts) Berichtet über den Zustand des Bildschirms. Die Elemente der Liste bedeuten der Reihe nach: Hintergrund des Grafikfensters (0/1); Fensterauftailung (ts ss oder fs); Anzahl der Textzeilen bei ss oder setsplit; Art der Bildschirmbegrenzung (fence window oder wrap); letztlich das Achsenverhältnis (Standart 0.46875).
show	show [a b c]	Zum Kenntlich machen von Listen. Die Klammern werden mit ausgegeben. Hier: [a b c]

shuffle	shuffle [a b c d]	Wirft Elemente einer Liste nach Zufallskriterien durcheinander, und gibt sie als Liste aus. Hier z.B. [b d a c]
sin	sin 30	Ausgabe ist der Sinuswert im Gradmaß des angegebenen Winkels Hier Ausgabe: 0.5
ss	ss	(Standart split) Grafik- und Textbereich des Monitors werden nach dem Standart aufgeteilt.
st	st	(show turtle) Macht den mit ht versteckten Grafikcursor wieder sichtbar.
stop	stop	Bricht Prozeduren ab.
text	text "test	Hier wird der Text der Prozedur "test" ausgegeben.
tf	tf	(turtle facts) Liefert Information über die Zustände der Turtle. 1.u.2. Position im Koordinatensystem, 3. Richtung der Turtle (seth), 4. zeigt, ob pd, pe, px oder pu aktiviert sind. 5.: Farbe des Zeichenstifts (Joyce 0/1) 6.: Zeigt ob Turtle sichtbar ist (true oder false; st/ht)
thing	thing "a	Hat den Wert der Liste a zur Ausgabe.
throw	throw "anfang	(s. catch) Hier würde die Prozedur an die Stelle einer untergeordneten Prozedur springen, die mit catch "anfang bezeichnet wurde. Eine Prozedur kann auch wie bei einem Abbruch mit der Stoptaste verlassen werden, ohne daß die Stopped-Meldung auf dem Monitor erscheint. Dies wird durch die Vokabel "TOPLEVEL (groß schreiben!) erreicht. throw "TOPLEVEL kehrt in den Direktmodus zurück.
to	to "test	To kennzeichnet den Namen einer Prozedur. Wird im Direktmodus to "test eingegeben, stellt Logo ein eingeschränktes Editierfeld zur Verfügung, in dem nicht zeilenweise gesprungen werden kann.
towards	towards [10 10]	Richtet die Spitze des Grafikcursors auf die in der nachfolgenden Liste angegebenen Koordinatenpunkte.
trace*	trace	Es wird ein spezieller Überwachungszustand eingeschaltet, der dem Anwender zeigt, an welcher Stelle sich eine ablaufende Prozedur befindet und die Werte der Variablen angibt (s. notrace).
ts	ts	(text screen) Der gesamte Monitor wird als Textbereich genutzt
type	type [a b c]	Stellt a b c auf dem Monitor dar, ohne daß ein Zeilenumbruch erfolgt. Die Klammern werden weggelassen.
uc	uc "mwst	(upper case) Die Ausgabe des nachstehenden Wortes erfolgt in Großbuchstaben. Hier: MWST
watch*	watch	Überwachungsmodus wie trace. Es werden jedoch die Verschachtelungstiefe und momentan ausgeführte Programmzeile angezeigt. Erst nach Druck auf Return wird der nächste Befehl des beobachteten Programms ausgeführt. (s. nowatch)
where* (pr memberp "b "abc where)		Memberp würde in diesem Beispiel feststellen, ob das Element b in abc vorhanden ist. Bei positiver Beantwortung gibt where jetzt den Standort von b in abc - also 2 - aus.
window	window	Der Grafikcursor kann in diesem Modus über den sichtbaren Bereich hinaus gesteuert werden.
word	pr word "com "puter	Ähnlich wie "se" eine Liste erstellt, erzeugt word aus Elementen ein Wort. Hier "computer
wordp*	wordp :a	Als Prüfwort fragt wordp nach, ob es sich bei der Eingabe um ein Wort handelt. Ausgabe ist true oder false.
wrap	wrap	Verläßt in diesem Modus der Grafikcursor den sichtbaren Bereich des Monitors, erscheint er am gegenüberliegenden Punkt des Bildschirms.

B A S I C

Vorweg

Eine derartig ausführliche Einführung in BASIC, wie sie im vorherigen Kapitel über Logo gegeben wurde, erschien nicht nötig, da das Benutzer-Handbuch schon über viele Aspekte der Programmierung Auskunft gibt. Einiges wird jedoch selbst dem fortgeschrittenen Einsteiger nicht unbedingt klar, so daß auch grundsätzliche Dinge wie Speichern, Laden oder Aufruf von Programmen etc. bei der Arbeit mit BASIC vorweg kurz anzusprechen sind.
Hingegen ist es auf jeden Fall angebracht, die einzelnen BASIC-Befehle noch einmal aufzuführen und mit einem kleinen Beispielprogramm zu versehen, da das Benutzer-Handbuch den Anwender oft genug im unklaren darüber läßt, wie denn Befehle zu handhaben sind. Außerdem kommen bei vielen Anweisungen eine Unzahl von Einzelaspekten und Feinheiten hinzu, die im Benutzer-Handbuch gar keine Erwähnung finden oder nur sehr verwirrend dargestellt sind.
Die im folgenden alphabetisch angeführten Befehle sollen dem Anwender, der sich schon ein wenig in BASIC auskennt, eine zusätzliche Hilfe bei der Programmierung sein und dem Neuling durch die Programmbeispiele Strukturen des BASIC verdeutlichen.
Ebenso wie im Logo-Teil finden sich die Listings zu den Beispielen ready to run auf der zum Buch erhältlichen Diskette.

In vielen Fällen weiß der Programmierer bei seiner Arbeit zwar, was er erreichen will, dafür aber nicht, welcher Befehl ihn seinem Ziel näherbringt. Bei solchen Problemen soll die Zusammenstellung der BASIC-Befehle nach Einsatzgebieten weiterhelfen.
Ist das Benutzer-Handbuch in punkto Erklärung zu BASIC-Befehlen schon recht schwach, so ist die Dokumentation zu Jetsam im Kapitel 8 des Benutzer-Handbuches kaum noch eine solche zu nennen. Hier wird auf kaum 20 Seiten versucht, dem Anwender das Prinzip der Jetsamverwaltung zu erläutern.

Um diesem Manko entgegenzutreten, erwies es sich als nötig, ein wenig genauer die Programmierung mit Jetsam an beispielhaften Programmzeilen zu erklären. Außerdem wurde auf die zum Buch erhältliche Diskette das Programm "Generjet" mit aufgenommen, das dem Anwender Jetsamstrukturen erklärt und außerdem bei der Erstellung von Jetsamprogrammen interaktiv weiterhilft. Dieses Programm stellt ein wertvolles "tool" dar, ist aber leider so umfangreich, daß das Listing im Buch zu viel Platz in Anspruch genommen hätte. (vgl. dazu auch die Anleitung im Text).

Allgemeine Einführung

in BASIC

Im folgenden soll eine Einführung in die Programmiersprache BASIC gegeben werden, die sich an den absoluten Laien richtet, der zum ersten Mal an einem Computer sitzt und keine Vorkenntnisse besitzt.

Um die Programmiersprache BASIC benutzen zu können, muß sie erst einmal in den Computer geladen werden. Dies geschieht in drei Schritten:

1. Betriebssystem CP/M Plus laden
2. BASIC-Diskette einlegen
3. BASIC laden

Zu 1.: Das Laden des Betriebssystems geschieht beim JOYCE durch Anschalten des Rechners und Einlegen einer Systemdiskette in Laufwerk A:, wobei die Seite mit dem Betriebssystem zum Bildschirm zeigen muß. Bei den dem JOYCE beigefügten Disketten befindet es sich auf Seite 2.

Zu 2.: Jetzt muß die BASIC-Diskette ins Laufwerk eingelegt werden. Wenn Sie gerade das Betriebssystem geladen haben, können Sie diesen Schritt überspringen, weil sich BASIC zusammen mit dem System auf einer Diskettenseite befindet.

Zu 3.: Das eigentliche Programm wird durch Eingabe von **BASIC** und einem abschließenden Drücken der [RETURN]- oder [ENTER]-Taste geladen. Der JOYCE sucht nun das Programm auf der Diskette. Falls vorhanden, lädt er es in den Speicher und startet es.

Wenn diese drei Schritte erfolgreich absolviert wurden, müßte jetzt die Einschaltmeldung des Mallard-80 BASIC auf dem Bildschirm erscheinen. Zum Schluß steht dann:

```
OK
■
```

Wenn Sie an dieser Stelle angelangt sind, befinden Sie sich im Direktmodus von BASIC. Das heißt, daß Sie nun durch das ■ aufgefordert werden, BASIC irgendwelche Kommandos zu geben, durch die BASIC weiß, was es jetzt zu tun hat.

An dieser Stelle fangen wir am besten mit einem Miniprogramm an:

```
10 PRINT "Dies ist mein erstes Programm."
20 END
```

Sobald Sie eine Zeile fertig eingegeben haben, muß sie mit einem abschließenden Drücken der Taste [RETURN] oder [ENTER] im Speicher fixiert werden.

An diesem Beispiel sieht man, daß jede Zeile, die BASIC abarbeiten soll, mit einer Zeilennummer anfängt. Danach schließen sich dann die Kommandos an. In unserem Beispiel ist es das Kommando **PRINT**. Durch **PRINT** wird BASIC signalisiert, das dem Kommando Folgende auf dem Bildschirm auszugeben. Auf dem Bildschirm erscheint also:

```
Dies ist mein erstes Programm.
```

In Zeile 20 steht das Kommando **END**. Es sagt BASIC, daß das Programm an dieser Stelle zu Ende ist. Nach dessen Ausführung springt BASIC wieder in den Direktmodus.

Um jetzt das Programm zu starten und dessen Wirkung zu beobachten, müssen Sie **RUN** eingeben und mit [RETURN] abschließen. Das Programm wird gestartet und auf dem Bildschirm erscheint:

```
Dies ist mein erstes Programm.
Ok
■
```

Das zweite Beispielprogramm soll nun etwas komplexer als das erste ausfallen: als Neuerung kommen Variablen.


```
10 a=10
20 b=15
30 c=a*b
40 PRINT "Ergebnis: ";c
50 END
```

In Zeile 10 wird der Variablen **a** der Wert 10, in Zeile 20 der Variablen **b** der Wert 15 zugewiesen. Danach wird in Zeile 30 die Multiplikation von **a** und **b** ausgeführt und das Ergebnis der Variablen **c** zugewiesen. Zum Schluß wird das Ergebnis auf dem Bildschirm ausgegeben. Das Semikolon bewirkt dabei, daß der Wert der Variablen **c** von dem Text nicht durch zusätzliche Leerzeichen getrennt wird.

Wenn das Programm korrekt eingegeben und mit **RUN** gestartet wurde, müßte auf dem Bildschirm

```
Ergebnis:  150
Ok
■
```

erscheinen. Ist dies nicht der Fall, so haben Sie sich in irgendeiner Zeile vertippt. Um jetzt nicht die ganze Zeile noch einmal von vorne neu eingeben zu müssen, stellt BASIC das Kommando **EDIT** zur Verfügung. Sollten Sie sich also zum Beispiel in der Zeile 40 vertippt haben, so müssen Sie

```
EDIT 40
```

eingeben, gefolgt von einem [RETURN]. Auf dem Bildschirm erscheint dann die entsprechende Zeile, in der Sie mit den Cursortasten beliebig hin- und herfahren und die entsprechende Stelle ändern können. Um das Ändern etwas zu erleichtern, gibt es einen Überschreibmodus, der durch Drücken der [+]-Taste (links neben Space) eingeschaltet und durch nochmaliges Drücken wieder abgeschaltet werden kann. Beim Überschreibmodus können die alten Zeichen von neuen überschrieben werden, d.h. der nachfolgende Text wird nicht nach rechts verschoben. Wenn die Zeile korrigiert wurde, muß man wieder [RETURN] betätigen, um sie in der neuen Form abzuspeichern. Um sich davon überzeugen zu können, daß die Zeile im Programm auch wirklich geändert wurde, verwendet man den Befehl **LIST**. Er bewirkt, daß das komplette Programm auf dem Bildschirm ausgegeben wird. Zusätzlich kann man sich jedoch

auch nur einen Ausschnitt des Programms anschauen. Dazu teilt man **LIST** noch einen Zeilenbereich mit. Um zum Beispiel die Zeilen 10 bis 30 aufzulisten, wird

```
LIST 10-30
```

(wieder + [RETURN]) eingegeben. Weitere Möglichkeiten des **LIST**-Kommandos entnehmen Sie bitte der ausführlichen Erläuterung im BASIC-Kommando-Teil dieses Buches, in dem auch alle anderen in der BASIC-Einführung verwendeten Befehle erklärt werden.

Unser bisheriges Programm läßt jedoch noch einigen Komfort vermissen. Es wäre zum Beispiel sinnvoll, daß man die beiden Werte, mit denen das Programm rechnen soll, eingeben kann. Dazu müssen wir jedoch ein weiteres BASIC-Kommando kennenlernen: **INPUT**. Damit wird es möglich, Werte und Texte Variablen zuzuweisen. Wenn das Beispielprogramm entsprechend geändert wird, sieht es folgendermaßen aus:

```
10 INPUT "Wert a: ",a
20 INPUT "Wert b: ",b
30 c=a*b
40 PRINT "Ergebnis: ";c
50 END
```

Die beiden **INPUT**-Anweisungen in Zeile 10 und Zeile 20 bewirken, daß zuerst auf dem Bildschirm der Text zwischen den Anführungszeichen erscheint und man hinter dem Text die Werte eingeben kann (eine ausführlichere Beschreibung mit weiteren Optionen entnehmen Sie bitte dem Kommando-Teil dieses Buches). Wenn man das Programm mit **RUN** (+[RETURN]) startet, könnte ein Testlauf folgendermaßen aussehen:

```
Wert a: -6
Wert b: 10
Ergebnis: -60
Ok
■
```

Der nächste wichtige Schritt im Leben eines Programmierers ist, seine Arbeit dauerhaft auf einem Datenträger zu sichern. Zu diesem Zweck wird von BASIC der Befehl **SAVE** bereitgestellt. Benutzt man ihn, so wird das komplette Programm in der derzeitigen Form mit einem bestimmten Namen auf der Diskette abgespeichert. Wenn Sie das Programm zum

Beispiel unter dem Dateinamen "RECHNE" speichern wollen, so müssen Sie nur

```
SAVE "RECHNE
```

(+ [RETURN]) eingeben. Das Programm wird jetzt unter dem Dateinamen "RECHNE.BAS" auf Diskette geschrieben. Das ".BAS" besagt, daß es sich bei dieser Datei um ein BASIC-Programm handelt.

Genauso wichtig wie das Speichern von Programmen ist das spätere Laden derselben. Es geschieht mit dem Kommando **LOAD**. Um "RECHNE.BAS" wieder zu laden, genügt ein einfaches

```
LOAD "RECHNE
```

Das ".BAS" braucht man nicht mit anzugeben, da es von BASIC selbständig ergänzt wird.

Das Programm ist nun wieder zur weiteren Bearbeitung in den Speicher geladen worden.

Eine Erweiterung des Programms wird im folgenden dargestellt: es soll möglich sein, die Verknüpfungsart der Zahlen frei wählen zu können.

```
10 INPUT "Wert a: ",a
20 INPUT "Wert b: ",b
30 INPUT "Rechenart (+ , - , * , /): ",r$
40 IF r$="+" THEN c=a+b
50 IF r$="-" THEN c=a-b
60 IF r$="*" THEN c=a*b
70 IF r$="/" THEN c=a/b
80 PRINT "Ergebnis: ";c
90 END
```

Man sieht, daß das Programm um die Zeilen 30-70 ergänzt wurde. Hauptbestandteil sind dabei die **IF**-Zeilen, in denen die eingegebene Rechenart geprüft und ausgeführt wird. In der Variablen **r$** steht das Symbol der auszuführenden Rechenart, das man über die in Zeile 30 stehende **INPUT**-Anweisung eingegeben hat. Das Symbol wird mit der Bedingung ("+","-", "*", "/") verglichen. Falls diese wahr ist, wird der Variablen **c** das Ergebnis aus der Verknüpfung von **a** und **b** zugewiesen.

In dem dritten Beispiel soll nun das Berechnungsproblem der Primzahlen abgehandelt werden. Grundlage ist die Eigenschaft einer Primzahl: eine Primzahl hat keinen Teiler außer 1 und ihrer selbst. Das heißt umgekehrt, daß alle Vielfachen einer Primzahl keine Primzahlen sein können ! Beispiel: man lege sich eine Tabelle aller Zahlen ab 2 (1 ist keine Primzahl) an. Dann streiche man alle Vielfachen von 2 aus (2 bleibt stehen) und prüfe ob die Zahl 3 ausgestrichen ist. Sie ist es nicht und ist daher eine Primzahl. Als nächstes streiche man alle Vielfachen von 3 aus. Dann prüfe man die Zahl 4. Da sie ausgestrichen ist, ist sie keine Primzahl. Nun wird die Zahl 5 geprüft... usw.

```
10 DEFINT z
20 grenze=1000
30 DIM z(grenze)
40 i=2
50 IF z(i)=0 THEN PRINT i ELSE GOTO 90
60 FOR u=i TO grenze STEP i
70     z(u)=1
80 NEXT
90 i=i+1
100 IF i<=grenze THEN 50
110 END
```

In Zeile 10 wird die Feldvariable **z** als Integervariable (ganzzahlige Variable) definiert und in Zeile 20 die obere Primzahlberechnungsgrenze auf 1000 (maximal 15675) gesetzt. In Zeile 30 wird die Feldvariable **z** als ein Feld von 0 bis 1000 definiert. In Zeile 40 beginnt die eigentliche Berechnung, angefangen bei der ersten Primzahl 2. Zeile 50 prüft, ob die Variable als ein Vielfaches einer anderen Zahl definiert ist. Ist sie es, werden in den Zeilen 60-80 alle Vielfachen der Zahl als Vielfache markiert, indem die entsprechende Feldvariable auf eins gesetzt wird. In Zeile 90 wird die zu prüfende Zahl um eins erhöht und anschließend geprüft, ob die obere Grenze überschritten wurde und daher in Zeile 110 das Programm beendet werden muß.

Dieses Verfahren hat den Vorteil, das es relativ schnell arbeitet. Aber auch der Nachteil liegt klar auf der Hand: man hat eine obere Grenze, bis zu der man Primzahlen berechnen kann, weil der Speicherbereich für die Feldvariable **z** zu knapp wird.

Im vierten Beispiel treten die genannten Nachteile nicht auf, allerdings dauert es im Gegensatz zum vorherigen Beispiel mit steigender Primzahl immer länger, bis die jeweils nächste gefunden wird.

Das Prinzip dieses Verfahrens ist, daß geprüft wird, ob bis zur Hälfte der Zahl ein ganzzahliger Teiler vorhanden ist. Wenn es keinen Teiler gibt, muß es sich um eine Primzahl handeln.

```
10 i=2
20 FOR u=2 TO i/2
30     IF i/u = INT(i/u) THEN 60
40 NEXT
50 PRINT i
60 i=i+1
70 GOTO 20
```

Da durch den Sprung in Zeile 70 zum Anfang des Programms eine Endlosschleife realisiert ist, kann das Programm irgendwann durch Betätigen des [STOP]-Taste oder durch [ALT]+[C] abgebrochen werden.

Wenn Sie die Primzahlen in den beiden letzten Beispielen lieber hintereinander statt untereinander auf den Bildschirm bringen möchten, müssen Sie jeweils an das Ende der PRINT-Anweisung ein Semikolon setzen:

```
50 IF z(i)=0 THEN PRINT i; ELSE GOTO 90
```

bzw.

```
50 PRINT i;
```

Zum Schluß dieser Kurz-Einführung soll noch gezeigt werden, wie man Daten auf dem Drucker ausgibt, wenn man seine Programme bzw. Primzahlen gerne dauerhaft auf Papier haben möchte.

Das Ausgeben eines Programms ist denkbar einfach: ähnlich dem Auflisten auf dem Bildschirm mit dem **LIST**-Kommando kann das Programm mit **LLIST** auf dem Drucker ausgegeben werden. Das **LLIST**-Kommando ist genauso wie das **LIST**-Kommando zu

verwenden, was heißt, daß ohne Einschränkungen Zusätze wie die Wahl eines Zeilenbereiches verwendet werden können.

Auch das **PRINT**-Kommando wird einfach mit einem L davor, also **LPRINT**, auf den Drucker umgelenkt. Außerdem kann es wie das **LLIST**-Kommando ohne Einschränkungen gegenüber dem einfachen **PRINT** verwendet werden.

```
LLIST
```

```
LLIST 10-30
```

```
50 IF z(i)=0 THEN LPRINT i; ELSE GOTO 90
```

bzw.

```
50 LPRINT i;
```


Zusammenfassung aller BASIC-Befehle nach Funktionsgruppen

Abfrage

IF	Bedingung prüfen

Bildschirmausgabe

POS	aktuelle Position des Cursors
PRINT	Text ausgeben
SPC	Leerstellen ausgeben
TAB	auf Tabulatoren springen
WIDTH	Bildschirmzeilenbreite bestimmen
WRITE	Text ausgeben

Diskettenarbeit

OPEN	Datei eröffnen
CLOSE	Datei schließen
DIR	Directory ausgeben
FILES	Directory ausgeben
DISPLAY	Datei auflisten
TYPE	Datei auflisten
EOF	prüfen, ob Dateiende erreicht
ERA	Datei löschen
KILL	Datei löschen
FIND$	Datei finden
INPUT #	Satz aus Datei lesen
INPUT$	n Zeichen aus Datei lesen
LINE INPUT	Datei zeilenweise auslesen
PRINT #	in Datei schreiben
WRITE #	in Datei schreiben
LOAD	Programm laden
RUN	Programm laden und starten
SAVE	Programm abspeichern
LOF	Dateilänge ermitteln
NAME	Datei umbenennen
REN	Datei umbenennen
OPTION FILES	Laufwerk und Benutzernummer ändern
RESET	Diskettenlaufwerk zurücksetzen

Drucker

LLIST	Programm auf Drucker ausgeben
LPOS	aktuelle Position des Druckkopfes
LPRINT	Text auf Drucker ausgeben
WIDTH LPRINT	Druckzeilenlänge bestimmen

Fehlerbehandlung

ERL	Fehlerzeile
ERR	Fehlernummer
ERROR	Fehler erzeugen
ON ERROR GOTO	Fehlerbehandlungsroutine anspringen
RESUME	normales Programm nach Fehler fortsetzen
TRON	Programmzeilenprotokoll einschalten
TROFF	Programmzeilenprotokoll ausschalten

Formatierte Zahlenausgabe

DEC$	Zahl in bestimmtes Format umformen
HEX$	Zahl in hexadezimales Äquivalent umformen
OCT$	Zahl in oktales Äquivalent umformen
PRINT USING	Zahl mit bestimmtem Format ausgeben

Konstanten

DATA	Konstantenablage
READ	Konstanten auslesen
RESTORE	Lesezeiger ausrichten

Maschinensprache

CALL	Unterprogramm aufrufen
DEF USR	Unterprogramm definieren
USR	Unterprogramm aufrufen
INP	Port auslesen
OUT	in Port schreiben
MEMORY	Programmeinstellungen und Speichergrenze ändern
OPTION INPUT	Adresse von **Maschinenprogramm INPUT** festlegen
OPTION LPRINT	Adresse von **Maschinenprogramm LPRINT** festlegen
OPTION PRINT	Adresse von **Maschinenprogramm PRINT** festlegen
PEEK	Speicher auslesen
POKE	in Speicher schreiben
VARPTR	Speicherstelle von Variablen ermitteln

Mathematische Funktionen

ABS	Betrag
ATN	Arcus Tangens
COS	Cosinus
EXP	Exponentialfunktion
LOG	natürlicher Logarithmus
LOG10	Logarithmus zur Basis 10
MOD	Modula
SGN	Signum

SIN	Sinus
SQR	Quadratwurzel
TAN	Tangens

Programmierung

AUTO	automatische Zeilennummerierung
CHAIN	Programmverknüpfung
CHAIN MERGE	Programmverknüpfung
MERGE	Programmverknüpfung
DELETE	Programmbereich löschen
EDIT	Zeile ändern
LIST	Programmbereich auflisten
LLIST	Programmbereich ausdrucken
RENUM	Programm neu nummerieren

Programmsteuerung

CONT	Programm nach Unterbrechung fortsetzen
END	Programmende
NEW	Programm komplett löschen
OPTION RUN	Programm kann nicht mit einer Tastenkombination angehalten werden
OPTION STOP	hebt OPTION RUN auf
REM	Programmanmerkungen
RUN	Programm starten
STOP	Programm anhalten
SYSTEM	BASIC beenden und zu CP/M Plus springen

Schleifen

FOR	Schleife mit n Programmschritten
NEXT	Ende der FOR-Schleife
WHILE	Schleife solange Bedingung wahr
WEND	Ende der WHILE-Schleife

Speicherplatz

CLEAR	Variablen löschen
FRE	freien Speicherplatz ermitteln
HIMEM	obere Speichergrenze ermitteln
MEMORY	obere Speichergrenze festlegen

Sprünge

GOSUB	Unterprogramm aufrufen
RETURN	Unterprogramm beenden
GOTO	Zeile anspringen
ON x GOSUB	bestimmtes Unterprogramm aufrufen
ON x GOTO	bestimmte Zeile anspringen

Stringumwandlung in Zahl

CVD	String in doppelt genaue Zahl
CVI	String in Integerzahl
CVIK	Schlüsselstring in Integerahl
CVS	String in einfach genaue Zahl
CVUK	Schlüsselstring in Integerzahl

Stringverarbeitung

ASC	ASCII-Wert eines Zeichens ermitteln
CHR$	Zeichen mit best. ASCII-Code darstellen
INSTR	Zeichenkette in einem String suchen
LEFT$	linken Teilstring ermitteln
LEN	Länge eines Strings ermitteln
LOWER$	String in Kleinbuchstaben umwandeln
MID$	Teilstring ermitteln
RIGHT$	rechten Teilstring ermitteln
SPACE$	Leerstellen erzeugen
STR$	Umwandlung einer Zahl in einen String
STRING$	n Zeichen eines Zeichens darstellen
STRIP$	7. Bit eines Zeichens auf 0 setzen
SWAP	Inhalt zweier Variablen ohne dritte austauschen
UPPER$	String in Großbuchstaben umwandeln
VAL	Wert eines Zahlenstringes ermitteln

Voreinstellungen

COMMON	Variablen als konsistent definieren
COMMON RESET	inkonsistente Variablen löschen
DEF FN	Benutzerfunktion definieren
DEFINT	Integervariablen definieren
DEFSNG	einfach genaue Variablen definieren
DEFDBL	doppelt genaue Variablen definieren
DEFSTR	Stringvariablen definieren
DIM	Felder dimensionieren
ERASE	dimensionierte Felder löschen
OPTION BASE	Feldanfang festlegen
OPTION NOT TAB	Tabulator setzen nicht möglich
OPTION TAB	Tabulator setzen möglich

Zahlenumwandlung in String

MKD$	Zahl in doppelt langen String
MKI$	Zahl in String
MKIK$	Integerzahl in Schlüsselstring
MKS$	Zahl in String umwandeln
MKUK$	Integerzahl in Schlüsselstring

Zahlenverarbeitung

CDBL	Zahl in doppelt genaue Zahl umwandeln
CINT	Zahl in ganze Zahl umwandeln
CSNG	Zahl in einfach genaue Zahl umwandeln
FIX	Zahl in Richtung Null runden
INT	Nachkommastellen einer Zahl abschneiden
MAX	größte Zahl aus mehreren heraussuchen
MIN	kleinste Zahl aus mehreren heraussuchen
ROUND	Zahl runden
SWAP	Inhalt zweier Variablen ohne dritte austauschen
UNT	Zahl in vorzeichenlose Integerzahl umwandeln
VAL	Wert eines Zahlenstringes ermitteln

Zeicheneingabe

INKEY$	einzelnes Zeichen von der Tastatur lesen
INPUT	Zeichenkette von der Tastatur lesen
INPUT$	n Zeichen von der Tastatur lesen
LINE INPUT	komplette Zeichenfolge bis zum nächsten RETURN von der Tastatur lesen

Zufall

RANDOMIZE	Zufallsgenerator initialisieren
RND	Zufallszahl erzeugen

Detaillierte Erläuterungen aller BASIC-Befehle

ABS

ABS (x) gibt den absoluten Wert von **x** zurück. Ein negatives **x** wird durch **ABS (x)** positiv, ein positives **x** positiv übergeben.

Beispiel:

```
10 x=-1.23456
30 PRINT "ABS (";x;") ergibt";ABS (x)
40 PRINT "ABS (4-14) ergibt";ABS (4-14)
50 END
```

RUN:

```
ABS (-1.23456 ) ergibt 1.23456
ABS (4-14) ergibt 10
Ok
■
```

ADDKEY - ADDREC

Bei diesen Kommandos handelt es sich um **JETSAM**-Befehle. Genaueres erfahren Sie im **JETSAM**-Teil auf den Seiten 175 und 177.

ASC

ASC (zeichen$) übergibt den ASCII-Code des ersten Zeichens von **zeichen$**. Der String-Ausdruck **zeichen$** muß mindestens die Länge eines Zeichens haben. Falls man nicht das erste Zeichen von **zeichen$** benötigt, muß man das entsprechende Zeichen mit der Funktion **MID$** (s. dort) heraussuchen.

Beispiel:

```
10 a$="a"                                                    |
20 b$=CHR$(164)                                              | CHR$(164)=
30 c$="Test"                                                 |
40 d$=MID$(c$,2,1)                                           | d$="a" (s. MID$)
50 PRINT "ASC (";CHR$(34);a$;CHR$(34);") ergibt";ASC(a$)     | CHR$(34)="
60 PRINT "ASC (";CHR$(34);b$;CHR$(34);") ergibt";ASC(b$)     |
70 PRINT "ASC (";CHR$(34);c$;CHR$(34);") ergibt";ASC(c$)     |
80 PRINT "ASC (";CHR$(34);d$;CHR$(34);") ergibt";ASC(d$)     |
90 END                                                       |
```

RUN:

```
ASC ("a") ergibt 97                                          |
ASC (" ") ergibt 164                                         |
ASC ("Test") ergibt 84                                       | ASC("T")=84
ASC ("e") ergibt 101                                         | 2. Zeich. von Test
Ok
■
```

Die Gegenfunktion zu **ASC** ist **CHR$**. Mit ihr ist es möglich, das zum entsprechenden ASCII-Code gehörige Zeichen darzustellen (genaue Beschreibung siehe dort).

Im Anhang dieses Buches befindet sich eine vollständige ASCII-Code Tabelle, die auch den ASCII-Bereich 128-159 (Grafik-Sonderzeichen) wiedergibt.

ATN

ATN (x) übergibt den **Arcus Tangens** von **x**. **x** muß dabei größer als $-1.70141178*10^{38}$ und kleiner als $1.70141178*10^{38}$ sein. Zu beachten ist, daß sich alle Angaben auf das **Bogenmaß** beziehen.

Beispiel:

```
10 x=0.546302
20 y=-0.546302
30 PRINT "ATN (";x;") ergibt ";ATN (x)
40 PRINT "ATN (";y;") ergibt ";ATN (y)
50 END
```

RUN:

```
ATN ( 0.546302 ) ergibt  0.5
ATN (-0.546302 ) ergibt -0.5
Ok
■
```

Besonderheiten:

Die Kreisfunktion π (Pi) läßt sich wie folgt berechnen:

```
10 pi=4*ATN(1)
20 PRINT pi
```

A U T O

Durch **AUTO zeilennummer,steprate** wird BASIC veranlaßt, bei der Erstellung eines Programms nach jedem [RETURN] eine neue Zeilennummer zu erzeugen.

Für **zeilennummer** ist dabei die Zeilennummer einzusetzen, bei der die automatische Zeilennummerierung beginnen soll. Wird sie weggelassen, wird als erste Zeilennummer 10 angenommen.

Als **steprate** muß der Abstand von der letzten zur folgenden Zeilennummer angegeben werden. Wird sie weggelassen, wird als Steprate 10 angenommen. Falls das Komma, aber nicht die Steprate vorhanden ist, wird die Steprate des letzten **AUTO**-Kommandos benutzt. Ist es das erste **AUTO**-Kommando, wird auch hier 10 angenommen.
Eine Steprate von 0 ist nicht zulässig.

Nach Erscheinen der neuen Zeilennummer kann wie gewohnt programmiert werden.

Die automatische Zeilennummerierung läßt sich mit [STOP] oder mit [ALT]+[C] abbrechen.

Beispiel:

```
AUTO

AUTO 0,5

AUTO 30,

AUTO 100
```

B U F F E R S

Bei diesem Kommando handelt es sich um einen **JETSAM**-Befehl. Genaueres erfahren Sie im **JETSAM**-Teil auf Seite 168.

C A L L

Durch **CALL adr(HL,DE,BC)** ist es möglich, eigene Maschinenprogramme von BASIC aus aufzurufen.
adr muß vor dem **CALL**-Aufruf definiert werden. Es ist zweckmäßig, dies in hexadezimaler Schreibweise zu tun, z.B. adr=**&HF400**.
HL,DE,BC übergibt beim Aufruf die entsprechenden Werte den entsprechenden Doppelregistern. **HL,DE,BC** müssen ebenfalls, falls sie benutzt werden, als Variablen übergeben werden.

Beispiele:

```
10 MEMORY &HF3FF:screen=&HF400                  | obere Speichergrenze
                                                | absenken
20 FOR i=&HF400 TO &HF400+63                    |
30  READ a$                                     | Maschinenprogramm
40  wert=VAL("&H"+a$)                           | in den
50  control=control+wert                        | Speicher
60  POKE i,wert                                 | übertragen
70 NEXT                                         |
80 IF control<>5187 THEN PRINT "DATAFEHLER !!!":END |
90 CALL screen                                  | Programm aufrufen
100 END
110 DATA 0E,09                                  | LD C,9
120 DATA 11,09,F4                               | LD DE,text
130 DATA CD,05,00                               | CALL BDOS
140 DATA C9                                     | RET
150 DATA 1B,45,1B                               | text: 1BH,'E',1BH,
160 DATA 48,1B,59,30                            | 'H',1BH,'Y',30H,
170 DATA 37,44,69,65,73,20,69,73,74,20,65,69,6E | 37H,'Dies ist ein
180 DATA 20,60,69,74,20,43,41,4C,4C,20,61,75,66 | mit CALL auf
190 DATA 67,65,72,75,66,65,6E,65,73,20,50,72,6F | gerufenes Pro
200 DATA 67,72,61,60,6D,2E,0D,0A,24             | gramm.',0DH,0AH,'$'
```

RUN:

```
Dies ist ein mit CALL aufgerufenes Programm.
Ok
■
```

C D B L

Die Funktion **CDBL (x)** gibt für den numerischen Ausdruck **x** eine doppelt genaue Zahl zurück.
In der Praxis hat diese Funktion jedoch keine Bedeutung, da sie nicht hinreichend genau arbeitet.

■

Beispiel:

```
10 a=1.234                          |
20 b#=CDBL (a)                      | Das # steht für die Kennzeichnung einer
30 PRINT "Aus";a;"wird";b#          | doppeltgenauen Variable
40 END
```

RUN:

```
Aus 1.234 wird 1.233999967575073
Ok
■
```

C H A I N

Mit **CHAIN datei$,zeilennummer,ALL** läßt sich aus einem laufenden Programm ein anderes anbinden, wobei alle Variablen erhalten bleiben können.

Die Variable **datei$** gibt das zu ladende Programm an. Ist in **datei$** keine Extension zugeordnet, wird **.BAS** verwendet.

zeilennummer gibt die Zeile an, bei der nach dem Laden das Programm beginnen soll. Wird **zeilennummer** nicht angegeben, beginnt die Ausführung mit der niedrigsten Zeilennummer, bei **zeilennummer=65535** geht das Programm in das Betriebssystem CP/M+ zurück.

Bei Angabe von **ALL** bleiben alle Variablen des laufenden Programms erhalten. Wird **ALL** nicht angegeben, werden alle Variablen gelöscht, sofern sie nicht mit **COMMON** entsprechend gesichert wurden.

Besonderheiten:

- alle Zeilen des aktuellen Programms werden gelöscht
- alle Einstellungen für **DEFINT**, **DEFSNG**, **DEFDBL** und **DEFSTR** werden gelöscht
- die **OPTION BASE**-Spezifikation wird gelöscht, wenn nicht Felder in das neue Programm übertragen wurden.
- alle Benutzerfunktionen (**DEF FN**) werden vergessen
- **ON ERROR GOTO** wird abgeschaltet
- offene Dateien werden nicht geschlossen
- **RESTORE** wird ausgeführt
- alle aktiven **FOR-**, **WHILE-** und **GOSUB**-Kommandos werden vergessen

Beispiele:

```
CHAIN "TEST"                        | Das Programm TEST.BAS wird zum aktiven
                                    | Programm herangebunden und gestartet. Alle
                                    | Variablen sind gelöscht.

CHAIN "TEST",,ALL                   | Das Programm TEST.BAS wird zum aktiven
                                    | Programm herangebunden und gestartet. Alle
                                    | Variablen bleiben bestehen.

CHAIN "TEST",10,ALL                 | Das Programm TEST.BAS wird zum aktiven
                                    | Programm herangebunden und  ab Zeile 10
                                    | gestartet. Alle Variablen bleiben beste-
                                    | hen.
```

C H A I N M E R G E

Mit **CHAIN MERGE datei$,zeilennummer,ALL,DELETE zeilennummerbereich** läßt sich an ein laufendens Programm ein anderes anbinden.

Die Variable **datei$** gibt das zu ladende Programm an. Ist in **datei$** keine Extension zugeordnet, wird **.BAS** verwendet.

zeilennummer gibt die Zeile an, bei der nach dem Laden das Programm weitergeführt werden soll. Wird **zeilennummer** nicht angegeben, beginnt die Ausführung mit der niedrigsten Zeilennummer, bei **zeilennummer=65535** geht das Programm in das Betriebssystem CP/M+ zurück.

Bei Angabe von **ALL** bleiben alle Variablen des laufenden Programms erhalten. Wird **ALL** nicht angegeben, werden alle Variablen gelöscht, sofern sie nicht mit **COMMON** entsprechend gesichert wurden.

DELETE zeilennummerbereich veranlaßt die Löschung der mit **DELETE** angegebenen Zeilen, bevor das neue Programm geladen wird. **zeilennummerbereich** spezifiziert alle Zeilen, deren Nummern im gegebenen Bereich liegen. Dieser Bereich kann eines der folgenden Formate annehmen:

zeilennummer	nur diese Nummer fällt in den Bereich
zeilennummer-zeilennummer	Zeilen von der ersten Nummer bis einschließlich der letzten
zeilennummer-	Zeilen von der bestimmten Zeile bis zum Ende des Programms
-zeilennummer	Zeilen vom Programmanfang bis zu der bestimmten Zeile

Besonderheiten:

- nur die in **DELETE** genannten Zeilen werden gelöscht
- alle Einstellungen für **DEFINT**, **DEFSNG**,**DEFDBL** und **DEFSTR** **bleiben erhalten**
- **OPTION BASE**-Spezifikation bleibt erhalten
- alle Benutzerfunktionen (**DEF FN**) werden vergessen
- **ON ERROR GOTO** wird abgeschaltet
- offene Dateien werden nicht geschlossen
- **RESTORE** wird ausgeführt
- alle aktiven **FOR**-, **WHILE**- und **GOSUB**-Kommandos werden vergessen

Beispiele:

```
CHAIN MERGE "TEST",100,,DELETE 100-200     | Die Zeilen 100 bis 200 werden gelöscht. Das
                                           | Programm TEST.BAS wird herangebunden und ab
                                           | Zeile 100 gestartet. Die Variablen werden
                                           | gelöscht.

CHAIN MERGE "M:TEST.BLD",,ALL,DELETE 1000- | Das aktive Programm wird ab Zeile 1000
                                           | gelöscht. Das Programm TEST.BLD wird
                                           | herangebunden. Die Variablen bleiben
                                           | erhalten.
```

C H R $

Die Funktion **CHR$ (x)** übergibt das dem ASCII-Code **x** zugehörige Zeichen. Das ganzzahlige **x** muß dabei zwischen 0 und 255 liegen.

Zeichen, deren ASCII-Code kleiner als 32 ist, können mit **CHR$(27)+CHR$(x) dargestellt werden.**

CHR$ (x) verhält sich zur Funktion **ASC (x$)** umgekehrt.
Eine vollständige ASCII-Code-Tabelle (ink. Grafik-Sonderzeichen) befindet sich im Anhang des Buches.

Beispiel:

```
10 a=97
20 b=164
30 c=84
40 PRINT "CHR$ (";a;") ergibt ";CHR$(a)
50 PRINT "CHR$ (";b;") ergibt ";CHR$(b)
60 PRINT "CHR$ (";c;") ergibt ";CHR$(c)
70 END
```


RUN:

```
CHR$ ( 97 ) ergibt a
CHR$ ( 164 ) ergibt
CHR$ ( 84 ) ergibt T
Ok
■
```

C I N T

Die Funktion **CINT (x)** rundet den Wert **x** auf eine Integerzahl. **x** muß dabei zwischen -32768 und +32767 liegen.
Werden diese Grenzen nicht beachtet, bricht BASIC das Programm mit der Fehlermeldung **Improper argument** ab.

Beispiel:

```
10 a=123.456
20 b=-456.789
30 PRINT "CINT (";a;") ergibt ";CINT (a)
40 PRINT "CINT (";b;") ergibt ";CINT (b)
50 END
```

RUN:

```
CINT ( 123.456 ) ergibt  123
CINT (-456.789 ) ergibt -457
OK
■
```

C L E A R

CLEAR,High-Memory,Stack-Größe, Dateizahl, maximale Satzlänge löscht alle Variablen, vergißt alle aktuellen **FOR**-, **WHILE**- und **GOSUB**-Kommandos, schließt alle offenen Dateien und verändert die angegebenen Parameter.

High-Memory bezieht sich auf die obere vom BASIC benutzbare Speichergrenze; **High-Memory** muß somit in der Form einer Adresse eingegeben werden. Wird diese Angabe weggelassen, bleibt der aktuelle Wert eingestellt.

Stack-Größe gibt die Anzahl der von BASIC frei für das Stapelregister verwendbaren Bytes an. Der Stack ist der Speicherbereich, der von BASIC für die Verwaltung von Schleifen, **GOSUB**-Routinen, etc. benötigt wird. Falls **Stack-Größe** angegeben wird, muß mindestens ein Wert von 256 verwendet wer-

den. Wird diese Angabe weggelassen, bleibt der aktuelle Wert eingestellt.

Dateizahl gibt die maximale Anzahl gleichzeitig zu bearbeitender Dateien an. Ohne diese Angabe bleibt der aktuelle Wert eingestellt.

maximale_Satzlänge gibt die maximale Größe eines frei adressierbaren Satzes an. Ohne diese Angabe bleibt der aktuelle Wert eingestellt.

Die Standardeinstellungen:

High-Memory	&HF605
Stack-Größe	512 Bytes
Dateizahl	3
max._Satzlänge	128 Bytes

Beispiel:

```
CLEAR                    | nur Variablen werden gelöscht

CLEAR,&HF605,,6          | alle Variablen werden gelöscht, High-Memory auf
                         | &HF605, 6 Dateien

CLEAR,&HEFFF,512,3,256   | alle Variablen werden gelöscht, High-Memory auf
                         | &HEFFF, Stack-Größe auf 512 Bytes, 3 Dateien,
                         | 256 Bytes maximale Satzlänge

CLEAR,,,10               | alle Variablen werden gelöscht, 10 Dateien

10 a=100                 | Initialisierung
20 b=200                 |
30 PRINT "Vor CLEAR:"    |
40 PRINT a;b             | Kontrollabfrage: a,b erhalten
50 CLEAR                 | alle Variablen löschen
60 PRINT "Nach CLEAR:"   |
70 PRINT a;b             | Kontrollabfrage: a,b gelöscht
80 END                   |
```

RUN:

```
Vor CLEAR:
 100  200
Nach CLEAR:
 0  0
```

Hinweis: Ein weiteres Kommando zum Ändern der Parameter ist **MEMORY**. Hierbei werden jedoch keine Variablen gelöscht.

C L O S E

Durch **CLOSE dateinummerliste** werden die mit **dateinummerliste** spezifizierten Dateien geschlossen.

dateinummerliste ist in der Form **#nr_a,#nr_b,#nr_c,..** anzugeben. Wird **dateinummerliste** weggelassen, werden sämtliche offenen Dateien geschlossen.

Beispiel:

```
10 OPEN "O",#1,"M:TEST1"                          | Datei öffnen: #1
20 OPEN "O",#2,"M:TEST2"                          | Datei öffnen: #2
30 PRINT #1,"Dies ist eine Testzeile für Datei 1." | Test für #1
40 PRINT #2,"Dies ist eine Testzeile für Datei 2." | Test für #2
50 CLOSE #1,#2                                    | #1, #2 schließen
60 TYPE M:TEST1                                   | Inhaltskontrolle #1
70 TYPE M:TEST2                                   | Inhaltskontrolle #2
80 END                                            | Ende
```

RUN:

```
Dies ist eine Testzeile für Datei 1.
Dies ist eine Testzeile für Datei 2.
Ok
■
```

C O M M O N

Mit **COMMON variablenliste** können Variablen spezifiziert werden, die während der Ausführung von **COMMON RESET**, **CHAIN** oder **CHAIN MERGE** nicht gelöscht werden sollen. **variablenliste** enthält dabei Variablen jeglichen Typs.
Die Ausführung eines **COMMON**-Befehles im Direktmodus (Zeitpunkt, zu dem man ein Programm eingeben kann; also die Zeit, zu der das Programm nicht abläuft) hat keine Wirkung.

Beispiel:

```
10 COMMON a,a$                          | a,a$ als konsistent definieren
20 a=100                                | Wertzuweisung
30 a$="Test_1"                          | Wertzuweisung
40 b=200                                | Wertzuweisung
50 b$="Test_2"                          | Wertzuweisung
60 PRINT "Vor COMMON RESET: ";a;a$;b;b$ | Kontrolle: a,a$,b,b$          Ok
70 COMMON RESET                         | Inkonsistente Variablen löschen
80 PRINT "Nach COMMON RESET:";a;a$;b;b$ | Kontrolle: a,a$ Ok; b,b$      gelöscht
90 END
```

RUN:

```
Vor COMMON RESET:  100 Test_1 200 Test_2
Nach COMMON RESET: 100 Test_1 0
Ok                 |    |    |    |
■                  a----a$   b---b$
                     |         |
                 gesichert  ungesichert
```

C O M M O N R E S E T

Durch Verwendung von **COMMON RESET** werden alle Variablen gelöscht, die nicht durch **COMMON** spezifiziert wurden. Außerdem werden alle aktiven **FOR-**, **WHILE-** und **GOSUB-**Kommandos vergessen.

Beispiel:

```
10 COMMON a,a$                          | a,a$ als konsistent definieren
20 a=100                                | Wertzuweisung
30 a$="Test_1"                          | Wertzuweisung
40 b=200                                | Wertzuweisung
50 b$="Test_2"                          | Wertzuweisung
60 PRINT "Vor COMMON RESET: ";a;a$;b;b$ | Kontrolle: a,a$,b,b$            Ok
70 COMMON RESET                         | Inkonsistente Variablen löschen
80 PRINT "Nach COMMON RESET:"a;a$;b;b$  | Kontrolle: a,a$ Ok; b,b$        gelöscht
90 END
```

RUN:

```
Vor COMMON RESET:  100 Test_1 200 Test_2
Nach COMMON RESET: 100 Test_1 0
Ok                 |    |    |    |
■                  a----a$   b---b$
                     |         |
                 gesichert  ungesichert
```

C O N S O L I D A T E

Bei diesem Kommando handelt es sich um einen **JETSAM**-Befehl. Genaueres erfahren Sie im **JETSAM**-Teil auf Seite 172.

C O N T

Wenn das Programm durch irgendeinen Umstand (Break (durch [ALT]+[C] oder [STOP]), **STOP**, **END**, Error) angehalten wurde, kann es durch Eingabe von **CONT** im Direktmodus fortgesetzt werden. Falls das Programm in dieser Pause geändert wurde, wird das **CONT**-Kommando mit der Fehlermeldung **Cannot CONTinue** zurückgewiesen.

Beispiel:

```
10 PRINT "Dies ist ein Programm."
20 STOP
30 PRINT "Dies ist das durch CONT fortgesetzte Programm."
40 END
```

RUN:

```
Dies ist ein Programm.
Break in 20
Ok
■
CONT (muß von Ihnen eingegeben werden)
Dies ist das durch CONT fortgesetzte Programm.
Ok
■
```

C O S

COS (x) übergibt den Cosinus von **x**. **x** muß dabei größer als -205884.2734375 und kleiner als 205884.2734375 sein.
Zu beachten ist, daß sich alle Angaben auf das **Bogenmaß** beziehen.

Beispiel:

```
10 x=3.1415927
20 y=6.2831853
30 PRINT "COS (";x;") ergibt ";COS (x)
40 PRINT "COS (";y;") ergibt ";COS (y)
50 END
```

RUN:

```
COS ( 3.1415927 ) ergibt -1
COS ( 6.2831853 ) ergibt  1
Ok
■
```

C R E A T E

Bei diesem Kommando handelt es sich um einen **JETSAM**-Befehl. Genaueres erfahren Sie im **JETSAM**-Teil auf Seite 170.

C S N G

CSNG (x) übergibt den einfach genauen Wert von **x**.

Beispiel:

```
10 x#=0.12345678901234#                  | #=Kennzeichnung doppelt genaue Zahl
20 PRINT "CSNG (";x#;") ergibt";CSNG (x#) |
30 END
```

RUN:

```
CSNG ( 0.12345678901234 ) ergibt 0.1234568
Ok
■
```

C V D

CVD (zahl$) wandelt die durch **MKD$** erzeugte Zeichenkette **zahl$** in eine doppelt genaue Zahl zurück.

Beispiel:

```
10 a#=0.1234567890123456#              | #=Kennzeichnung doppelt genaue Zahl
20 zahl$=MKD$ (a#)                     |
30 PRINT "CVD (";CHR$(34);zahl$;CHR$(34);") ergibt";CVD (zahl$)
40 END
```


RUN:

```
CVD ("Ê¯7¯360") ergibt 0.1234567890123456
Ok       |                    |
■        |                    |
         |                    |
Dies ist der durch MKD$ (s. dort)| erzeugte 8 Byte lange String. Die hier angezeigten
Zeichen entsprechen den ASCII-Ze | ichen der einzelnen Bytes.
                                 |
         Dies ist die durch CVD aus dem String zurückgewandelte doppelt genaue Zahl.
```

C V I

CVI (integerzahl$) wandelt die durch **MKI$** erzeugte Zeichenkette **integerzahl$** in eine Integerzahl zurück.

Beispiel:

```
10 a%=-1                       | !=Kennzeichnung einer den Integerbereich überschreitenden
20 b%=32767!                   |   Zahl
30 zahl1$=MKI$ (a%)
40 zahl2$=MKI$ (b%)
50 PRINT "CVI (";CHR$(34);zahl1$;CHR$(34);") ergibt ";CVI (zahl1$)
60 PRINT "CVI (";CHR$(34);zahl2$;CHR$(34);") ergibt ";CVI (zahl2$)
70 END
```

RUN:

```
CVI ("ÿÿ") ergibt -1
CVI ("ß0") ergibt  32767
Ok     |              |
■      |              |
       |              |
Dies ist der durch M | KI$ (s. dort) erzeugte 2 Byte lange String. Die hier
angezeigten Zeichen  | entsprechen den ASCII-Zeichen der einzelnen Bytes.
                     |
         Dies ist die durch CVI aus dem String zurückgewandelte Integerzahl.
```

C V I K

CVIK (integerzahl$) wandelt die durch **MKIK$** erzeugte Zeichenkette **integerzahl$** in eine Integerzahl zurück.

Beispiel:

```
10 a!=-1                    | !=Kennzeichnung einer den Integerbereich überschreitenden
20 b!=32767!                |   Zahl
30 zahl1$=MKIK$ (a!)
40 zahl2$=MKIK$ (b!)
50 PRINT "CVIK ("+CHR$(34)+zahl1$+CHR$(34)+") ergibt "+CVIK (zahl1$)
60 PRINT "CVIK ("+CHR$(34)+zahl2$+CHR$(34)+") ergibt "+CVIK (zahl2$)
70 END
```

RUN:

```
CVIK ("Ø▒") ergibt -1
CVIK ("▒▒") ergibt  32767
Ok     |             |
■      |             |
       |             |
Dies ist der durch M | KIK$ (s. dort) erzeugte 2 Byte lange String. Die hier
angezeigten Zeichen  | entsprachen den ASCII-Zeichen der einzelnen Bytes.
                     |
        Dies ist die durch CVIK aus dem String zurückgewandelte Integerzahl.
```

C V S

CVS (zahl$) wandelt die durch **MKS$** erzeugte Zeichenkette **zahl$** in eine einfach genaue Zahl zurück.

Beispiel:

```
10 a=12345.67
20 b=-12345.67
30 zahl1$=MKS$ (a)
40 zahl2$=MKS$ (b)
50 PRINT "CVS ("+CHR$(34)+zahl1$+CHR$(34)+") ergibt "+CVS (zahl1$)
60 PRINT "CVS ("+CHR$(34)+zahl2$+CHR$(34)+") ergibt "+CVS (zahl2$)
70 END
```


RUN:

```
CVS ("¿ê▒▒") ergibt  12345.67
CVS ("¿êA▒") ergibt -12345.67
Ok     |             |
■      |             |
       |             |
Dies ist der durch MKS$ | (s. dort) erzeugte 4 Byte lange String. Die hier ange-
zeigten Zeichen entspre | chen den ASCII-Zeichen der einzelnen Bytes.
                        |
        Dies ist die durch CVS aus dem String zurückgewandelte einfach genaue Zahl.
```

C V U K

CVUK (integerzahl$) wandelt die durch **MKUK$** erzeugte Zeichenkette **integerzahl$** in eine Integerzahl zurück.

Beispiel:

```
10 a=12345
20 b=54321!
30 zahl1$=MKUK$ (a)
40 zahl2$=MKUK$ (b)
50 PRINT "CVUK ("+CHR$(34)+zahl1$+CHR$(34)+") ergibt"+CVUK (zahl1$)
60 PRINT "CVUK ("+CHR$(34)+zahl2$+CHR$(34)+") ergibt"+CVUK (zahl2$)
70 END
```

RUN:

```
CVUK ("09") ergibt 12345
CVUK ("11") ergibt 54321
Ok     |             |
■      |             |
       |             |
Dies ist der durch M | KUK$ (s. dort) erzeugte 2 Byte lange String. Die hier
angezeigten Zeichen  | entsprachen den ASCII-Zeichen der einzelnen Bytes.
                     |
        Dies ist die durch CVUK aus dem String zurückgewandelte Integerzahl.
```

D A T A

Dieser Befehl wird in der Form **DATA konstantenliste** verwendet. **konstantenliste**, die eine programminterne Ablage von Konstanten darstellt, kann sich dabei aus Zahlen oder Buchstaben zusammensetzen. Alle aufeinanderfolgenden Elemente müssen dabei durch Komma getrennt werden. Leerzeichen vor oder nach einem Element werden ignoriert.
Falls in einem String Kommas oder Doppelpunkte vorkommen, muß der String mit Anführungszeichen eingegeben werden, da BASIC sonst das Komma bzw. den Doppelpunkt als Ende des Strings interpretiert. Das abschließende Anführungszeichen kann weggelassen werden, wenn es das letzte Zeichen einer Zeile ist.

Beispiel:

```
10 RESTORE 130                          | DATA-Zeiger auf Zeile 130 richten
20 FOR i=1 to 3
30  READ a
40  PRINT a
50 NEXT
60 PRINT
70 RESTORE 140                          | DATA-Zeiger auf Zeile 140 richten
80 FOR i=1 to 4
90  READ a$
100  PRINT a$
110 NEXT
120 END
130 DATA 10,-1.234,-97531
140 DATA Test,Autobahnraststätte
150 DATA "Kommatest,Doppelpunkttest:",*6,19,23,30,36,45 / 5
```

RUN:

```
 10
-1.234
-97531

Test
Autobahnraststätte
Kommatest,Doppelpunkttest:
6,19,23,30,36,45 / 5
Ok
■
```


D E C $

DEC$(a,format$) wird für die Umwandlung der Zahl **a** in das durch **format$** angegebene Format verwendet. a ist dabei ein beliebiger numerischer Ausdruck.

Für **format$** sind verschiedene Optionen einsetzbar:
Jedes **#** repräsentiert die Position einer Ziffer. In **format$** muß mindestens ein # enthalten sein.
. legt die Position des Dezimalpunktes fest. In **format$** darf höchstens ein . enthalten sein.
Ein **,** bewirkt die Aufteilung der Vorkommastellen in Dreiergruppen, die durch Komma getrennt werden. Das , muß vor den . gesetzt werden.
Mit **$$** kann man vor die erste Ziffer bzw. den Dezimalpunkt ein $ setzen. Die $$ müssen vor dem Zahlenformat stehen.
Bei Verwendung von ****** werden führende Leerstellen mit Sternen aufgefüllt. Die ** müssen vor dem Zahlenformat stehen.
Durch ****$** werden $$ und ** kombiniert. Auch ****$** muß vor dem Zahlenformat stehen.
Mit **+** wird das Vorzeichen der Zahl ausgegeben. Steht + vor dem Zahlenformat, wird das Vorzeichen vor die Zahl oder das $ gestellt. Steht + am Ende des Zahlenformats, wird das Vorzeichen nachgestellt.
Ein **-** bewirkt die nachgestellte Ausgabe eines Minuszeichens oder eines Leerzeichens für positive Zahlen. - muß nach dem Zahlenformats stehen.
Bei Verwendung von **↑↑↑↑** wird die Zahl in Exponentialdarstellung ausgegeben. ↑↑↑↑ muß direkt hinter dem Zahlenformat stehen.

Beispiel:

```
10 DEFDBL a                             | a=doppelt genau
20 a=-123456.789#                       | # wird von BASIC gesetzt
30 PRINT DEC$(a,"########")             | im folgenden werden die
40 PRINT DEC$(a,"########.######")      | verschiedenen Formate durch-
50 PRINT DEC$(a,"########,.######")     | getestet
60 PRINT DEC$(a,"$$########,.######")   |
70 PRINT DEC$(a,"$$########.######")    |
80 PRINT DEC$(a,"**########,.######")   |
90 PRINT DEC$(a,"**########.######")    |
100 PRINT DEC$(a,"**$########,.######")  |
110 PRINT DEC$(a,"**$########.######")   |
120 PRINT DEC$(a,"+$$########,.######")  |
130 PRINT DEC$(a,"$$########.######+")   |
140 PRINT DEC$(a,"$$########.######-")   |
150 PRINT DEC$(a,"$$########.######↑↑↑↑-")|
160 END
```

RUN:

```
 -123457
 -123456.789000
 -123,456.789000
  -$123,456.789000
  -$123456.789000
***-123,456.789000
***-123456.789000
***-$123,456.789000
***-$123456.789000
   -$123456.789000
   $123456.789000-
   $123456.789000-
$123456789.0000000-03-
Ok
■
```

D E F F N

Mit **DEF FNx(a,b,c,...)=funktion(a,b,c,...)** (**DEF**ine **FN**ktion) läßt sich eine Funktion definieren, die bei späterer Gelegenheit universell aufgerufen werden kann. Prinzipiell gibt es vier verschiedene Grundmöglichkeiten, das **DEF FN**-Kommando einzusetzen:

Einfache numerische Verwendung
`DEF FNx=a*b-c+d`: Nach Aufruf durch **wert=FNx** wird **a*b-c+d** ausgerechnet und **wert** übergeben.

Numerische Verwendung mit Variablenübergabe
`DEF FNx(y)=(y+1)↑2`: Nach Aufruf durch **wert=FNx(y)** wird die Funktion **(y+1)↑2** ausgerechnet und das Ergebnis **wert** übergeben.

Einfache Stringverwendung
`DEF FNx$=a$+b$`: Nach Aufruf durch **wert$=FNx$** wird ein String, der aus **a$+b$** besteht, an **wert$** übergeben.

Stringverwendung mit Variablenübergabe
`DEF FNx$(y)=a$(y)+b$(y)`: Nach Aufruf durch **wert$=FNx$(y)** wird ein String, der aus **a$(y)+b$(y)** besteht, **wert$** übergeben.

Der Typ der Funktion muß mit dem Typ der Variable kompatibel sein (numerisch oder String). Im übrigen kann die Funktion auch andere Funktionen aufrufen. Im Direktmodus ist das Kommando ungültig.

Beispiel:

```
10 DEF FNa(x)=x^2-4*x+2                                          | verschiedene
20 DEF FNb(x)=FNa(x)+10                                          | Funktionen defi-
30 DEF FNy$(x)=STRING$(x,"J")                                    | nieren
40 DEF FNz$(x,y)=STR(NG$(x,CHR$(y))                              |
50 DEF FNlocate$(x,y)=CHR$(27)+"Y"+CHR$(32+x)+CHR$(32+y)         | (Cursor positionieren)
60 PRINT CHR$(27)+"E"+CHR$(27)+"H"                               | Bildschirm löschen
70 PRINT "FNa(2)=";FNa(2)                                        | Funktionen mit
80 PRINT "FNb(2)=";FNb(2)                                        | Werten testen
90 PRINT "FNy$(10)=";FNy$(10)                                    |
100 PRINT "FNz$(10,255)=";FNz$(10,255)                           |
110 PRINT "FNlocate$(0,0)+FNz$(10,255)=";FNlocate$(0,0)+FNz$(10,255)
120 END
```

RUN:

```
(Bildschirm gelöscht)
FNa(2)=-2
FNb(2)= 8
FNy$(10)=JJJJJJJJJJ
FNz$(10,255)=
FNlocate$(0,0)+FNz$(10,255)=

Ok
■
```

D E F U S R

DEF USR nummer=adresse legt bis zu 10 (0-9) verschiedene Maschinen-Unterprogramme fest. **nummer** bezeichnet dabei das Unterprogramm, **adresse** den entsprechenden Beginn desselben. Wird **nummer** nicht angegeben, wird der Wert 0 angenommen. Das Unterprogramm wird durch die Funktion **USR** aufgerufen.

Beispiele:

```
10 MEMORY &HF3FF:DEF USR 1=&HF400
20 FOR i=&HF400 TO &HF400+62
30  READ a$
40  wert=VAL("&H"+a$)
50  POKE i,wert
60 NEXT
70 a=USR 1(b) 'a und b stellen Hilfsvariablen dar, um das Maschinenunterprogramm
              'über die USR-Funktion zu starten.
80 END
90 DATA 0E,09                                                    | LD C,9
100 DATA 11,09,F4                                                | LD DE,text
110 DATA CD,05,00                                                | CALL BDOS
```

```
120 DATA C9                                          | RET
130 DATA 1B,45,1B,48                                 | text: 1BH,'E',1BH,'H',
140 DATA 1B,59,30,37,44,69,65,73,20                  | 1BH,'Y',30H,37H,'Dies
150 DATA 69,73,74,20,65,69,6E,20,6D,69,74,20,55,53,52| ist ein mit USR
160 DATA 20,61,75,66,67,65,72,75,66,65,6E,65,73,20,50|  aufgerufenes P
170 DATA 72,6F,67,72,61,6D,6D,2E,0D,0A,24            | rogramm.',0DH,0AH,'$'
```

RUN:

```
Dies ist ein mit USR aufgerufenes Programm.
Ok
■
```

D E F D B L

Mit DEFDBL buchstabenbereich lassen sich die mit **buchstabenbereich** bezeichneten Variablen als doppelt genaue Variablen vordefinieren.
buchstabenbereich muß entweder in der Form eines einzelnen Buchstabens eingegeben werden oder in der Form einer Buchstabenliste, also Buchstabe1-Buchstabe2 (einschließlich).
In der Praxis hat dieses Kommando jedoch keine Bedeutung, da Mallard-80 BASIC mit doppelt genauen Zahlen nicht hinreichend genau arbeitet.

Beispiel:

```
DEFDBL b                  | b ist doppelt genau

DEFDBL a-z                | die Variablen a bis z sind doppelt genau

10 DEFDBL b
20 a=1.234
30 b=CDBL(a)
40 PRINT "Aus";a;"wird";b
50 END
```

RUN:

```
Aus 1.234 wird 1.233999967575073
Ok
■
```


D E F I N T

Mit DEFINT buchstabenbereich lassen sich die mit **buchstabenbereich** bezeichneten Variablen als Integervariablen vordefinieren.
buchstabenbereich muß entweder in der Form eines einzelnen Buchstabens eingegeben werden oder in der Form einer Buchstabenliste, also Buchstabe1-Buchstabe2 (einschließlich).
Eine Integervariable ist eine Variable, die nur ganzzahlige Werte im Bereich von -32768 bis +32767 annehmen kann.
Durch eine Voreinstellung mit **DEFINT** läßt sich die Rechenzeit eines Programmes erheblich verkürzen, was sich besonders bei **FOR..NEXT**-Schleifen positiv bemerkbar macht.

Beispiel:

```
DEFINT b                  | b ist als integer definiert

DEFINT e-z                | die Variablen e bis z sind als integer definiert

10 DEFINT b-c
20 a=1.23456
30 b=1.23456
40 c=4.99999
50 PRINT "a (einfach genau) hat den Wert";a
60 PRINT "b (integer) hat den Wert";b
70 PRINT "c (integer) hat den Wert";c
80 END
```

RUN:

```
a (einfach genau) hat den Wert 1.23456
b (integer) hat den Wert 1
c (integer) hat den Wert 5
Ok
■
```

D E F S N G

Mit DEFSNG buchstabenbereich lassen sich die mit **buchstabenbereich** bezeichneten Variablen als einfach genaue Variablen vordefinieren.
buchstabenbereich muß entweder in der Form eines einzelnen Buchstabens oder in der Form einer Buchstabenliste, also Buchstabe1-Buchstabe2 (einschließlich), eingegeben werden.
Die Voreinstellung für alle Variablen ist die einfach genaue Variable. Dieser Zustand wird durch **LOAD**, **RUN**, **CHAIN**, **NEW** und **CLEAR** eingestellt.

Beispiel:

```
DEFSNG b                    | b ist als einfach genau definiert

DEFSNG e-z                  | die Variablen e bis z sind als einfach genau definiert

10 DEFSNG a-d
20 a=1.23456
30 b=-3.89673
40 c=4.99999
50 d=a*b/c
60 PRINT "a (einfach genau) hat den Wert ";a
70 PRINT "b (einfach genau) hat den Wert ";b
80 PRINT "c (einfach genau) hat den Wert ";c
90 PRINT "d (einfach genau) hat den Wert ";d
100 END
```

RUN:

```
a (einfach genau) hat den Wert  1.23456
b (einfach genau) hat den Wert -3.89673
c (einfach genau) hat den Wert  4.99999
d (einfach genau) hat den Wert -1.5840943
Ok
■
```

D E F S T R

Mit **DEFSTR buchstabenbereich** lassen sich die mit **buchstabenbereich** bezeichneten Variablen als Stringvariablen vordefinieren.
buchstabenbereich muß entweder in der Form eines einzelnen Buchstabens oder in der Form einer Buchstabenliste, also Buchstabe1-Buchstabe2 (einschließlich), eingegeben werden.
Die Folge ist also, daß man Stringvariablen nicht mehr durch ein $ kennzeichnen muß.

Beispiel:

```
DEFSTR b                    | b ist als Stringvariable definiert

DEFSTR e-z                  | die Variablen e bis z sind als Stringvariablen definiert

10 DEFSTR b
20 a=1.2345630
30 b="DEFSTR-Test"
40 PRINT "a (einfach genau) hat den Wert";a
50 PRINT "b (String) ist ";b
60 END
```


RUN

```
a (einfach genau) hat den Wert 1.23456
b (String) ist DEFSTR-Test
Ok
■
```

D E L E T E

Durch das Kommando **DELETE zeilennummernbereich, startzeilennummer** lassen sich die mit **zeilennummernbereich** festgelegten Zeilen aus dem aktuellen Programm löschen. Das Programm fährt dann mit der durch **startzeilennummer** bezeichneten Zeile fort.

zeilennummernbereich spezifiziert alle Zeilen, deren Nummern im gegebenen Bereich liegen. Dieser Bereich kann eines der folgenden Formate annehmen:

zeilennummer	nur diese Nummer fällt in den Bereich.
zeilennummer-zeilennummer	Zeilen von der ersten Nummer bis einschließlich der letzten.
zeilennummer-	Zeilen von der bestimmten Zeile bis zum Ende des Programms.
-zeilennummer	Zeilen vom Programmanfang bis zu der bestimmten Zeile.

Falls der Parameter **startzeilennummer** weggelassen wird, kehrt BASIC nach der Ausführung von **DELETE** in den Direktmodus zurück. Wird als **startzeilennummer** 0 angegeben, beginnt BASIC die Ausführung des Programms mit der niedrigsten Zeilennummer.

Als Nebenwirkungen des **DELETE**-Kommandos werden alle Benutzerfunktionen (s. **DEF FN**) vergessen, die **ON ERROR GOTO**-Fehlerbehandlungsroutine wird abgeschaltet, ein **RESTORE** wird durchgeführt und es werden alle aktiven **FOR**-, **WHILE**- und **GOSUB**-Schleifen vergessen.

Beispiel:

```
DELETE 40           | Löschen der Zeile 40

DELETE 10-30        | Löschen der Zeilen 10 bis 30

DELETE 100-         | Löschen des Programms ab Zeile 100

DELETE -100         | Löschen des Programms bis Zeile 100

DELETE 40,50        | Löschen von Zeile 40, fortfahren des Programms mit Zeile 50

DELETE 10-30,40     | Löschen der Zeilen 10 bis 30, fortfahren ab Zeile 40
```

```
DELETE 100-,10      | Löschen des Programms ab Zeile 100, weiter ab Zeile 10

DELETE -100,150     | Löschen des Programms bis Zeile 100, weiter ab Zeile 150
```

D E L K E Y

Bei diesem Kommando handelt es sich um einen **JETSAM**-Befehl. Genaueres erfahren Sie im **JETSAM**-Teil auf Seite 192.

D I M

Mit **DIM variable(a,b,..),..** wird der nötige Speicherbereich für **variable** festgelegt - **variable** wird **DIM**ensioniert. **a,b,..** stellt dabei eine Liste von Integerzahlen dar, die den Maximalwert der entsprechenden Dimension festlegen.
Ohne **DIM**ensionierung beträgt die Voreinstellung für jede Dimension maximal 10. Um sie zu vergrößern oder zu verkleinern (um Speicherplatz einzusparen) muß das **DIM**-Kommando benutzt werden.
Es ist nicht möglich, eine bereits ge**DIM**te Variable erneut zu dimensionieren. Wird es trotzdem versucht, reagiert der Computer mit der Fehlermeldung **Array already dimensioned** (Fehler 10). Wenn der Computer **Subscript out of range** (Fehler 9) meldet, ist der vordimensionierte Bereich unter- (s. **OPTON BASE**) oder überschritten worden.

Beispiel:

```
10 OPTION BASE 1                        | Dimensionen beginnen bei 1
20 DIM a(20,20),b(40)                   | Festlegung der Dimensionen von a und b
30 FOR i=1 to 20                        | Schleifen,
40   FOR u=1 to 20                      | um die Variablen
50      a(i,u)=u                        | mit Werten aufzufüllen
60   NEXT                               | Ende Schleife 2
70 NEXT                                 | Ende Schleife 1
80 FOR i=1 to 40                        | Anfang Schleife 3
90   b(i)=40-i                          | Wertzuweisung
100 NEXT                                | Ende Schleife 3
110 PRINT "a(10,15)=";a(10,15)          | Kontrolle einzelner
120 PRINT "b(19)=";b(19)                | Werte
130 END                                 | Programmende
```


RUN:

```
a(10,15)= 15
b(19)= 21
Ok
■
```

D I R

DIR dateinamenliste listet die mit **dateinamenliste** spezifizierten Dateien auf dem Bildschirm auf, sofern sie sich im Directory des entsprechenden Laufwerkes befinden. **dateinamenliste** ist in der Form **Datei1.Ext,Datei2.Ext,Datei3.Ext,..** einzugeben, wobei auch Wildcards (?,*) auf übliche Weise verwendet werden dürfen.

DIR interpretiert den Rest der aktuellen Zeile als Argument, unabhängig, ob es sich dabei um Dateinamen handelt oder nicht.

Beispiel:

```
DIR BASIC.COM

DIR BASIC.COM,RPED.BAS

DIR *.COM

DIR B???????.C?M
```

Wenn sich **BASIC.COM** auf der Diskette befindet, sollte in allen Fällen mindestens

```
BASIC.COM
Ok
■
```

auf dem Bildschirm erscheinen.

D I S P L A Y

DISPLAY datei$** zeigt den Inhalt von **datei$ an der Konsole. **DISPLAY** arbeitet prinzipiell wie das CP/M-Kommando **TYPE**. Die Auflistung kann duch [ALT]+[C] (STOP) abgebrochen, durch [ALT]+[S] (f3/f4) angehalten und durch [ALT]+[Q] (f5/f6) fortgesetzt werden. Zu beachten ist, daß **datei$** keine Wildcards (?,*) enthalten darf, sondern eine exakte Dateispezifikation darstellen muß.

Beispiel:

```
10 datei$="PROFILE.GER"
20 DISPLAY datei$
30 PRINT
40 DISPLAY "PROFILE.GER"
50 END
```

RUN:
(**PROFILE.GER** muß sich auf der Diskette im aktuellen Laufwerk befinden!)

```
setdef m:,* [order = (sub,com) temporary = m:)
pip
<m:=basic.com[o]
<m:=dir.com[o]
<m:=erase.com[o]
<m:=paper.com[o]
<m:=pip.com[o]
<m:=rename.com[o]
<m:=setkeys.com[o]
<m:=show.com[o]
<m:=submit.com[o]
<m:=type.com[o]
<

setdef m:,* [order = (sub,com) temporary = m:]
pip
<m:=basic.com[o]
<m:=dir.com[o]
     .....
     .....
<m:=type.com[o]
<
Ok
■
```


E D I T

Mit dem Kommando **EDIT zeilennummer** läßt sich die mit **zeilennummer** spezifizierte Programmzeile ändern.

Beispiel:

```
10 a$="TEST?"
20 PRINT a$
30 END
EDIT 10 (muß von Ihnen eingegeben werden)
10 a$="TEST?" (jetzt können Sie die Zeile korrigieren)
```

RUN:

```
TEST (bzw. Ihre Änderung)
Ok
■
```

E N D

Durch das **END**-Kommando wird das Programm an dieser Stelle ordnungsgemäß beendet. Alle noch offenen Dateien werden geschlossen, danach kehrt das Programm in den Direktmodus zurück. Steht am Ende des Programms kein **END**, werden evtl. offene Dateien erst beim nächsten **RUN** oder **SYSTEM** geschlossen.

Es ist ratsam, aus strukturellen Gründen ein Programm grundsätzlich mit **END** abzuschließen, auch wenn es in den meisten Fällen aus programmtechnischen Gründen nicht unbedingt erforderlich ist.

Beispiel:

```
10 'Hier steht Ihr Programm          | Ihr Programm kann in den Zeilen 0-9998
9999 END                             | stehen
```

RUN:

```
Ihr Programm...
Ok
■
```

E O F

EOF (dateinummer) prüft, ob das Dateiende der durch **dateinummer** spezifizierten Datei erreicht wurde. Ist dies der Fall, wird -1, ansonsten 0 übergeben.

Beispiel:

```
10 DIM inhalt$(50)
20 satz=0
30 OPEN "I",#1,"PROFILE.GER"
40 WHILE NOT EOF(1)                        | Grundsatzbeispiel zum Einlesen von
50   satz=satz+1                           | Daten aus einer Datei unbekannter
60   LINE INPUT #1,inhalt$(satz)           | Länge / Dimension / Dateneinträge
70 WEND
80 CLOSE
90 FOR i=1 to satz
100   PRINT "Satz ";DEC$(i,"###");" = ";inhalt$(i)
110 NEXT
120 END
```

RUN:

```
Satz  1 = setdef m:,* [order = (sub,com) temporary = m:]
Satz  2 = pip
Satz  3 = <m:=basic.com[o]
Satz  4 = <m:=dir.com[o]
Satz  5 = <m:=erase.com[o]
Satz  6 = <m:=paper.com[o]
Satz  7 = <m:=pip.com[o]
Satz  8 = <m:=rename.com[o]
Satz  9 = <m:=setkeys.com[o]
Satz 10 = <m:=show.com[o]
Satz 11 = <m:=submit.com[o]
Satz 12 = <m:=type.com[o]
Satz 13 = <
Ok
■
```

E R A

ERA dateinamenliste löscht die mit **dateinamenliste** spezifizierten Dateien. **dateinamenliste** ist in der Form **Datei1.Ext, Datei2.Ext,Datei3.Ext,...** einzugeben, wobei auch Wildcards (?,*) auf übliche Weise verwendet werden dürfen. Hierbei ist jedoch Vorsicht geboten, da keine Sicherheitsabfrage wie unter CP/M Plus stattfindet.

ERA interpretiert den Rest der aktuellen Zeile als Argument, unabhängig ob es sich dabei um Dateinamen handelt oder nicht.

Beispiel: **(ACHTUNG: ES IST SINNVOLL, DIE BEISPIELE NICHT AUSZUPROBIEREN. SIE DIENEN LEDIGLICH DAZU, DIE SYNTAX DES ERA-KOMMANDOS ZU VERSTEHEN !!)**

```
ERA ED.COM

ERA ED.COM,PROFILE.GER

ERA *.COM

ERA E???????.C?M
```

In allen Fällen würde **mindestens ED.COM gelöscht** werden.

E R A S E

Durch **ERASE variable,...** wird die vorherige Dimensionierung der **variable** durch ein **DIM**-Kommando rückgängig gemacht, so daß der ehemals belegte Speicherplatz für neue Zwecke nutzbar wird.
Es ist nicht möglich, eine bereits ge**ERASE**te Variable erneut zu löschen.

Beispiel:

```
10 DIM a(20,20),b(40)                      | Variablenfelder festlegen
20 FOR i=1 to 20                           |
30   FOR u=1 to 20                         | Variablen in Schleifen
40     a(i,u)=u                            |
50   NEXT                                  | mit irgendwelchen Werten
60 NEXT                                    |
70 FOR i=1 to 40                           | auffüllen
80   b(i)=40-i                             |
90 NEXT                                    |
100 PRINT "a(10,15)=";a(10,15)             | und anzeigen
110 PRINT "b(19)=";b(19)
120 PRINT "Noch freier Speicherplatz vor ERASE:";FRE("");"Bytes"
130 ERASE a,b
140 PRINT "Freier Speicherplatz nach ERASE:";FRE("");"Bytes"
150 END
```

RUN:

```
a(10,15)= 15
b(19)= 21
Noch freier Speicherplatz vor ERASE: 29316 Bytes          | Normalwerte bei Grund-
Freier Speicherplatz nach ERASE: 31264 Bytes              | einstellung
Ok
■
```

E R L

Die Funktion **ERL** (**ER**ror **L**ine) wird dazu benutzt, festzustellen, in welcher Zeile der letzte Fehler auftrat. Es wird also die fehlerhafte Zeilennummer übergeben.

Wenn **ERL** in Vergleichsoperationen benutzt wird, muß es auf der linken Seite der zu vergleichenden Ausdrücke stehen, damit BASIC die rechte Seite als Zeilennummer erkennt, so daß es bei späterer Anwendung des **RENUM**-Befehles keine Probleme gibt und der Programmablauf nicht gestört wird.

Beispiel:

```
10 ON ERROR GOTO 100                          | Bei Fehler Zeile 100 anspringen
20 PRINT "Diese Zeile ist korrekt,"
30 PRINT "die nächste nicht."
40 PRINZ "Hier in Zeile 40 ist »PRINZ« ein SYNTAX-ERROR"
50 END
100 PRINT "In Zeile";ERL;"tritt der Fehler mit der Nummer";ERR;"auf."
110 RESUME NEXT
```

RUN:

```
Diese Zeile ist korrekt,
die nächste nicht.
In Zeile 40 tritt der Fehler mit der Nummer 2 auf.
Ok
■
```

E R R

Die Funktion **ERR** wird dazu benutzt, festzustellen, welcher Fehler zuletzt auftrat. **ERR** liefert den entsprechenden Fehlercode. Eine ausführliche Beschreibung finden Sie im Anhang II Ihres mitgelieferten BASIC-Benutzer-Handbuches.

Beispiel:

```
10 ON ERROR GOTO 100                          | bei Fehler Zeile 100 anspringen
20 PRINT "Diese Zeile ist korrekt,"
30 PRINT "die nächste nicht."
40 PRINZ "Hier in Zeile 40 ist »PRINZ« ein SYNTAX-ERROR"
50 END
100 PRINT "In Zeile";ERL;"tritt der Fehler mit der Nummer";ERR;"auf."
110 RESUME NEXT
```

RUN:

```
Diese Zeile ist korrekt,
die nächste nicht.
In Zeile 40 tritt der Fehler mit der Nummer 2 auf.
Ok
■
```

E R R O R

ERROR fehlernummer simuliert den mit **fehlernummer** spezifizierten Fehler. **fehlernummer** stellt eine Integerzahl im Bereich von 1 bis 255 dar. Eine ausführliche Beschreibung der Fehlercodes finden Sie im Anhang II Ihres BASIC-Handbuches.

Beispiel:

```
10 ON ERROR GOTO 100                          | bei Fehler Zeile 100 anspringen
20 PRINT "Diese Zeile ist korrekt,"
30 PRINT "in der nächsten wird ein SYNTAX-ERROR simuliert."
40 ERROR 2                                    | 2 = SYNTAX ERROR
50 END
100 PRINT "In Zeile";ERL;"tritt der Fehler mit der Nummer";ERR;"auf."
110 RESUME NEXT
```

RUN:

```
Diese Zeile ist korrekt,
in der nächsten wird ein SYNTAX-ERROR simuliert.
In Zeile 40 tritt der Fehler mit der Nummer 2 auf.
```

E X P

Die Funktion **EXP (p)** erhebt die Zahl e in die **p**-te Potenz. e (= 2.718281828) ist diejenige Zahl, deren natürlicher Logarithmus 1 ist.
p darf dabei nicht größer als +88.0296... und nicht kleiner als -88.7228... sein, da **EXP** sonst entweder einen Überlauffehler erzeugt, oder 0 als Ergebnis errechnet.
Die Funktion **EXP** liefert nur einfach genaue Zahlen.

Beispiel:

```
10 PRINT "EXP (8) ergibt:";EXP (8)
20 END
```

RUN:

```
EXP (8) ergibt: 2980.958
Ok
■
```

F E T C H K E Y $ - F I E L D

Bei diesen Kommandos handelt es sich um **JETSAM**-Befehle. Genaueres erfahren Sie im **JETSAM**-Teil auf den Seiten 173 und 178.

F I L E S

FILES datei$** listet die mit **datei$ spezifizierte(n) Dateie(n) auf dem Bildschirm auf. In **datei$** dürfen auch auf übliche Weise Wildcards (?,*) verwendet werden.

Beispiel:

```
FILES "BASIC.COM"

FILES "*.COM"

FILES "B???????.C?M"
```



```
10 datei$="BASIC.COM"
20 FILES datei$
30 END
```

Wenn sich BASIC.COM auf der Diskette befindet, sollte in allen Fällen (ggf. nach **RUN**) mindestens

```
BASIC.COM
Ok
■
```

auf dem Bildschirm erscheinen.

F I N D $

Die Funktion **FIND$(maske$,n-te_Datei)** sucht nach der **n-ten_Datei**, auf die die durch **maske$** spezifizierte Maske zutrifft. Falls sie eine solche findet, übergibt sie einen 12 Zeichen langen String mit der gefundenen Datei, in dem auch evtl. Bit 7 gesetzt sein kann. Andernfalls wird ein leerer String übergeben.

Bit 7 gesetzt in

Byte 10:	Read Only-Attribut
Byte 11:	System-Attribut
Byte 12:	Archive-Attribut

In **maske$** dürfen auch auf übliche Weise Wildcards (?,*) verwendet werden.

n-te_Datei muß eine Integerzahl zwischen 1 und 255 ergeben. Falls es weggelassen wird, sucht **FIND$** nach der ersten zutreffenden Datei.

Beispiel:

```
10 maske$="*.COM"
20 n=1
30 datei$=FIND$ (maske$,n)
40 IF datei$<>"" THEN PRINT datei$:n=n+1:GOTO 30
50 PRINT "Gefundene Dateien:";n
60 END
```

RUN:

```
Hier wird die erste .COM-Datei gezeigt.
                . . . . . .
Hier wird die n-te .COM-Datei gezeigt.
Gefundene Dateien: n
Ok
■
```

F I X

Die Funktion **FIX (zahl)** rundet **zahl** auf eine ganze Zahl in Richtung 0 ab, es werden also die Nachkommastellen von **zahl** abgeschnitten.

Beispiel:

```
10 a=12.34567
20 b=-123456.7
30 PRINT "FIX (";a;") =";FIX(a)
40 PRINT "FIX (";b;") =";FIX(b)
50 END
```

RUN:

```
FIX ( 12.34567 ) = 12
FIX (-123456.7 ) =-123456
Ok
■
```

F O R

Mit **FOR variable=startwert TO endwert STEP schrittweite** lassen sich ohne große Probleme Schleifen realisieren.

variable ist eine Kontrollvariable für den aktuellen Zählerstand der Schleife. Sie darf nur eine ganze oder einfach genaue Variable sein.

startwert stellt den Anfangswert der Schleife dar.

endwert stellt den Endwert der Schleife dar. Die Schleife wird genau dann beendet, wenn **variable** dem Wert nach den **endwert** erreicht oder beim nächsten Durchgang durch ein zu großes Inkrement überschreiten würde.

schrittweite stellt ein Inkrement dar, um welches **variable** bei jedem Schleifendurchgang erhöht bzw. erniedrigt wird. Ist es nicht angegeben wird +1 angenommen.

Ist der **startwert** bei einer positiven **schrittweite** größer als der **endwert**, wird die Schleife übersprungen. Dasselbe gilt auch für den Fall, daß der **startwert** bei einer negativen **schrittweite** kleiner als der **endwert** ist.
Zu beachten ist, daß jede **FOR**-Schleife durch ein **NEXT**-Kommando beendet werden muß.

Beispiel:

```
10 DIM a(20,20),b(40)
20 FOR i=1 to 20
30   FOR u=20 to 1 STEP -1
40       a(i,u)=u
50   NEXT
60 NEXT
70 FOR i=1 to 40
80   b(i)=40-i
90 NEXT
100 PRINT "a(10,15)=";a(10,15)
110 PRINT "b(19)=";b(19)
120 END
```

RUN:

```
a(10,15)= 15
b(19)= 21
Ok
■
```

F R E

FRE ("") bestimmt den noch zur freien Benutzung verfügbaren Speicherplatz, wobei eine "garbage collection" (Speicheraufräumung) durchgeführt wird. Soll sie nicht bei der Speicherplatzbestimmung durchgeführt werden, muß **FRE(0)** verwendet werden..

Beispiel:

```
10 DIM a(20,20),b(40),c(500),a$(20)        | ordentlich Speicherplatz verbrauchen...
20 FOR i=1 to 20
30   a$(i)=STRING$(255,"T")
40 NEXT
50 PRINT "Noch freier Speicherplatz vor ERASE:";FRE("");"Bytes"
60 ERASE a,b,c,a$
70 PRINT "Freier Speicherplatz nach ERASE:";FRE("");"Bytes"
80 END
```

RUN:

```
Noch freier Speicherplatz vor ERASE: 22171 Bytes
Freier Speicherplatz nach ERASE: 31344 Bytes
Ok
■
```

G E T

Bei diesem Kommando handelt es sich um einen Befehl, um in Jetsam-Dateien bzw. Dateien mit wahlfreiem Zugriff Datensätze einzulesen. Genaueres erfahren Sie im Teil über **JETSAM** bzw. **Dateien mit wahlfreiem Zugriff** auf den Seiten 189 und 206.

G O S U B

Das **GOSUB zeilennummer**-Kommando wird dazu benutzt, um aus dem laufenden Programm ein Unterprogramm aufzurufen und dann (nach Beendigung des Unterprogramms durch **RETURN**) mit dem unmittelbar auf **GOSUB** folgenden Befehl fortzufahren.

Beispiel:

```
10 PRINT "Hier läuft das Hauptprogramm."
20 PRINT "Gleich wird das Unterprogramm aufgerufen."
30 GOSUB 100
40 PRINT "Jetzt läuft das Hauptprogramm weiter."
50 END
100 PRINT CHR$(7);"---- UNTERPROGRAMM ----";CHR$(7)
110 RETURN
```

RUN:

```
Hier läuft das Hauptprogramm.
Gleich wird das Unterprogramm aufgerufen.
(Pieps)---- UNTERPROGRAMM ----(Pieps)
Jetzt läuft das Hauptprogramm weiter.
Ok
■
```


G O T O

Das **GOTO zeilennummer**-Kommando stellt einen unbedingten Sprung zur durch die Zahl **zeilennummer** angegebenen Zeile dar.

Beispiel:

```
10 PRINT "Hier läuft das Programm."
20 PRINT "Jetzt springt es zur Zeile 100."
30 GOTO 100
40 END
100 PRINT CHR$(7);"Dies ist Zeile 100.";CHR$(7)
110 GOTO 40
```

RUN:

```
Hier läuft das Programm.
Jetzt springt es zur Zeile 100.
(Pieps)Dies ist Zeile 100.(Pieps)
Ok
■
```

H E X $

Die Funktion **HEX$ (zahl,stellen)** dient dazu, **zahl** in einen hexadezimalen String umzuwandeln, wobei **stellen** die Länge des hexadezimalen Stringes angibt. Falls die Länge unterschritten wird, wird der String bis zur vorgegebenen Länge mit führenden Nullen aufgefüllt. Ist der String jedoch zu lang, wird nichts abgeschnitten.
Zu beachten ist, daß **zahl** nur zwischen 0 und 65535 und **stellen** nur zwischen 0 und 16 liegen darf.

Beispiel:

```
10 a=256
20 b=65535
30 PRINT "HEX$(";a;") ergibt ";HEX$(a,4)
40 PRINT "HEX$(";b;") ergibt ";HEX$(b,4)
50 END
```

RUN:

```
HEX$( 256 ) ergibt 0100
HEX$( 65535 ) ergibt FFFF
Ok
■
```

H I M E M

Die **HIMEM**-Funktion ermittelt die höchste für BASIC benutzbare Speicheradresse. Sie bezeichnet den Bereich, in dem BASIC Speicherplatz für zahlreiche Zwecke reservieren kann, z.B. Schleifen, Variablen (insbesondere String-Variablen), etc.

Beispiel:

```
10 PRINT HEX$(HIMEM)
20 END
```

RUN:

```
F605 (in der Regel)
Ok
■
```

I F

Das **IF**-Kommando ermöglicht bedingt auszuführende Befehle.
Die Form ist grundsätzlich: **IF vergleichsoperation THEN anweisung ELSE anweisung**. Wenn das Ergebnis der **vergleichsoperation** wahr ist, wird die mit **THEN** verbundene **anweisung** ausgeführt. Ist das Ergebnis unwahr, wird die mit **ELSE** (sofern vorhanden) verbundene **anweisung** ausgeführt. Als Vergleichsoperator kann folgendes eingesetzt werden: = (gleich), < (kleiner), > (größer), <> (ungleich), **AND**, **NOT**, **OR**, **XOR** (letzteres bezieht sich auf eine bitweise Verknüpfung).
Die **anweisung** besteht aus ein oder mehreren Befehlen, die voneinander durch Doppelpunkte getrennt werden. Sie kann aber auch aus einer Zeilennummer bestehen: die Formen **THEN zeilennummer** und **ELSE zeilennummer** sind gleichbedeutend mit **THEN GOTO zeilennummer** und **ELSE GOTO zeilennummer**. Statt **THEN zeilennummer** kann aber auch **GOTO zeilennummer** verwendet werden.
Zu beachten ist, daß der Wirkungsbereich des **IF**-Kommandos erst durch das Zeilenende beendet wird. Es ist also in einer Zeile nicht möglich, nach einem IF-Kommando unabhängig von diesem arbeitende Befehle zu verwenden.

Beispiel:

```
10 INPUT "Ihr Name: ",name$
20 PRINT "Herr / Frau ([H] / [F])";
30 t$=UPPER$(INPUT$(1))
40 IF t$<>"H" AND t$<>"F" THEN 30
50 PRINT t$:PRINT
60 PRINT "Guten Tag, ";
70 IF t$="H" THEN PRINT "Herr "; ELSE PRINT "Frau";
80 PRINT name$
90 END
```

RUN:

```
Ihr Name: Mustermann
Herr / Frau ([H] / [F])? H

Guten Tag, Herr Mustermann
Ok
■
```

I N K E Y $

Mit **INKEY$** kann abgefragt werden, ob ein Zeichen von der Tastatur anliegt. Ist dies der Fall, wird es durch **INKEY$** übergeben. Ist dies nicht der Fall, wird ein leerer String übergeben. Sofern **OPTION RUN** aktiv ist, werden alle Zeichen übergeben, die von der Tastatur kommen.Ist es nicht aktiv, rufen ↑C ([ALT]+[C]) und ↑S ([ALT]+[S]) ihre üblichen Funktionen hervor (↑C bricht das Programm ab, ↑S läßt das Programm bis zum nächsten Tastendruck warten).

Beispiel:

```
10 PRINT "Abbruch mit ↑C (STOP)"
20 t$=INKEY$
30 WHILE t$=""
40 t$=INKEY$
50 WEND
60 PRINT "Es wurde die Taste mit dem ASCII-Code";ASC(t$);"gedrückt."
70 GOTO 20
```

RUN:

```
Abbruch mit ↑C (Stop)
Es wurde die Taste mit dem ASCII-Code xx gedrückt.
Es wurde die Taste mit dem ASCII-Code yy gedrückt.
↑C              | Sie haben die STOP-Taste gedrückt
Break in 40
Ok
■
```

I N P

Die **INP (kanalnummer)**-Funktion dient dazu, aus dem durch **kanalnummer** (0..255) spezifizierten Kanal des I/O-Portes einen Wert (0..255) zu lesen.

Beispiel:

```
10 INPUT "Kanalnummer (0..255): ",kanal
20 IF kanal<0 OR kanal>255 THEN 10
30 PRINT "Am Kanal";kanal;"des I/O-Portes liegt der Wert";INP (kanal);"an."
40 END
```

RUN:

```
Kanalnummer (0..255): xxx
Am Kanal xxx des I/O-Portes liegt der Wert yyy an.
Ok
■
```

I N P U T

INPUT ;"Text in Anführungszeichen";variablenliste dient zur Dateneingabe von der Konsole (meistens die Tastatur), wobei auf dem Bildschirm die eingegebenen Daten erscheinen und editiert werden können.

Das erste Semikolon bewirkt, daß der Cursor nach Beendigung der Eingabe nicht in die nächste Zeile springt. Es kann auch weggelassen werden.

Wenn ein Text (der von Anführungszeichen eingeschlossen sein muß) angegeben wird, wird er vor der Dateneingabe auf dem Bildschirm ausgegeben. Unmittelbar darauf folgt dann die Dateneingabe.

Steht vor der **variablenliste** ein Semikolon, wird vor der

Dateneingabe ein Fragezeichen ausgegeben. Steht statt des Semikolons ein Komma, entfällt das Fragezeichen.

Die Variablen der **variablenliste** enthalten nach Beendigung der Eingabe die einzelnen eingegebenen Werte. **variablenliste** ist in der Form **variable1,variable2,variable3,..** anzugeben. Selbstverständlich kann die **variablenliste** auch nur aus einer einzigen Variable bestehen.

Zu beachten ist, daß mit der Eingabe eines Kommas auch ein Element folgen muß. Wenn mehr Elemente eingegeben werden, als entsprechende Variablen vorhanden sind, oder wenn der Datentyp des eingegebenen Elementes nicht mit dem der Variablen übereinstimmt, wird die Fehlermeldung **?Redo from start** ausgegeben und das **INPUT**-Kommando erneut gestartet.

Die Dateneingabe wird durch einen Wagenrücklauf ([RETURN] / [ENTER]) abgeschlossen.

Beispiel:

```
INPUT a$
?■

INPUT "Text ";a$
Text ?■

INPUT "Text ",a$,b$,c
Text Test1,Test2,600■

INPUT ;"Zahl ",b:PRINT " Hier erfolgte kein Carriage Return !"
Zahl xxx Hier erfolgte kein Carriage Return !
```

I N P U T

INPUT #dateinummer,variablenliste wird zum Lesen von Daten aus sequentiellen Dateien benutzt.
dateinummer nennt die Datei, aus der die Daten gelesen werden sollen.
variablenliste stellt eine Liste von Variablen dar, die festlegt, in welche Variablen die Daten gelesen werden sollen. Sie ist in der Form **variable1,variable2,variable3,..** anzugeben. Dabei geben die Variablen den Datentyp der zu lesenden Variablen an.

Die drei Datentypen:

numerischer Wert: Ein numerisches Element wird beendet durch Leerstellen, Komma, Wagenrücklauf oder Dateiende.

String mit "": Alle Zeichen zwischen dem führenden und dem abschließenden Anführungszeichen bilden den String, in dem auch Wagenrücklauf oder Zeilenvorschub eingeschlossen sein

dürfen. Der String wird durch Anführungszeichen oder das Dateiende abgeschlossen.

einf. String: Der einfache String beginnt bei dem eigentlichen Text (führende Leerzeichen werden ignoriert) und endet durch Komma, Wagenrücklauf oder Dateiende. Strings sind auf 255 Zeichen begrenzt und werden nach dem 255. Zeichen automatisch abgeschlossen.

Beispiel:

```
10 DIM inhalt$(10)
20 satz=0
30 OPEN "I",#1,"RPED.SUB"
40 WHILE NOT EOF(1)
50   satz=satz+1
60   INPUT #1,inhalt$(satz)
70 WEND
80 CLOSE
90 FOR i=1 to satz
100   PRINT "Satz ";DEC$(i,"##");" = ";inhalt$(i)
110 NEXT
120 END
```

RUN:

```
Satz 01 = basic rped
Ok
■
```

I N P U T $

INPUT$ (max_länge,#dateinummer) wird zur Stringeingabe mit fester Länge von der Konsole (meistens Tastatur) oder aus einer Datei benutzt.

Die **max_länge** bezeichnet die Länge des zu lesenden Strings, die zwischen 1 und 255 liegen muß.

Die **dateinummer** bezeichnet die Nummer der Datei, aus der gelesen werden soll. Wird sie nicht angegeben, liest **INPUT$** den String von der Konsole. Sofern **OPTION RUN** aktiv ist, werden alle Zeichen, die von der Tastatur kommen, aufgenommen. Ist es nicht aktiv, rufen ↑C und ↑S ihre üblichen Funktionen hervor (↑C bricht das Programm ab, ↑S läßt das Programm bis zum nächsten Tastendruck warten).

Beispiel:

```
10 DIM zeichen$(100)
20 OPEN "I",#1,"PROFILE.GER"
30 FOR zeichen=1 to 100                    | Hier wird die Datei
40   zeichen$(zeichen)=INPUT$ (1,#1)       | zeichenweise eingelesen
50 NEXT
60 CLOSE
70 FOR i=1 to 100
80   PRINT zeichen$(i);
90 NEXT
100 END
```

RUN:

```
setdef m:,* [order = (sub,com) temporary = m:]
pip
<m:=basic.com[o]
<m:=dir.com[o]
<m:=erase.com[o]
Ok
■
```

I N S T R

Die **INSTR (begin,zu_durchsuchender_String,gesuchter_String)**-Funktion wird dazu benutzt, ab der mit **begin** genannten Stelle im **zu_durchsuchenden_String** nach dem **gesuchten_String** zu suchen.

Wird **begin** (1..255) weggelassen, beginnt die Suche nach **gesuchter_String** beim ersten Zeichen des **zu_durchsuchenden_Strings**.

Wird der gesuchte String gefunden, übergibt **INSTR** die Position des ersten Auftretens des gesuchten Strings. Ist der gesuchte String nicht enthalten, wird der Wert 0 übergeben.

Sind im **zu_durchsuchenden_String** ab **begin** keine Zeichen mehr vorhanden, wird ebenfalls der Wert 0 übergeben. Wenn der **gesuchte_String** ein leerer ist, wird er sofort gefunden, falls der zuletzt genannte Fall nicht eintritt.

Beispiel:

```
10 begriff$="Bergstiefel"
20 such$="stiefel"
30 posi=INSTR(begriff$,such$)
40 PRINT "Das Wort ";such$;" befindet sich in ";begriff$;" an";posi;"ter Stelle."
50 END
```

RUN:

```
Das Wort stiefel befindet sich in Bergstiefel an 5 ter Stelle.
Ok
■
```

I N T

INT (zahl) rundet **zahl** nach unten zur nächsten ganzen Zahl ab.

Beispiel:

```
10 a=12.3456
20 b=-12.3456
30 PRINT "INT (";a;") ergibt ";INT(a)
40 PRINT "INT (";b;") ergibt ";INT(b)
```

RUN:

```
INT ( 12.3456 ) ergibt 12
INT (-12.3456 ) ergibt -13
Ok
■
```


K I L L

KILL dateiname$** löscht die mit **dateiname$ spezifizierte Datei, wobei auch Wildcards (?,*) auf übliche Weise verwendet werden dürfen. Hiermit ist jedoch Vorsicht geboten, da keine Sicherheitsabfrage wie unter CP/M Plus stattfindet.

Beispiel: **(ACHTUNG: ES IST SINNVOLL, DIE BEISPIELE NICHT AUSZUPROBIEREN. SIE DIENEN LEDIGLICH DAZU, DIE SYNTAX DES KILL-KOMMANDOS ZU VERSTEHEN !!)**

```
KILL "ED.COM"

datei$="ED.COM":KILL datei$

datei$="*.COM":KILL datei$

KILL "E???????.C?M"
```

In allen Fällen würde **mindestens** ED.COM gelöscht werden.

L E F T $

LEFT$ (wort$,n) ermittelt von **wort$** die ersten **n** linken Zeichen. **n** muß dabei zwischen 0 und 255 liegen.

wort$ stellt einen beliebigen String dar, der zwischen 1 und 255 Zeichen lang sein muß.

Ist **n** größer als die gesamte Länge von **wort$**, wird ein String übergeben, der aus **wort$** und der Anzahl Leerstellen besteht, die der Länge von **wort$** bis zur Länge **n** fehlen.

Beispiel:

```
10 text$="JOYCE - mehr als ein Textsystem."
20 links$=LEFT$(text$,5)
30 PRINT links$
40 END
```

RUN:

```
JOYCE
Ok
■
```

L E N

LEN (text$) ermittelt die gesamte Länge (0..255) von **text$**, eingeschlossen sind auch Zeichen, die nicht ohne weiteres auf dem Bildschirm darstellbar sind (Zeichen, deren ASCII-Code kleiner als 32 ist).

Liefert **LEN** als Ergebnis 0, zeigt das, daß **text$** ein leerer String ist, also keine Zeichen enthält.

Beispiel:

```
10 text$="JOYCE"
20 PRINT "Die Länge des Wortes ";text$;" beträgt";LEN (text$);" Zeichen"
30 END
```

RUN:

```
Die Länge des Wortes JOYCE beträgt 5 Zeichen
Ok
■
```

L E T

Das **LET**-Kommando wurde früher zur Wertzuweisung für Variablen benutzt. In Mallard 80-BASIC hat es jedoch keine besondere Funktion und kann daher ignoriert werden.

L I N E I N P U T

LINE INPUT ;"Text in Anführungszeichen";variable$ dient zur Dateneingabe von der Konsole (meistens die Tastatur), wobei auf dem Bildschirm die eingegebenen Daten erscheinen und editiert werden können. **LINE INPUT** übernimmt vollständig die eingegebene Zeile, wobei also auch z.B. Kommas mitgelesen werden. Dadurch ist es jedoch nicht möglich, mit einem **LINE INPUT**-Kommando mehreren Variablen verschiedene Texte zuzuweisen. Auch kann aus diesen technischen Gründen immer nur ein **Text** eingegeben werden, der dann einer Stringvariable zugewiesen wird.

Das erste Semikolon bewirkt, daß der Cursor nach Beendigung der Eingabe nicht in die nächste Zeile springt. Es kann auch bei Bedarf weggelassen werden, damit ein Carriage Return

erfolgt.

Wenn ein Text (der von Anführungszeichen eingeschlossen sein muß) angegeben wird, wird er vor der Dateneingabe auf dem Bildschirm ausgegeben. Unmittelbar darauf folgt dann die Dateneingabe.

Steht vor **variable$** ein Semikolon, wird vor der Dateneingabe ein Fragezeichen ausgegeben. Steht statt des Semikolons ein Komma, entfällt das Fragezeichen.

variable$ enthält nach Beendigung der Eingabe, die durch einen Wagenrücklauf ([RETURN] / [ENTER] / [ALT]+[M]) abgeschlossen wird, den eingegebenen Text.

Beispiel:

```
LINE INPUT a$
?■

LINE INPUT "Text ";a$
Text ?■

LINE INPUT ;"Zahl ",b:PRINT " Hier erfolgte kein Carriage Return !"
Zahl xx■ Hier erfolgte kein Carriage Return !
```

L I N E I N P U T

LINE INPUT #dateinummer,text$ wird zum Lesen von Daten aus einer Datei benutzt.

dateinummer nennt die Datei, aus der die Daten gelesen werden sollen.

text$ stellt die Variable dar, in die die Daten gelesen werden sollen. Die Daten, die immer vom Typ "String" sind, enden entweder mit einem Wagenrücklauf (=Zeilenende) oder automatisch mit dem 255. Zeichen.

Beispiel:

```
10 DIM inhalt$(50)                                  | Initialisierung
20 satz=0                                           |
30 OPEN "I",#1,"PROFILE.GER"                        |
40 WHILE NOT EOF(1)                                 | Leseroutine
50   satz=satz+1                                    | Datensatzzähler
60   LINE INPUT #1,inhalt$(satz)                    | Auslesen der Datei
70 WEND                                             | Ende Routine
80 CLOSE                                            | Datei schließen
90 FOR i=1 to satz                                  | Schleife zum
100   PRINT "Satz ";DEC$(i,"##");" = ";inhalt$(i)   | Auslesen der
110 NEXT                                            | einzelnen Sätze
120 END                                             | Programmende
```

RUN:

```
Satz  1 = setdef a:,* (order = (sub,com) temporary = m:]
Satz  2 = pip
Satz  3 = <m:=basic.com[o]
Satz  4 = <m:=dir.com[o]
Satz  5 = <m:=erase.com[o]
Satz  6 = <m:=paper.com[o]
Satz  7 = <m:=pip.com[o]
Satz  8 = <m:=rename.com[o]
Satz  9 = <m:=setkeys.com[o]
Satz 10 = <m:=show.com[o]
Satz 11 = <m:=submit.com[o]
Satz 12 = <m:=type.com[o]
Satz 13 = <
Ok
■
```

L I S T

Durch das Kommando **LIST zeilennummernbereich** lassen sich die mit **zeilennummernbereich** festgelegten Zeilen aus dem aktuellen Programm auf der Konsole (meistens Bildschirm) auflisten.

zeilennummernbereich spezifiziert alle Zeilen, deren Nummern im gegebenen Bereich liegen. Dieser Bereich kann eines der folgenden Formate annehmen:

zeilennummer	nur diese Nummer fällt in den Bereich
zeilennummer-zeilennummer	Zeilen von der ersten Nummer bis einschließlich der letzten
zeilennummer-	Zeilen von der bestimmten Zeile bis zum Ende des Programms
-zeilennummer	Zeilen vom Programmanfang bis zu der bestimmten Zeile

Beispiel:

```
LIST 40          | Auflisten der Zeile 40

LIST 10-30       | Auflisten der Zeilen 10 bis 30

LIST 100-        | Auflisten des Programms ab Zeile 100

LIST -100        | Auflisten des Programms bis Zeile 100
```


L L I S T

Durch das Kommando **LLIST zeilennummernbereich** lassen sich die mit **zeilennummernbereich** festgelegten Zeilen aus dem aktuellen Programm auf dem LST-Kanal (meistens Drucker) ausgeben.
zeilennummernbereich spezifiziert alle Zeilen, deren Nummern im gegebenen Bereich liegen. Dieser Bereich kann eines der folgenden Formate annehmen:

zeilennummer	nur diese Nummer fällt in den Bereich
zeilennummer-zeilennummer	Zeilen von der ersten Nummer bis einschließlich der letzten
zeilennummer-	Zeilen von der bestimmten Zeile bis zum Ende des Programms
-zeilennummer	Zeilen vom Programmanfang bis zu der bestimmten Zeile

Beispiel:

```
LLIST 40         | Ausdrucken der Zeile 40

LLIST 10-30      | Ausdrucken der Zeilen 10 bis 30

LLIST 100-       | Ausdrucken des Programms ab Zeile 100

LLIST -100       | Ausdrucken des Programms bis Zeile 100
```

L O A D

Mit **LOAD datei$,R** läßt sich eine Datei von der Diskette laden und bei der Angabe des **,R** (Run)-Parameters sofort starten. Ein evtl. schon im Speicher befindliches Programm wird überschrieben (= gelöscht!).

datei$** gibt den Namen der Datei, die geladen werden soll. Es kann dabei am Anfang von **datei$ auch eine Laufwerksbezeichnung mit angegeben werden. Wird keine Extension mit angegeben, wird **.BAS** angenommen.

Zu beachten ist, daß alle Variablen und Benutzerfunktionen gelöscht, alle **DEFINT-, DEFSNG-, DEFDBL-, DEFSTR-** und **OPTION BASE**-Einstellungen zurückgesetzt werden.

Beispiel:

```
LOAD "RPED

LOAD "RPED.BAS

LOAD "A:RPED

LOAD "A:RPED.BAS

LOAD "RPED.BAS",R
```

In sämtlichen Fällen wird versucht, **RPED.BAS** zu laden.

L O C - L O C K

Bei diesen Kommandos handelt es sich um Befehle, die bei der Verarbeitung von Jetsam-Dateien bzw. Dateien mit wahlfreiem Zugriff verwendet werden. Genaueres erfahren Sie im Teil über **JETSAM** bzw. **Dateien mit wahlfreiem Zugriff** auf den Seiten 194 und 208.

L O F

LOF (dateinummer) wird zum Ermitteln der Länge (Length Of File) der mit **dateinummer** gekennzeichneten Datei benutzt. Der übergebene Wert bezieht sich dabei auf die Dateilänge in Records (1 Record = 128 Bytes).

ACHTUNG: Die LOF-Funktion arbeitet nur bis zu einer maximalen Dateilänge von 128 Records (=16384 Bytes = 16 Kbyte) korrekt. Eine größere Datei wird konstant mit 128 Records angegeben. Der Grund für diese Fehlfunktion liegt im Aufbau des Directory-Eintrags einer Datei auf der Diskette. Für jede Datei, die größer als 128 Records ist, wird für jede angefangenen 128 Records ein neuer Directory-Eintrag angelegt. **LOF** ermittelt jedoch immer nur die Länge des gerade aktuellen Directory-Eintrages, der für die Daten zuständig ist, die man liest. Wenn man jetzt also die gesamte Länge einer Datei benötigt, muß man die einzelnen Längen der Directory-Einträge addieren.

Beispiel:

```
10 OPEN "I",#1,"RPED.BAS"
20 PRINT "Die Länge der Datei beträgt";LOF(1);"Records =";LOF(1)*128;"Bytes."
30 CLOSE
40 END
```


RUN:

```
Die Länge der Datei beträgt 56 Records = 7168 Bytes.
OK
■
```

L O G

LOG (zahl) berechnet den natürlichen Logarithmus von **zahl**. **LOG** liefert einen einfach genauen Wert, da **zahl**, die größer als 0 sein muß, vor der Berechnung des Logarithmus in eine einfach genaue Zahl umgewandelt wird.

Beispiel:

```
10 PRINT "LOG (148.4132) ergibt:";LOG (148.4132)
20 END
```

RUN:

```
LOG (148.4132) ergibt: 5
Ok
■
```

L O G 1 0

LOG10 (zahl) berechnet den dekadischen Logarithmus (zur Basis 10) von **zahl**. **LOG10** liefert einen einfach genauen Wert, da **zahl**, die größer als 0 sein muß, vor der Berechnung des Logarithmus in eine einfach genaue Zahl umgewandelt wird.

Beispiel:

```
10 PRINT "LOG10 (100) ergibt:";LOG10 (100)
20 END
```

RUN:

```
LOG10 (100) ergibt: 2
Ok
■
```

L O W E R $

LOWER$ (text$) wandelt alle Großbuchstaben in **text$** in Kleinbuchstaben um. Als Großbuchstaben werden nur die Zeichen mit dem ASCII-Code zwischen 65 und 90 (A-Z) anerkannt und umgewandelt.

Beispiel:

```
10 text$="JOYCE"
20 PRINT text$;" in Kleinbuchstaben umgewandelt ergibt ";LOWER$(text$)
30 END
```

RUN:

```
JOYCE in Kleinbuchstaben umgewandelt ergibt joyce
OK
■
```

L P O S (0)

LPOS(0) ermittelt die aktuelle logische Druckposition des Druckers. Das heißt, daß die ermittelte Position nicht unbedingt etwas mit der tatsächlichen zu tun hat, weil jedes Zeichen, dessen ASCII-Code kleiner als 32 ist, nicht mitgezählt wird. Eine Ausnahme machen nur die Zeichen mit dem ASCII-Code 8, 9 und 13.

Die Bedeutung:

ASCII:
- 8 entspricht einem Rückschritt, der den Zähler um 1 verringert, falls die Position nicht schon 1 ist.
- 9 entspricht einem TAB, der in Leerstellen umgewandelt wird, von denen jede gezählt wird
- 13 entspricht einem Wagenrücklauf, der die Position auf 1 setzt.

Beispiel:

```
10 LPRINT "Dies ist ein Test.";
20 PRINT "Die aktuelle Druckposition beträgt:";LPOS(0)
30 END
```


RUN:

```
Dies ist ein Test. (Erscheint auf dem Drucker)
Die aktuelle Druckposition beträgt 19
Ok
■
```

L P R I N T

LPRINT text; bewirkt die Ausgabe des textes auf dem Drucker.

text kann dabei eine Variablenliste oder ein fester Ausdruck sein. Die Variablenliste kann aus Variablen vom Typ "string" und "numerisch" bestehen. Wenn die einzelnen Variablen dabei durch ein Komma getrennt werden, wird jedesmal für ein Komma ein Tabulator eingefügt. Wenn die Variablen durch ein Semikolon getrennt werden, werden die Werte der Variablen ohne zusätzliche Leerstellen aneinandergereiht gedruckt. In diesem Fall ist jedoch zu beachten, daß eine Variable numerischen Typs eine Leerstelle vor (für Vorzeichen) und nach der Zahl beansprucht. Der feste Ausdruck ist entweder eine Zahl oder ein Text in Anführungsstrichen.

Ein Semikolon am Ende des Textes bewirkt, daß kein Wagenrücklauf ausgeführt wird. Ein Komma würde einem Tabulator gleichkommen, wobei auch hier kein Wagenrücklauf ausgeführt wird. Wenn am Ende kein Zeichen steht, wird ein ganz normaler Zeilenvorschub ausgeführt.

Beispiel:

```
10 a=8256
20 a$="SCHNEIDER JOYCE PCW"
30 LPRINT "Dieses Beispiel läuft auf einem ";a$;a
40 a=10:b=20:c=-30
50 LPRINT a,b,c,999;
60 LPRINT "Test"
70 END
```

RUN:

(Erscheint auf dem Drucker):

```
Dieses Beispiel läuft auf einem SCHNEIDER JOYCE PCW 8256
 10            20           -30            999 Test
```

L S E T

Bei diesem Kommando handelt es sich um einen **JETSAM**-Befehl. Genaueres erfahren Sie im **JETSAM**-Teil auf Seite 173.

M A X

MAX (zahlenliste) ermittelt den größten in **zahlenliste** enthaltenen Wert. **zahlenliste** ist in der Form **zahl1,zahl2, zahl3,..** anzugeben.

Beispiel:

```
10 a=10:b=20:c=-30
20 PRINT "MAX (";a;",";b;",";c;") ist";MAX (a,b,c)
30 END
```

RUN:

```
MAX ( 10 , 20 ,-30 ) ist 20
Ok
■
```

M E M O R Y

MEMORY High-Memory,Stack-Größe,Dateizahl,maximale_Satzlänge verändert die angegebenen Parameter.

High-Memory bezieht sich auf die obere vom BASIC benutzbare Speicheradresse (Sie bezeichnet den Bereich, in dem BASIC Speicherplatz für zahlreiche Zwecke reservieren kann, z.B. Schleifen, Variablen (insbesondere String-Variablen), etc.). Wird diese Angabe weggelassen, bleibt der aktuelle Wert eingestellt.

Stack-Größe gibt die Anzahl der von BASIC frei für das Stapelregister verwendbaren Bytes an. Der Stack ist der Speicher, der von BASIC für die Verwaltung von Schleifen, **GOSUB**-Routinen, etc. benutzt wird. Falls **Stack-Größe** angegeben wird, muß mindestens ein Wert von 256 verwendet werden. Auch werden dabei alle aktiven **FOR-**, **WHILE-**, und **GOSUB**-Kommandos vergessen. Wird diese Angabe weggelassen, bleibt der aktuelle Wert eingestellt.

Dateizahl gibt die maximale Anzahl gleichzeitig zu bearbeitender Dateien an. Ohne diese Angabe bleibt der aktuelle Wert eingestellt.

maximale_Satzlänge gibt die maximale Größe eines frei adressierbaren Satzes an. Ohne diese Angabe bleibt der aktuelle Wert eingestellt.

Die Standardeinstellungen:

High-Memory:	&HF605
Stack-Größe:	512 Bytes
Dateizahl:	3
max._Satzlänge:	128 Bytes

Beispiel:

```
MEMORY &HF605,,6          | High-Memory auf &HF605, maximale Dateizahl 6

MEMORY &HEFFF,512,3,256   | High-Memory auf &HEFFF, Stack-Größe auf 512 Bytes, 3 Dateien,
                          | maximale Satzlänge 128 Bytes

MEMORY ,,10               | 10 Dateien
```

```
10 PRINT "Alte obere Speicheradresse: ";HEX$(HIMEM)
20 MEMORY &HF5FF
30 PRINT "Neue obere Speicheradresse: ";HEX$(HIMEM)
40 END
```

RUN:

```
Alte obere Speicheradresse: F605
Neue obere Speicheradresse: F5FF
Ok
■
```

M E R G E

MERGE datei$** wird zum Anbinden der Datei **datei$ an das aktuelle Programm benutzt.

datei$ bezeichnet das anzubindende Programm. Wird die Extension weggelassen, wird **.BAS** angenommen. Auch kann vor dem eigentlichen Dateinamen eine Laufwerksbezeichnung mit angegeben werden. Während des Anbindens durch **MERGE** werden die Programmzeilen ersetzt, die mit gleicher Zeilennummer bereits existieren. Nach der Ausführung des Befehls, kehrt BASIC wieder in den Direktmodus zurück.
Zu beachten ist jedoch, daß alle Variablen, Benutzerfunktionen, **OPTION BASE-**, **DEFINT-**, **DEFSNG-**, **DEFDBL-** und **DEFSTR-**Kommandos zurückgesetzt werden, die **ON ERROR GOTO**-Funktion abgeschaltet und ein **RESTORE**-Befehl ausgeführt wird und daß alle offenen Dateien erhalten bleiben.

Beispiel:

```
10000 'Hier steht ihr Programm
MERGE "RPED.BAS
Ok
OPEN "O",#1,"M:PASS.OFF  'Dieser Teil ist nötig, da
Ok
MERGE "M:PASS.OFF        'RPED.BAS geschützt ist...
Ok
LIST
1 OPTION NOT TAB:OPTION RUN              | Dieser Teil stammt von
2 DEFINT a-z:.....                       | RPED.BAS
     ............
10000 'Hier steht ihr Programm
Ok
■
```

M I D $

Mit **MID$ (wort$,start,länge)** läßt sich aus dem String **wort$** ein Teilstück heraussuchen, das an der Position **start** beginnt und dann eine Länge von **länge** erhält.

wort$ stellt den String dar, aus dem ein Teil herauskopiert werden soll. Er muß mindestens eine Länge von 1 Zeichen haben und kann bis zur Maximalgrenze von 255 Zeichen heranreichen.

start bezeichnet die Stelle, ab der der neue String gebildet werden soll. **start** muß eine Integerzahl zwischen 1 und 255 ergeben. Ist **start** größer als die gesamte Länge von **wort$**, wird ein neuer String mit der Länge 0 gebildet

länge gibt die Länge des herauszusuchenden Teilstrings an. Überschreitet **länge** die verbleibende Länge des Strings ab der Position **start**, reicht der neue String ab **start** bis zum Ende des ürsprünglichen Strings. Dies ist auch der Fall, wenn **länge** nicht angegeben wird (dann ist **MID$** äquivalent zu **RIGHT$**).

Beispiel:

```
10 text1$="JOYCE - mehr als ein Textsystem."
20 text2$=" PCW "
30 mitte$=MID$(text1$,9,23)
40 MID$(text1$,1,5)=text2$
50 PRINT mitte$
60 PRINT text1$
70 END
```


RUN:

```
mehr als ein Textsystem
 PCW  - mehr als ein Textsystem.
Ok
■
```

M I N

MIN (zahlenliste) ermittelt den kleinsten in **zahlenliste** enthaltenen Wert. **zahlenliste** ist in der Form **zahl1,zahl2, zahl3,..** einzugeben.

Beispiel:

```
10 a=10:b=20:c=-30
20 PRINT "MIN (";a;",";b;",";c;") ist ";MIN (a,b,c)
30 END
```

RUN:

```
MIN ( 10 , 20 ,-30 ) ist -30
Ok
■
```

M K D $ - M K U K $

Bei diesen Kommandos handelt es sich um **JETSAM**-Befehle. Genaueres erfahren Sie im **JETSAM**-Teil auf den Seiten 195 bis 197.

M O D

Mit **a MOD b** läßt sich der ganzzahlige Rest von **a** ermitteln, der entsteht, wenn man **a** durch **b** teilt.

Beispiel:

```
10 INPUT "Zahl: ",a
20 INPUT "Teiler: ",b
30 PRINT "Der ganzzahlige Rest von ";a;"durch ";b;"ist: ";a MOD b
40 END
```

RUN:

```
Zahl: 10
Teiler: 3
Der ganzzahlige Rest von  10 durch  3 ist:  1
Ok
■
```

N A M E

Durch Verwendung von **NAME altdatei$ AS neudatei$** läßt sich eine Datei umbenennen. **altdatei$** bezeichnet dabei die Datei, die umbenannt werden soll. Sie wird dann in **neudatei$** umbenannt. Wichtig ist, daß eine Datei mit dem Namen **neudatei$** nicht schon auf dem aktuellen Laufwerk existieren darf. Es ist außerdem nicht empfehlenswert, eine offene Datei umzubenennen.

Beispiel:

```
10 altname$="RPED.BAS"                  | Initialisierung
20 neuname$="R.BAS"                     |
30 DIR *.BAS                            | alle .BAS Dateien auflisten
40 PRINT                                | Neue Zeile nach DIR-Auflistung
50 NAME altname$ AS neuname$            | RPED.BAS umbenennen
60 DIR *.BAS                            | erneut alle .BAS Dateien heraussuchen
70 END                                  | Programmende
```

RUN:

```
RPED    .BAS  ... (mindestens RPED.BAS, sofern RPED.BAS auf Diskette vorhanden)
R       .BAS  ... (mindestens R.BAS, sofern RPED.BAS auf Diskette vorhanden)
Ok
■
```


N E W

Mit dem **NEW**-Kommando läßt sich ein im Speicher befindliches Programm löschen. Außerdem werden alle Variablen und Benutzerfunktionen vergessen, alle **OPTION BASE-**, **DEFINT-**, **DEFSNG-**, **DEFDBL-** und **DEFSTR-**Kommandos zurückgesetzt, die **ON ERROR GOTO**-Funktion wird abgeschaltet und ein **RESTORE**-Befehl ausgeführt. Alle offenen Dateien werden geschlossen.

Nach der Ausführung des **NEW**-Befehls springt BASIC immer in den Direktmodus zurück, da ein evtl. laufendes Programm gelöscht wurde.

Beispiel:

```
10 a=100
20 PRINT a
30 NEW
40 END
```

RUN:

```
100
Ok
LIST            | (zur Kontrolle; von Ihnen einzugeben)
Ok
PRINT a         | (zur Kontrolle; von Ihnen einzugeben)
0
Ok
■
```

N E X T

Mit **NEXT variable** wird die zu **variable** gehörige **FOR**-Schleife geschlossen (siehe **FOR**). **variable** kann auch weggelassen werden, da sich ein **NEXT**-Kommando immer auf das letzte **FOR** bezieht. Die (einzig) erlaubte Schachtelung von **FOR-NEXT** Schleifen ist:

```
FOR v1=a to b
     FOR v2=a to b
          FOR v3=a to b
               |
               |
          NEXT v3
     NEXT v2
NEXT v1
```

Beispiel:

```
10 DIM a(20,20),b(40)
20 FOR i=1 to 20
30   FOR u=20 to 1 STEP -1
40     a(i,u)=u
50   NEXT u
60 NEXT i
70 FOR i=1 to 40
80   b(i)=40-i
90 NEXT
100 PRINT "a(10,15)=";a(10,15)
110 PRINT "b(19)=";b(19)
120 END
```

RUN:

```
a(10,15)= 15
b(19)= 21
Ok
■
```

O C T $

Die Funktion **OCT$ (zahl,stellen)** dient dazu, **zahl** in einen oktalen String umzuwandeln, wobei **stellen** die Länge des oktalen Strings angibt. Falls die Länge unterschritten wird, wird der String bis zur vorgegebenen Länge mit führenden Nullen aufgefüllt. Ist der String jedoch zu lang, wird nichts abgeschnitten.

Zu beachten ist, daß **zahl** nur zwischen 0 und 65535 und **stellen** nur zwischen 0 und 16 liegen darf.

Beispiel:

```
10 a=256
20 b=65535
30 PRINT "OCT$(";a;") ergibt ";OCT$(a,4)
40 PRINT "OCT$(";b;") ergibt ";OCT$(b)
50 END
```

RUN:

```
OCT$( 256 ) ergibt 0400
OCT$( 65535 ) ergibt 177777
Ok
■
```

O N x G O S U B

Mit **ON x GOSUB zeile1,zeile2,zeile3,...** läßt sich abhängig von **x** eine der **zeilen** als Unterprogramm anspringen (siehe **GOSUB**). Beträgt **x** eins wird **zeile1** aufgerufen, bei **x**=zwei wird **zeile2** aufgerufen, bei **x**=drei **zeile3**, usw.

Beispiel:

```
10 PRINT "Bitte geben Sie eine Zahl ein (1..5):";     | zahl
20 t$=INPUT$(1):IF t$<"1" OR t$>"5" THEN 20           | wird ein
30 zahl=VAL(t$):PRINT zahl                            | Wert zugewiesen
40 PRINT "Sie haben die Zahl ";                       |
50 ON zahl GOSUB 80,90,100,110,120                    |Sprung zum entsp. U-Prg.
60 PRINT " eingegeben."                               |
70 END                                                | Programmende
80 PRINT "eins";:RETURN                               | Je nach Zahl
90 PRINT "zwei";:RETURN                               | wird der zu-
100 PRINT "drei";:RETURN                              | gehörige String
110 PRINT "vier";:RETURN                              | ausgegeben und
120 PRINT "fünf";:RETURN                              | RETURN
```

RUN:

```
Bitte geben Sie eine Zahl ein (1..5):x
Sie haben die Zahl x_in_Buchstaben eingegeben.
Ok
■
```

O N x G O T O

Mit **ON x GOTO zeile1,zeile2,zeile3,...** läßt sich abhängig von **x** eine der **zeilen** anspringen (siehe **GOTO**). Beträgt **x** eins wird **zeile1** angesprungen, bei **x**=zwei wird **zeile2** angesprungen, bei **x**=drei **zeile3**, usw.

Beispiel:

```
10 PRINT "Bitte geben Sie eine Zahl ein (1..5):";          | zahl
20 t$=INPUT$(1):IF t$<"1" OR t$>"5" THEN 20                | wird ein
30 zahl=VAL(t$):PRINT zahl                                 | Wert zugewiesen
40 PRINT "Sie haben die Zahl ";                            |
50 ON zahl GOTO 60,70,80,90,100                            |Sprung zum entsp. U-Prg.
60 PRINT "eins";:GOTO 110                                  | je nach zahl
70 PRINT "zwei";:GOTO 110                                  | wird
80 PRINT "drei";:GOTO 110                                  | der entsprechende
90 PRINT "vier";:GOTO 110                                  | Wert ausgegeben
100 PRINT "fünf";:GOTO 110                                 | und zu 110 gesprungen
110 PRINT " eingegeben."                                   | Programmabschluß
120 END                                                    |
```

RUN:

```
Bitte geben Sie eine Zahl ein (1..5):x
Sie haben die Zahl x_in_Buchstaben eingegeben.
Ok
■
```

O N E R R O R G O T O

ON ERROR GOTO zeilennummer wird zur Fehlerbehandlung während eines laufenden Programms benutzt.

Prinzipiell gibt es zwei Wege, Fehler, die während eines Programms auftreten, zu behandeln: den normalen Fehlermodus (Abbruch des Programms und Ausgabe des Fehlers) und den Behandlungsmodus, der die Möglichkeit bietet, nach dem Finden eines Fehlers in ein Unterprogramm zu springen, welches mit **zeilennummer** beginnt.

Während der normale Fehlermodus von Anfang an eingestellt ist, wird der Behandlungsmodus erst durch ein **ON ERROR GOTO**-Kommando aktiviert. Ist dieser aktiviert und BASIC findet einen Fehler, werden **ERR** und **ERL** (siehe dort) gesetzt, um die Art des Fehlers zu bestimmen. Außerdem springt das Programm zu dem ab **zeilennummer** stehenden Unterprogramm, welches zweckmäßig mit dem **RESUME**-Kommando (sieh dort) abzuschließen ist. Abgeschaltet wird der Behandlungsmodus durch **ON ERROR GOTO 0**; der normale Fehlermodus ist dann wieder aktiv.

Beispiel:

```
10 ON ERROR GOTO 100                        | ab 100: Fehlerbehandlungsroutine
20 PRINT "Diese Zeile ist korrekt,"
30 PRINT "in der nächsten wird ein SYNTAX-ERROR simuliert."
40 ERROR 2
50 PRINT "Das Programm wird fortgesetzt."
60 END
100 PRINT "In Zeile";ERL;"tritt der Fehler mit der Nummer";ERR;"auf."
110 RESUME NEXT                             | Programm fortsetzen
```

RUN:

```
Diese Zeile ist korrekt,
in der nächsten wird ein SYNTAX-ERROR simuliert.
In Zeile 40 tritt der Fehler mit der Nummer 2 auf.
Das Programm wird fortgesetzt.
Ok
■
```

O P E N

Mit **OPEN art$,#dateinummer,datei$** wird eine Datei für die Ein- oder Ausgabe geöffnet und der Kennzeichnung **#dateinummer** zugewiesen.

art$** gibt an, ob die Datei zum sequentiellen Lesen oder Schreiben geöffnet werden soll. Soll sie gelesen werden, muß für **art$ "I" (I=INPUT) angegeben werden. Entsprechend muß zum Schreiben einer Datei für **art$ "O"** (O=OUTPUT) verwendet werden. Man muß jedoch darauf achten, daß beim öffnen einer Datei zum Schreiben eine Datei mit gleichem Dateinamen überschrieben (gelöscht) wird. Ein versehentliches Verwechseln von **I** und **O** kann also fatale Folgen haben !

Nach dem öffnungsvorgang erhält die Datei **#dateinummer** als Kennzeichnung. **dateinummer** kann jedoch nicht beliebig groß gewählt werden. Als Maximalwert voreingestellt ist 3 (maximal 3 Dateien zur gleichzeitigen Bearbeitung). Wenn man mehr gleichzeitig geöffnete Dateien braucht, kann man mit dem **CLEAR-** oder **MEMORY**-Kommando die entsprechende Zahl einstellen. Da BASIC für die Verwaltung der maximalen Dateizahl einen Teil des Speichers benutzt, sollte man die maximale Dateizahl nie überdimensionieren.

datei$** gibt an, welche Datei gelesen oder geöffnet werden soll. **datei$ muß eine eindeutige Dateikennzeichnung ergeben. Es kann jedoch auch eine Laufwerkskennzeichnung mit angegeben werden.

Beispiel:

```
10 OPEN "O",#1,"M:TEST.DAT"          | in Laufwerk M: die Datei TEST.DAT zum
                                     | Schreiben einrichten und der Dateinummer
                                     | 1 zuweisen

10 OPEN "I",#2,"PROFILE.GER"         | im aktuellen Laufwerk die Datei
                                     | PROFILE.GER zum Lesen öffnen und der
                                     | Dateinummer 2 zuweisen
```

Hinweis Das **OPEN**-Kommando wird auch zum Eröffnen von indizierten Dateien und Dateien mit wahlfreiem Zugriff benutzt. Eine genaue Beschreibung in diesem Zusammenhang befindet sich im Teil über **JETSAM** bzw. **Dateien mit wahlfreiem Zugriff** in diesem Buch.

O P T I O N B A S E

Mit **OPTION BASE anfang** läßt sich der Beginn einer Tabelle auf den durch **anfang** spezifizierten Wert festlegen, der nur 0 oder 1 betragen darf.

Wenn durch das **DIM**-Kommando eine Tabelle eingerichtet wird, beginnt sie an der durch **RUN** voreingestellten Position 0; es ist also möglich, **variable(0)** zu benutzen. In einigen Fällen, reicht es jedoch, sie erst ab der Position 1 beginnen zu lassen, nicht zuletzt um Speicherplatz einzusparen. Wichtig ist jedoch, daß nur ein **OPTION BASE**-Kommando pro Programm verwendet werden darf und daß es **vor** einem **DIM**-Kommando stehen muß.

Beispiel:

```
10 OPTION BASE 1
20 DIM a(20,20),b(40)
30 FOR i=1 to 20
40   FOR u=1 to 20
50     a(i,u)=u
60   NEXT
70 NEXT
80 FOR i=1 to 40
90   b(i)=40-i
100 NEXT
110 PRINT "a(10,15)=";a(10,15)
120 PRINT "b(19)=";b(19)
130 END
```


RUN:

```
a(10,15)= 15
b(19)= 21
Ok
■
```

O P T I O N F I E L D

Bei diesem Kommando handelt es sich um einen **JETSAM**-Befehl. Genaueres erfahren Sie im **JETSAM**-Teil auf Seite 193.

O P T I O N F I L E S

Mit **OPTION FILES einstellung$** kann das aktuelle Bezugslaufwerk und die aktuelle Usernummer geändert werden.

einstellung$** kann nur das neue Laufwerk oder die neue Usernummer enthalten. Wenn das Laufwerk geändert werden soll, muß **einstellung$ den Kennbuchstaben des neuen Laufwerks enthalten. Wenn die Usernummer geändert werden soll, muß in **einstellung$** die neue Usernummer stehen. Die vorgenommenen Änderungen bleiben jedoch nur während der Arbeit mit BASIC erhalten. Springt man wieder ins Betriebssystem zurück, sind wieder die Werte eingestellt, wie sie es vor dem Aufruf von BASIC waren.

Beispiel:

```
OPTION FILES "B"          | Laufwerk B:

OPTION FILES "5"          | Userbereich 5
```

O P T I O N I N P U T

Mit **OPTION INPUT = keyboard_status,get_character** läßt sich ein Maschinenunterprogramm für die Konsoleneingabe in BASIC einbinden.

keyboard_status bezeichnet die Adresse des Unterprogramms, welches von BASIC regelmäßig aufgerufen wird, um festzustellen, ob eine Taste gedrückt wurde (es ist daher aus Geschwindigkeitsgründen sinnvoll, es kurz zu halten). Das Ma-

schinenprogramm benötigt keine Einsprungbedingungen. Die Aussprungbedingungen sind jedoch folgende: wenn eine Taste gedrückt wurde, muß das Carry-Flag gesetzt sein (andernfalls muß es rückgesetzt sein) und das A-Register muß das Zeichen von der Tastatur enthalten. Wenn kein Zeichen anliegt, kann es jedoch auch zerstört sein. Die drei Doppelregister BC, DE und HL, sowie die restliches Flags können in jedem Fall zerstört sein. Alle anderen Register müssen jedoch immer erhalten bleiben.

get_character bezeichnet die Adresse des Unterprogramms, welches von BASIC nur dann aufgerufen wird, wenn es ein Zeichen benötigt. Die Routine muß warten, bis ein Zeichen von der Tastatur anliegt und es dann übergeben. Auch dieses Unterprogramm benötigt keine Einsprungbedingungen. Die Aussprunghedingung ist, daß das A-Register das Zeichen von der Tastatur enthält (z.B. durch die BDOS-Funktion 1). Die drei Doppelregister BC, DE und HL und alle Flags können wiederum zerstört sein, die restlichen Register müssen jedoch erhalten bleiben.

OPTION INPUT und **RUN** heben die Umleitung der Aufrufe zu Ihrer Routine auf und lassen BASIC wieder die Originalroutine benutzen.

Beispiel:

```
OPTION INPUT = &HF500,&HF5A0
```

(Auf ein ausführliches Beispiel wird an dieser Stelle verzichtet, da die Originalroutinen von BASIC diese Aufgabe ausreichend gut bewältigen. Ich nehme an, daß derjenige Leser, der eine andere Routine benötigt, in der Lage ist, sich mit diesen Informationen eine eigene Routine zu basteln.)

O P T I O N L P R I N T

Mit **OPTION LPRINT = ausgabe** läßt sich ein an den Drucker ausgegebenes Zeichen (durch **LPRINT**) auf Ihre Routine umlenken.

ausgabe bezeichnet die Adresse Ihres Maschinenunterprogramms, welches das für den Drucker bestimmte Zeichen verarbeitet. Der Einsprung: das C-Register enthält das an den Drucker zu schickende Zeichen. Der Aussprung: die Register A, BC, DE und HL sowie alle Flags können zerstört sein.

OPTION LPRINT und **RUN** heben die Umleitung der Aufrufe zu Ihrer Routine auf und lassen BASIC wieder die Originalroutine benutzen.

Beispiel:

```
10 FOR adr=&HF500 to &HF506
20   READ byte
30   POKE adr,byte
40 NEXT
50 OPTION LPRINT = &HF500
60 LPRINT "Test für den AUXOUT:-Kanal."
70 END
80 DATA &H59            :'LD E,C
90 DATA &HOE,&H04       :'LD C,04    Umleitung von LPRINT
100 DATA &HCD,&H05,&H00 :'CALL BDOS  auf den AUXOUT:-Kanal
110 DATA &HC9           :'RET
```

RUN:

```
Test für den AUXOUT:-Kanal.
```

O P T I O N N O T T A B

OPTION NOT TAB schaltet das **TAB**-Kommando (siehe dort) beim Ausdruck auf dem Bildschirm oder auf dem Drucker ab. Dieser Zustand bleibt solange erhalten, bis ein **RUN**- oder ein **OPTION TAB**-Kommando ausgeführt wird.
Die **TAB**-Abschaltung wird nötig, wenn man z.B. das Zeichen mit dem ASCII-Code 9 (↓) darstellen möchte oder verhindern will, daß bei versehentlichem Drücken der TAB-Taste während eines **INPUT**-Kommandos Verwirrung gestiftet wird, weil man nicht korrekt zurücklöschen kann (durch [←DEL]).

Beispiel:

```
10 INPUT "Probieren Sie ein paar Buchstaben und die TAB-Taste aus und drücken Sie dann
DEL",a$
20 OPTION NOT TAB
30 INPUT "Das gleiche noch einmal",a$
40 END
```

RUN:

```
Probieren Sie die TAB-Taste aus und drücken Sie dann DEL_ (Nach drücken der
TAB-Taste können Sie nicht bis zur Ausgangsposition des Cursors zurücklöschen.)
Das gleiche noch einmal_ (Jetzt geht es.)
```

O P T I O N P R I N T

Mit **OPTION PRINT = ausgabe** läßt sich ein an den Bildschirm ausgegebenes Zeichen (durch **PRINT**) auf Ihre Routine umlenken.

ausgabe bezeichnet die Adresse Ihres Maschinenunterprogramms, welches das für den Bildschirm bestimmte Zeichen verarbeitet. Der Einsprung: das C-Register enthält das an den Bildschirm zu schickende Zeichen. Der Aussprung: die Register A, BC, DE und HL sowie alle Flags können zerstört sein.

OPTION PRINT und **RUN** heben die Umleitung der Aufrufe zu Ihrer Routine auf und lassen BASIC wieder die Originalroutine benutzen.

Beispiel:

```
10 FOR adr=&HF500 to &HF506
20   READ byte
30   POKE adr,byte
40 NEXT
50 OPTION PRINT = &HF500
60 PRINT "Test für den AUXOUT:-Kanal."
70 END
80 DATA &H59           :'LD E,C
90 DATA &H0E,&H04      :'LD C,04    Umleitung von PRINT
100 DATA &HCD,&H05,&H00 :'CALL &DOS  auf den AUXOUT:-Kanal
100 DATA &HC9          :'RET
```

RUN:

```
Test für den AUXOUT:-Kanal.
```

O P T I O N R U N

Durch ein **OPTION RUN**-Kommando in einem laufenden Programm wird verhindert, daß es durch ↑C (=[ALT]+[C] / [STOP]) oder ↑S (=[ALT]+[S]) angehalten werden kann. ↑C bricht normalerweise ein Programm ab, ↑S läßt es auf den nächsten Tastendruck warten). **OPTION RUN** bewirkt auch, daß das Programm schneller abläuft, da BASIC nicht mehr die Tastatur abfragen muß (nach ↑C und ↑S). Man sollte mit diesem Kommando jedoch vorsichtig umgehen. Wenn BASIC in einer Endlosschleife aktiv und **OPTION RUN** wirksam ist, bleibt einem zum Abbrechen nichts anderes übrig, als den Computer aus- und anzuschalten, was einen evtl. unwiederruflichen Datenverlust zur Folge hat.
Um **OPTION RUN** zu deaktivieren wird **OPTION STOP** verwendet.

Beispiel:

```
10 OPTION RUN
20 t$=INPUT$(1)
30 PRINT ASC(t$)
40 END
```

RUN:

```
x (Testen Sie versch. Tasten aus, auch ↑C und ↑S !)
```

O P T I O N S T O P

OPTION STOP schaltet die Wirkung von **OPTION RUN** wieder ab, d.h. ↑C (=[ALT]+[C] / [STOP]) und ↑S (=[ALT]+[S]) haben wieder ihre alte Wirkung (↑C bricht ein Programm ab, ↑S läßt es auf den nächsten Tastendruck warten).

Beispiel:

```
10 OPTION RUN
20 'Hier steht Ihr Programm, das Sie nicht abbrechen / anhalten können
30 OPTION STOP
40 'Jetzt können Sie es wie gewohnt abbrechen / anhalten
50 END
```

O P T I O N T A B

OPTION TAB schaltet das **TAB**-Kommando (siehe dort) beim Ausdruck auf dem Bildschirm oder auf dem Drucker wieder ein, wenn es durch **OPTION NOT TAB** ausgeschaltet wurde.

Beispiel:

```
10 OPTION NOT TAB
20 INPUT "Probieren Sie ein paar Buchstaben und  die TAB-Taste aus und drücken Sie dann
DEL",a$
30 OPTION TAB
40 INPUT "Das gleiche noch einmal",a$
50 END
```

RUN:

```
Probieren Sie die TAB-Taste aus und Drücken Sie dann DEL_ (Nach Drücken der
TAB-Taste können Sie bis zur Ausgangsposition des Cursors
zurücklöschen.)
Das gleiche noch einmal_ (Jetzt geht es nicht mehr.)
```

O S E R R

Die **OSERR**-Funktion dient zur näheren Identifizierung des Fehlers 21. Da keine weiteren Informationen über diesen Fehler vorliegen und er wahrscheinlich in der CP/M-Version nie auftreten wird, hat diese Funktion keine Bedeutung.

O U T

OUT portnummer,wert sendet **wert** (0..255) zum durch **portnummer** (0..255) gewählten Ausgabekanal des Prozessors.

Beispiel:

```
10 FOR wert=0 to 255
20   OUT 246,wert
30   FOR u=0 to 30
40   NEXT
50 NEXT
60 OUT 246,0
70 END
```

RUN:

(Der aktuelle Bildschirminhalt rollt nach oben und erscheint wieder am unteren Bildschirmrand.)

P E E K

PEEK (adresse) dient zum Lesen eines Wertes aus der durch **adresse** spezifizierten Speicherstelle. **adresse** muß eine ganze Zahl zwischen 0 und 65535 (0000H..0FFFFH) ergeben. **PEEK** liefert einen Wert (Byte) zwischen 0 und 255 (00H..0FFH).

Beispiel:

```
10 PRINT CHR$(27)+"E"                    | Bildschirm löschen
20 stu$=HEX$(PEEK(&HFBF6),2)             | Stunden auslesen
30 min$=HEX$(PEEK(&HFBF7),2)             | Minuten auslesen
40 sek$=HEX$(PEEK(&HFBF8),2)             | Sekunden auslesen
50 zeit$=stu$+":"+min$+":"+sek$          | zeit$ zusammensetzen
60 PRINT CHR$(27)+"H";zeit$              | Zeit links oben ausgeben
70 IF INKEY$="" THEN 20                  | sooft wiederholen bis Tastendruck
80 END                                   | Programmende
```

RUN:

Der Bildschirm wird gelöscht, in der linken oberen Ecke erscheint die aktuelle Uhrzeit. (Der Abbruch der Uhr geschieht durch Tastendruck).
Siehe auch **POKE** (Beispiel).

P O K E

POKE adresse,wert dient zum Schreiben eines Wertes in die durch **adresse** spezifizierte Speicherstelle. **adresse** muß eine ganze Zahl zwischen 0 und 65535 (0000H.. 0FFFFH) ergeben. **wert** muß eine ganze Zahl zwischen 0 und 255 (=00H..0FFH) ergeben. Es stellt das Byte dar, welches geschrieben (ge**POKE**d) werden soll.

Beispiel:

```
10 INPUT "Stunde: ",stu$:POKE &HFBF6,VAL("&H"+stu$)
20 INPUT "Minute: ",min$:POKE &HFBF7,VAL("&H"+min$)
30 INPUT "Sekunde:",sek$:POKE &HFBF8,VAL("&H"+sek$)
40 END
```

RUN:

```
Stunde: stunden
Minute: minuten
Sekunde:sekunden
Ok
■
```

Siehe auch **PEEK** (Beispiel).

P O S (0)

POS(0) ermittelt die aktuelle logische Position des Cursors. Das heißt, daß die ermittelte Position nicht unbedingt etwas mit der tatsächlichen zu tun hat, weil jedes Zeichen, dessen ASCII-Code kleiner als 32 ist, nicht mitgezählt wird. Eine Ausnahme machen nur die Zeichen mit dem ASCII-Code 8, 9 und 13.

Die Bedeutung:

ASCII:	8	entspricht einem Rückschritt, der den Zähler um 1 verringert, falls die Position nicht schon 1 ist
	9	entspricht einem TAB, der in Leerstellen umgewandelt wird, von denen jede gezählt wird
	13	entspricht einem Wagenrücklauf, der die Position auf 1 setzt

Beispiel:

```
10 PRINT "Dies ist ein Test.";              | der Cursor befindet sich nun am Zeilenende
20 x=POS(0)
30 PRINT "Die aktuelle Position des Cursors beträgt:";x
40 END
```

RUN:

```
Dies ist ein Test.
Die aktuelle Position des Cursors beträgt 19
Ok
■
```

P R I N T

PRINT text; bewirkt die Ausgabe des **textes** auf der Konsole (meistens Bildschirm).

text kann dabei eine Variablenliste oder ein fester Ausdruck sein. Die Variablenliste kann aus Variablen vom Typ **string** und **numerisch** bestehen. Wenn die einzelnen Variablen dabei durch ein Komma getrennt werden, wird jedesmal für ein Komma ein Tabulator eingefügt. Wenn die Variablen durch ein Semikolon getrennt werden, werden die Werte der Variablen ohne zusätzliche Leerstellen aneinandergereiht gedruckt. Bei diesem Fall muß man jedoch beachten, daß eine Variable numerischen Typs eine Leerstelle vor (für Vorzeichen) und nach der Zahl beansprucht. Der feste Ausdruck ist entweder eine Zahl oder ein Text in Anführungsstrichen.

Ein Semikolon am Ende des Textes bewirkt, daß kein Wagenrücklauf ausgeführt wird. Ein Komma würde einem Tabulator gleichkommen, wobei auch hier kein Wagenrücklauf ausgeführt wird. Wenn am Ende kein Zeichen steht, wird ein ganz normaler Zeilenvorschub ausgeführt.

Beispiel:

```
10 a=8256
20 a$="SCHNEIDER JOYCE PCW"
30 PRINT "Dieses Beispiel läuft auf einem ";a$;a
40 a=10:b=20:c=-30
50 PRINT a,b,c,999;
60 PRINT "Test"
70 END
```

RUN:

```
Dieses Beispiel läuft auf einem SCHNEIDER JOYCE PCW 8256
 10            20           -30            999 Test
Ok
■
```

P R I N T

PRINT #dateinummer,text; bewirkt die Ausgabe des **textes** in eine Datei.
text kann dabei eine Variablenliste oder ein fester Ausdruck sein. Die Variablenliste kann aus Variablen vom Typ **string** und **numerisch** bestehen. Wenn die einzelnen Variablen dabei durch ein Komma getrennt werden, wird jedesmal für ein Komma ein Tabulator eingefügt. Wenn die Variablen durch ein Semikolon getrennt werden, werden die Werte der Variablen ohne zusätzliche Leerstellen aneinandergereiht gedruckt. Bei diesem Fall muß man jedoch beachten, daß eine Variable numerischen Typs eine Leerstelle vor (für Vorzeichen) und nach der Zahl beansprucht. Der feste Ausdruck ist entweder eine Zahl oder ein Text in Anführungsstrichen.

Ein Semikolon am Ende des Textes bewirkt, daß kein Wagenrücklauf ausgeführt wird. Ein Komma würde einem Tabulator gleichkommen, wobei auch hier kein Wagenrücklauf ausgeführt wird. Wenn am Ende kein Zeichen steht, wird ein ganz normaler Zeilenvorschub ausgeführt.

Beispiel:

```
10 OPEN "O",#1,"M:TEST"
20 a=8256
30 a$="SCHNEIDER JOYCE PCW"
40 PRINT #1,"Dieses Beispiel läuft auf einem ";a$;a
50 a=10:b=20:c=-30
60 PRINT #1,a,b,c,999;
70 PRINT #1,"Test"
80 CLOSE
90 DISPLAY "M:TEST"
100 END
```

RUN:

```
Dieses Beispiel läuft auf einem SCHNEIDER JOYCE PCW 8256
 10            20           -30            999 Test
Ok
■
```

P U T

Bei diesem Kommando handelt es sich um einen Befehl, um in Jetsam-Dateien bzw. Dateien mit wahlfreiem Zugriff Datensätze zu schreiben. Genaueres erfahren Sie im Teil über **JETSAM** bzw. **Dateien mit wahlfreiem Zugriff** auf den Seiten 190 und 206.

R A N D O M I Z E

Mit **RANDOMIZE zahl** wird der Zufallszahlengenerator auf einen bestimmten Anfangswert gesetzt. **zahl** stellt eine beliebige ganze Zahl dar. Wird **zahl** weggelassen, reagiert BASIC mit der Frage **"Random Number Seed ?"** und fordert Sie damit zur manuellen Eingabe von **zahl** auf.

BASIC macht eine neue Zufallszahl immer von der letzten abhängig. Wenn also bei jeder Zufallszahlenreihe mit dem gleichen Anfangswert begonnen wird, sind auch die nachfolgenden "Zufallszahlen" identisch. Um diesem Manko entgegenzuwirken, gibt es das **RANDOMIZE**-Kommando, mit dem sich der Anfangswert (**zahl**) unbegrenzt variieren läßt.

Beispiel:

```
10 wert=PEEK(&HFBF8)                    | aktuelle Sekunden
20 RANDOMIZE wert
30 FOR i=1 to 20
40   PRINT INT(RND(1)*10)
50 NEXT
60 END
```

RUN:

Es erscheint eine Zufallszahlenreihe von 20 Elementen. Wenn Sie dieses Beispielprogramm mehrmals hintereinander austesten, werden Sie feststellen, daß die neue Zufallszahlenreihe in den meisten Fällen von der vorherigen abweicht Wiederholungsquote: alle 60 Läufe eine Wiederholung (da von Sekunden abhängig) .

R A N K S P E C

Bei diesem Kommando handelt es sich um einen **JETSAM**-Befehl. Genaueres erfahren Sie im **JETSAM**-Teil auf Seite 174.

R E A D

READ variablenliste liest die in den **DATA**-Zeilen gespeicherten Informationen und weist sie den in **variablenliste** genannten Variablen zu. **variablenliste** ist in der Form **variable1,variable2,variable3,...** anzugeben. Dabei ist selbstverständlich, daß die Variablentypen mit den Typen der durch READ gelesenen Informationen übereinstimmen müssen. Auch muß man darauf achten, daß man nicht versucht, hinter dem letzten **DATA**-Element zu lesen. Versucht man es trotzdem, reagiert BASIC mit dem Error 4: **DATA exhausted** (was auch sonst ?). Vgl. hierzu auch: **RESTORE**.

Beispiel:

```
10 RESTORE 130
20 FOR i=1 to 3
30  READ a
40  PRINT a
50 NEXT
60 PRINT
70 RESTORE 140
80 FOR i=1 to 4
90  READ a$
100  PRINT a$
110 NEXT
120 END
130 DATA 10,-1.234,-97531
140 DATA Test,Autobahnraststätte
150 DATA "Kommatest,Doppelpunkttest:",*6,19,23,30,36,45 / 5
```

RUN:

```
 10
-1.234
-97531

Test
Autobahnraststätte
Kommatest,Doppelpunkttest:
6,19,23,30,36,45 / 5
Ok
■
```

R E M

Mit **REM kommentar** läßt sich ein BASIC-Programm mit Kommentaren versehen. Nach dem **REM**-Kommando wird jedes Zeichen als Kommentar aufgefaßt, egal ob es sich dabei tatsächlich um einen Kommentar handelt. Das **REM**-Kommando kann grundsätzlich an jeder Stelle des Programms stehen, jedoch darf es nicht hinter einem **DEL**, **ERA**, **DIR**, **REN** und **TYPE**-Kommando stehen, weil diese Befehle den Rest der Zeile als Parameter interpretieren. Statt **REM** kann auch vereinfacht ein Apostroph (') verwendet werden. Er hat die gleiche Wirkung wie **:REM**. Der Apostroph darf jedoch nicht am Ende einer **DATA**-Zeile ohne führenden Doppelpunkt stehen, da er sonst von **READ** als Beginn eines einfachen Stringes interpretiert wird.

Beispiel:

```
10 REM Dies ist ein Test

10 PRINT "Test":REM Dies ist ein Test

10 'Dies ist ein Test

10 PRINT "Test" 'Dies ist ein Test

10 DATA 1,2,3:REM Dies ist ein Test

10 DATA 1,2,3:'Dies ist ein Test
```

R E N

Durch Verwendung von **REN neuname.ext = altname.ext** läßt sich eine Datei umbenennen. **altname.ext** bezeichnet dabei die Datei, die umbenannt werden soll. Sie wird dann in **neuname .ext** umbenannt. Wichtig ist dabei, daß eine Datei mit dem Namen **neuname.ext** nicht schon auf dem aktuellen Laufwerk existieren darf. Es ist außerdem nicht empfehlenswert, eine offene Datei umzubenennen.

Beispiel:

```
10 DIR *.BAS
20 PRINT
30 REN R.BAS=RPED.BAS
40 DIR *.BAS
50 END
```

RUN:

```
RPED    .BAS  ... (mindestens RPED.BAS, sofern RPED.BAS auf Diskette vorhanden)
R       .BAS  ... (mindestens R.BAS)
Ok
■
```

R E N U M

RENUM neubeginn,altbeginn,abstand nummeriert das aktuelle Programm nach den angegebenen Parametern neu durch, wobei auch die Zeilennummern in **ELSE-**, **GOSUB-**, **GOTO-**, **LIST-**, **LLIST-**, **RESTORE-**, **RESUME-**, **RUN-**, **THEN-** und **DELETE**-Kommandos automatisch angepaßt werden. Außerdem geschieht dieser Vor-

gang bei einem **DELETE** eines **CHAIN MERGE**-Kommandos.

neubeginn bezeichnet die Zeilennummer, ab der das neu numerierte Programm abgelegt werden soll. Wird diese Angabe weggelassen, wird das Programm ab Zeile 10 abgelegt.

altbeginn bezeichnet die Zeilennummer, bei der die Neunumerierung beginnen soll. Wird diese Angabe weggelassen, beginnt die Numerierung bei der ersten Zeile des Programms.

abstand bezeichnet den Zeilenabstand, den die Zeilen nach der Neunumerierung haben sollen. Wird diese Angabe weggelassen, beträgt der Zeilenabstand 10.

Findet BASIC während der Ausführung des **RENUM**-Kommandos einen Verweis zu einer nicht vorhandenen Zeile, gibt es die Fehlermeldung **Undefined line xxxx in yyyy** aus. **xxxx** gibt die nicht existierende Zeilennummer an, **yyyy** die (neu numerierte) Zeile, in der der Fehler auftritt. Nach der Ausführung des **RENUM**-Kommandos springt BASIC in den Direktmodus zurück.

Beispiel:

```
7 '
16 '
21 '
55 '
```

(Probieren Sie die diversen Beispiele aus.)

```
RENUM          | erste Zeile 10, Zeilenabstand 10

RENUM 10       | erste Zeile 10, Zeilenabstand 10

RENUM 10,16    | erste Zeile 10 (ehemals 16), Zeilenabstand 10

RENUM ,,5      | erste Zeile 10, Zeilenabstand 5

RENUM 1,,1     | erste Zeile 1, Zeilenabstand 1
```

R E S E T

RESET laufwerk$** setzt das mit **laufwerk$ spezifizierte Laufwerk zurück und schließt dabei alle offenen Dateien.

laufwerk$ gibt das Diskettenlaufwerk an, was zurückgesetzt werden soll. Die Laufwerksbezeichnung muß dabei zwischen A und P liegen.

Beispiel:

```
RESET "A"

10 lw$="M"
20 RESET lw$
30 END
```

R E S T O R E

RESTORE zeilennummer setzt den **DATA**-Zeiger auf die durch **zeilennummer** spezifizierte **DATA**-Zeile.

zeilennummer bezeichnet die Zeile, auf die der **DATA**-Zeiger gerichtet werden soll. Die **zeilennummer** muß dabei als Konstante eingegeben werden. Wird **zeilennummer** weggelassen, wird der **DATA**-Zeiger auf das erste **DATA**-Element gesetzt. Der **DATA**-Zeiger bezieht sich auf die einzelnen **DATA**-Elemente. Mit jedem **READ**-Kommando wird er auf das folgende Element gesetzt. Das **RESTORE**-Kommando wird z.B. nötig, wenn man am Ende einer **DATA**-Liste angelangt ist und sie noch einmal (von vorne) lesen möchte.

Beispiel:

```
10 RESTORE 130
20 FOR i=1 to 3
30  READ a
40  PRINT a
50 NEXT
60 PRINT
70 RESTORE 140
80 FOR i=1 to 4
90  READ a$
100  PRINT a$
110 NEXT
120 END
130 DATA 10,-1.234,-97531
140 DATA Test,Autobahnraststätte
150 DATA "Kommatest,Doppelpunkttest:",*6,19,23,30,36,45 / 5
```

RUN:

```
 10
-1.234
-97531

Test
Autobahnraststätte
Kommatest,Doppelpunkttest:
6,19,23,30,36,45 / 5
Ok
■
```

R E S U M E

Wenn ein Fehler in einem Programm auftrat und er mit einer **ON ERROR GOTO**-Routine behandelt wurde, bietet **RESUME** die Möglichkeit, das Programm normal fortzusetzen.

Dazu gibt es drei Möglichkeiten:

RESUME	**RESUME** springt zu dem Befehl zurück, bei dem der Fehler aufgetreten ist.
RESUME zeilennr	**RESUME** springt zu der durch **zeilennr** spezifizierten Zeile.
RESUME NEXT	**RESUME** springt zu dem Befehl, der unmittelbar auf den fehlerhaften folgt.

Beispiel:

```
10 ON ERROR GOTO 100   | ab 100: Fehlerbehandlungsroutine
20 PRINT "Diese Zeile ist korrekt,"
30 PRINT "in der nächsten wird ein SYNTAX-ERROR simuliert."
40 ERROR 2
50 PRINT "Das Programm wird fortgesetzt."
60 END
100 PRINT "In Zeile";ERL;"tritt der Fehler mit der Nummer";ERR;"auf."
110 RESUME NEXT                | Programm nach der fehlerhaften Zeile fortsetzen
```

RUN:

```
Diese Zeile ist korrekt,
in der nächsten wird ein SYNTAX-ERROR simuliert.
In Zeile 40 tritt der Fehler mit der Nummer 2 auf.
Das Programm wird fortgesetzt.
Ok
■
```


R E T U R N

RETURN beendet ein durch **GOSUB** oder **ON x GOSUB** aufgerufenes Unterprogramm. Nach dem **RETURN**-Kommando setzt BASIC das Programm mit dem unmittelbar auf **GOSUB** folgenden Befehl fort.

Beispiel:

```
10 PRINT "Hier läuft das Hauptprogramm."
20 PRINT "Gleich wird das Unterprogramm aufgerufen."
30 GOSUB 100
40 PRINT "Jetzt läuft das Hauptprogramm weiter."
50 END
100 PRINT CHR$(7);"---- UNTERPROGRAMM ----";CHR$(7)
110 RETURN
```

RUN

```
Hier läuft das Hauptprogramm.
Gleich wird das Unterprogramm aufgerufen.
(Pieps)---- UNTERPROGRAMM ----(Pieps)
Jetzt läuft das Hauptprogramm weiter.
Ok
■
```

R I G H T $

RIGHT$ (wort$,n) ermittelt vom rechten Ende **n** Zeichen von **wort$**. **n** muß dabei zwischen 0 und 255 liegen.
Ist **n** größer als die gesamte Länge von **wort$**, so wird **wort$** ohne Veränderungen als String übergeben.

Beispiel:

```
10 text$="JOYCE  mehr als ein Textsystem."
20 rechts$=RIGHT$(text$,24)
30 PRINT rechts$
40 END
```

RUN:

```
mehr als ein Textsystem.
Ok
■
```

R N D

Die Funktion **RND (x)** übergibt eine Zufallszahl, die nur in dem Sinne zufällig ist, als daß man die Errechnungsprinzipien für die Zufallszahl von BASIC nicht kennt. Es ist nur bekannt, daß BASIC eine Zufallszahl aus der vorherigen ableitet. Beginnt man also mit der gleichen Anfangszahl (s. **RANDOMIZE**), sind auch die nachfolgenden Zufallszahlen gleich.

Für die Form von RND gibt es vier Möglichkeiten:

RND	Hier wird eine neue Zufallszahl erzeugt.
RND (positiv.x)	Wie **RND (positiv.x** heißt: **positiv.x** ist größer als 0).
RND (0)	Hier wird immer die letzte Zufallszahl übergeben.
RND (negativ.x)	Für jedes **negativ.x** wird eine neue Zufallszahl erzeugt (**negativ.x** heißt: **negativ.x** ist kleiner als 0).

Jede Form von **RND** übergibt eine Zahl einfacher Genauigkeit, die größer gleich 0 und kleiner als 1 ist.

Beispiel:

```
10 wert=PEEK(&HFBF8) 'aktuelle Sekunden
20 RANDOMIZE wert
30 FOR i=1 to 20
40   PRINT INT(RND(1)*10)
50 NEXT
60 END
```

RUN:

Es erscheint eine Zufallszahlenreihe von 20 Elementen. Wenn Sie dieses Beispielprogramm mehrmals hintereinander austesten, werden Sie feststellen, daß die neue Zufallszahlenreihe in den meisten Fällen von der vorherigen abweicht (Wiederholungsquote: alle 60 Läufe eine Wiederholung (da von Sekunden abhängig)).

R O U N D

ROUND (zahl,stellen) rundet **zahl** auf die durch **stellen** angegebenen Vor- bzw. Nachkommastellen.

zahl stellt die Zahl dar, die gerundet werden soll. Es kann sich dabei um eine beliebige Zahl handeln.

stellen gibt an, auf wieviel Stellen vor oder nach dem Komma gerundet werden soll. Dabei ergeben sich drei Möglichkeiten:

stellen > 0	Hier wird **zahl** auf die angegebenen Nachkommastellen gerundet.
stellen = 0	**zahl** wird zu einer ganzen Zahl gerundet.
stellen < 0	**zahl** wird auf die positiven angegebenen Vorkommastellen gerundet. Dabei werden die Vorkommastellen gleich 0 gesetzt (1200000).

In jedem Fall muß **stellen** jedoch zwischen -39 und +39 liegen. Wenn Sie **stellen** nicht angeben, wird **stellen = 0** angenommen.

Beispiel:

```
10 a=1234.567
20 PRINT "ROUND (";a;", 1) ergibt: ";ROUND (a,1)
30 PRINT "ROUND (";a;", 0) ergibt: ";ROUND (a,0)
40 PRINT "ROUND (";a;",-2) ergibt: ";ROUND (a,-2)
50 END
```

RUN:

```
ROUND ( 1234.567 , 1) ergibt: 1234.6
ROUND ( 1234.567 , 0) ergibt: 1235
ROUND ( 1234.567 ,-2) ergibt: 1200
OK
■
```

R S E T

Bei diesem Kommando handelt es sich um einen **JETSAM**-Befehl. Genaueres erfahren Sie im **JETSAM**-Teil auf Seite 173.

R U N

Prinzipiell startet das **RUN**-Kommando ein vorhandenes Programm und initialisiert den Speicher, so wie er nach dem Laden von BASIC war (Löschung aller Variablen, Variableneinstellungen, etc.). Unberührt von der Initialisierung bleiben lediglich die Einstellungen des **MEMORY**-Kommandos.

RUN hat jedoch auch zwei Zusatzfunktionen:

RUN datei$** Das unter der Bezeichnung **datei$ abgespeicherte BASIC-Programm wird geladen und ausgeführt. Ist in **datei$** keine Extension angegeben, wird **.BAS** angenommen. Zusätzlich kann vor dem Dateinamen auch das Laufwerk mit angegeben werden.

RUN zeilennr BASIC beginnt die Ausführung des im Speicher befindlichen Programmes bei der angegebenen Zeilennummer (**zeilennr**) [wird **zeilennr** weggelassen, beginnt die Ausführung bei der ersten Zeilennummer].

Beispiel:

```
RUN                 | das im Speicher befindliche Programm wird gestartet

RUN "RPED"          | RPED.BAS wird geladen und gestartet

RUN "RPED.BAS"      | RPED.BAS wird geladen und gestartet

RUN 100             | das aktuelle Programm wird ab Zeile 100 gestartet
```

S A V E

SAVE "datei.ext",A sichert das im Speicher befindliche Programm unter der Dateikennzeichnung **datei.ext** auf Diskette. Ist **.ext** nicht angegeben, wird .BAS angenommen. **,A** bedeutet, daß das Programm im ASCII-Format abgespeichert wird, so daß man es z.B mit einem Texteditor bearbeiten oder mit **TYPE** ansehen könnte. Statt **,A** kann man auch **,P** verwenden, wobei das Programm in geschützter Form gespeichert wird. Nach einem späteren Laden läßt sich daß Programm nicht ohne weiteres auflisten. Will man es "entschützen", benutze man die folgende Befehlsfolge: **OPEN "O",#1,"M:PASS.OFF" :MERGE "M:PASS.OFF"**. Da diese zwei Kommandos den Schutz lahmlegen, ist seine Anwendung witzlos geworden. Benutzt man jedoch weder **,A** noch **,P**, wird das Programm im Normalformat abgespeichert, das nur von BASIC verarbeitet werden kann.

Beispiel:

```
SAVE "TEST                | das aktuelle Programm wird unter TEST.BAS gespeichert

SAVE "M:TEST              | das aktu. Programm wird in M unter TEST.BAS gespeichert

SAVE "TEST.BAS"           | das aktuelle Programm wird unter TEST.BAS gespeichert

SAVE "TEST.ASC",A     | das aktuelle Programm wird im ASCII-Format abgelegt

SAVE "TEST.PAS",P     | das aktuelle Programm wird geschützt abgelegt
```

S E E K K E Y - S E E K S E T

Bei diesen Kommandos handelt es sich um **JETSAM**-Befehle. Genaueres erfahren Sie im **JETSAM**-Teil auf den Seiten 180 bis 189.

S G N

SGN (x) [Signum (x)] prüft, ob **x** positiv, null oder negativ ist. Ist **x** positiv, übergibt **SGN** den Wert 1. Wenn **x** negativ ist, wird -1 übergeben. Wenn **x** 0 ist, wird ebenfalls 0 übergeben.

Beispiel:

```
10 a(1)=10
20 a(2)=0
30 a(3)=-6.92
40 FOR i=1 to 3
50   IF SGN(a(i))=-1 THEN PRINT a(i);"ist negativ."
60   IF SGN(a(i))=0 THEN PRINT a(i);"ist null."
70   IF SGN(a(i))=1 THEN PRINT a(i);"ist positiv."
80 NEXT
90 END
```

RUN:

```
 10 ist positiv.
 0 ist null.
-6.92 ist negativ.
Ok
■
```

S I N

SIN (x) übergibt den Sinus von **x**. **x** muß dabei größer sein als -205884.2734375 und kleiner als 205884.2734375.
Zu beachten ist, daß sich alle Angaben auf das **Bogenmaß** beziehen.

Beispiel:

```
10 x=3.1415927
20 y=4.7123889
30 PRINT "SIN (";x;") ergibt ";SIN (x)
40 PRINT "SIN (";y;") ergibt ";SIN (y)
50 END
```

RUN:

```
SIN ( 3.1415927 ) ergibt  0
SIN ( 4.7123889 ) ergibt -1
Ok
■
```

S P A C E $

SPACE$ (anzahl) erzeugt einen String mit **anzahl** Leerstellen. **anzahl** muß dabei zwischen 0 und 255 liegen.

Beispiel:

```
10 text$="JOYCE"
20 PRINT text$
30 text$=text$+SPACE$ (10)+text$
40 PRINT text$
50 END
```

RUN:

```
JOYCE
JOYCE          JOYCE
Ok
■
```


S P C

SPC (anzahl) gibt bei einem **PRINT**, **LPRINT** oder **PRINT#**-Befehl **anzahl** Leerstellen auf dem entsprechendem Medium (Bildschirm, Drucker oder Datei) aus. **SPC** kann nur in diesem Zusammenhang benutzt werden.

anzahl gibt die zu übergebenden Leerstellen an. Ist **anzahl** größer als die eingestellte Zeilenbreite des entsprechenden Mediums, wird von **anzahl** sooft die Zeilenbreite abgezogen, bis **anzahl** kleiner als die Zeilenbreite ist.

Nach einem **SPC**-Kommando wird grundsätzlich ein Semikolon angenommen, sofern es nicht schon angegeben ist. **SPC** kann natürlich aber auch von einem Komma gefolgt werden (die Wirkung von ; und , siehe **PRINT**).

Beispiel:

```
10 text$="JOYCE"
20 PRINT text$;SPC(10);text$
30 END
```

RUN:

```
JOYCE          JOYCE
Ok
■
```

S Q R

SQR (zahl) errechnet die Quadratwurzel der gegebenen **zahl**. Den Gesetzen der Mathematik folgend darf **zahl** nur größer oder gleich Null sein. Durch die rechnerischen Beschränkungen von BASIC übergibt SQR nur einen Wert einfacher Genauigkeit. **zahl** darf jedoch eine Zahl beliebigen Typs sein.

Beispiel:

```
10 zahl=256
20 PRINT "Die Quadratwurzel von";zahl;"beträgt";SQR (zahl)
30 END
```

RUN:

```
Die Quadratwurzel von 256 beträgt 16
Ok
■
```

Hinweis: **Wenn man als Ergebnis von SQR** ein "glattes" Ergebnis erwarten kann, sollte man das Ergebnis mit **ROUND** gegen evtl. Rechenungenauigkeiten von BASIC absichern (8.9999999 statt 9).

S T O P

STOP bewirkt eine Unterbrechung des laufenden Programms und läßt BASIC in den Direktmodus zurückkehren. Dieses Kommando kann dazu nützlich sein, das Programm an einer bestimmten Stelle anzuhalten, um dann z.B. Variablen zu setzen oder abzufragen. Durch Eingabe von **CONT** wird das Programm forgesetzt (siehe dort).

Beispiel:

```
10 PRINT "Dies ist ein Programm."
20 STOP
30 PRINT "Dies ist das nach STOP fortgesetzte Programm."
40 END
```

RUN:

```
Dies ist ein Programm.
Break in 20
Ok
CONT (muß von Ihnen eingegeben werden)
Dies ist das nach STOP fortgesetzte Programm.
Ok
■
```

S T R $

STR$ (zahl) wandelt **zahl** in einen String um. Er erhält das Format, wie es durch ein **PRINT**-Kommando erzeugt worden wäre (positive Zahlen erzeugen einen String mit führender Leerstelle, negative Zahlen erzeugen ein führendes Minuszeichen).

Beispiel:

```
10 a=1.234
20 b=-1.234
30 a$=STR$ (a)
40 b$=STR$ (b)
50 PRINT "*";a$;"*"
60 PRINT "*";b$;"*"
70 END
```

RUN:

```
* 1.234*
*-1.234*
Ok
■
```

S T R I N G $

STRING$ (zeichenzahl,zeichen$) oder **STRING$ (zeichenzahl, zeichen_ASCII_Code)** bildet einen String, der aus **zeichenzahl** durch **zeichen$ / zeichen_ASCII_Code** spezifizierten Zeichen besteht.

zeichenzahl gibt an, wieviele Zeichen vom Typ **zeichen$** erzeugt werden sollen. **zeichenzahl** darf nicht größer als 255 sein, da dies die Maximallänge eines Stringes ist.

zeichen$** gibt das Zeichen an, aus welchem der neue String **zeichenzahl**-mal bestehen soll. Wenn **zeichen$ aus mehreren Zeichen besteht, wird das erste Zeichen vervielfältigt.

zeichen_ASCII_Code gibt den ASCII-Code des Zeichens an, aus welchem der neue String **zeichenzahl**-mal bestehen soll.

Beispiel:

```
10 text1$=STRING$(20,"=")
20 text2$=STRING$(20,61)                    | "=" hat ASCII-Code 61
30 PRINT text1$
40 PRINT text2$
50 END
```

RUN:

```
====================
====================

Ok
■
```

S T R I P $

STRIP$ (text$) setzt in allen Zeichen von **text$** das 7. Bit auf Null zurück. **text$** stellt einen beliebigen String dar (max. Länge: 255 Zeichen).

Erklärung: Jedes Zeichen setzt sich aus 8 Bits (0..7) zusammen (=1 Byte). Ein Bit kann nur zwei Zustände annehmen: gesetzt (=1) oder nicht gesetzt (=0). Aus der Folge der gesetzten und nicht gesetzten Bits wird eine Dezimalzahl gebildet (z.B.: 11111111 dual = 255 dezimal). Das 7. Bit ist für Zahlen zwischen 128 und 255 verantwortlich. Wird es vom gesetzten in den ungesetzten Zustand gewandelt, verringert sich die Zahl um 128: 01111111 dual = 127 dezimal. In der Praxis wird das 7. Bit z.B. für eine Dateikennzeichnung in der Extension benutzt. Ist es im ersten Zeichen der Extension gesetzt, ist die Datei mit einem Read Only-Attribut versehen. Im zweiten Zeichen kennzeichnet es das System-Attribut der Datei, im dritten das Archive-Attribut.

Beispiel:

```
10 maske$="*.*"
20 n=1
30 datei$=FIND$(maske$,n)
40 stripdat$=STRIP$(datei$)
50 IF datei$<>"" THEN PRINT "Mit Bit 7:";datei$ : PRINT "Ohne Bit 7:";stripdat$: n=n+1 :
GOTO 30
60 END
```

RUN: (evtl. LOCOSCRIPT-Disk einlegen)

```
Mit Bit 7: datei1.axt
Ohne Bit 7: datei1.ext
      ......
Ok
■
```

S W A P

SWAP (var1,var2) tauscht den Inhalt der Variablen **var1** und **var2** ohne Umweg über eine dritte Variable aus. Vorraussetzung ist, daß **var1** und **var2** vom selben Variablentyp sind.

Beispiel:

```
10 a=-4.567
20 b=123.456
30 PRINT a;b
40 SWAP a,b
50 PRINT a;b
60 END
```

RUN:

```
-4.567  123.456
123.456  -4.567
Ok
■
```

S Y S T E M

SYSTEM veranlaßt BASIC, in das Betriebssystem (CP/M Plus) zurückzuspringen. Alle offenen Dateien werden dabei geschlossen.

Beispiel:

```
10 PRINT "Ende BASIC."
20 SYSTEM
```

RUN:

```
Ende BASIC.
A>■
```

T A B

TAB (position) gibt bei einem **PRINT**, **LPRINT** oder **PRINT#** bis **position** Leerstellen auf dem entsprechendem Medium (Bildschirm, Drucker oder Datei) aus. **TAB** kann nur in diesem Zusammenhang benutzt werden.

position gibt die Druckposition an, bis zu der (von der aktuellen Position) Leerstellen ausgegeben werden. Auf dem Bildschirm z.B. entspricht **position** der Spalte, zu der der Cursor rücken soll. Ist **position** größer als die eingestellte Zeilenbreite des entsprechenden Mediums, wird von **position** sooft die Zeilenbreite abgezogen, bis **position** kleiner als die Zeilenbreite ist. Ist **position** größer als die aktuelle Druckposition, wird ein Zeilenvorschub ausgeführt und es werden Leerstellen ausgegeben, bis **position** erreicht ist.

Nach einem **TAB**-Kommando wird grundsätzlich ein Semikolon angenommen, sofern es nicht schon angegeben ist. **TAB** kann natürlich aber auch von einem Komma gefolgt werden (die Wirkung von ; und , siehe **PRINT**).

Beispiel:

```
10 text$="JOYCE"
20 PRINT text$;TAB(10);text$                    | das zweite JOYCE wird an 10. Stelle
30 END                                          | ausgegeben
```

RUN:

```
JOYCE     JOYCE
Ok
■
```

T A N

TAN (x) übergibt den Tangens von **x**. **x** muß dabei größer sein als -205884.2734375 und kleiner als 205884.2734375.

Zu beachten ist, daß sich alle Angaben auf das **Bogenmaß** beziehen.

Beispiel:

```
10 pi=4*ATN(1)
20 x=0.25*pi
30 y=0.75*pi
40 PRINT "TAN (";x;") ergibt ";TAN (x)
50 PRINT "TAN (";y;") ergibt ";TAN (y)
60 END
```

RUN:

```
TAN ( 0.7853982 ) ergibt  1
TAN ( 2.3561945 ) ergibt -1
Ok
■
```

T R O F F

TROFF (**TR**ace **OFF**) schaltet den TRACE-Modus (siehe **TRON**) ab. Das Programm läuft danach wieder normal weiter.

Beispiel:

```
10 TRON
20 PRINT CHR$(27)+"2"+chr$(0)                   | amerikanischen Zeichensatz einschalten
30 FOR f=1 to 2
40 '
50 NEXT
60 TROFF
70 PRINT CHR$(27)+"2"+chr$(2)                   | deutschen Zeichensatz einschalten
80 END
```

RUN:

```
[20] [30] [40] [50] [30] [40] [50] [60]
Ok
■
```

T R O N

TRON schaltet den TRACE-Modus ein (**TRace ON**). Er bietet die Möglichkeit, den Programmablauf Zeile für Zeile zu überwachen. Dabei wird die aktuelle Zeilennummer in eckigen Klammern (beim amerikanischen Zeichensatz) auf dem Bildschirm angezeigt, bevor die eigentliche Verarbeitung der Zeile beginnt. Abgeschaltet wird der TRACE-Modus durch **TROFF**, **CHAIN**, **LOAD**, **NEW** und **RUN datei$**.

Beispiel:

```
10 TRON
20 PRINT CHR$(27)+"2"+CHR$(0)                  | amerikanischen Zeichensatz einschalten
30 FOR i=1 to 2
40 '
50 NEXT
60 TROFF
70 PRINT CHR$(27)+"2"+CHR$(2)                  | deutschen Zeichensatz einschalten
80 END
```

RUN:

```
[20] [30] [40] [50] [30] [40] [50] [60]

Ok
■
```

T Y P E

TYPE datei.ext zeigt den Inhalt von **datei.ext** an der Konsole. **TYPE** arbeitet genau wie direkt unter CP/M Plus. Die Auflistung kann duch [ALT]+[C] (STOP) abgebrochen, durch [ALT]+[S] (f3/f4) angehalten und durch [ALT]+[Q] (f5/f6) fortgesetzt werden. Zu beachten ist, daß **datei.ext** keine Wildcards (?,*) enthalten darf, sondern eine exakte Dateispezifikation darstellen muß. Der Rest der Zeile wird von **TYPE** als Parameter interpretiert, egal ob es sich tatsächlich um solche handelt.

Beispiel:

```
10 TYPE PROFILE.GER
20 END
```


RUN:

```
(PROFILE.GER muß sich auf der Diskette im aktuellen Laufwerk befinden!)
setdef m:,* [order = (sub,com) temporary = m:]
pip
<m:=basic.com[o]
<m:=dir.com[o]
<m:=erase.com[o]
<m:=paper.com[o]
<m:=pip.com[o]
<m:=rename.com[o]
<m:=setkeys.com[o]
<m:=show.com[o]
<m:=submit.com[o]
<m:=type.com[o]
<
Ok
■
```

U N T

UNT (zahl) wandelt **zahl** in eine vorzeichenlose Integerzahl um.

zahl muß eine ganze Zahl im Bereich von 0 bis 65535 (00H - 0FFFFH) sein.

Die vorzeichenlose Integerzahl ist ein Wert im Bereich von -32768..+32767. Für **zahl**en zwischen 0 und 32767 liefert **UNT** die unveränderte Zahl. Bei 32768 und größer geht es dann mit -32767 aufwärts bis 0 weiter.

Beispiel:

```
10 INPUT "Zahl: ",a
20 IF a<0 OR a>65535 THEN 10                   | evtl. Fehleingabe abfangen
30 PRINT "UNT (";a;") = ";UNT (a)
40 END
```

RUN:

```
Zahl: x
UNT ( x ) = y
Ok
■
```

U P P E R $

UPPER$ (text$) wandelt alle Kleinbuchstaben in **text$** in Großbuchstaben um. Als Kleinbuchstaben werden alle Zeichen mit dem ASCII-Code zwischen 97 und 122 (a-z) erkannt.

Beispiel:

```
10 text$="Joyce"
20 PRINT text$;" in Großbuchstaben umgewandelt ergibt ";UPPER$(text$)
30 END
```

RUN:

```
joyce in Großbuchstaben umgewandelt ergibt JOYCE
OK
■
```

U S I N G

Das **USING format$**-Kommando wird nur im Zusammenhang mit **PRINT** oder **LPRINT** benutzt. Es bewirkt die Anpassung der Ausgabe der Variablen auf dem entsprechenden Medium an das durch **format$** gekennzeichnete Format.
Für **format$** sind verschiedene Optionen einsetzbar:

Jedes # repräsentiert die Position einer Ziffer (Zahlenformat). In **format$** muß mindestens ein # enthalten sein.

. legt die Position des Dezimalpunktes fest. In **format$** darf höchstens ein . enthalten sein.

Ein , bewirkt die Aufteilung der Vorkommastellen in Dreiergruppen, die durch Komma getrennt werden. Das , muß vor den . gesetzt werden.

Mit $$ kann man vor die erste Ziffer bzw. den Dezimalpunkt ein $ setzen. Die $$ müssen vor dem Zahlenformat stehen.

Bei Verwendung von ** werden führende Leerstellen mit Sternen aufgefüllt. Die ** müssen vor dem Zahlenformat stehen.

Durch **$ werden $$ und ** kombiniert. Auch **$ muß vor dem Zahlenformat stehen.

Mit + wird das Vorzeichen der Zahl ausgegeben. Steht + vor dem Zahlenformat, wird das Vorzeichen vor die Zahl oder das $ gestellt. Steht + am Ende des Zahlenformats, wird das Vorzeichen nachgestellt.

Ein - bewirkt die nachgestellte Ausgabe eines Minuszeichens für negative oder eines Leerzeichens für positive Zahlen. - muß hinter dem Zahlenformat stehen.

Bei Verwendung von ↑↑↑↑ wird die Zahl in Exponentialdarstellung ausgegeben. ↑↑↑↑ muß direkt hinter dem Zahlenformat angegeben werden.

Beispiel:

```
10 DEFDBL a                                  | a=doppelt genau
20 a=-123456.789#                            | # wird von BASIC gesetzt
30 PRINT USING "########";a                  | im folgenden werden die verschie-
40 PRINT USING "########.######";a           | densten Formate durchgetestet
50 PRINT USING "########,.######";a          |
60 PRINT USING "$$########,.######";a        |
70 PRINT USING "$$########.######";a         |
80 PRINT USING "**########,.######";a        |
90 PRINT USING "**########.######";a         |
100 PRINT USING "**$########,.######";a      |
110 PRINT USING "**$########.######";a       |
120 PRINT USING "++$$########,.######";a     |
130 PRINT USING "$$########.######+";a       |
140 PRINT USING "$$########.######-";a       |
150 PRINT USING "$$########.######↑↑↑↑-";a   |
160 END
```

RUN:

```
 -123457
 -123456.789000
 -123,456.789000
  -$123,456.789000
  -$123456.789000
***-123,456.789000
***-123456.789000
***-$123,456.789000
***-$123456.789000
   -$123456.789000
   $123456.789000-
   $123456.789000-
$123456789.0000000-03-
Ok
■
```

V A L

VAL (zahl$) wandelt den String **zahl$** in eine äquivalente Variable numerischen Typs um.

zahl$** stellt eine Zahl dar, die durch einen String charakterisiert wird (s. **STR$). Durch **VAL** wird sie einer Variablen numerischen Typs übergeben, mit der man wieder auf übliche Weise rechnen kann.

Beispiel:

```
10 a$="12.345"
20 b$="-10.987"
30 a=VAL (a$)
40 b=VAL (b$)
50 c=a*b
60 PRINT a;b
70 PRINT c
80 END
```

RUN:

```
 12.345 -10.987
-135.6345
Ok
■
```

V A R P T R

VARPTR (variable) ermittelt die Speicheradresse von **variable** (Adresse, an der **variable** im Speicher steht).

variable ist eine Variable beliebigen Typs.

Aufbau der Variablendatenfelder im Speicher:

INTEGER: Die Variablen werden im Lowbyte- / Highbyteformat abgespeichert.

STRING: Das erste Byte bezeichnet die Länge des Strings (daher auch die maximale Stringlänge von 255 = 0FFH Zeichen). Das zweite und dritte Byte gibt die Adresse (Lowbyte- / Highbyteformat) des Strings im Speicher an.

EINFACHE GENAUIGKEIT: Die Bytes eins bis drei geben die Zahl an. Byte 4 gibt den Exponenten an. Er ist um 128 größer als der tatsächliche. Byte 4 = 0 bedeutet, daß die Zahl gleich Null ist.

DOPPELTE GENAUIGKEIT: Die Bytes eins bis sieben geben die Zahl an. Byte 8 gibt den Exponenten an. Er ist um 128 größer als der tatsächliche. Byte 8 = 0 bedeutet, daß die Zahl gleich Null ist.

Wenn statt **variable** eine Dateinummer einer geöffneten Datei angegeben ist [**VARPTR (#dateinummer)**], wird die Anfangsadresse des zugehörigen Dateipuffers übergeben.

Da sich die Variablen immer im Hauptspeicher (TPA) befinden, übergibt **VARPTR** auch nur eine Adresse zwischen 0 und 65535 (=0FFFFH).

Beispiel:

```
10 a$="Dies ist ein Test."
20 stelle=VARPTR (a$)
30 PRINT a$
40 POKE stelle,4        | vortäuschen, daß a$ nur 4 Zeichen lang ist
50 PRINT a$
60 END
```

RUN:

```
Dies ist ein Test.
Dies
Ok
■
```

V E R S I O N

VERSION (parameter) dient zur Ermittlung des verwendeten Computertyps (CP/M = JOYCE oder MS-DOS System). Da dieses Buch nur für den JOYCE geschrieben ist und hoffentlich jeder Benutzer selbst weiß, daß er einen JOYCE hat, spielt das **VERSION**-Kommando keine Rolle.

W E N D

WEND beendet die zum letzten **WHILE** gehörige **WHILE-WEND**-Schleife (s. **WHILE**). Wenn BASIC ein **WHILE**-Kommando findet, prüft es, ob ein zugehöriges **WEND** vorhanden ist. Ist dies nicht der Fall, wird das Programm abgebrochen. Wenn BASIC fündig wird, nimmt es grundsächlich das nächstbeste passende **WEND**. Es nimmt also keine Rücksicht darauf, ob sich das vom Programmierer gewünschte **WEND** vielleicht in einer Unterroutine befindet.

Die Schachtelung einer **WHILE-WEND**-Schleife kann nur folgendermaßen geschehen:

```
WHILE bedingung 1
   WHILE bedingung 2
      WHILE bedingung 3
            |
            |
      WEND
   WEND
WEND
```

Beispiel:

```
10 WHILE a$<>"Ende"     | wiederhole die Schleife solange, bis a$="Ende"
20   INPUT "Bitte Wort eingeben ( Ende beendet ): ",a$
30 WEND
40 PRINT "Programm beendet."
50 END
```

RUN:

```
Bitte Wort eingeben ( Ende beendet ): Test
Bitte Wort eingeben ( Ende beendet ): Ende
Programm beendet.
Ok
■
```

W H I L E

WHILE bedingung markiert den Anfang einer **WHILE-WEND**-Schleife. Die **WHILE-WEND**-Schleife ist im Prinzip nichts anderes als eine Endlosschleife, aus der mit einem IF-Kommando herausgesprungen werden kann:

```
10 IF bedingung THEN 40          10 WHILE bedingung
20 'Kommandoteil                 20 'Kommandoteil
30 GOTO 10                       30 WEND
40 'weiter                       40 'weiter
```


Die Schleife wird solange ausgeführt, bis **bedingung** nicht mehr erfüllt ist. **bedingung** ist in den meisten Fällen eine Vergleichsoperation, z.B. **WHILE a>b**. Die Schleife wird also solange wiederholt, bis **a** nicht mehr größer als **b** ist, denn das Kommando sagt: wiederhole, solange **a** größer als **b** ist.

Beendet wird die Schleife durch das **WEND**-Kommando (s. dort).

Beispiel:

```
10 WHILE a$<>"Ende"
20   INPUT "Bitte Wort eingeben ( Ende beendet ): ",a$
30 WEND
40 PRINT "Programm beendet."
50 END
```

RUN:

```
Bitte Wort eingeben ( Ende beendet ): Test
Bitte Wort eingeben ( Ende beendet ): Ende
Programm beendet.
Ok
■
```

W I D T H

Mit **WIDTH zeilenbreite** kann man die Zeilenbreite der Konsole (Bildschirm) verändern.

zeilenbreite gibt an, nach wieviel Zeichen BASIC eine neue Zeile beginnen soll. **zeilenbreite** muß zwischen 1 und 255 liegen.

Beispiel:

```
WIDTH 40          | Bildschirm 40 Zeichen breit

WIDTH 80          | Bildschirm 80 Zeichen breit

WIDTH 255         | Bildschirm 255 Zeichen breit (nach 90 Zeich. neue Zeile)

WIDTH 255,255     | Bei Bildschirmausgaben durch CHR$(27)+"Y" wird nicht unkontrol-
                  | liert ein Carriage Return durchgeführt
```

W I D T H L P R I N T

Mit **WIDTH LPRINT zeilenbreite** kann man die Zeilenbreite des Druckers verändern.

zeilenbreite gibt an, nach wieviel Zeichen der Drucker eine neue Zeile beginnen soll. **zeilenbreite** muß zwischen 1 und 255 liegen.

Für den Ausdruck von BASIC-Programmen (Listings) hat dieses Kommando eine große Bedeutung. Der voreingestellte Wert für die Zeilenbreite (132) ist größer, als der Drucker Zeichen in einer Zeile unterbringen kann. Daher wird beim Ausdruck des Listings teilweise an einer ungewünschten Stelle ein Zeilenvorschub eingefügt, was die Struktur des Listings erheblich durcheinander bringt. Da der Drucker ein sog. 80-Zeichen Drucker ist (also 80 Zeichen pro Zeile drucken kann), muß man, um einen problemlosen Ausdruck zu gewährleisten, mit **WIDTH LPRINT** die **zeilenbreite** auf 80 Zeichen einstellen (**WIDTH LPRINT 80**).

Beispiel:

```
WIDTH LPRINT 40          | Druckerzeilenbreite 40 Zeichen

WIDTH LPRINT 80          | Druckerzeilenbreite 80 Zeichen

WIDHTH LPRINT 255        | wichtig für Ausdruck von Grafiken
```

W R I T E

Der **WRITE**-Befehl arbeitet ähnlich wie der **PRINT**-Befehl. Die Gemeinsamkeit liegt darin, daß man mit beiden Befehlen irgendetwas (Variablen, Texte) auf die Konsole (Bildschirm) bringen kann. Der Unterschied liegt jedoch in der Form. Während bei **PRINT** alles ohne irgendwelchen Zusatz auf die Konsole gebracht wird, trennt der **WRITE**-Befehl bei mehreren Variablen grundsätzlich die einzelnen durch ein Komma. Ist die Variable eine Stringvariable, kleidet er sie auf der Konsole in Anführungsstriche ein. Mehrere Variablen können durch ein Komma oder ein Semikolen im **WRITE**-Befehl getrennt werden. Sie haben nur trennende Bedeutung (der einzelnen Elemente) und haben keinen Einfluß (wie bei **PRINT**) auf den Ausdruck der Variablen.

Beispiel:

```
10 a=10:b=20:c=30
20 text$="Dies ist ein Test."
30 WRITE a,b,c,text$
40 WRITE a,"abcdefg",b
50 END
```


RUN:

```
10,20,30,"Dies ist ein Test."
10,"abcdefg",20
OK
■
```

W R I T E

Der **WRITE #**-Befehl arbeitet ähnlich wie der **PRINT #**-Befehl. Die Gemeinsamkeit liegt darin, daß man mit beiden Befehlen irgendetwas (Variablen, Texte) in eine Datei schreiben kann. Der Unterschied liegt jedoch in der Form. Während bei **PRINT #** alles ohne irgendwelchen Zusatz in die Datei geschrieben wird, trennt der **WRITE #**-Befehl bei mehreren Variablen grundsätzlich die einzelnen durch ein Komma. Ist die Variable eine Stringvariable, kleidet er sie in der Datei in Anführungsstriche ein. Mehrere Variablen können durch ein Komma oder ein Semikolen im **WRITE #**-Befehl getrennt werden. Sie haben nur trennende Bedeutung (der einzelnen Elemente) und keinen Einfluß (wie bei **PRINT #**) auf den Ausdruck der Variablen.

Beispiel:

```
10 OPEN "O",#1,"M:TEST"          | Datei #1 eröffnen
20 a=10:b=20:c=30                | Variablen definieren
30 text$="Dies ist ein Test."    |
40 WRITE #1,a,b,c,text$          | und in Datei schreiben
50 WRITE #1,a,"abcdefg",b        | auch "gemischtes" in Datei schreiben
60 CLOSE                         | Datei schließen
70 DISPLAY "M:TEST"              | und Inhalt kontrollieren
80 END                           |
```

RUN:

```
10,20,30,"Dies ist ein Test."
10,"abcdefg",20
OK
■
```

Dateiverarbeitung mit Jetsam

Die Verarbeitung von Jetsam-Dateien unterscheidet sich grundlegend von der Verarbeitung von ASCII-Dateien, die nur sequentiell, d.h. Satz für Satz verarbeitet werden können, und auch von der Verarbeitung von Dateien mit wahlfreiem Zugriff, bei denen über die Satznummer die Datensätze der Datei in beliebiger Reihenfolge verarbeitet werden können. Auf die Daten in einer Jetsam-Datei kann nicht direkt, sondern nur indirekt über Schlüssel zugegriffen werden, die in einer Schlüsseldatei abgelegt sind. Dies hat den Vorteil, daß man auf die Daten aufgrund von eingegebenen Suchmerkmalen, den Schlüsseln, zugreifen kann, was besonders bei größeren Datenbeständen eine schnellere Verarbeitung ermöglicht, da nur auf die Daten zugegriffen wird, die tatsächlich verarbeitet werden sollen.

Eine Jetsamdatei besteht aus 1.: einer Datendatei, in der die eigentlichen Datensätze mit den Daten abgelegt sind, und 2: aus einer Schlüsseldatei, in der die Schlüssel sind, über die auf die Datensätze in der Datendatei zugegriffen werden kann. In letzterer werden auch die Satznummern der Datensätze in Schlüsselreihen abgelegt. Zur Verarbeitung unter Jetsam müssen beide Teildateien, die zusammen als eine einheitliche Jetsam-Datei zu sehen sind, geöffnet sein, damit sie dem Programm zur Verfügung stehen. Mit den Jetsam-Funktionen zum Suchen nach Daten wird intern die Satznummer bereitgehalten, über die durch ein nachfolgendes **GET** dann die Daten aus der Datendatei eingelesen und dem Programm zur Verfügung gestellt werden.

Die Datendatei entspricht ihrem Aufbau nach einer Datei für wahlfreien Zugriff, nur daß die ersten 128 Bytes für die spezielle Jetsamverarbeitung reserviert sind.

Ein direkter Zugriff auf die Datendatei (ohne über die Schlüsseldatei zu gehen) empfiehlt sich nicht, da durch eine

solche Arbeitsweise nicht sichergestellt ist, daß nur gültige Datensätze verarbeitet werden. Sind durch Löschen von Datensätzen Daten in der Datendatei ungültig geworden, sind die als ungültig gekennzeichneten Sätze nur über die Schlüsseldatei erkennbar. Wird nämlich z.B. ein Datensatz gelöscht, so wird die Datendatei nicht verändert! Es werden lediglich sämtliche Schlüssel zu dem Datensatz in der Schlüsseldatei gelöscht, so daß auf den Satz nicht mehr über einen Schlüssel zugegriffen werden kann!

Daten- und Schlüsseldatei müssen stets stimmig (konsistent) zueinander sein. Dies wird durch den Befehl **CONSOLIDATE** und bei Beendigung der Bearbeitung durch **CLOSE Liste von: Dateisymbol** sichergestellt. Sollten durch einen Fehler die beiden Teildateien unstimmig zueinander werden, sind die Daten verloren und können, falls überhaupt, nur sehr mühsam gerettet werden. Es empfiehlt sich daher, diese Dateien möglichst vor jeder Verarbeitung zu sichern.

Um Datensätze verarbeiten zu können, muß sich das Programm, das mit einer Jetsam-Datei arbeitet, auf den gewünschten Datensatz positionieren. Eine solche Positionierung geschieht z.B. durch einen Suchvorgang (**SEEK**-Funktionen). Die Position innerhalb der Jetsam-Datei, die das Programm eingenommen hat, wird auch aktuelle Position genannt. Über diese aktuelle Position weiß Jetsam, mit welchem Datensatz gearbeitet werden soll, so daß der zum Schlüssel gehörende Datensatz über ein **GET** eingelesen werden kann.

Fast alle Jetsam-Funktionen und -Befehle (Unterschied zwischen Funktion und Befehl vgl.: Ende des Kapitels) arbeiten über die aktuelle Position in der Jetsam-Datei, die durch die Schlüsselreihe, den Schlüsselwert und die Satznummer des Datensatzes in der Datendatei bestimmt wird.

Bei der Verarbeitung von Jetsam-Dateien sind die Schlüssel, die zum Wiederauffinden der Daten dienen, sehr wichtig. Es ist möglich, in Jetsam bis zu acht Schlüsselreihen zu

benutzen (Reihe 0 bis 7). Eine Schlüsselreihe kann man sich als Nachschlagelexikon vorstellen. Jeder Eintrag in diesem Lexikon, ein sogenannter Schlüsseleintrag, der den Schlüsselwert und, für den Benutzer verborgen, die Satznummer des zugehörigen Datensatzes enthält, wird von Jetsam selbständig in der richtigen Sortierfolge (aufsteigende Sortierung) in der Schlüsseldatei vorgehalten. Der Benutzer oder das Programm braucht sich um die Sortierung der Schlüssel nicht zu kümmern. Durch die Satznummer wird vom Schlüsseleintrag auf den Beginn des Datensatzes in der Datendatei verwiesen. Jede Schlüsselreihe ist von den anderen Schlüsselreihen unabhängig.

Vom BASIC-Programm aus muß dann umsortiert werden, wenn eine andere Sortierfolge gewünscht wird, z.B. eine absteigende Sortierung der Daten.

Ist es zulässig, daß in einer Schlüsselreihe mehrere Schlüsseleinträge mit identischen Schlüsselwerten stehen und wird über diese Schlüsselreihe auf die Daten sequentiell zugegriffen, so werden die Sätze innerhalb des Schlüsselsatzes (siehe unten) in der Reihenfolge der Satznummern bereitgestellt. Möchte man diese Datensätze nach einem anderen Sortierbegriff als dem Jetsam-Schlüssel sortiert haben, muß diese Sortierung im Programm selbst vorgenommen werden.

Beispiel:

In einer Adreßdatei gibt es als Schlüssel den Namen und die Postleitzahl. Jeder Name darf in der Datei nur einmal vorkommen (durch **RANKSPEC** für die Schlüsselreihe angegeben). Die Postleitzahl darf als Schlüssel mehrfach vorkommen.
Liest man sämtliche Adressen aus der Datei über den Namen, bekommt man die Namen alphabetisch aufsteigend sortiert - Hubert, Maier, Meier, Krause, Schulze.
Liest man einige Adressen gezielt nach einer bestimmten Postleitzahl aus der Datei, erhält man die Daten in der Reihenfolge ihrer Satznummern (in Klammern angegeben): Krause (1), Hubert (2), Schulze (3), Meier (4), Maier (5).
Möchte man alle Adressen mit einer bestimmten Postleitzahl nach Namen alphabetisch sortiert bekommen, müssen die Adressen durch eine Sortierroutine entsprechend nach den Namen sortiert werden, oder man liest sämtliche Datensätze sequentiell nach dem Namen und gibt nur die Adressen aus, die die gewünschte Postleitzahl enthalten.
Welchen Weg man wählt, hängt im wesentlichen von der Größe der Datei ab - bei einer großen Datei ist es besser, gezielt über die Postleitzahl zu lesen und die Adressen umzusortieren; bei einer kleinen Datei kann es vertretbar sein, alle Namen zu lesen, die Datensätze über ein **GET** zu lesen und die Postleitzahl abzufragen.

Ein Schlüsselsatz wird dann in einer Schlüsselreihe gebildet, wenn mehrere Schlüsseleinträge denselben Schlüsselwert haben.

Je Schlüsselreihe sollte nur eine Art von Schlüssel vorgesehen werden. Z.B. Reihe 0: Name und Vorname als kombinierter Schlüssel, Reihe 1: Postleitzahl, Reihe 2: Ort, Reihe 3: Telefonvorwahl. Man könnte sämtliche Schlüssel auch in Reihe 0 einstellen, was jedoch die Handhabung der Datei sehr erschweren würde, denn es müssen nicht zu jedem Datensatz sämtliche Schlüssel ausgefüllt sein (nicht alle Adressaten haben Telefon) und beim Löschen eines Satzes müßte sehr sorgfältig geprüft werden, ob man auch alle Schlüssel gelöscht hat, die in einer Schlüsselreihe eingetragen sind.- Siehe auch die Ausführungen zu **ADDKEY** und **DELKEY** -.

Enthalten die Datensätze sehr viele verschiedene Datenelemente, so sind die Datenelemente, über die als Schlüssel nach den Datensätzen zugegriffen werden soll, sorgfältig auszuwählen. Kann man nicht alle Schlüssel in den acht Schlüsselreihen unterbringen, so sollte man prüfen, ob man aus der einen Jetsam-Datei nicht zwei Dateien machen könnte.

Ggf. wäre zu überlegen, ob die Daten nicht anders gegliedert werden können.

Die Schlüssel sollten auf jeden Fall so gewählt werden, daß die meisten Zugriffe auf die Daten abgedeckt werden. Es gilt, die Frage zu beantworten: "Wer braucht welche Daten mit welchen Suchmerkmalen mit welcher Häufigkeit (laufend, täglich, wöchentlich, monatlich etc.)?" In einem Lager eines Fertigungsbetriebes z.B. werden zu jedem Artikel die Artikelnummer, Bezeichnung des Artikels, Kurzbezeichnung des Artikels, Lieferant, Bestand, Mindestbestellmenge, Preis, MWSt.-Satz, Einkaufs- oder Verkaufsartikel, Einkaufsdatum und Verkaufsdatum festgehalten. Je nach Vorgehensweise des Betriebes wird sehr viel über die Artikelnummer oder auch über die Artikelkurzbezeichnung auf die Artikeldaten zugegriffen. Für die Rechnungsabteilung wird es interessant sein, über den Namen des Lieferanten Artikeldaten auszuwählen, um z.B. Rechnungen zu prüfen.
Um möglichst allen Erfordernissen des Betriebes gerecht zu werden, muß sorgfältig darüber nachgedacht werden, welche Datenelemente als Schlüssel dienen sollen, damit die Artikeldaten in den verschiedenen Bereichen des Betriebes schnell und gezielt zur Verfügung stehen; hier z.B. können die Artikelnummer, die Artikelkurzbezeichnung und der Name des Lieferanten als Schlüssel auf die Artikeldaten dienen.

In den Jetsam-Befehlen und -Funktionen sind Sperren anzugeben. Es gibt folgende Sperren, die über die entsprechende Kennziffer anzugeben sind:

0 **keine Sperre** einrichten
1 **Lesesperre** einrichten, d.h. ein anderer Benutzer darf lesen, aber nicht schreiben
2 **Schreibsperre** einrichten, d.h. ein anderer Benutzer darf weder lesen noch schreiben.

Die Sperren können für die gesamte Datei (bei **CREATE** und **OPEN**) oder für einzelne Sätze (bei den übrigen Jetsam-Befehlen und -Funktionen) gesetzt werden.

Die Sperren haben eigentlich nur bei Mehrbenutzer-Systemen Bedeutung, wenn also mehrere Benutzer gleichzeitig auf eine Datei zugreifen können. Der JOYCE ist in der Regel jedoch ein Einbenutzer-System, so daß die Sperren eigentlich ohne Belang sind. Für eine ordnungsgemäße Verarbeitung sollten die Sperren aber doch sinnvoll angegeben werden.

Bei den **SEEK**-Funktionen genügen Lesesperren. Bei Funktionen, die die Datei verändern, sollten Schreibsperren auf den Datensatz gesetzt werden. Sollte Ihr Programm dann einmal auf einem Mehrbenutzer-System eingesetzt werden, ist dadurch sofort eine fehlerfreie Verarbeitung sichergestellt.

Wie bei jeder Verarbeitung von Dateien, muß auch für Jetsam die Jetsam-Datei angelegt und eröffnet, d.h. für das Programm zugänglich gemacht werden. Es müssen Datensätze geschrieben, verändert, gelesen und gelöscht werden können. Für all diese Aufgaben gibt es auf die Belange der Jetsam-Dateien zugeschnittene BASIC-Befehle, die im folgenden ausführlich erläutert werden.
Hier die BASIC-Befehle, die nur für die Verarbeitung von Jetsam-Dateien vorgesehen sind, in alphabetischer Reihenfolge und gegliedert nach Jetsam-Funktion und Jetsam-Kommando:

Jetsam-Funktion
- ADDKEY
- ADDREC
- CONSOLIDATE
- DELKEY
- FETCHKEY$
- FETCHRANK
- FETCHREC
- LOCK
- RANKSPEC
- SEEKKEY
- SEEKNEXT
- SEEKPREV

SEEKRANK
SEEKREC
SEEKSET

Jetsam-Kommando
BUFFERS
CLOSE
CREATE
FIELD
GET
OPEN
OPTION FIELD
PUT

Außerdem sind noch diese Befehle von Bedeutung:
CLEAR CVD CVI CVIK CVS CVUK FIELD LSET MEMORY MKD$ MKI$ MKIK$ MKS$ MKUK$

Die Befehle sollen an einer Beispieldatei verständlich gemacht werden;
Es ist eine kleine Adreßdatei mit Name, Vorname, Postleitzahl, Ort, und einer Kennung, ob es sich um einen Lieferanten, Kunden oder Geschäftsfreund handelt.
Als Schlüssel wird benutzt:
In Reihe 0 jeweils die ersten 5 Buchstaben des Names und Vornamens als kombinierter Schlüssel; es dürfen in dieser Schlüsselreihe keine Schlüsseleinträge mit identischem Schlüsselwert vorkommen.
In Reihe 1 steht die Postleitzahl.
In Reihe 2 steht die Kennung, ob "Lieferant" (L), "Kunde" (K) oder "Geschäftsfreund" (G) zutrifft.

Sämtliche Schlüsselreihen enthalten 5 Schlüsseleinträge.

Reihe 0	ElektMaier	InstaKraus	LudwiElekt	Mayr GroßH	XaverUnsin
Satznr.	4	2	6	3	5
Reihe 1	2390	2390	8000	8000	8000
Satznr.	2	4	3	5	6
Reihe 2	G	K	K	L	L
Satznr.	5	2	4	3	6

Die aktuelle Position in der Datei wird durch ↓ angezeigt.

Auf die Darstellung der Datendatei wird der Einfachheit halber verzichtet.

In Reihe 1 haben wir zwei Schlüsselsätze, einmal mit "2390" und "8000", in Reihe 2 ebenfalls 2 Schlüsselsätze mit "K" und "L". Die nur einmal vorkommenden Schlüsselwerte könnte man ebenfalls als Schlüsselsätze betrachten; diese Schlüsselsätze umfassen statt mehrerer nur einen Schlüsseleintrag.

Die Befehle werden an Hand eines kleinen Beispielprogramms erläutert, das Sie auf der zum Buch erhältlichen Diskette unter dem Namen JETSAM.BAS finden. Die BASIC-Zeilen sind diesem Programm entnommen. Das vollständige Listing ist am Ende dieses Kapitels abgedruckt. Wenn Sie das Programm unter BASIC ablaufen lassen, werden alle aufgeführten Beispiele zu den Funktionen durchgeführt und jeweils die aktuelle Position und der Antwortcode angezeigt.

Um das Programm ablaufen zu lassen, starten Sie CP/M, laden BASIC,

```
A>BASIC
```

legen die zum Buch zu beziehende Diskette mit Seite B in Laufwerk A: - oder Sie geben das Listing ein, das Sie am Ende des Kapitels finden, speichern es mit **SAVE "jetsam** - und starten es mit

```
OK
. run "jetsam".
```

Hier nun die Befehle im einzelnen:

Zu Beginn der Jetsam-Verarbeitung sind die Systemvariablen einzustellen.

```
BUFFERS Zahl der Puffer für die Schlüsseldatei[,Zahl der Satzsperren]
```

oder

```
BUFFERS ,Zahl der Satzsperren
```

Beispiel:

```
30 BUFFERS 20
```

Durch diesen Befehl wird die Größe des Dateipuffers für die Schlüsseldatei festgelegt und ein entsprechend großer Bereich im Speicher reserviert. Bei der Verarbeitung von Jetsam-Dateien geschieht der Zugriff auf die Daten über die Schlüsseldatei. Für eine zügige Verarbeitung der Schlüsseldatei ist es günstig, wenn ein möglichst großer Pufferbereich zur Verfügung steht, um möglichst wenig zeitraubende Zugriffe auf Diskette und möglichst viele schnelle Zugriffe auf den Puffer zu haben.
Dieser Befehl ist Voraussetzung, um mit der Schlüsseldatei arbeiten zu können.

Beim **OPEN**-Befehl der Jetsam-Datei versucht Jetsam möglichst viele Einträge der Schlüsseldatei in den durch **BUFFERS** reservierten Speicherbereich einzustellen, um möglichst viele Schüsseleinträge sofort bereit zu halten, damit so wenig wie möglich aus der Schlüsseldatei nachgelesen werden muß, was die Programmlaufzeit verlängern würde.

Da ein Puffer 128 Bytes umfaßt, wird durch das obige Beispiel ein Speicherbereich von 20 * 128 = 2560 Bytes als Puffer für die Schlüsseldatei reserviert. Dieser Speicherbereich steht damit dem BASIC-Programm nicht zur Verfügung.

Im **BUFFERS**-Befehl können auch Pufferbereiche für die Speicherung von Satzsperren reserviert werden. Dies ist jedoch nur für Mehrbenutzer-Systeme bedeutsam. Ein solcher Speicherbereich umfaßt jeweils 7 Bytes.

BUFFERS muß vor einem **OPEN** oder **CREATE** der Jetsam-Datei stehen.

Einige allgemeine Hinweise zum Dateipuffer:
Jeglicher Zugriff zum Lesen oder Schreiben einer Datei, auch bei sequentiellen Dateien oder Dateien mit wahlfreiem Zugriff, wird nie direkt auf die Datei, die auf der Diskette gespeichert ist, durchgeführt, sondern immer über einen vom Betriebssystem oder auch vom BASIC eingerichteten Dateipuffer. Dieser Puffer steht im Arbeitsspeicher an einer bestimmten Stelle, in die vom Betriebssystem die einzulesenden Daten aus der Datei hineingestellt werden. Dann erst werden die Daten dem Programm zur Verfügung gestellt.
Sollen Informationen auf Diskette gespeichert werden, schreibt das Programm die Daten in den Dateipuffer. Ist der Puffer vollständig gefüllt bzw. ist man am Ende des Progamms angekommen, liest das Betriebssystem die Daten aus dem Pufferbereich und schreibt sie auf Diskette.
Sinn dieser Vorgehensweise ist, daß der tatsächliche physikalische, mechanische Zugriff stets nur von den Routinen des Betriebssystems durchgeführt wird, und daß sich nicht jedes Programm um die Handhabung der Datei und des Diskettenlaufwerks selbst kümmern muß.

Soll ein Datensatz aus einer Datei gelesen werden, prüft das Betriebssystem zunächst, ob der Datensatz schon im Pufferbereich der Datei zu finden ist. Ggf. wird nur ein Zeiger, der auf den Beginn des angeforderten Datensatzes zeigt, umgesetzt und die Daten dadurch für das Anwendungsprogramm zugänglich.
Erst wenn festgestellt wird, daß die Daten nicht im Pufferbereich zu finden sind, werden ein Zugriff auf die Datei auf Diskette durchgeführt und so viele Daten eingelesen, bis dieser Bereich neu gefüllt ist. Der Zeiger wird auf den Beginn des angeforderten Satzes gesetzt, in diesem Fall meist der Beginn des Pufferbereichs.

Soll zum ersten Mal mit einer Jetsam-Datei gearbeitet werden, müssen die Daten- und die Schlüsseldatei durch den Befehl **CREATE** angelegt werden. Bei ASCII-Dateien und Dateien mit wahlfreiem Zugriff genügt der **OPEN**-Befehl zum Anlegen der Datei. Aufgrund der besonderen Erfordernisse bei der Verarbeitung mit Jetsam ist jedoch eine besondere Vorbereitung der Dateien zur Verarbeitung erforderlich.

In der Datendatei sind die ersten 128 Bytes der Datei, bzw. so viele Bytes wie in der Satzlänge beim **CREATE** angegeben wurde, für die Jetsam-Routinen reserviert, d.h., für den Anwender nicht zugänglich. Sollen Datensätze verarbeitet werden, die länger als 128 Bytes sind, ist der Maximalwert von 128 durch einen **MEMORY**- (**MEMORY ,,,maximale Satzlänge**) oder **CLEAR**-Befehl (**CLEAR,,,,maximale Satzlänge**) entsprechend heraufzusetzen. Außerdem sind in jedem Datensatz die ersten zwei Bytes, die die Satznummer des Datensatzes enthalten, für Jetsam reserviert und damit ebenfalls nicht direkt dem Anwender zugänglich, da für ihn die Satznummer in der Regel nicht interessant ist. (Vgl. aber **FETCHREC**.)

Außerdem wird die Schlüsseldatei eingerichtet, die besonders auf die Verarbeitung der Jetsam-Befehle und -Funktionen eingerichtet ist. Die Datei wird zur Aufnahme der Schlüsselwerte und Satznummern in den Schlüsselreihen vorbereitet.

Nach Durchführung des Befehls stehen die Schlüssel- und die Datendatei zur Verarbeitung zur Verfügung, so daß der **OPEN**-Befehl nach dem **CREATE** nicht mehr erforderlich ist.

```
CREATE Datei-Symbol, Name der Datendatei, Name der Schlüsseldatei, Sperre der
Jetsam-Datei, [Länge der Datensätze in der Datendatei (einschließlich 2 Byte
für die Satznummer)], [Benutzer-String]
```

Wurde die Jetsam-Datei in einem ersten Programmlauf durch **CREATE** eingerichtet, müssen bei einem erneuten Programmlauf die Daten- und Schlüsseldatei für die Verarbeitung eröffnet werden.


```
OPEN "K",Datei-Symbol, Name der Datendatei, Name der Schlüsseldatei, Sperre
der Jetsam-Datei, [Länge der Datensätze in der Datendatei (einschließlich 2
Byte für die Satznummer)], [Benutzer-String]
```

Voraussetzung für den **OPEN**-Befehl ist, daß die Datendatei und die Schlüsseldatei (= Indexdatei) in einem vorangegangenen Programmlauf durch ein **CREATE** angelegt und zur Jetsam-Verarbeitung vorbereitet wurden.

Durch das "K" in der **OPEN**-Anweisung wird kenntlich gemacht, daß eine Jetsam-Datei zu eröffnen ist (K für **K**ey - engl. Schlüssel)

Durch Angabe des Benutzer-Strings kann in die Schlüsseldatei z.B. eine Kurzbezeichnung der Jetsam-Datei geschrieben werden.

Beispiel:

```
50 IF FIND$("TEST.DAT")<>"" AND FIND$("TEST.IND")<>""
THEN OPEN "K",#1,"TEST.DAT","TEST.IND",2
ELSE CREATE #1,"TEST.DAT","TEST.IND",2
```

Hier wird zunächst über **FIND$** geprüft, ob die Datendatei **TEST.DAT** und die Schlüsseldatei **TEST.IND** vorhanden sind. Ist dies der Fall, werden die Dateien durch **OPEN** eröffnet, sonst durch **CREATE** eingerichtet. Es wird jeweils eine Schreibsperre auf die Jetsam-Datei gelegt. Eine Satzlänge und ein Benutzer-String wird nicht angegeben.

Bei Programmende ist die Jetsam-Datei zu schließen.

```
CLOSE Liste von: Dateisymbol
```

Um die Verarbeitung der Jetsam-Datei zu beenden, ist es wichtig, daß das Datei-Symbol mit angegeben wird; für Dateien mit wahlfreiem Zugriff und einfach sequentielle Dateien ist die Angabe des Datei-Symbols nicht erforderlich.

Nur so ist sichergestellt, daß die Schlüssel- und Datendatei stimmig (konsistent) bleiben und für eine Weiterverarbeitung in weiteren Programmläufen oder durch andere Programme zur Verfügung stehen.

Beispiel:

```
260 CLOSE #1
```

Die unter #1 eingerichtete bzw. eröffnete Jetsam-Datei wird geschlossen. Alle noch nicht auf Diskette geschriebenen Pufferbereiche werden jetzt auf die Diskette gebracht.

Auch während des Programmlaufs ist sicherzustellen, daß die Schlüssel- und Datendatei zueinander stimmig bleiben.

```
CONSOLIDATE (Datei-Symbol)
```

Jede Veränderung wie

Einfügen eines Satzes in die Datendatei
Einfügen von Schlüsseln in die Schlüsseldatei
Überschreiben von Daten in der Datendatei
Löschen von Schlüsseln in der Schlüsseldatei

führt dazu, daß die Daten- und Schlüsseldatei als nicht stimmig markiert werden. Dies bedeutet, daß aufgrund der Veränderung nicht gewährleistet ist, daß alle (Schlüssel-)Informationen, die in den durch den **BUFFERS**-Befehl eingerichteten Pufferbereich hineingestellt wurden, auch auf Diskette geschrieben und damit gesichert sind.

Um zu gewährleisten, daß Daten- und Schlüsseldatei stets zueinander stimmig sind, wird dringend empfohlen, nach jedem Befehl, der die Jetsam-Datei verändert (**ADDKEY, ADDREC, DELKEY, PUT, RANKSPEC**), ein **CONSOLIDATE** durchzuführen. Der Befehl bewirkt einen physikalischen Zugriff auf das Diskettenlaufwerk zum Wegschreiben der Pufferbereiche in die Dateien. Zur Beschleunigung des Programms sollten die Daten- und Schlüsseldatei zur Verarbeitung im Laufwerk M: zur

Verfügung stehen, so daß sich die **CONSOLIDATE**-Funktion kaum auf die Laufzeit des Programms auswirkt.

Beispiel:

```
280 rc=CONSOLIDATE(#1)
```

Auf die unter #1 eingerichtete bzw. eröffnete Datei wird ein **CONSOLIDATE** durchgeführt. Da dies eine Funktion ist, muß das Funktionsergebnis einer Variablen - hier **rc** -, zugewiesen werden. Bei Einzelbenutzer-Systemen, wie es der Benutzung des JOYCE (leider) meistens entspricht, ist das Funktionsergebnis stets 0. Bei Mehrbenutzer-Systemen wird angegeben, durch wieviele Benutzer die Datei als inkonsistent markiert wurde, d.h., wie viele Benutzer seit dem **OPEN/CREATE** bzw. seit dem letzten **CONSOLIDATE** die Datei verändert haben.
Um Daten in die Jetsam-Datei schreiben und auch wieder auslesen zu können, muß der Datensatz definiert sein. Dies geschieht durch das Kommando **FIELD**. Damit werden die Datenfelder und ihre jeweilige Länge in Bytes (Feldgröße) festgelegt. Die für Jetsam reservierten ersten beiden Bytes eines Datensatzes können nicht im **FIELD**-Kommando angesprochen werden. Die Satzlänge des Datensatzes errechnet sich somit als Summe der Feldgrößen der Datenfelder zuzüglich der 2 für Jetsam reservierten Bytes.

```
FIELD Datei-Symbol,Länge von Feld AS Name von Feld
```

Der unterstrichene Teil des Befehles kann beliebig oft wiederholt werden.
Um Werte in die Datenfelder zu schreiben, dürfen nur die Befehle **LSET** für linksbündige Wertzuweisung, **RSET** für rechtsbündige Wertzuweisung oder **MID$** benutzt werden:

```
LSET Feld aus der FIELD-Anweisung=String-Ausdruck
RSET Feld aus der FIELD-Anweisung=String-Ausdruck
MID$ (Feld aus der FIELD-Anweisung, Startposition ab der der String-Ausdruck
stehen soll[,Länge in der der String-Ausdruck übernommen werden soll])=String-
Ausdruck
```

Beispiel: - nicht in JETSAM.BAS enthalten -

Durch **FIELD** wurde ein Datenfeld mit 10 **AS name$** bestimmt.

```
10 LSET name$="Hugo"
```
setzt **name$** auf "Hugo ".

```
10 RSET name$="Hugo"
```
setzt **name$** auf " Hugo" .

```
10 MID$(name$,3,2)="Hugo"
```
setzt **name$** auf " Hu ".

MID$ wird hierbei als Kommando und nicht als Funktion verwendet.

Beispiel:

```
60 FIELD #1,20 AS xname$,20 AS xvorname$,2 AS xplz$,20 AS xort$,1 AS xkunde$
870 LSETxname$="Installation":LSETxvorname$="Krause":xplz=2390:LSET xort$=
"Flensburg":LSET xkunde$="K"
1000 LSET xplz$=MKI$(xplz)
```

In Zeile 60 wird zunächst der Datensatz beschrieben. Der Datensatz ist 20 + 20 + 2 + 20 + 1 zuzüglich 2 Bytes für die Satznummer, also 65 Bytes lang. In Zeile 870 und 1000 werden jeweils über **LSET** den Datenfeldern Werte zugewiesen.

Mit der Funktion **RANKSPEC** wird für eine Schlüsselreihe festgelegt, ob in der Schlüsselreihe identische Schlüsselwerte vorkommen dürfen oder nicht. Diese Kennung der Schlüsselreihe macht sich bei den Befehlen **ADDREC** bzw. **ADDKEY** ggf. durch den Fehlercode 116 bemerkbar.

Die Funktion wirkt sich stets nur für nachfolgende Veränderungen der Jetsam-Datei aus. Wenn man es möchte, könnte man eine Schlüsselreihe, die so gekennzeichnet ist, daß keine mehrfachen Schlüssel vorkommen dürfen, vorübergehend so kennzeichnen, daß mehrfache Schlüssel zulässig sind, dann beliebig viele Datensätze mit gleichen Schlüsselwerten in die Datei einstellen und danach die Schlüsselreihe wieder so kennzeichnen, daß keine mehrfachen Schlüssel zulässig sind.

Nach Ausführung sollte die Jetsam-Datei durch den **CONSOLIDATE**-Befehl wieder stimmig markiert werden.

RANKSPEC(Datei-Symbol,Schlüsselreihe,Information (identische Schlüsselwerte in der Schlüsselreihe zulässig (Wert 0) oder nicht (Wert1))

Beispiel:

```
70 ergebnis=RANKSPEC(#1,0,1)
```

Da standardmäßig angenommen wird, daß mehrfache Schlüssel zulässig sind, braucht im Programm nur angegeben werden, in welchen Schlüsselreihen mehrfache Schlüssel nicht zulässig sein sollen. Hier wird für Schlüsselreihe 0 festgelegt, daß nur einmalige Schlüssel zulässig sind und bei Einfügen eines Schlüssels in Reihe 0 ggf. als Antwortcode 116 auszugeben ist.

Durch die Funktion **ADDREC** werden zwei Aufgaben zugleich erledigt:

1. Der Datensatz wird in die Datendatei eingestellt.
2. Es wird eine Schlüsselinformation für den Datensatz in die Schlüsseldatei eingestellt - Schlüsselreihe und Schlüsselwert - und Jetsam vergibt eine Satznummer für den Datensatz, die in der Schlüsseldatei zu dem Schlüssel abgelegt wird. Die niedrigste Satznummer, die von Jetsam vergeben wird, ist 2. Der Schlüsselwert wird in aufsteigender Sortierfolge in die Schlüsseldatei eingefügt.

Nur durch diesen Befehl können Datensätze in die Datendatei geschrieben werden. Wird ein Fehlercode 116, 130 oder 131 zurückgemeldet, wurde der Datensatz nicht geschrieben und auch keine Schlüsselinformation in die Schlüsseldatei eingestellt.
Der Fehlercode 116 ist sehr hilfreich für Schlüsselreihen, die durch **RANKSPEC** so gekennzeichnet wurden, daß in der Schlüsselreihe nur eindeutige Schlüssel zulässig sind, d.h., ein Schlüssel darf nur einmal in der Schlüsselreihe vorkommen. Jetsam macht den Benutzer dadurch darauf aufmerksam, daß ein Datensatz mit einem solchen Schlüssel bereits in der Jetsam-Datei vorhanden ist und dieser

Datensatz aufgrund der Kennung der Schlüsselreihe durch **RANKSPEC** nicht in die Jetsamdatei eingefügt werden darf.

```
ADDREC(Datei-Symbol,Sperre für den neuen Satz,Schlüsselreihe in die der
Schlüsselwert eingetragen werden soll,Schlüsselwert)
```

Beispiel:

```
980 key$=LEFT$(xname$,5):IF LEN(xname$)<5 THEN key$=key$+SPACE$(5-LEN(xname$))
990 key$=key$+LEFT$(xvorname$,5):IF LEN(xvorname$)<5 THEN key$=key$+SPACE$(5-
LEN(xvorname$))
1010 ergebnis=ADDREC(#1,2,0,key$)
```

In den Zeilen 980 und 990 wird zunächst der Schlüsselwert aus den ersten 5 Buchstaben des Namens und Vornamens - ggf. aufgefüllt mit Leerstellen - gebildet, der dann in Zeile 1010 beim **ADDREC** als Schlüsselwert für die Schlüsselreihe 0 mitgegeben wird.

Die Beispieldatei baut sich in der Schlüsselreihe 0 Stück für Stück durch **ADDREC** wie folgt auf:

Reihe 0	InstaKraus				
Satznr.	2				
Reihe 0	InstaKraus	Mayr Groß			
Satznr.	2	3			
Reihe 0	ElektMaier	InstaKraus	Mayr Groß		
Satznr.	4	2	3		
Reihe 0	ElektMaier	InstaKraus	Mayr Groß	XaverUnsin	
Satznr.	4	2	3	5	
Reihe 0	ElektMaier	InstaKraus	LudwiElekt	Mayr Groß	XaverUnsin
Satznr.	4	2	6	3	5

Mit der Funktion **ADDKEY** können weitere Schlüsselinformationen zu einem Datensatz in der Schlüsseldatei abgelegt werden. Über diese Schlüssel kann später wieder auf den Datensatz zugegriffen werden.

Durch die Funktion **ADDREC** wird bereits eine Schlüsselinformation zu einem Datensatz in die Schlüsseldatei geschrieben. Man möchte jedoch meistens über weitere Schlüssel auf den Datensatz zugreifen können. Diese zusätzlichen Schlüssel werden durch **ADDKEY** in die Schlüsseldatei eingefügt.
Es können weitere Schlüssel zu dem Datensatz in anderen Schlüsselreihen geschrieben werden. Dabei sollte nicht die Schlüsselreihe, in der durch **ADDREC** bereits ein Schlüssel eingestellt wurde, genommen werden.

Soll jedoch ein bestehender Schlüssel zu einem Datensatz in einer Schlüsselreihe geändert werden, so ist durch **ADDKEY** zunächst der neue Schlüssel in die Schlüsseldatei einzustellen, so daß vorübergehend in einer Schlüsselreihe zu einem Datensatz mehrere verschiedene Schlüssel vorhanden sind. Danach wird durch **DELKEY** der alte Schlüssel gelöscht. Diese Vorgehensweise ist erforderlich, damit nicht versehentlich ein Datensatz gelöscht wird. Hat ein Datensatz nur einen Schlüssel und würde zuerst der alte Schlüssel über **DELKEY** gelöscht, so wäre der Datensatz gelöscht und stünde nicht mehr zur Verfügung, selbst wenn unmittelbar danach durch ein **ADDKEY** wieder ein Schlüssel zu dem Datensatz eingefügt werden würde, was jedoch dann schon nicht mehr möglich ist.

Wenn man möchte, kann man sich jedoch die Eigenschaft zu Nutze machen, daß in einer Schlüsselreihe zu einem Datensatz mehrere verschiedene Schlüssel abgelegt werden können. Jedoch wird dadurch ggf. die Handhabung der Jetsam-Datei erschwert und fehleranfällig.

```
ADDKEY(Datei-Symbol,Sperre auf den Datensatz,Schlüsselreihe in der der
Schlüsselwert eingefügt werden soll,der einzufügende Schlüsselwert,Satznummer
des Datensatzes)
```

Beispiel:

```
1030 key$=MKIK$(xplz)
1040 satznr=FETCHREC(#1):ergebnis=ADDKEY(#1,2,1,key$,satznr)
1060 key$=xkunde$
1070 satznr=FETCHREC(#1):ergebnis=ADDKEY(#1,2,2,key$,satznr)
```

In den Zeilen 1030 und 1060 wird der neue Schlüsselwert gebildet; zum einen die Postleitzahl, die hier in verdichteter Form als Jetsam-Schlüssel benutzt wird (**MKIK$** siehe unten), und zum anderen die Kennung auf Kunde, Lieferant bzw. Geschäftsfreund.

In Zeile 1040 bzw. 1070 wird zunächst die aktuelle Satznummer des unmittelbar vorher durch **ADDREC** eingefügten Satzes ermittelt und dann wird die Schlüsselinformation in Reihe 1 bzw. Reihe 2 mit der ermittelen Satznummer in die Schlüsseldatei geschrieben. Als Sperre wurde hier eine Schreibsperre angegeben. Die Schlüsselreihen 1 und 2 bauen sich entsprechend wie Schlüsselreihe 0 Stück für Stück auf.

Die Positionierung des Programms, die aktuelle Position in einer Jetsam-Datei, wird von den drei Komponenten **Reihe, Schlüsselwert** und **Satznummer** bestimmt.

Die aktuelle Position in einer Jetsam-Datei wird durch die Befehle **FETCHKEY$, FETCHRANK** und **FETCHREC** ermittelt:

```
FETCHKEY$(Datei-Symbol)
FETCHRANK(Datei-Symbol)
FETCHREC(Datei-Symbol)
```

Diese drei Funktionen stellen jeweils eine Komponente der aktuellen Position zur Verfügung. Die Schlüsselreihe wird durch **FETCHRANK**, der Schlüsselwert durch **FETCHKEY$** und die Satznummer durch **FETCHREC** zur Verfügung gestellt.

Durch die Verarbeitung im Programm sind meistens nicht alle Komponenten der Positionierung bekannt. Oft kennt man nur

den Schlüsselwert, weil man in der Regel über einen Schlüsselwert auf die Datensätze zugreift. Seltener arbeitet man mit der Satznummer, die z.B. in der Suchfunktion **SEEKREC** benötigt wird. Die Schlüsselreihe ist meist bei einer sequentiellen Verarbeitung der Jetsam-Datei von Bedeutung (**SEEKRANK**).

Bei einigen Funktionen ist es sinnvoll und wichtig, die fehlenden Komponenten der aktuellen Position in der Jetsam-Datei zu kennen, wie z.B. für die Funktion **DELKEY** dann, wenn in der Schlüsselreihe zu mehreren Einträgen identische Schlüsselwerte zugelassen sind (kein **RANKSPEC** für die Schlüsselreihe wirksam). Es besteht sonst die Gefahr, daß die Funktion **DELKEY** fehlerhaft arbeitet und Schlüsselwerte unkontrolliert aus der Schlüsseldatei entfernt werden und damit die Jetsam-Datei unbrauchbar wird.

Beispiel:

Reihe 0	ElektMaier	InstaKraus	LudwiElekt	Mayr Groß	XaverUnsin
Satznr.	4	2	6	3	5
			↓		
Reihe 1	2390	2390	8000	8000	8000
Satznr.	2	4	3	5	6
Reihe 2	G	K	K	L	L
Satznr.	5	2	4	3	6

```
310 reihe=FETCHRANK(#1)
320 key$=FETCHKEY$(#1)
330 IF reihe=1 THEN key$=STR$(CVIK(FETCHKEY$(#1)))
340 PRINT "Die aktuelle Position ist auf Reihe"reihe",Schlüssel "key$" und
Satznummer"FETCHREC(#1)
```

FETCHKEY$ stellt als Schlüssel 8000, **FETCHRANK** als Reihe 1, und **FETCHREC** als Satznummer der aktuellen Position 3 bereit. Da die Postleitzahl über die Funktion **MKIK$** verdichtet

gespeichert wurde (siehe Beispiel zu **ADDKEY**), muß sie hier über **CVIK** wieder aufbereitet werden.
- **MKIK$** und **CVIK** siehe unten -

Die Suchfunktionen richten eine aktuelle Position in der Jetsam-Datei ein.

```
SEEKKEY(Datei-Symbol,Sperre auf den Datensatz, zu durchsuchende
Schlüsselreihe,zu suchender Schlüsselwert)
```

Mit dieser Funktion werden Datensätze aufgrund eines bestimmten Schlüsselwertes in der Jetsam-Datei gesucht. Neben dem eigentlichen Schlüsselwert ist die Schlüsselreihe anzugeben, in der nach Schlüsseleinträgen mit dem angegebenen Schlüsselwert gesucht werden soll.

Die aktuelle Position wird auf den gefundenen Schlüssel in der Jetsam-Datei eingerichtet, so daß der Datensatz über ein **GET** eingelesen werden kann.

Beispiel:

Reihe 0	ElektMaier	InstaKraus	LudwigElekt	Meyr Groß	XaverUnsin
Satznr.	4	2	6	3	5
Reihe 1	2390	2390	8000	8000	8000
Satznr.	2	4	3	5	6
				↓	
Reihe 2	G	K	K	L	L
Satznr.	5	2	4	3	6

```
420 ergebnis=SEEKKEY(#1,0,1,MKIK$(2300))
430 ergebnis=SEEKKEY(#1,0,2,"M")
440 ergebnis=SEEKKEY(#1,0,2,"L")
```

Das erste **SEEKKEY** ergibt den Antwortcode 105 - der Schlüsselwert wurde nicht gefunden, da in Reihe 1 kein Schlüsseleintrag mit dem Wert 2300 vorhanden ist. Die aktuelle Position wird auf Reihe 1, Schlüssel 2390, Satznummer 2 gesetzt, weil der erste Schlüsseleintrag mit dem Wert 2390 einem Schlüsseleintrag mit dem Wert 2300 folgen würde.

Das **SEEKKEY** in Zeile 430 ergibt den Wert 103 - Schlüsselwert wurde nicht in dieser Reihe gefunden, es wurde das Ende der Datei erreicht. Es gibt keine höheren Schlüsselwerte in dieser oder einer folgenden Reihe, d.h., es konnte keine aktuelle Position eingerichtet werden.

Das **SEEKKEY** in Zeile 440 ergibt den Antwortcode 0 und setzt die aktuelle Position wie angezeigt auf Reihe 2, Schlüssel L, Satznummer 3, weil dies der erste Schlüsseleintrag mit dem gewünschten Schlüsselwert ist.

Mit **SEEKNEXT** wird auf den nächstfolgenden Schlüsseleintrag zugegriffen. Abhängig vom Antwortcode läßt sich ermitteln, ob der neue Schlüsseleintrag denselben Schlüsselwert hat wie der vorhergehende Schlüsseleintrag (Antwortcode = 0). In dieser Schlüsselreihe sind identische Schlüsselwerte zu verschiedenen Schlüsseleinträgen zulässig (siehe **RANKSPEC**) und es wurde ein weiterer Schlüsseleintrag mit demselben Schlüsselwert gefunden. Man befindet sich noch im selben Schlüsselsatz.
Durch den Antwortcode 101 wird angezeigt, daß der Schlüsselwert des Schlüsseleintrags vom vorherigen Schlüsselwert abweicht und in derselben Schlüsselreihe ist.
Wechselte die aktuelle Position in eine andere Schlüsselreihe, so ist der Antwortcode 102.

Gibt man bei dieser Funktion Werte für die Index-Position mit, wird ab dem damit angegebenen Schlüsseleintrag auf den nächsten Schlüsseleintrag zugegriffen (Bsp.: **ergebnis = SEEKNEXT(#1,0,1,"HUGO",43)**: Suche nach dem nächsten Schlüsseleintrag, der auf den Schlüsseleintrag in Schlüsselreihe 1 mit dem Schlüsselwert "HUGO" und der Satznummer 43 folgt).

Antwortcode 103 wird ausgegeben, wenn man sämtliche Schlüsseleinträge der Schlüsseldatei abgearbeitet hat und damit das Dateiende erreicht ist.

SEEKNEXT(Datei-Symbol,Sperre auf den Datensatz [,Schlüsselreihe, Schlüsselwert, Satznummer der aktuellen Position von der bei der Suche ausgegangen werden soll])

Beispiel:

Reihe 0	ElektMaier	InstaKraus	LudwigElekt	Mayr Groß	XaverUnsin
Satznr.	4	2	6	3	5
Reihe 1	2390	2390	8000	8000	8000
Satznr.	2	4	3	5	6
		↓			
Reihe 2	G	K	K	L	L
Satznr.	5	2	4	3	6

```
470 ergebnis=SEEKKEY(#1,0,2,"K")
480 ergebnis=SEEKNEXT(#1,0)
490 ergebnis=SEEKNEXT(#1,0)
500 ergebnis=SEEKNEXT(#1,0)
510 ergebnis=SEEKNEXT(#1,0)
520 ergebnis=SEEKNEXT(#1,0)
530 ergebnis=SEEKKEY(#1,0,0,0,"XaverUnsin"):ergebnis=SEEKNEXT(#1,0)
540 ergbenis=SEEKNEXT(#1,0,0,"WalteElekt",6)
550 ergebnis=SEEKNEXT(#1,0,2,"H",5)
```

Durch **SEEKKEY** wurde eine aktuelle Position in der angezeigten Datei eingerichtet.

Das erste **SEEKNEXT** ergibt den Antwortcode 0 und setzt die aktuelle Position auf Reihe 2, Schlüssel K und Satznummer 4.

Ein weiteres **SEEKNEXT** ergibt den Antwortcode 101, denn die aktuelle Position geht auf Reihe 2, Schlüssel L, Satznummer

3. Da der Schlüsselwert sich verändert hat, ist der bisherige Schlüsselsatz verlassen worden.

Mit einem weiteren **SEEKNEXT** erhalten wir wieder den Antwortcode 0, die aktuelle Position wechselt auf Reihe 2, Schlüssel L und Satznummer 6.

Ein nochmaliges **SEEKNEXT** (Zeile 510) ergibt den Antwortcode 103, weil das Dateiende erreicht wurde und keine aktuelle Position mehr eingerichtet werden kann. Wird jetzt wieder ein **SEEKNEXT** ohne Angabe einer Index-Position durchgeführt, käme als Antwortcode 115, weil von einer unbestimmten aktuellen Position aus kein **SEEKNEXT** durchgeführt werden kann.

Befinden wir uns mit der aktuellen Position am Ende einer Schlüsselreihe (siehe Zeile 530 - hier wird durch das **SEEKKEY** die aktuelle Position am Ende von Reihe 0 eingerichtet) und führen dann ein **SEEKNEXT** durch, erhalten wir als Antwortcode 102, weil die Schlüsselreihe gewechselt hat, und die aktuelle Position steht dann auf Reihe 1, Schlüssel 2390, Satznummer 2.

Bei dem **SEEKNEXT** in Zeile 540 erhalten wir den Antwortcode 105, weil die angegebene Indexposition nicht vorhanden ist. In diesem Fall bleibt die aktuelle Position unbestimmt.

In Zeile 550 erhalten wir ebenfalls Antwortcode 105, jedoch würde eine aktuelle Position auf Reihe 2, Schlüssel K Satznummer 2 eingerichtet werden, weil der Schlüsselwert K dem gewünschten Schlüsselwert H folgt - siehe S. 326 im BASIC-Benutzer-Handbuch.

Die Funktion **SEEKPREV** arbeitet genauso wie **SEEKNEXT**, jedoch wird nicht weiter zum Dateiende hin gesucht, sondern rückwärts zum Dateianfang. Der Antwortcode 103 deutet darauf hin, daß der Dateianfang erreicht wurde: vgl. dazu Zeile 650 im folgenden Beispiel - Die Position wird auf den ersten Schlüsseleintrag in der Datei gesetzt. Mit dem folgenden

SEEKPREV wird versucht, über den Dateianfang hinaus rückwärts zu suchen, was den Antwortcode 103 ergibt.

SEEKPREV(Datei-Symbol,Sperre auf den Datensatz (,Schlüsselreihe, Schlüsselwert, Satznummer der aktuellen Position von der bei der Suche ausgegangen werden soll])

Beispiel:

Reihe 0	ElektMaier	InstaKraus	LudwiElekt	Mayr Groß h	XaverUnsin
Satznr.	4	2	6	3	5
Reihe 1	2390	2390	8000	8000	8000
Satznr.	2	4	3	5	6
				↓	
Reihe 2	G	K	K	L	L
Satznr.	5	2	4	3	6

```
580 ergebnis=SEEKKEY(#1,0,2,"L")
590 ergebnis=SEEKPREV(#1,0)
600 ergebnis=SEEKPREV(#1,0)
610 ergebnis=SEEKPREV(#1,0)
620 ergebnis=SEEKPREV(#1,0)
630 ergebnis=SEEKPREV(#1,0,0,"WalteElekt",6)
640 ergebnis=SEEKPREV(#1,0,2,"H",5)
650 ergebnis=SEEKKEY(#1,0,0,"ElektMaier"):ergebnis=SEEKPREV(#1,0)
```

Durch **SEEKKEY** wurde eine aktuelle Position in der angezeigten Datei eingerichtet.

Mit dem ersten **SEEKPREV** in Zeile 590 wird die aktuelle Position auf Reihe 2, Schlüssel K und Satznummer 4 gesetzt und ergibt den Antwortcode 101, weil der Schlüsselwert gewechselt hat.

Ein weiteres **SEEKPREV** ergibt den Antwortcode 0 und die aktuelle Position geht auf Reihe 2, Schlüssel K, Satznummer 2. Weder Schlüsselwert noch Schlüsselreihe haben gewechselt.

Mit dem **SEEKPREV** in Zeile 610 wechselt die aktuelle Position auf Reihe 2, Schlüssel G, Satznummer 5 mit Antwortcode 101, da der Schlüsselwert sich geändert hat.

Ein nochmaliges **SEEKPREV** setzt die aktuelle Position auf Reihe 1, Schlüssel 8000 und Satznummer 6. Antwortcode ist hier 102, weil die Schlüsselreihe gewechselt hat.

Zu den anderen Antwortcodes siehe die Ausführungen zu **SEEKNEXT**. In Zeile 630 wird Antwortcode 105 ausgegeben, die aktuelle Position ist unbestimmt. In Zeile 640 wird Antwortcode 105 ausgegeben, jedoch ist die aktuelle Position auf Reihe 2, Schlüssel K, Satznummer 2.

Die Funktion **SEEKRANK** hat nichts mit Seekrankheit zu tun, sondern ist als SEEK-RANK zu lesen: "Suche eine Schlüsselreihe!"

Durch diese Funktion wird die aktuelle Position des Programms in der Datei auf den Anfang einer Schlüsselreihe gesetzt. Dies ist sinnvoll, wenn man die Daten der Jetsam-Datei sequentiell, abhängig von einer bestimmten Schlüsselreihe abarbeiten möchte. Mit **SEEKRANK** geht man auf den ersten Schlüsseleintrag der Schlüsselreihe und mit **SEEKNEXT** von einem Schlüsseleintrag zum nächsten in der Schlüsselreihe, bis die Funktion **SEEKNEXT** den Antwortcode 102 oder 103 zurückmeldet - d.h., es sind sämtliche Schlüsseleinträge dieser Schlüsselreihe abgearbeitet worden bzw. das Ende der Datei wurde erreicht.

SEEKRANK(Datei-Symbol, Sperre auf den ersten Satz in der Schlüsselreihe, Schlüsselreihe)

Beispiel:

Reihe 0	ElektMaier	InstaKraus	LudwiElekt	Mayr Großh	XaverUnsin
Satznr.	4	2	6	3	5
	↓				
Reihe 1	2390	2390	8000	8000	8000
Satznr.	2	4	3	5	6
Reihe 2	G	K	K	L	L
Satznr.	5	2	4	3	6

```
680 ergebnis=SEEKRANK(#1,0,1)
690 ergebnis=SEEKRANK(#1,0,4)
```

Das **SEEKRANK** in Zeile 680 ergibt den Antwortcode 0 und setzt wie angezeigt die aktuelle Position auf den Beginn der Schlüsselreihe 1, d.h., auf Reihe 1, Schlüssel 2390 und Satznummer 2.

In Zeile 690 ergibt **SEEKRANK** den Antwortcode 103, weil zu der angegebenen Schlüsselreihe keine Schlüsseleinträge gefunden werden können.

Gäbe es in unserer Beispieldatei die Schlüsselreihen 0 und 1 nicht, ergäbe **ergebnis=SEEKRANK(#1,0,0)** den Antwortcode 102 und würde die aktuelle Position auf Reihe 2, Schlüssel G und Satznummer 5 einrichten. Die gewünschte Schlüsselreihe konnte nicht gefunden werden, jedoch konnte in einer folgenden Schlüsselreihe eine aktuelle Position eingerichtet werden.

Hat man sich die Index-Position, d.h. Reihe, Schlüsselwert und Satznummer, einer Position in der Datei zwischengespeichert, kann die durch diese drei Werte bestimmte Position in der Datei als aktuelle Position durch

```
SEEKREC(Datei-Symbol,Sperre auf den Datensatz (,Schlüsselreihe, Schlüsselwert,
Satznummer der Position die als aktuelle Position eingerichtet werden soll))
```


wieder hergestellt werden, um z.B. eine sequentielle Verarbeitung der Datei über **SEEKNEXT** von der zwischengespeicherten Position ab fortzusetzen.

Beispiel

Reihe 0	ElektMaier	InstaKraus	LudwiElekt	Mayr Großh	XaverUnsin
				↓	
Satznr.	4	2	6	3	5
Reihe 1	2390	2390	8000	8000	8000
Satznr.	2	4	3	5	6
Reihe 2	G	K	K	L	L
Satznr.	5	2	4	3	6

```
720 reihe=0:schluessel$="Mayr Großh":satznummer=3
730 ergebnis=SEEKREC(#1,0,reihe,schluessel$,satznummer)
740 ergebnis=SEEKREC(#1,0,0,"XaverFranz",6)
750 ergebnis=SEEKREC(#1,0,2,"M",6)
```

In Zeile 720 werden in Variablen die Werte für eine Index-Position angegeben, die dann in Zeile 730 im **SEEKREC** zur Einrichtung einer aktuellen Position benutzt werden. Antwortcode ist 0, die aktuelle Position wird auf Reihe 0, Schlüssel "Mayr Großh" mit Satznummer 3 - wie angezeigt - eingerichtet.

In Zeile 740 wird Antwortcode 105 ausgegeben, die Indexposition konnte nicht gefunden werden, weil in Reihe 0 kein entsprechender Schlüsseleintrag vorhanden ist.

In Zeile 750 wird Antwortcode 103 ausgegeben, die Indexposition konnte nicht gefunden werden - es wurde das Ende der Datei erreicht.

Jetsam faßt gleiche Schlüsselwerte in einer Reihe zu sogenannten Schlüsselsätzen zusammen. **SEEKSET** setzt die aktuelle Position auf den Beginn des nächsten, der aktuellen

Position folgenden Schlüsselsatzes. Ggf. besteht ein Schlüsselsatz nur aus einem Schlüsseleintrag. Die aktuelle Position wird stets auf den nächsten, vom momentanen Schlüsselwert abweichenden Schlüsseleintrag gesetzt.

SEEKSET ist ähnlich wie **SEEKNEXT**, nur daß **SEEKNEXT** stets zum nächsten Schlüsseleintrag geht, egal wie der neue Schlüsseleintrag aussieht, während **SEEKSET** die Schlüsseleinträge überspringt, die denselben Schlüsselwert haben wie der aktuelle Schlüsselwert.

Aufgrund der Antwortcodes läßt sich erkennen, ob die aktuelle Position noch in derselben Schlüsselreihe ist (Code 101) oder ob die aktuelle Position in eine andere Schlüsselreihe gewechselt hat (Code 102).

Antwortcode 103 deutet darauf hin, daß das Ende der Datei erreicht wurde.

Antwortcode 0 wird bei dieser Funktion nicht ausgegeben.

```
SEEKSET(Datei-Symbol, Sperre auf den Datensatz [Schlüsselreihe, Schlüsselwert,
Satznummer der aktuellen Position von der bei der Suche ausgegangen werden
soll])
```

Beispiel:

Reihe 0	ElektMeier	InstaKraus	LudwiElekt	Meyr GroßH	XeverUnsin
Satznr.	4	2	6	3	5
	↓				
Reihe 1	2390	2390	8000	8000	8000
Satznr.	2	4	3	5	6
Reihe 2	G	K	K	L	L
Satznr.	5	2	4	3	6

```
780 ergebnis=SEEKKEY(#1,0,1,MKI$(2390))
790 ergebnis=SEEKSET(#1,0)
800 ergebnis=SEEKSET(#1,0)
810 ergebnis=SEEKSET(#1,0)
820 ergebnis=SEEKSET(#1,0)
830 ergebnis=SEEKSET(#1,0)
840 ergebnis=SEEKSET(#1,0)
```


Von der aktuellen Position ausgehend ergibt **SEEKSET** in Zeile 790 den Antwortcode 101, d.h., die neue aktuelle Position auf Reihe 1, Schlüssel 8000, Satznummer 3 ist in der Schlüsselreihe geblieben.

Ein nochmaliges **SEEKSET** in Zeile 800 setzt die aktuelle Position auf Reihe 2, Schlüssel G und Satznummer 5 und ergibt den Antwortcode 102, da die Schlüsselreihe gewechselt hat.

Mit zwei weiteren **SEEKSET** (Zeilen 810 und 820) springt man vor auf Reihe 2, Schlüssel L und Satznummer 3, jeweils mit Antwortcode 101.

Ein weiteres **SEEKSET** (Zeile 830) ergibt den Antwortcode 103, da das Ende der Jetsam-Datei erreicht wurde, die aktuelle Position ist unbestimmt.

Von einer unbestimmten aktuellen Position aus kann kein **SEEKSET** ausgeführt werden - Antwortcode 115 (Zeile 840). Zu Antwortcode 115 siehe die Erläuterungen bei **SEEKNEXT**. Statt **SEEKNEXT** ist dort **SEEKSET** einzusetzen.

Wurde durch eine der **SEEK**-Funktionen die aktuelle Position eingerichtet, kann durch den **GET**-Befehl der zugehörige Datensatz eingelesen werden, denn bisher wurde nur der Schlüssel aus der Schlüsseldatei gelesen, aber noch nicht der Datensatz aus der Datendatei.

GET stellt den Datensatz aus der Datendatei, der zum aktuellen Schlüsseleintrag (Reihe, Schlüsselwert und Satznummer) gehört, in einen Satzpuffer. Dieser steht dann dem Programm

zur Verfügung. Über die im **FIELD**-Kommando festgelegten Feldnamen der Datenfelder des Datensatzes im Satzpuffer kann auf den Inhalt des Datensatzes zugegriffen werden.
Der Satzpuffer ist ähnlich wie der Pufferbereich, der durch **BUFFERS** für die Schlüsseldatei bestimmt wird, ein besonderer Speicherbereich, über den Datensätze eingelesen und geschrieben werden.

Durch Angabe einer Satznummer kann jeder beliebige Datensatz eingelesen werden. Die Satznummer sollte jedoch durch die Funktion **FETCHREC** ermittelt sein, damit nicht ungültige Datensätze verarbeitet werden.

```
GET Dateisymbol [, Satznummer des zu lesenden Satzes]
```

oder

```
GET Datei-Symbol, [Satznummer des zu lesenden Satzes], Sperre auf den
Datensatz
```

Beispiel:

```
350 IF ergebnis<103 THEN GET #1
```

Hier wird durch das vorangestellte **IF** auf den Ergebniswert der letzten Jetsam-Funktion sichergestellt, daß nur dann ein **GET** durchgeführt wird, wenn eine aktuelle Position eingerichtet werden konnte. Andernfalls würde die Fehlermeldung **record number error** ausgegeben werden.

Durch **PUT** werden bestehende Datensätze einer Jetsam-Datei überschrieben, d.h., der Dateninhalt des Datensatzes wird verändert.
Mit diesem Befehl können keine neuen zusätzlichen Datensätze in die Jetsam-Datei eingestellt werden. Dies ist nur durch den Befehl **ADDREC** mögich.

```
PUT Datei-Symbol [, Satznummer des zu verändernden Satzes]
```

oder

```
PUT Datei-Symbol, [Satznummer des zu verändernden Satzes], Sperre auf den
Datensatz
```


Wird keine Satznummer mitgegeben, wird der Datensatz, der zur aktuellen Position in der Jetsam-Datei gehört, überschrieben. Durch Angabe einer Satznummer kann jeder beliebige Datensatz überschrieben werden, jedoch sollte die Satznummer durch **FETCHREC** bestimmt werden, um keine ungültigen Datensätze zu verarbeiten.

Beispiel:

```
1110 PRINT"Lesen der Daten zum Geschäftsfreund": ergebnis=SEEKKEY(#1,2,2,"G"):
GET #1
1120 PRINT"Verändern der Ortsangabe von "xort$" auf ";:LSET xort$="München
340":PRINT xort$
1130 PRINT"Zurückschreiben des Satzes":PUT #1
1140 PRINT"Lesen der Daten zu Xaver Unsinn": ergebnis=SEEKKEY(#1,1,0,"Xaver
Unsin"):GET #1
1150 PRINT xvorname$" "xname$" "CVI(xplz$)" "xort$" "xkunde$
```

Wie an den erläuternden Texten der **PRINT**-Befehle zu sehen ist, wird hier ein Datensatz zunächst über **SEEKKEY** und **GET** eingelesen - Zeile 1110, und dann durch **LSET** und **PUT** ein Feldinhalt verändert sowie der Datensatz in die Datendatei zurückgeschrieben - Zeilen 1120 und 1130. In Zeile 1140 wird der Datensatz über einen anderen Schlüssel erneut eingelesen und in Zeile 1150 werden alle Datenfelder ausgegeben, um die Veränderung der Ortsangabe zu prüfen.

In den Datensätzen sind in der Regel sämtliche Schlüssel enthalten, durch die man diesen Datensatz über die Schlüsseldatei ermitteln kann. Wird durch ein **PUT** ein Datenfeld, das als Schlüssel benutzt wird, verändert, muß zusätzlich über **ADDKEY/DELKEY** (siehe dort) die Schlüsselinformation in der Schlüsseldatei angepaßt werden, denn Jetsam paßt die Schlüsselinformation nicht von selbst an.
Da dies sehr leicht vergessen wird, sollte ein Datensatz nicht über **PUT** geändert werden, sondern:

1. durch Lesen des zu ändernden Datensatzes (**SEEK**-Funktion und **GET**),
2. Löschen sämtlicher Schlüssel zum alten Datensatz

(**DELKEY**),

3. Verändern des Inhalts des Datensatzes im Satzpuffer (**LSET, RSET, MID$**) und
4. als neuen Datensatz über **ADDREC/ADDKEY** wieder in die Jetsam-Datei

zurückgeschrieben werden. Diese Vorgehensweise ist letztlich sicherer, als die Arbeitsweise mit **PUT** und Anpassen der Schlüsselinformationen in der Schlüsseldatei.

Zum Löschen von Datensätzen wird die Funktion **DELKEY** benutzt.

Die Datensätze selbst werden in der Jetsam-Datei nicht gelöscht. Es werden lediglich sämtliche Schlüssel zu dem Datensatz in der Schlüsseldatei durch **DELKEY** entfernt. Damit steht der dazugehörige Datensatz nicht mehr zur Verfügung, weil Jetsam die Satznummer als wieder belegbar gekennzeichnet hat. Solange die Satznummer nicht wieder belegt wird, bleibt dieser Datensatz als ungültiger Datensatz in der Datendatei stehen.

Entgegen der Darstellung im BASIC-Benutzer-Handbuch wird durch das Löschen sämtlicher Schlüssel einer Jetsam-Datei die Jetsam-Datei nicht gelöscht. Es sind dann nur sämtliche Datensätze als gelöscht gekennzeichnet und die Jetsam-Datei steht wieder so zur Verfügung, wie nach der Durchführung eines **CREATE**. Ggf. ist die Jetsam-Datei zu löschen, indem die Daten- und die Schlüsseldatei über **KILL** gelöscht werden.

Die aktuelle Position in der Jetsam-Datei wird durch **DELKEY** verändert. Die Antwortcodes geben darüber näher Auskunft. Siehe auch die Ausführungen im Beispiel zu **SEEKNEXT**.

```
DELKEY(Datei-Symbol,Sperre auf den Datensatz der zu der auf den zu löschenden
Schlüssel nachfolgenden Indexposition gehört[,Schlüsselreihe in der der
Schlüssel gelöscht werden soll,zu löschender Schlüsselwert,Satznummer des
Datensatzes zu dem der Schlüssel gelöscht werden soll])
```


Beispiel

```
1190 PRINT"Lesen des ersten Lieferanten":ergebnis=SEEKKEY(#1,2,2,"L")
1200 PRINT"Löschen des Schlüssels - hier der Kennung Lieferant":satznr=
FETCHREC(#1):ergebnis=DELKEY(#1,2,2,"L",satznr)
1210 PRINT"Lesen der Postleitzahl zu diesem Datensatz als Schlüssel":ergebnis
=SEEKKEY(#1,2,1,MKI$(8000))
1220 PRINT"Löschen des Schlüssels - hier der Postleitzahl":ergebnis=DELKEY
(#1,2,1,MKI$(8000),satznr)
1230 PRINT"Lesen des kombinierten Schlüssels Name und Vorname":ergebnis=
DELKEY(#1,2,0,"Mayr Großh",satznr)
1240 PRINT"Löschen des Schlüssels - hier Name und Vorname":ergebnis=DELKEY
(#1,2,0,"Mayr Großh",satznr)
1250 PRINT"Erst jetzt sind alle Informationen zu Großhandel Mayr gelöscht!!"
1260 ergebnis=SEKKEY(#1,2,0,"Mayr Großh")
1270 ergebnis=SEEKKEY(#1,2,1,MKI$(8000))
1280 ergebnis=SEEKKEY(#1,2,2,"L")
```

Die erläuternden **PRINT**-Texte beschreiben, was in der Zeile gerade geschieht. Je Schlüsselreihe wird zunächst der Schlüssel gelesen, die Satznummer des Datensatzes ermittelt (nur in Zeile 1200) und dann wird über **DELKEY** der Schlüssel aus der Schlüsseldatei gelöscht.
Sind in allen Schlüsselreihen die Schlüssel zu Mayr Großhandel gelöscht, so ist der Datensatz in der Jetsam-Datei als ungültig gekennzeichnet, da wir keinen Schlüssel mehr haben, über den auf den Datensatz zugegriffen werden kann. Dies wird auch durch **SEEKKEY** in den Zeilen 1260 bis 1280 angezeigt. In Zeile 1260 erhalten wir Antwortcode 105, in Zeile 1270 und 1280 bekommen wir Antwortcode 0, jedoch nur, weil noch andere Schlüsseleinträge mit dem selben Schlüsselwert, wie ihn der gelöschte Datensatz hatte, vorhanden sind, d.h., es wird auf andere Datensätze hingewiesen.

Der Befehl **OPTION FIELD** sollte normalerweise nicht mehr benutzt werden, damit das Programm nicht unnötig fehleranfällig wird. Er ist nur aus Verträglichkeitsgründen zu älteren BASIC-Versionen aufgenommen worden, in denen es möglich war, die zwei für Jetsam in den Datensätzen

reservierten Bytes, in denen die Satznummer notiert ist, dem Benutzer zugänglich zu machen. Auf eine weitere Beschreibung des Befehls wird daher verzichtet. Ggf. kann die Satznummer jetzt besser über **FETCHREC** ermittelt werden.

Der Befehl **LOCK** hat nur in einem Mehrbenutzer-System Bedeutung, um Satzsperren gezielt einzurichten oder aufzuheben.

Die Satzsperren spielen in fast allen Jetsam-Befehlen und -Funktionen eine Rolle. Diese Sperren sind in der Regel jedoch nur vorübergehend eingerichtet. Durch eine der **SEEK**-Funktionen wird eine Lesesperre über einen Satz errichtet, die erst wieder aufgehoben wird, wenn auf einen anderen Satz durch eine **SEEK**-Funktion zugegriffen wird.

Durch die **LOCK**-Funktion kann ein Benutzer versuchen, eine Sperre auf einen Datensatz zu ändern oder zu löschen, um den Satz wieder zur allgemeinen Benutzung durch andere Benutzer freizugeben. Die Satznummer, die mitgegeben wird, sollte durch die Funktion **FETCHREC** ermittelt sein.

Es kann hierdurch nur eine Sperre für einen einzelnen Datensatz errichtet werden, nicht auf die gesamte Jetsam-Datei. Eine Sperre auf die gesamte Datei wird durch **CREATE** oder durch **OPEN** eingerichtet.

```
LOCK(Datei-Symbol,Art der Sperre,Satznummer des Datensatzes zu dem die Sperre
eingerichtet werden soll)
```

Wird als Sperre der Wert 0 angegeben, wird eine vorher eingerichtete Sperre gelöscht. Mit dem Wert 1 wird eine bestehende Sperre auf den Datensatz zur Lesesperre geändert, d.h. ein anderer Benutzer darf den Datensatz lesen, aber nicht verändern. Mit dem Wert 2 wird eine bestehende Sperre auf den Datensatz zu einer Schreibsperre geändert, d.h. ein anderer Benutzer darf den Datensatz weder lesen noch verändern.

Die von den Jetsam-Funktionen zurückgemeldeten Antwortcodes hinsichtlich des Setzens von Sperren (Antwortcodes 130 bis 133) sind bei dem zum JOYCE mitgelieferten BASIC unwichtig, da das BASIC ein Einzelbenutzersystem ist (siehe den Befehl **VERSION** mit dem Parameter 2: **VERSION(2)** meldet eine 1 zurück - siehe S. 351 im BASIC-Benutzer-Handbuch). Wer jedoch auf dem JOYCE BASIC-Programme mit Jetsam entwickelt, die später in eine Mehrbenutzerumgebung übernommen werden, sollte diese Antwortcodes berücksichtigen und entsprechende Routinen im Programm vorsehen, damit auf diese Bedingungen vom Programm oder vom Benutzer her eingegangen werden kann.

In den Jetsam-Funktionen **ADDKEY, DELKEY, LOCK** und **SEEK** sind in der Regel Angaben zum Setzen von Sperren zu machen. Damit die Funktionen ordnungsgemäß arbeiten können, sind die Sperren auch in der Einzelbenutzerumgebung wichtig und sollten der Funktion entsprechend gesetzt werden (z. B. Schreibsperre bei **ADDREC** und nicht nur Lesesperre).

Platz sparen ist die Devise

Um die Datensätze nicht unnötig groß definieren zu müssen, gibt es zur Speicherung von Zahlen einige Möglichkeiten, diese in verdichteter Form im Speicher abzulegen:

Doppelt genaue Zahlen

MKD$ 8 Byte langer String Rückumwandlung über **CVD**

```
10 dopp$=MKD$(11/3)
20 erg#=CVD(dopp$)
```

Einfach genaue Zahlen

MKS$ 4 Byte langer String Rückumwandlung über **CVS**

```
10 einf$=MKS$(11/3)
20 erg!=CVS(einf$)
```

Integer-Werte

MKI$ 2 Byte langer String Rückumwandlung über **CVI**

```
10 int$=MKI$(32500)
```

```
20 ergX=CVI(int$)
```

Möglicher Zahlenbereich -32768 bis +32767

Interne Darstellung:

Integer-Zahlen werden im Hexadezimalen Zahlensystem durch Werte von 0000H bis 0FFFFH dargestellt. (Siehe BASIC-Funktion **HEX$**). Die interne Darstellung umfaßt daher stets nur zwei Bytes. Durch **MKI$** wird diese interne Darstellung in einen String-Ausdruck überführt. Dabei werden die beiden Bytes miteinander vertauscht.

Beispiel: Dezimalzahl 32500 ist hexadezimal 7EF4H; Speicherung durch **MKI$** als 0F47EH. Die Funktion **CVI** wandelt den String 0F47EH wieder zurück in die Zahl 32500.

MKIK$ 2 Byte langer String Rückumwandlung über **CVIK**

```
10 intk$=MKIK$(11/3)
20 ergX=CVIK(intk$)
```

Der hiermit erzeugte String kann als Jetsam-Schlüssel benutzt werden, d.h., dieser String kann über **ADDREC/ADDKEY** in die Schlüsseldatei eingefügt werden.

Möglicher Zahlenbereich -32768 bis +32767

Interne Darstellung:

Die erste Hälfte des ersten Bytes wird zur Darstellung des Vorzeichens benutzt (8 bis F für positive Zahlen und 0 bis 7 für negative Zahlen). In den verbleibenden 1 1/2 Byte wird die Zahl als normale hexadezimale Zahl dargestellt.

Beispiel: Dezimalzahl +32500 wird zu 0FEF4H; Dezimalzahl -32500 (hexadezimal 810CH) wird zu 010CH; +10 (000AH) wird zu 800AH und -10 (0FFF6H) wird zu 7FF6H.

MKUK$ 2 Byte langer String Rückumwandlung über **CVUK**

```
10 intk2$=MKUK$(11/3)
20 ergX=CVUK(intk2$)
```


Diese Funktion arbeitet wie **MKIK$**, nur daß der Zahlenbereich aufgrund des fehlenden Vorzeichens 0 bis 65535 beträgt. Der erzeugte String kann ebenfalls als Jetsam-Schlüssel verwendet werden.

Interne Darstellung:

Die interne Darstellung ist mit der hexadezimalen Zahl identisch.

Die durch die **MKX$**-Funktionen erzeugten Strings sollten nur in der Weise verwendet werden, daß sie anderen Strings zugewiesen werden.

z.B.: **xzahl$=MKI$(integerzahl)**

Die hierdurch erreichte Platzersparnis ist recht beachtlich, denn in Normaldarstellung werden z.B. 2 Byte Länge mit Zahlen ab 100 überschritten, weil diese Zahlen schon mindestens drei Stellen umfassen.

Hier ein kleiner Überblick, an welcher Stelle in einem Programm die Jetsam-Befehle anzuwenden sind:

Anfangsroutinen

BUFFERS

OPEN oder **CREATE**, abhängig davon, ob die Schlüsseldatei und Datendatei bereits vorhanden sind oder nicht (feststellen über **FIND$**, jeweils für die Schlüssel- und Datendatei getrennt)

FIELD-Befehle für die Datensätze der Jetsam-Datei

RANKSPEC zur Angabe, ob für bestimmte Schlüsselreihen nur eindeutige oder mehrfache Schlüssel zulässig sind.

Verarbeitungsroutinen

Einlesen von Daten

Über eine **SEEK**-Funktion (**SEEKKEY, SEEKNEXT, SEEKPREV, SEEKRANK, SEEKREC** und/oder **SEEKSET**) die aktuelle Position in der Jetsam-Datei auf den gewünschten Satz einrichten und durch ein **GET** den Datensatz einlesen.

Verändern von Daten

Einfügen

Eingaberoutine zur Entgegennahme der neuen Daten, z.B. **INPUT**

Einfügen der eingegebenen Daten in eine Jetsam-Datei:

ADDREC zum Einfügen eines Schlüssels in die Schlüsseldatei und des Datensatzes in die Datendatei

ADDKEY zum Einfügen der weiteren Schlüssel in die Schlüsseldatei

Überschreiben

Ermitteln der richtigen Satznummer über **FETCHREC**

Neuschreiben des Datensatzes durch **PUT #1, Satznummer**

Soll ein Schlüsselfeld überschrieben werden, dann zusätzlich Einfügen des neuen Schlüssels über **ADDKEY** und erst dann Löschen des alten Schlüssels über **DELKEY**

Löschen

Löschen sämtlicher Schlüssel zum Datensatz durch entsprechend häufiges Ausführen von **DELKEY** für jeden Schlüssel des zu löschenden Datensatzes. Stets die durch **FETCHKEY$, FETCHRANK** und **FETCHREC** ermittelte Index-Position angeben, um ein korrektes Löschen der Schlüssel sicherzustellen.

Die Routinen zum Einfügen, Überschreiben und Löschen sollten stets mit **CONSOLIDATE** abgeschlossen werden.

Enderoutinen

Das Schließen einer Jetsam-Datei muß ausdrücklich durch **CLOSE** mit Angabe der Dateinummer geschehen, da sonst die Schlüssel- und Datendatei als nicht stimmig (nicht konsistent zueinander) gekennzeichnet sind.

Unterschied von Funktion und Kommando im BASIC

Vielen ist sicherlich aufgefallen, daß im BASIC-Benutzer-Handbuch bei der ausführlichen Beschreibung der einzelnen Befehle rechts oben in der Ecke steht, ob es sich um eine Funktion oder ein Kommando handelt.

Bei einer Funktion wird nach Ausführung des Befehls in einer Variablen ein Ergebnis an das Programm bzw. an den Anwender zurückgegeben. Bei den Jetsam-Funktionen werden die Antwortcodes der Funktion bereitgestellt. Bei anderen Funktionen wird in der Variable ein Ergebnis bereitgestellt, das dann weiter verarbeitet werden kann.
Der Befehlsaufbau sieht grundsätzlich wie folgt aus:

```
funktionsergebnis=BASIC-FUNKTION(funktionsausgangswert)
```

Eine Funktion kann also nie für sich allein ausgeführt werden, sondern nur in Verbindung mit einer Wertzuweisung des Funktionsergebnisses an eine Variable - wie hier durch das Gleichheitszeichen an die Variable **funktionsergebnis**. In

Klammern sind die Ausgangswerte der Funktion anzugeben, aus denen das Ergebnis erzeugt werden soll.

Beispiel:

Berechnen des Cosinuswertes eines Winkels im Bogenmaß

```
10 cosinus=COS(2)
```

Die numerische Variable **cosinus** nimmt das Funktionsergebnis auf. Mit **COS** wird die BASIC-Routine zur Berechnung eines Cosinuswertes aufgerufen. In Klammern ist der Wert angegeben, zu dem der Cosinus berechnet werden soll, hier die Zahl 2. Durch das Gleichheitszeichen wird der Cosinuswert der Variablen **cosinus** zugewiesen und steht somit dem Anwender als Rechenergebnis zur Verfügung.

Eine Besonderheit sind die Jetsam-Funktionen. Sie sind Funktionen, weil von ihnen Funktionsergebnisse erzeugt werden, die abgefragt werden können - die Antwortcodes. Neben der Erzeugung des Antwortcodes bewirken sie einen Zugriff auf die Jetsam-Datei, suchen nach Daten, speichern Daten, löschen Daten etc. Abhängig von den Antwortcodes kann das Programm dann mit den Daten weiter arbeiten.

Die Antwortcodes der Jetsam-Funktionen lassen sich in folgende Gruppen einteilen:

Funktion wurde erfolgreich durchgeführt - Antwortcode ist 0

Funktion wurde erfolgreich durchgeführt - zusätzlich wird darauf hingewiesen, daß der Schlüsselwert (Antwortcode 101) oder die Schlüsselreihe (Antwortcode 102) gewechselt hat.

Funktion konnte nicht erfolgreich durchgeführt werden:
Antwortcode 103 zeigt an, daß das Ende der Datei erreicht wurde
Antwortcode 105 sagt aus, daß der Schlüssel nicht gefunden wurde
Antwortcode 115 zeigt an, daß keine aktuelle Position eingerichtet ist

Antwortcodes ab 130 sind nur in Mehrbenutzer-Systemen von Bedeutung, weil sie Auskunft darüber geben, ob die Sperren wie gewünscht eingerichtet werden konnten oder nicht.

Ein Kommando steht immer für sich allein. Es wird kein Ergebniswert erzeugt, der einer Variablen zugewiesen werden müßte. Es wird stets direkt etwas bewirkt.

Beispiel:

Schließen von Dateien

```
CLOSE
```

Sämtliche Dateien sind geschlossen. Soll nur eine bestimmte Datei geschlossen werden kann zur genaueren Bestimmung der Wirkungsweise des Kommandos die Dateinummer mitgegeben werden (**CLOSE #1**). Eine solche Möglichkeit zur Steuerung der Wirkungsweise der Kommandos findet sich bei vielen BASIC-Kommandos.

Hier das vollständige Listing des Jetsam-Beispielprogramms JETSAM.BAS:

```
10 e$=CHR$(27):cl$=e$+"H"+e$+"J"
20 OPTION FILES "M:"
30 BUFFERS 20
40 PRINT"Beispielprogramm für die Jetsam-Funktionen und Befehle":GOSUB 380
50 IF FIND$("test.dat")<>"" AND FIND$("test.ind")<>"" THEN OPEN
"K",#1,"test.dat","test.ind",2 ELSE CREATE #1,"test.dat","test.ind",2
60 FIELD #1,20 AS xname$,20 AS xvorname$,2 AS xplz$,20 AS xort$,1 AS xkunde$
70 ergebnis=RANKSPEC(#1,0,1)
80 PRINT cl$
90 GOSUB 860'Addrec und Addkey
100 PRINT cl$
110 GOSUB 410'Seekkey
120 PRINT cl$
130 GOSUB 460'Seeknext
140 PRINT cl$
150 GOSUB 570'Seekprev
160 PRINT cl$
170 GOSUB 670'Seekrank
180 PRINT cl$
190 GOSUB 710'Seekrec
200 PRINT cl$
210 GOSUB 770'Seekset
220 PRINT cl$
230 GOSUB 1100'Put
240 PRINT cl$
250 GOSUB 1180'Delkey
260 CLOSE #1
270 OPTION FILES "A:":END
280 rc=CONSOLIDATE(#1)
290 RETURN
300 'Anzeige der aktuellen Position
310 reihe=FETCHRANK(#1)
320 key$=FETCHKEY$(#1)
330 IF reihe=1 THEN key$=STR$(CVI%(FETCHKEY$(#1)))
340 PRINT "Die aktuelle Position ist auf Reihe"reihe", Schlüssel "key$" und
Satznummer"FETCHREC(#1)"."
350 IF ergebnis<103 THEN GET #1:PRINT xvorname$" "xname$" "CVI(xplz$)" "xort$"
"xkunde$:satznr=FETCHREC(#1):GET #1,satznr:PRINT xvorname$" "xname$" "CVI(xplz$)" "xort$"
"xkunde$" "satznr
360 'Ausgabe des Antwortcodes der Jetsamfunktion
370 PRINT"Antwortcode ist"ergebnis;
380 PRINT"Weiter mit beliebiger Taste"
390 t$="":WHILE t$="":t$=INKEY$:WEND
400 RETURN
410 PRINT"Beispiele zu SEEKKEY
420 ergebnis=SEEKKEY(#1,0,1,MKI$(2300)):GOSUB 310
430 ergebnis=SEEKKEY(#1,0,2,"M"):GOSUB 310
440 ergebnis=SEEKKEY(#1,0,2,"L"):GOSUB 310
450 RETURN
460 PRINT"Beispiele zu SEEKNEXT
470 ergebnis=SEEKKEY(#1,0,2,"K"):GOSUB 310
480 ergebnis=SEEKNEXT(#1,0):GOSUB 310
490 ergebnis=SEEKNEXT(#1,0):GOSUB 310
500 ergebnis=SEEKNEXT(#1,0):GOSUB 310
510 ergebnis=SEEKNEXT(#1,0):GOSUB 310
520 ergebnis=SEEKNEXT(#1,0,0):GOSUB 310
530 ergebnis=SEEKKEY(#1,0,0,"XaverUnsin"):GOSUB 310:ergebnis=SEEKNEXT(#1,0):GOSUB 310
540 ergebnis=SEEKNEXT(#1,0,0,"WalteElekt",6):GOSUB 310
550 ergebnis=SEEKNEXT(#1,0,2,"M",5):GOSUB 310
560 RETURN
570 PRINT"Beispiele zu SEEKPREV
580 ergebnis=SEEKKEY(#1,0,2,"L"):GOSUB 310
590 ergebnis=SEEKPREV(#1,0):GOSUB 310
600 ergebnis=SEEKPREV(#1,0):GOSUB 310
610 ergebnis=SEEKPREV(#1,0):GOSUB 310
620 ergebnis=SEEKPREV(#1,0):GOSUB 310
```



```
630 ergebnis=SEEKPREV(#1,0,0,"WalteElekt",6):GOSUB 310
640 ergebnis=SEEKPREV(#1,0,2,"M",5):GOSUB 310
650 ergebnis=SEEKKEY(#1,0,0,"ElektMaier"):GOSUB 310:ergebnis=SEEKPREV(#1,0):GOSUB 310
660 RETURN
670 PRINT"Beispiele zu SEEKRANK
680 ergebnis=SEEKRANK(#1,0,1):GOSUB 310
690 ergebnis=SEEKRANK(#1,0,4):GOSUB 310
700 RETURN
710 PRINT"Beispiele zu SEEKREC
720 reihe=0:schluessel$="Mayr Groß":satznummer=3
730 ergebnis=SEEKREC(#1,0,reihe,schluessel$,satznummer):GOSUB 310
740 ergebnis=SEEKREC(#1,0,0,"XaverFranz",6):GOSUB 310
750 ergebnis=SEEKREC(#1,0,2,"M",6):GOSUB 310
760 RETURN
770 PRINT"Beispiele zu SEEKSET
780 ergebnis=SEEKKEY(#1,0,1,MKI$(2390)):GOSUB 310
790 ergebnis=SEEKSET(#1,0):GOSUB 310
800 ergebnis=SEEKSET(#1,0):GOSUB 310
810 ergebnis=SEEKSET(#1,0):GOSUB 310
820 ergebnis=SEEKSET(#1,0):GOSUB 310
830 ergebnis=SEEKSET(#1,0):GOSUB 310
840 ergebnis=SEEKSET(#1,0):GOSUB 310
850 RETURN
860 PRINT"Beispiele für ADDREC und ADDKEY
870 LSET xname$="Installation":LSET xvorname$="Krause":xplz=2390:LSET
xort$="Flensburg":LSET xkunde$="K"
880 GOSUB 980
890 LSET xname$="Mayr":LSET xvorname$="Großhandel":xplz=8000:LSET xort$="München 50":LSET
xkunde$="L"
900 GOSUB 980
910 LSET xname$="Elektro":LSET xvorname$="Maier":xplz=2390:LSET xort$="Flensburg":LSET
xkunde$="K"
920 GOSUB 980
930 LSET xname$="Xaver":LSET xvorname$="Unsinn":xplz=8000:LSET xort$="München 200":LSET
xkunde$="G"
940 GOSUB 980
950 LSET xname$="Ludwig":LSET xvorname$="Elektronik":xplz=8000:LSET xort$="München
150":LSET xkunde$="L"
960 GOSUB 980
970 RETURN
980 key$=LEFT$(xname$,5):IF LEN(xname$)<5 THEN key$=key$+SPACE$(5-LEN(xname$))
990 key$=key$+LEFT$(xvorname$,5):IF LEN(xvorname$)<5 THEN
key$=key$+SPACE$(5-LEN(xvorname$))
1000 LSET xplz$=MKI$(xplz)
1010 ergebnis=ADDREC(#1,2,0,key$):GOSUB 310
1020 GOSUB 280:IF ergebnis=116 THEN 1090
1030 key$=MKI$(xplz)
1040 satznr=FETCHREC(#1):ergebnis=ADDKEY(#1,2,1,key$,satznr)
1050 GOSUB 310:GOSUB 280
1060 key$=xkunde$
1070 satznr=FETCHREC(#1):ergebnis=ADDKEY(#1,2,2,key$,satznr)
1080 GOSUB 310:GOSUB 280
1090 RETURN
1100 PRINT"Beispiel für PUT
1110 PRINT"Lesen der Daten zum Geschäftsfreund":ergebnis=SEEKKEY(#1,2,2,"G"):GOSUB 310:GET
#1
1120 PRINT"Verändern der Ortsangabe von "xort$" auf ":LSET xort$="München 340":PRINT
xort$
1130 PRINT"Zurückschreiben des Satzes":PUT #1
1140 PRINT"Lesen der Daten zu Xaver Unsinn":ergebnis=SEEKKEY(#1,1,0,"XaverUnsin"):GET #1
1150 PRINT xvorname$" "xname$" "CVI(xplz$)" "xort$" "xkunde$
1160 PRINT"Weiter mit beliebiger Taste":t$="":WHILE t$="":t$=INKEY$:WEND
1170 RETURN
1180 PRINT"Beispiel zu DELKEY
1190 PRINT"Lesen des ersten Lieferanten":ergebnis=SEEKKEY(#1,2,2,"L"):GOSUB 310
1200 PRINT"Löschen des Schlüssels - hier der Kennung
Lieferant":satznr=FETCHREC(#1):ergebnis=DELKEY(#1,2,2,"L",satznr):GOSUB 280:GOSUB 310
1210 PRINT"Lesen der Postleitzahl zu diesem Datensatz als
Schlüssel":ergebnis=SEEKKEY(#1,2,1,MKI$(8000)):GOSUB 310
1220 PRINT"Löschen des Schlüssels - hier der
Postleitzahl":ergebnis=DELKEY(#1,2,1,MKI$(8000),satznr):GOSUB 280:GOSUB 310
1230 PRINT"Lesen des kombinierten Schlüssels Name und
```

```
Vorname":ergebnis=SEEKKEY(#1,2,0,"Mayr Großh"):GOSUB 310
1240 PRINT"Löschen des Schlüssels - hier Name und Vorname":ergebnis=DELKEY(#1,2,0,"Mayr
Großh",satznr):GOSUB 280:GOSUB 310
1250 PRINT"Erst jetzt sind alle Informationen zu Großhandel Mayr gelöscht!!"
1260 ergebnis=SEEKKEY(#1,2,0,"Mayr Großh"):GOSUB 310
1270 ergebnis=SEEKKEY(#1,2,1,NK1K$(8000)):GOSUB 310
1280 ergebnis=SEEKKEY(#1,2,2,"L"):GOSUB 310
1290 RETURN
```


Verarbeitung von Dateien mit wahlfreiem Zugriff

Bedeutsame Befehle:

- CLOSE
- FIELD
- GET
- INPUT #
- INPUT$
- LOC
- PRINT #
- PUT
- OPEN
- WRITE #

Das Programm RANDOM.BAS am Ende dieses Kapitels ist ein kleines Beispielprogramm, das die Wirkung der Befehle zeigt.

Wahlfreier Zugriff bedeutet, daß die Datei gleichzeitig gelesen und beschrieben werden kann.

OPEN "R",#1,datei$,168 eröffnet eine Datei für wahlfreien Zugriff mit einer Satzlänge von 168 Stellen. Da hier die Standardeinstellung von 128 Stellen Satzlänge überschritten wird, muß die maximale Satzlänge vorher durch ein **CLEAR,,,,168** oder **MEMORY,,,168** heraufgesetzt werden.

Ist die Datei noch nicht vorhanden, wird sie automatisch angelegt.

```
OPEN "R",Datei-Symbol,Name der Random-Datei[,maximale Länge des Datensatzes]
```

Beispiel

```
10 OPEN "R",#1,"test.rdm"
```

Hier wird unter dem Datei-Symbol **#1** die Datei TEST.RDM eingerichtet und für die Verarbeitung im Programm eröffnet.

GET #1 liest einen Satz der für wahlfreien Zugriff geöffneten Datei in den Satzpuffer.

Wurde der Satzpuffer durch ein **FIELD**-Kommando in verschiedene Datenfelder unterteilt, kann direkt auf die Datenfelder zugegriffen werden.

Wurde kein **FIELD**-Kommando benutzt, kann der Satzpuffer durch **INPUT #** oder **INPUT$** gelesen und mit **PRINT #** oder **WRITE #** geschrieben werden.

```
GET Datei-Symbol der Random-Datei[,Satznummer des zu lesenden Satzes]
```

Beispiel

```
65 GET #1,2
```

Hier wird aus der Random-Datei der Satz mit der Satznummer 2 in den Satzpuffer gelesen.

FIELD ist im Kapitel über Jetsam im einzelnen beschrieben. **INPUT #, INPUT$, PRINT #** und **WRITE #** sind im BASIC-Teil dieses Buches beschrieben.

PUT #1 schreibt den aktuellen Inhalt des Satzpuffers in die Random-Datei. Der Inhalt des Satzpuffers wird mit der aktuellen Satznummer in der Random-Datei abgelegt.

Put #1,35 schreibt den Inhalt des Satzpuffers in Satz Nr. 35 der für wahlfreien Zugriff geöffneten Datei.

Der Satzpuffer kann nur durch die Kommandos **LSET, RSET, MID$, PRINT #** und **WRITE #** gefüllt werden. **LSET, RSET** und **MID$** beziehen sich dabei auf die im **FIELD**-Kommando beschriebenen Datenfelder des Satzpuffers. Ist kein **FIELD**-Kommando vorgesehen, kann der Satzpuffer über **PRINT #** und **WRITE #** beschrieben werden.

WRITE # füllt jeweils den Puffer bis zum Ende des Puffers mit Leerstellen und Wagenrücklauf am Ende auf, so daß nach jedem **WRITE #** ein **PUT** kommen muß; sonst wird die Fehlermeldung **record overflow** ausgegeben.

PRINT füllt den Puffer nur soweit, wie im **PRINT**-Befehl angegeben: d.h., daß mehrere **PRINT**-Befehle den Satzpuffer auffüllen können, bis er voll ist. Dann erst braucht der Pufferinhalt mit **PUT** auf die Datei geschrieben werden. Nach jedem **PRINT** wird automatisch Wagenrücklauf und Zeilenvorschub eingefügt, es sei denn, der **PRINT**-Befehl wird mit Komma oder Semikolon abgeschlossen. Das Schreiben des Puffers mit dem **PRINT**-Befehl ist genauso zu betrachten, als ob man eine kleine einfache sequentielle Datei füllt, die in ihrer Größe begrenzt ist (hier die Größe des Satzpuffers = Satzlänge)

Zu den Befehlen **LSET, RSET** und **MID$** siehe im Kapitel über Jetsam.

Beispiel

```
20 FIELD #1,100 AS text$
25 FOR x=1 to 30:LSET text$=" ":PUT #1:NEXT
30 LSET text$="Dies ist Satz Nummer 1":PUT #1,1
40 LSET text$="Dies ist Satz Nummer 2":PUT #1
50 LSET text$="Dies ist Satz Nummer 3":PUT #1
60 LSET text$="Dies ist Satz Nummer 4":PUT #1
```

Hier wird zunächst in Zeile 20 der Satzpuffer durch das **FIELD** definiert. In Zeile 25 wird die Random-Datei vorbereitet, so daß sie bis zu 30 Sätze aufnehmen kann. Diese Vorbereitung ist wichtig, um sicherzustellen, daß auf allen Sätzen der richtige Inhalt geschrieben wird - wird die Random-Datei gezielt auf den Sätzen Nummer 1, 5, 15, und 20 beschrieben, steht in den anderen Sätzen zufälliger Inhalt, wenn die Datei nicht entsprechend vorbereitet wurde. Lassen Sie das Programm RANDOM.BAS einmal ohne Zeile 25 laufen und sehen Sie sich den Unterschied zu den Sätzen Nr. 5 und 6 an.

In Zeile 30 wird im **PUT** gezielt auf Satz Nummer 1 geschrieben. Ohne Angabe der Satznummer würde Satz Nummer 31 beschrieben werden, denn durch die Schleife in Zeile 25 wird die aktuelle Satznummer automatisch jeweils um 1 auf 30 hochgezählt.

In den Zeilen 40 - 60 kann dann wieder die Satznummer weggelassen werden, denn in Zeile 30 wird durch Angabe der Satznummer die 1 zur aktuellen Satznummer gemacht.

satznr=LOC(1) gibt im Feld satznr die Satznummer des zuletzt mit **GET** gelesenen oder mit **PUT** geschriebenen Satzes an. Dies kann hilfreich sein, wenn man z.B. die Nummer des aktuellen Satzes zwischenspeichern möchte, weitere Sätze aus der Datei einliest und später diesen Satz, verändert durch ein **PUT #1,satznr**, zurückschreiben möchte.

```
LOC(Dateinummern-Ausdruck)
```

Beispiel

```
74 PRINT"Der Satz mit der Satznummer"LOC(1)" wurde verarbeitet
83 GET #1:PRINT"Lesen ohne Satznummer - aktuelle Nummer ist"LOC(1)
90 PRINT "Der Satz mit der Satznummer"LOC(1)" ist der zuletzt verarbeitete
Satz"
```

LOC wird hier jeweils benutzt, um die aktuelle Satznummer des gerade verarbeiteten Datensatzes auszugeben.

LOC kann auch bei Dateien benutzt werden, die für sequentielle Ein- (**OPEN "I"**) oder Ausgabe (**OPEN "O"**) eröffnet sind. **saetze=LOC(2)** ergibt dann die Anzahl der in die Datei #2 geschriebenen oder bereits gelesenen 128 Bytes langen Datenblöcke.

CP/M unterteilt jede Datei in Datenblöcke zu jeweils 128 Bytes, egal wie lang die Datensätze oder die Datenelemente in der Datei sind. Ein String von 255 Bytes Länge belegt in einer Datei 2 solcher Dateiblöcke. Das 256. Byte ist ein

systeminternes Längenfeld des Strings, damit CP/M bzw. BASIC erkennen kann, wie lang der String ist. Dieses systeminterne Feld wird durch die Funktion **laenge=LEN(string$)** abgefragt und im Feld **laenge** zur Verfügung gestellt.

Hier das vollständig Listing des Beispielprogramms RANDOM.BAS

```
10 OPTION FILES "M:":OPEN "R",#1,"test.rdm"
20 FIELD #1,100 AS text$
25 FOR x=1 TO 30:LSET text$=" ":PUT #1:NEXT
30 LSET text$="Dies ist Satz Nummer 1":PUT #1,1
40 LSET text$="Dies ist Satz Nummer 2":PUT #1
50 LSET text$="Dies ist Satz Nummer 3":PUT #1
60 LSET text$="Dies ist Satz Nummer 4":PUT #1
65 GET #1,2:PRINT text$
70 LSET text$="Dies ist Satz Nummer 5":PUT #1
74 PRINT "Der Satz mit der Satznummer"LOC(1)" wurde verarbeitet"
75 LSET text$="Dies ist Satz Nummer 6":PUT #1
80 LSET text$="Dies ist Satz Nummer 20":PUT #1,20
83 GET #1:PRINT"Lesen ohne Satznummer - aktuelle Nummer ist"LOC(1):PRINT text$
84 GET #1,3:PRINT text$
90 PRINT "Der Satz mit der Satznummer"LOC(1)" ist der zuletzt verarbeitete Satz"
100 CLOSE #1
110 TYPE test.rdm
120 OPTION FILES "A:":END
```

In Zeile 65 wird Satz Nummer 2 gelesen. Aktuelle Satznummer ist somit 2. In Zeile 70 wird im **PUT** keine Satznummer mitgegeben, so daß der Text "Dies ist Satz Nummer 5" auf Satz Nummer 3 geschrieben wird. Bei einem **PUT** wird zunächst die Satznummer erhöht und dann der Satz mit der erhöhten Satznummer in die Datei geschrieben, wenn keine Satznummer mitgegeben wurde.
In Zeile 75 wird dann der Text "Dies ist Satz Nummer 6" auf die Satznummer 4 geschrieben. Die Sätze 3 und 4 wurden bereits in den Zeilen 50 und 60 beschrieben und werden in den Zeilen 70 und 75 neu eingetragen.

Mit dem **TYPE** in Zeile 110 wird der Inhalt der Datei TEST.RDM angezeigt.

Generator für JETSAM-Verarbeitung

Um die Dateiverarbeitung mit Jetsam zu erleichtern, wurde der Programmgenerator GENERJET.BAS geschaffen, den Sie auf der zum Buch erhältlichen Diskette auf Seite B: finden. Mit diesem Generator wird ein lauffähiges BASIC-Programm erstellt, mit dem eine Jetsam-Datei verarbeitet wird, und die Daten auf Bildschirm und Drucker ausgegeben werden können.

Erläuterungen zum Generator

Der Generator erklärt sich weitgehend von selbst.

Der Generator wird von BASIC gestartet:

Laden Sie zunächst BASIC von CP/M, durch Eingabe von:

```
A>BASIC[ENTER]
```

[ENTER] bedeutet, daß Sie die [RETURN]- bzw. [ENTER]-Taste betätigen.

Schieben Sie jetzt die dem Buch beiligende Diskette mit Seite B in das Laufwerk A: und starten Sie GENREJET durch Eingabe von:

```
OK
run "generjet[ENTER]
```

Sie werden von Generjet begrüßt und nach Betätigen einer beliebigen Taste werden Sie aufgefordert, die notwendigen Eingaben zur Generierung Ihres Jetsam-Programmes zu machen. Im folgenden finden Sie die Erläuterungen zu den Eingaben. Die **fett** angegebenen Texte erscheinen bei Ihnen auf dem Bildschirm.

Es sind folgende Eingaben vorzunehmen - die Eingaben sind in der Regel mit [ENTER] abzuschließen:

Allgemeine Angaben:

Name des Programms:
Der Name darf bis zu 8 Stellen lang sein, längere Bezeichnungen würden zu Fehlern führen, da der Programmname als Name für die zu generierende Programmdatei und die Submitdatei zum Start des generierten Programms benutzt wird.
Ggf. werden Sie nach einem Pieps erneut aufgefordert, einen Namen einzugeben.

Ihr Name:
Ihr Name wird als Kommentarzeile zu Beginn des generierten Programms aufgenommen und für das Begrüßungsbild des generierten Programms ausgegeben.

Das heutige Datum:
Um im Programm genau festzuhalten, wann das generierte Programm erstellt wurde, geben Sie hier bitte das Tagesdatum ein. Die Stellenzahl ist nicht begrenzt. Sie sind daher frei, in welcher Form Sie das Datum eingeben (z.B. 24.12.1988 oder 24. Dezember 1988 etc.)

Kurzbeschreibung/Aufgabe des Programms:
In einer Länge von 244 Buchstaben (einschließlich Leerstellen) kann hier etwas ausführlicher die Aufgabe des Programms beschrieben werden. Dieser Text wird als Kommentarzeile zu Beginn des generierten Programms eingefügt und auf dem Begrüßungsbild beim generierten Programm mit ausgegeben.

Name der Jetsamdatei:
Der Name darf bis zu 8 Stellen lang sein (vgl. Anmerkungen zu **Name des Programms**). Dieser Name wird

ergänzt um die Erweiterungen ".DAT" für die Datendatei und ".IND" für die Schlüsseldatei.

Speicherung der Jetsam-Datei und des Programms auf Laufwerk A: oder B:

Es kann das Laufwerk eingegeben werden, auf dem die Jetsamdatei (Daten- und Schlüsseldatei) und die Programmdatei Ihres generierten Programms gespeichert werden sollen. Es wird nicht geprüft, ob ein Laufwerk "B:" vorhanden ist, falls "B" eingegeben wird. Ggf. tritt ein Fehler bei der Speicherung des generierten Programms auf.
Schieben Sie eine formatierte Diskette in das entsprechende Laufwerk und geben Sie "A" oder "B" ein. Es wird die Datei **Programmname.SUB** angelegt und auf Diskette gesichert.

Angaben zu den Daten, die mit dem zu generierenden Programm verarbeitet werden sollen:

Geben Sie die Feldnamen ein - die Schlüsselfelder bitte als erstes eingeben:

Es werden jetzt der Reihe nach die Feldnamen und weitere Angaben zu den Datenfeldern abgefragt. Danach wird die Endeverarbeitung zur Generierung eingeleitet (siehe unten).
Die Schlüsselfelder sind als erstes einzugeben. Beim ersten eingegebenen Feldnamen wird unterstellt, daß es ein Schlüsselfeld ist, bei dem nur eindeutige Schlüsselwerte zugelassen sind. Sind alle Schüsselfelder eingegeben worden, können die übrigen Datenfelder erstellt werden.

x. Feldname

Wird nur [ENTER] eingegeben, wird die Abfrage zu den Feldnamen beendet.
x. ist eine laufende Nummer der eingegebenen Felder.
Der erste eingegebene Feldname sollte ein

Schlüsselfeld mit eindeutigem Schlüssel sein, da dieser Schlüssel zum Einfügen des neuen Satzes in die Datendatei genommen wird. Damit wird erreicht, daß in der ersten Schlüsselreihe ein Schlüsselwert gespeichert wird, über den man jeden einzelnen Satz eindeutig identifizieren kann.
Für den ersten eingegebenen Feldnamen wird vom Generator daher unterstellt, das es ein Schlüsselfeld ist, bei dem gleiche Schlüsselwerte nicht zugelassen sind (eindeutige Schlüsselwerte).

Zeile und Spalte auf dem Bildschirm für feldname - zeile,spalte

Hier kann angegeben werden, an welcher Bildschirmposition zur Eingabe von Werten aufgefordert werden soll. Es wird damit eine Eingabemaske definiert.
Es ist die Bildschirmzeile anzugeben: Zeile 6 bis 27
Es ist die Spalte der Bildschirmzeile anzugeben: Spalte 0 bis 88
Beispiel: Eingabe: 6,4 - Es wird auf Zeile 6 ab Spalte 4 zur Eingabe aufgefordert werden.

Ist feldname ein Zahlenfeld ? (J/N)

Hier wird angegeben, ob das Feld Text oder nur Zahlen aufnehmen soll. Wird "J" eingegeben, wird bei der Generierung als Zahlenfeld fortgefahren, bei Eingabe eines "N" wird das Feld als Stringfeld generiert, das Texte aufnehmen kann.

Generierung als Zahlenfeld:

Wertebereich der Zahl:

1 - 0 bis 255
2 - -32768 bis +32767
3 - 0 bis 65535
4 - einfach genaue Zahl
5 - doppelt genaue Zahl

Auswahl 1 bis 5 eingeben

Hierdurch kann der Zahlentyp festgelegt werden, außerdem welche Minimum- und Maximumwerte grundsätzlich vorgesehen (Auswahl 1 bis 3) oder ob einfache oder doppelt genaue Zahlen (Auswahl 4 oder 5) verarbeitet werden sollen.

Abhängig von der Auswahl werden entsprechende Routinen vorgesehen, die die Zahlen in der Jetsam-Datei in komprimierter Form speichern, und somit Platz gespart wird. Ebenso werden entsprechende Routinen zur Aufbereitung der komprimierten Werte zur Anzeige und zum Druck vorgesehen.

Geben Sie eine der Zahlen 1 bis 5 zur Festlegung des Zahlentyps bzw. des Wertebereiches ein.

Möchten Sie Minimum und Maximum von feldname festlegen? (J/N)

Sollen keine Grenzwerte festgelegt werden, bleibt es bei der grundsätzlichen Festlegung (Auswahl 1 bis 3 der vorherigen Frage) bzw. es werden keine Grenzwerte vorgesehen (Auswahl 4 bzw. 5 der vorherigen Frage).

Geben Sie "J" oder "N" ein.

Minimal- und Maximalwert von feldname

Wurde gesagt, daß Grenzwerte festgelegt werden sollen (Auswahl "J" bei der vorherigen Frage), können diese jetzt eingegeben werden.

Im generierten Programm wird sichergestellt, daß diese Grenzwerte berücksichtigt werden.

Beispiel: Eingabe: 50,75 - Zu diesem Feld können nur Werte von 50 bis 75 eingegeben werden.

Wurde versehentlich der Maximalwert zuerst eingegeben (Eingabe: 75,50) wird dies berücksichtigt und die Zahlen vom Generator richtig eingesetzt, und zwar mit 50 als Minimal- und 75 als Maximalwert.

Werden bei den Zahlentypen 1 bis 3 durch die Minimal- und Maximalwerte die grundsätzlichen Wertebereiche unter- bzw. überschritten, wird zur Eingabe des gewünschten Zahlentyps zurückgesprungen.

Generierung als Stringfeld:

Maximale Datenlänge von feldname

Hier wird eingegeben, wie viele Stellen der Text maximal lang sein darf. Dies wird für eine Prüfung der Länge des eingegebenen Textes mit herangezogen. Außerdem wird der Text nur bis zu dieser Länge in der Jetsam-Datei abgespeichert.

Geben Sie die maximale Stellenzahl ein.

Gemeinsame Weiterverarbeitung für String- und Zahlenfelder

Soll feldname als Schlüsselbegriff verwendet werden ? (J/N)

Hiermit wird angegeben, ob dieses Feld als Schlüssel für die späteren Datenzugriffe - wie Anzeigen, Löschen, Verändern und Drucken - verwendet werden soll. Beim ersten eingegeben Feldnamen wird vom Generator stets in die Routinen für die Generierung als Schlüsselfeld verzweigt.

Wird diese Frage mit Nein beantwortet, wird für alle weiteren eingegebenen Feldnamen unterstellt, daß sie **nicht** als Schlüssel berücksichtigt werden sollen und diese Frage wird nicht mehr angezeigt.

Geben Sie "J" oder "N" ein.

Generierung des Feldes als Schlüsselbegriff:

Schlüsselgenerierung

Hiermit wird angezeigt, daß man sich in dem Teil des Generators befindet, durch den der Feldname als Schlüsselbegriff vorbereitet wird. Beim ersten Feldnamen wird stets in diesen Teil verzweigt.

Dieser Zahlentyp kann nicht als Schlüssel verwendet werden

Wurde bei einer einfach oder doppelt genauen Zahl gesagt, daß sie als Schlüssel benutzt werden soll, wird die Schlüsselgenerierung zurückgewiesen. Dieses Feld wird als einfaches Datenfeld berücksichtigt. Der nächste Feldname kann wieder als Schlüssel berücksichtigt werden.

Es wurden bereits alle 8 Schlüsselreihen benutzt. In welcher Reihe soll dieser Schlüssel zusätzlich aufgenommen werden (Reihe 1 bis 7; nicht aufnehmen = 9)

Jetsam verarbeitet normalerweise nur maximal 8 verschiedene Schlüssel in sogenannten Schlüsselreihen. Es können in einer Schlüsselreihe jedoch mehr als ein Schlüssel abgelegt werden. Wählen Sie eine beliebige Schlüsselreihe aus. Wird eine 9 eingegeben, wird dieser Feldname überhaupt nicht berücksichtigt und kann ggf. erneut eingegeben werden.
Geben Sie eine der Zahlen 1 bis 7 oder 9 ein.

Als einstellige Kennung für feldname wird buchstabe vorgeschlagen.
Nehmen Sie den Vorschlag an? (J/N)

Mit dieser Kennung wird für den Teil zum Anzeigen bzw. Drucken der gespeicherten Werte im generierten Programm die Wahl erleichtert und vereinfacht, nach welchem Schlüssel Daten gesucht und angezeigt bzw. gedruckt werden sollen.
Bitte entscheiden Sie, ob Sie den vorgeschlagenen Buchstaben für den Feldnamen als einstellige Kennung möchten oder nicht, indem Sie "J" oder "N" eingeben.
Standardmäßig wird das erste Zeichen des Feldnamens vorgeschlagen. Wird dieser bereits für ein anderes Schlüsselfeld als Kennung benutzt, wird ein anderes Zeichen aus dem Feldnamen als Kennung vorgeschlagen.

Bitte eine einstellige Kennung für feldname eingeben

Geben Sie eine Zahl oder einen Buchstaben ein. Groß- oder Kleinschreibung wird bei Buchstaben nicht berücksichtigt.
Mit dieser Kennung wird für den Teil zum Anzeigen bzw. Drucken der gespeicherten Werte im generierten Programm die Wahl erleichtert und vereinfacht, nach welchem Schlüssel Daten gesucht und angezeigt bzw. gedruckt werden sollen.
Diese Frage erscheint nur, wenn automatisch keine einstellige Kennung ermittelt werden konnte oder Sie bei der vorherigen Frage angegeben haben, daß Sie eine andere als die vorgeschlagene Kennung haben möchten.

buchstabe wird als Kennung bereits benutzt

Die eingegebene einstellige Kennung wird bereits benutzt. Bitte geben Sie nach Aufforderung eine andere einstellige Kennung ein.

Länge des Schlüsselbegriffs von feldname

Ist das Feld ein Stringfeld, kann hier festgelegt werden, wie lang der Schlüssel aus dem eingegebenen Text sein soll, beginnend mit der ersten Stelle des Textes.
Beispiel:

Eingabe zum Stringfeld - 30 Stellen lang

Eingabe zur Länge des Schlüssels - 9 Stellen lang

Damit werden nur die ersten 9 Stellen des Stringfeldes als Schlüsselwert genommen.
Die Länge des Schlüsselbegriffs darf die Länge des Stringfeldes nicht überschreiten. Ggf. wird durch ein Pieps auf eine Fehleintragung aufmerksam gemacht mit der Möglichkeit zur Korrektur der Länge des Schlüsselbegriffs.

Kann der Schlüssel für mehrere Datensätze mit demselben Wert vorkommen? (J/N)

Hier wird gefragt, ob zu diesem Schlüssel nur eindeutige Schlüsselwerte vorkommen dürfen (Auswahl "N"), oder ob ein Schlüsselwert zu verschiedenen Datensätzen denselben Schlüsselwert haben darf (mehrdeutige Schlüssel) (Auswahl "J").
Beim ersten Feldnamen wird unterstellt, daß nur eindeutige Schlüssel zugelassen sind. Diese Frage wird beim ersten Feldnamen daher nicht gestellt.
Geben Sie "J" oder "N" ein.

Endeverarbeitung zur Generierung

Möchten Sie das generierte Programm sofort starten ? (J/N)

Hier wird angegeben, ob Sie sofort beginnen möchten, mit Ihrem generierten Programm zu arbeiten, oder ob das Programm nur erstellt und gespeichert werden soll. Auch bei einem Sofortstart wird das generierte Programm zunächst abgespeichert, dann jedoch auch gleich gestartet.
Geben Sie "J" oder "N" ein.

Einen Moment bitte, das generierte Programm wird gesichert!

Diese Meldung teilt Ihnen mit, daß die Generierung abgeschlossen ist, und das Programm jetzt auf Diskette gespeichert wird. Außerdem wird die Jetsam-Datei mit der Schlüsseldatei **Dateiname.IND** und der Datendatei **Dateiname.DAT** auf Diskette angelegt. Bereits bestehende Dateinamen mit gleichem Namen werden vor dem Anlegen gelöscht!!

Das Programm steht auf Laufwerk laufwerk mit dem Namen programmname zur Verfügung!

Der Generator hat seine Arbeit getan. Das Programm steht zum einen im BASIC-Speicher bereit, so daß es mit **RUN** gestartet werden kann. Außerdem wurde das generierte Programm unter dem Namen **Programmname.BAS**

auf Diskette gespeichert.
Wird das generierte Programm erstmals von Diskette gestartet, müssen die Zeilen 1 bis 3 noch gelöscht werden, weil darin noch Anweisungen vom Generator enthalten sind:
Von BASIC her sind einzugeben:

```
OK
LOAD "laufwerk:programmname[ENTER]

OK
delete 1 - 3[ENTER]
SAVE "laufwerk:programmname[ENTER]

OK
```

Damit sind die Zeilen 1 bis 3 dauerhaft aus dem generierten Programm entfernt. Sonst würde jedesmal zu Beginn des Programms ein SAVE ausgeführt werden, der sehr viel Zeit kostet.

Soll der Generator neu gestartet werden, so muß RUN GENERJET unter BASIC eingegeben werden, damit BASIC den Generator neu von Diskette starten kann, denn mit Beendigung der Generierung ist der Generator aus dem Programmspeicher von BASIC entfernt worden.

Erläuterungen zum generierten Programm

Start des generierten Programms:

Auf der Arbeitsdiskette müssen folgende Dateien vorhanden sein:
PIP.COM - Dateikopierprogramm
BASIC.COM - BASIC
SUBMIT.COM - Programm zur Abarbeitung von *.SUB-Dateien
Programmname.SUB - Submitdatei zum Start des generierten Programms unter BASIC
Programmname.BAS - Das generierte Jetsamprogramm
Dateiname.DAT - Die verwendete Datendatei
Dateiname.IND - Die verwendete Schlüsseldatei

Die Dateien **Programmname.BAS**, **Programmname.SUB**, **Dateiname.DAT** und **Dateiname.IND** werden vom Generator angelegt. Kopieren Sie die drei COM-Dateien von Ihrer CP/M-Systemdiskette auf die Arbeitsdiskette, auf der die generierten Dateien stehen.

Das Programm wird von CP/M her gestartet:

A>programmname[ENTER]

Die Daten- und Schlüsseldatei werden zur schnelleren Bearbeitung in das Laufwerk M: kopiert. Dann wird BASIC mit dem generierten Jetsam-Programm gestartet. Nach Beendigung des Jetsamprogramms werden die Dateien zurück auf die Diskette kopiert und somit die Änderungen in den Dateien auf Diskette gesichert.

Die Konfiguration der Jetsamdateien und Kommandodateien kann nach Bedarf geändert werden. Es sind die Zeilen 340 - 360 von GENERJET anzupassen. Es wird von GENERJET folgende Konfiguration unterstellt:

*.COM-Dateien auf Laufwerk A:
programmname.* und Dateiname.* auf dem bei der Generierung eingegebenen Laufwerk A: oder B:.

Nach einer Generierung kann die Datei **Programmname.SUB** von jedem Texteditor, z.B. RPED, geändert werden.

Nach einer kurzen Begrüßung wird das Arbeitsmenü des Programms aufgebaut, von dem aus die Auswahlpunkte

E - Eingabe
V - Verändern
L - Löschen
A - Anzeigen
D - Drucken
X - Programmende

durch Eingabe des groß geschriebenen Buchstabens angewählt werden können.

Das "X" steht auch bei sämtlichen anderen Programmteilen zur Verfügung, um den jeweils angewählten Menüpunkt zu beenden und in das Arbeitsmenü zurückzukommen, um einen anderen Menüpunkt auszuwählen oder das Programm durch nochmalige Eingabe eines "X" ganz zu beenden.

Es wird jeweils rechts oben angezeigt, welcher Programmteil aktiv ist.

Zu den einzelnen Programmteilen:

<u>Eingabe</u>

Nacheinander geben Sie zu jedem Feld, daß Sie bei der Generierung angegeben haben, nach Aufforderung einen Wert ein. Jede Eingabe wird mit [ENTER] abgeschlossen.

Weitere Eingaben?
Weitere Daten können eingegeben werden, wenn ein "W", "E" oder Leertaste eingegeben wird. Die Eingabe eines "X" beendet den Eingabeteil.

Wert zum ersten Schlüssel bereits gespeichert
oder
Satz mit diesem Schlüssel bereits vorhanden

Sie haben für einen Schlüssel bei der Generierung angegeben, daß nur eindeutige Schlüsselwerte zugelassen sind. Es wurde versucht, die eingegebenen Daten in die Jetsamdatei zu schreiben, jedoch ist der Schlüssel bereits vergeben.
Es wurden keine Daten eingefügt. Durch Eingabe von "W", "E" oder Leertaste können die Daten mit veränderten Werten erneut eingegeben werden. Die Eingabe eines "X" beendet den Eingabeteil.

Verändern

Bitte Schlüsselwert eingeben

Zunächst wird nach dem Schlüsselwert gefragt, um den Datensatz einzulesen, der verändert werden soll.
Soll der Datensatz verändert werden, der über den Schlüssel NAME mit dem Wert "Hugo" gefunden werden kann, wird hier aufgefordert, den Namen einzugeben. Geben Sie dann bitte "Hugo" ein.

Eingabe der neuen Werte

Ähnlich wie im Eingabeteil werden Sie jetzt Feld für Feld aufgefordert, die **neuen** Werte einzugeben. Dabei wird jeweils der alte Feldinhalt angezeigt.
Soll der alte Feldinhalt übernommen, d.h., nicht verändert werden, braucht der alte Inhalt nicht erneut eingegeben werden; es genügt, [ENTER] zu betätigen. Die alten Daten werden dann in den neuen Datensatz übernommen.
Soll ein Wert verändert werden, so ist der neue Wert vollständig einzugeben. Es genügt nicht, nur einzelne Zeichen des alten Wertes abzuändern. Das würde zu einer fehlerhaften Speicherung der Daten führen.
Der alte Datensatz bleibt nicht bestehen, er wird

gelöscht. Der veränderte Datensatz wird neu in die Jetsam-Datei eingefügt.

Neue Werte erneut eingeben? (J/N)

Beim Verändern der Werte wurde ein Schlüsselwert verändert. Bei der Schlüsselreihe des Schlüssels sind nur eindeutige Schlüssel zugelassen, d.h., es dürfen nicht zwei gleiche Schlüsselwerte zu verschiedenen Datensätzen vorkommen. Der veränderte Datensatz konnte nicht in die Jetsamdatei geschrieben werden, weil der neue Schlüsselwert bereits in der Jetsamdatei vorhanden ist.
Bei Eingabe eines "J" können die Werte erneut überarbeitet werden, um die Daten richtig zu speichern. Soll der Datensatz auf die alten Werte zurück geändert werden, ist ebenfalls "J" einzugeben, um die Veränderungen rückgängig zu machen, denn der **alte Datensatz** wurde programmintern **bereits gelöscht!**
Bei "N" wird die Veränderung nicht beachtet, der alte bisherige Datensatz wurde jedoch gelöscht und steht nicht mehr zur Verfügung!

Weitere Veränderungen?

Weitere Daten können verändert werden, wenn ein "W", "V" oder Leertaste eingegeben wird. Die Eingabe eines "X" beendet den Programmteil zum Verändern von Daten.

Zu den Fehlermeldungen vgl. die Erläuterungen zu Eingabe.

Löschen

Bitte Schlüsselwert eingeben

Zunächst wird nach dem Schlüsselwert gefragt, um den Datensatz zu suchen und einzulesen, der gelöscht werden soll.
Soll der Datensatz gelöscht werden, der über den Schlüssel NAME mit dem Wert "Hugo" gefunden werden

kann, wird hier aufgefordert, den Namen einzugeben. Geben Sie dann bitte "Hugo" ein.

Löschung mit "L" bestätigen - andernfalls ENTER drücken

Es wird hier nochmals nachgefragt, ob die Daten tatsächlich gelöscht werden sollen. Bei Eingabe eines "L" werden die Daten aus der Jetsamdatei entfernt. Wird die Taste [RETURN] oder [ENTER] betätigt, werden die Daten nicht gelöscht.

Weitere Löschungen?

Weitere Daten können gelöscht werden, wenn ein "W", "L" oder Leertaste eingegeben wird. Die Eingabe eines "X" beendet den Programmteil zum Löschen von Daten.

Anzeigen

Anzeige von Schlüssel1 Schlüssel2 ...
Auswahl über 12 ...

Hier wird gefragt, nach welchem Schlüssel nach Daten gesucht werden soll. Durch Eingabe der bei der Generierung angegebenen einstelligen Kennung des Schlüssels wird der entsprechende Schlüssel ausgewählt. Unten auf dem Bildschirm werden sämtliche einstelligen Schlüsselkennungen angezeigt.
Durch Eingabe eines "X" wird der Programmteil zum Anzeigen von Daten beendet.

Alle oder Einzeln anzeigen?

Sollen alle Daten zu dem Schüssel angezeigt werden, ist ein "A" einzugeben. Nacheinander werden dann sämtliche Daten aufgelistet.

Bitte eine Taste drücken

Ist die letzte Zeile des Bildschirms erreicht, wird aufgefordert, eine beliebige Taste zu drücken, damit die nächste Bildschirmseite zur Anzeige weiterer Daten genutzt werden kann (Seitenwechsel).

Sämtliche Daten zu einem Datensatz werden stets zusammenhängend angezeigt, d.h.: keine Trennung der Daten durch Seitenwechsel.

Sollen nur bestimmte Daten ausgewählt werden, ist ein "E" einzugeben.

Bitte Schlüsselwert eingeben

Es wird nach dem Schlüsselwert gefragt, um den Datensatz zu suchen und einzulesen, der angezeigt werden soll.
Soll der Datensatz angezeigt werden, der über den Schlüssel NAME mit dem Wert "Hugo" gefunden werden kann, wird hier aufgefordert, den Namen einzugeben. Geben Sie dann bitte "Hugo" ein.
Es werden sämtliche Datensätze aufgelistet, die den eingegebenen Schlüsselwert bei diesem Schlüssel haben (alle Datensätze des Schlüsselsatzes). Ggf. wird ein Seitenwechsel durchgeführt wie bei 'Alle Anzeigen'.

Weitere Daten Anzeigen?
oder
Weitere Daten Anzeigen? In rückwärtiger Reihenfolge (Druck=T)?

Weitere Daten können auf dem Bildschirm angezeigt werden, wenn ein "W", "D", "A" oder Leertaste eingegeben wird. Die Eingabe eines "X" beendet den Programmteil zum Anzeigen von Daten.
mit dem Zusatz **"In rückwärtiger Reihenfolge (Druck=T)?"**:
Bei Eingabe eines "R" werden die Daten in umgekehrter Reihenfolge - rückwärts - erneut angezeigt. Bei Eingabe eines "T" werden die Daten in umgekehrter Reihenfolge auf dem Drucker ausgegeben - legen Sie dafür bitte Papier in Ihren Drucker ein.

Drucken

Bitte Papier einlagen und bestätigen

Legen Sie jetzt bitte das Papier in den Drucker ein. Durch Eingabe von "B" oder "P" wird dem Programm mitgeteilt, daß Papier eingelegt wurde. Die Eingabe eines "X" beendet den Programmteil zum Drucken von Daten.

Ansonsten läuft dieser Programmteil genauso ab wie der Programmteil Anzeigen, jedoch wird nichts auf dem Bildschirm angezeigt, sondern alles auf dem Drucker ausgegeben. Blattwechsel werden automatisch berücksichtigt. Die Daten zu einem Datensatz werden stets zusammenhängend ausgegeben, d.h. sie werden nicht durch Seitenwechsel getrennt. Auf ein Blatt werden maximal 66 Zeilen gedruckt.

Weitere Daten Drucken?
oder
Weitere Daten Drucken? In rückwärtiger Reihenfolge?

Weitere Daten können gedruckt werden, wenn ein "W", "D" oder Leertaste eingegeben wird. Die Eingabe eines "X" beendet den Programmteil zum Drucken von Daten.
Mit dem Zusatz **"In rückwärtiger Reihenfolge?"**:
Bei Eingabe eines "R" werden die Daten in umgekehrter Reihenfolge - rückwärts - erneut gedruckt.

Die Einstellung des Druckers ist vom Programm wie folgt vorgesehen:

Elite - 12 Zeichen je Zoll
Einzelblatt
Seitenlänge 76 Zeilen
Lücke am Seitenende (nicht bedruckbare Zeilen am Ende der Seite) 6 Zeilen
Linker Rand 10 Stellen
Zeilenlänge 86 Stellen

Möchten Sie eine andere Standardeinstellung, sind die Zeilen 19300 und 19310 im Programm zu ändern. Dabei helfen die vorbereiteten Steuercodes in den Zeilen 18730 bis 19280. Es

sind sämtliche möglichen Steuercodes zur Steuerung des Druckers vorhanden. Die Steuercodes BS, LF und CR sind bereits bei den Steuercodes für die Bildschirmsteuerung berücksichtigt und können auch für die Steuerung des Druckers benutzt werden.

Auf den folgenden Seiten wird der Aufbau des Generators und des generierten Programms näher erläutert.

Es wird die Einteilung des Bildschirms gezeigt und sämtliche Jetsam-Meldungen aufgeführt, die vom generierten Programm ausgegeben werden.

Außerdem wird erklärt, wie das Programmlisting von GENERJET ausgedruckt werden kann.

Dann folgt eine Liste der Variablen, die im generierten Programm und im Generator benutzt werden, und es werden die Zeilen genannt, in denen die einzelnen Programmteile zu finden sind.

Mögen die Programmroutinen in GENERJET Sie zu eigenen Jetsam-Programmen anregen und ermutigen!

Die Bildschirmzeilen werden wie folgt verwendet:

Zeile	Verwendung
0	Titelzeile
1	Leerzeile
2	Programmauswahlpunkte mit Anzeige des aktiven Programmteils
3- 5	Abfrage von steuernden Informationen bei Verändern, Anzeigen und Drucken
6-27	Datenteil für Anzeigen und Eingabe
28	Leerzeile
29	Abfrage auf "weiter arbeiten" im Programmmodus oder Ende des Programmmodus (Stets Auswahl "X")
30	Fehlermeldungen
31	Leerzeile

Auf Zeile 30 des Bildschirms werden zu sämtlichen Jetsam-Kommandos und -Funktionen aufgrund des Antwortcodes entsprechende Meldungen ausgegeben:

Antwort-code	Text
101	Neuer Schlüsselwert ist ...
102	Neue Schlüsselreihe ..., neuer Schlüsselwert ...
102	bei SEEKRANK: Keine Einträge in der angegebenen Schlüsselreihe. Position ist auf Reihe ...
103	Keine weiteren Werte gespeichert
105	Die aktuelle Position wurde nicht gefunden. Die aktuelle Position ist unbestimmt. oder Die aktuelle Position wurde nicht gefunden. Die Position wurde eingerichtet auf Reihe ... Satznummer ... mit dem Schlüssel ...
105	bei SEEKKEY: ... nicht gefunden. Position - Reihe ... Satznummer ... Schlüssel ...
115	Keine aktuelle Position. Eine Positionierung konnte nicht eingerichtet werden.
116	... ist bereits vorhanden. Mehrfachschlüssel sind nicht zulässig.
130	Die Datei bekam von einem anderen Benutzer eine Lesesperre
131	Der Satz, der zum Schlüssel gehört, konnte nicht schreibgesperrt werden
132	Eine Lesesperre konnte nicht eingerichtet werden
133	Eine Schreibsperre konnte nicht eingerichtet werden

Die Datei wurde von ... Benutzern verändert.

Programmaufbau

Um ein Listing von GENERJET.BAS zu erhalten, starten Sie bitte CP/M, indem Sie Ihren JOYCE einschalten und Ihre CP/M-Startdiskette in Laufwerk A: legen.

Tippen Sie dann:

```
A>BASIC[ENTER]
```

[ENTER] bedeutet, daß Sie die [RETURN]- bzw. [ENTER]-Taste betätigen.

Es wird BASIC in den Arbeitsspeicher geladen und meldet sich mit **OK**. Laden Sie dann GENERJET von der zum Buch erhältlichen Diskette:

```
OK
load "generjet[ENTER]

OK
```

Damit ist GENERJET geladen. Spannen Sie jetzt einen Bogen Papier in den Drucker und halten Sie 9 weitere Blätter Papier bereit. Tippen Sie dann:

```
llist[ENTER]
```

Das Programmlisting wird jetzt auf dem Drucker ausgegeben.

Es werden folgende Variablen benutzt:

im generierten Programm

abbruch	Flag, daß beim Einfügen eines Satzes bzw. Schlüssels ein schwerwiegender rc auftauchte
aendern	Flag für Programmteil Verändern
druck	Flag für Programmteil Drucken; Wert 2: Druck bei rückwärts lesen
dsn$	Dateibezeichnung der Jetsamdatei ohne Extension
ein$	Mögliche Auswahloptionen bei einer Frage an den Benutzer für INKEY$
ein	Weist auf die gewählte Auswahloption hin
einzeln	Flag für 'Anzeigen Einzelne'
einzig	Flag für RANKSPEC - eindeutige oder mehrdeutige Schlüssel möglich
fehler$	Hinweis auf den zulässigen Wertebereich bei Zahlen
frei	Freie Speicherstellen im BASIC-Speicher - für Garbage Collection
grenz1$,grenz2$	Hinweis auf maximal mögliche Stellenzahl bei Texteingaben
init	Flag für Begrüßungsteil
key$	Aktueller Schlüsselwert
modus$	Name des aktiven Programmteils
modus	Index zu modus$
pgm$	Programmname
rc1	Zwischenspeicher für rc
rc	Antwortcode der Jetsamfunktionen
reihe	Aktuelle Schlüsselreihe
satzlaenge	Länge des Jetsam-Datensatzes
satznr	Aktuelle Satznummer des Datensatzes
t$	Über INKEY$ von der Tastatur eingelesenes Zeichen
text$	Textkonstante für den Begrüßungsteil mit der Kurzbeschreibung des Programms
x$	Enthält CHR$(34), um " auf dem Bildschirm darstellen zu können
xrec	Zwischenspeicher für die Satznummer

xrank	Zwischenspeicher für die Reihe
xkey$	Zwischenspeicher für den Schlüssel
yrueck	Flag für rückwärts lesen
zahl$	Nimmt Zahlenwerte auf zur Prüfung auf numerischen Inhalt der Zahl, denn auch Zahlen werden in Stringvariablen eingegeben.
zz	Zeilenzähler für Bildschirm- und Druckerausgabe

im Generator

anzeige$	Schlüsselbezeichnungen
datum$	Eingegebenes Tagesdatum
dein$	Kürzel der Schlüsselbezeichnungen
dfeld$	Feldname in Großbuchstaben
dmax,dmin	Zwischenspeicher für Maximalwert und Minimalwert zu einem Zahlenfeld
dreihe$	Generiert eine Zeile zu Beginn des Eingabeteils, um bei anderen Programmmodi als 'Eingabe' zum Schlüsselfeld zu verzweigen, nach dem in der Jetsamdatei ein Datensatz gesucht werden soll
feldlaenge	Länge eines Datenfeldes in Bytes
feldname$	Feldname des Datenfeldes
gf	Index für gfield$
gfield$	Anweisungen für FIELD
imax,imin	Eingegebener Maximal- und Minimalwert bei Zahlenfeldern
keylen	Länge des Schlüsselbegriffs bei Textfeldern
kfeld$	Name des Schlüsselfeldes
lauf$	Laufwerksbezeichnung
ldfeld$	Feldname in Kleinbuchstaben
name$	Name des Benutzers des Generators
rueck$	Rückumwandlung von komprimiert gespeicherten Zahlen
schluessel	Flag für Schlüsselgenerierung
spalte	Spalte der Bildschirmzeile, wo der Feldname für die Eingaberoutine angezeigt werden soll
string	Flag für Generierung eines Stringfeldes

umw$	Umwandlung von Zahlen für die komprimierte Speicherung
x	Laufvariable für Schleifen
y$	Zwischenspeicher für t$ bei Vorschlag der einstelligen Kennung
zahl	Zahlentyp - CHR$, Integer, vorzeichenlose Integerzahl, einfach oder doppelt genaue Zahl
zaddkey	Zeilenzähler für ADDKEY-Befehle
zdelkey	Zeilenzähler für DELKEY-Befehle
zeile	Zeile, wo der Feldname für die Eingaberoutine angezeigt werden soll und für Generierung der RETURN-Zeilen
zfeld	Zähler für die Anzahl der generierten Felder
zfield	Zeilenzähler für FIELD-Befehle
zinput	Zeilenzähler für INPUT-Befehle
zleft	Zeilenzähler für LEFT$-Befehle
zlprint	Zeilenzähler für LPRINT-Befehle
zlset	Zeilenzähler für LSET-Befehle
zmax,zmin	Minimum und Maximum des jeweiligen Zahlentyps
zprint	Zeilenzähler für PRINT-Befehle
zrankspec	Zeilenzähler für RANKSPEC-Befehle
zreihe	Zähler der aktuell generierten Schlüsselreihe
zseekkey	Zeilenzähler für SEEKKEY-Befehle

Werden Variablen sowohl im generierten Programm als auch im Generator verwendet, sind sie beim generierten Programm beschrieben.

Alle übrigen Variablen sind Steuercodes für den Bildschirm oder Drucker - siehe die Kommentare im Programmlisting - Zeilen 18290 bis 19310.

Die einzelnen Programmroutinen sind zu finden in den Zeilen:

Zeilen	Programmteil
10 - 90	Hauptsteuerleiste Ansteuerung der Programmteile zur Initialisierung des Bildschirms und des Druckers, Aufbau des Begrüßungsbildes, Abfragen der Tastatur, Ansteuern des Generators und der Enderoutinen. In Zeile 80 wird das generierte Programm geladen und der Programmteil des Generators gelöscht.
100 - 1310	Generator mit folgenden Programmteilen:
100 - 370	Einleitung und Initialisierung für den Generator
380 - 950	Felder generieren
960 - 1200	Schlüssel generieren
1210 - 1270	Zahlenfelder verdichten und wieder aufbereiten
1280 - 1310	RETURN-Zeile berücksichtigen
10000 - 10580	Jetsam-Hauptprogramm mit folgenden Programmteilen:
10000 - 10090	Ansteurung der Programmteile für Eingabe, Verändern, Löschen, Anzeigen und Drucken mit Angabe des Programmmodus auf dem Bildschirm
10100 - 10170	Programmteil für die Eingabe von Daten von der Tastatur und Schreiben der Datendatei und Schlüsseldatei
10180 - 10230	Programmteil für das Verändern von Daten; zunächst werden die Daten eingelesen und sämtliche Schlüssel des alten Datensatzes werden gelöscht; dann werden die einzelnen Daten angezeigt mit der Möglichkeit, neue Werte einzugeben; nach Eingabe wird der überarbeitete Datensatz neu in die Jetsam-Datei eingefügt
10240 - 10310	Programmteil zum Löschen von Datensätzen; zunächst werden die zu löschenden Daten eingelesen, dann sämtliche Schlüssel zu dem Datensatz gelöscht.
10320 - 10510	Programmteil zum Anzeigen von Datensätzen auf

	dem Bildschirm
10520 - 10580	Programmteil zum Ausdrucken von Datensätzen auf dem Drucker
10590 - 11690	Hilfs- und Unterroutinen für die Jetsam-Verarbeitung
10590 - 10630	Einlesen eines bestimmten Zeichens von der Tastatur
10640 - 10670	Jetsam-Kommando GET
10680 - 10730	Jetsam-Kommando PUT
10740 - 10800	Jetsam-Funktion SEEKNEXT
10810 - 10870	Jetsam-Funktion SEEKPREV
10880 - 10920	Jetsam-Funktion SEEKSET
10930 - 11030	Jetsam-Fehlerroutinen
11040 - 11090	Jetsam-Fehlerroutinen für Mehrbenutzer-Systeme
11100 - 11150	Jetsam-Funktion RANKSPEC
11160 - 11190	Jetsam-Funktion CONSOLIDATE
11200 - 11250	Jetsam-Funktion SEEKRANK
11260 - 11310	Jetsam-Funktion SEEKKEY
11320 - 11370	Jetsam-Funktion DELKEY
11380 - 11440	Jetsam-Funktion ADDKEY
11450 - 11510	Jetsam-Funktion ADDREC
11520 - 11570	Ausgabe von Erläuterungen zur Bedienung des Programms bei Programmstart
11580 - 11600	Jetsam-Funktionen FETCHRANK, FETCHREC und FETCHKEY$
11610 - 11620	Bildschirm bis auf die Kopfzeilen löschen
11630 - 11690	Prüfung auf numerischen Inhalt bei eingegebenen Zahlenwerten - Exponentialdarstellung wird berücksichtigt
18000 - 18050	Programmende
18060 - 18180	Begrüßung und Initialisierung
18190 - 18270	Einlesen eines beliebigen Zeichens von der Tastatur
18280 - 18710	Sämtliche Steuercodes zur Bildschirmsteuerung
18720 - 19320	Sämtliche Steuercodes zur Druckersteuerung

Ab Zeile	wird generiert
20000	Satzbeschreibung - **FIELD**
21000	Eingabe - **INPUT**
21200	Datensatz füllen - **LSET**
21400	Schlüssel setzen - ggf. mit **MKI$, MKU$**
21600	Schlüssel einfügen - Aufruf **ADDKEY**
22000	Ausgabe auf Bildschirm - **PRINT**
	Die hier generierten Befehle können nach Bedarf geändert werden, falls z.B. mehrere Datenfelder in einer Bildschirmzeile angezeigt werden sollen etc. Wichtig ist, den Zeilenzähler **zz** richtig zu setzen.
23000	Ausgabe auf Drucker - **LPRINT**
	Die hier generierten Befehle können nach Bedarf geändert werden, falls z.B. mehrere Datenfelder in einer Druckzeile ausgegeben werden sollen etc. Wichtig ist, den Zeilenzähler **zz** richtig zu setzen.
25000	Eindeutige Schlüssel je Schlüsselreihe - **RANKSPEC**
26000	Schlüssel suchen - Aufruf **SEEKKEY**
27000	Schlüssel löschen - Aufurf **DELKEY**

Turbo-Pascal

Mittlerweile gibt es eine ganze Reihe von Joyce-Anwendern, die mit Turbo-Pascal arbeiten. Diesen bieten sich einige Vorteile, die Pascal gegenüber anderen Programmiersprachen auszeichnet.
Der Hauptvorteil von Turbo-Pascal ist, daß der eingegebene Programmtext automatisch in schnellen Z80-Code compiliert wird. Das Gegenstück zum Compiler ist der Interpreter, der sich Befehl für Befehl vom Programmtext holt und ihn dann auszuführen versucht (so beim mitgelieferten Mallard-80 BASIC oder bei LOGO). Turbo-Pascal hingegen übersetzt das Programm und sucht schon während des Übersetzens nach Fehlern und meldet sie. Dadurch wird das Programm nicht nur schnell, es kommt auch während der Ausführung seltener zu Abbrüchen.
Aber selbst ungewollte Programmabbrüche z.B. durch falsche Eingaben lassen sich durch entsprechende Programmierung abfangen.
Der größte Vorteil der in Pascal erstellten Programme besteht in ihrer Struktur. Als gegenteiliges Beispiel sei hier auf längere in BASIC geschriebene Programme hingewiesen. Bei ihnen wird der Text trotz guter Vorsätze zum Schluß meist durch häufige Zeilensprünge und Sprünge in Unterprogramme unübersichtlich, zumindest für nicht in das Programm eingearbeitete Personen. Bei Turbo-Pascal hingegen wird man von Anfang an gezwungen, strukturiert zu programmieren, da man keine Zeilennummern verwendet. Vor dem eigentlichen Programm wird festlegt, welche Variablen, Konstanten, etc. verwendet werden sollen. Auch kann man Unterprogramme definieren, die während des Hauptprogramms durch einfache Angabe ihres Namens aufgerufen werden. Der Vorteil dabei liegt darin, daß die Prozeduren vor dem Hauptprogramm definiert werden und der Anwender somit schnell über die Möglichkeiten des Programms informiert wird und nicht mit einer unidentifizierbaren Zeilennummer, sondern mit dem Namen konfrontiert wird. Zum Vergleich führe ich jetzt das schon von LOGO bekannte Programm zur

Dreiecksberechnung in der Turbo-Pascal Version an. Zwar werden an diesem Programm die tatsächlichen Vorzüge Turbo-Pascals nicht deutlich, doch wird hier ein ungefährer Einblick in die Art des Programmierens mit Pascal gewährt:

```
program Dreieck;

var seitea, seiteb, seitec: real;

begin
 clrscr;
 write('Seite a ?'); readln(Seitea);
 write('Seite b ?'); readln(Seiteb);
 seitec := sqrt (sqr (Seitea) + sqr (Seiteb));
 writeln('Seite c = ',Seitec:10:4);
end.
```

Zum Programm: Zunächst werden die Variablen definiert. Dann wird nach Ankündigung des Hauptprogramms mit **begin** der Bildschirm durch **clrscr** gelöscht. Danach wird eine Eingabe für die Seitenlänge von **a** und **b** verlangt. Seite c wird ausgehend von Pythagoras (a↑2+b↑2=c↑2 bzw. c=wurzel (a↑2+b↑2)) berechnet. Beim Beispiel ist zu beachten, daß die Wurzelfunktion durch **sqrt** dargestellt wird. Das bekannte **sqr** stellt unter Turbo-Pascal eine Quadratfunktion dar. Zum Schluß wird das Ergebnis auf dem Bildschirm ausgegeben, wobei ein Format von 10 Vorkomma- und 4 Nachkommastellen verwendet wird.
Zur weiteren Information über die Arbeit mit Pascal:

Turbo-Pascal systematisch
B.Ciric/D.Thies
Tewi München 1987

MI-C / C-Compiler

C ist eine der jüngsten Sprachen. Sie entstand als "Abfallprodukt" aus der Entwicklung eines Betriebssystems für Unixrechner. BCPL, das keine Strukturierung von Daten zuließ, wurde zu diesem Zweck weiterentwickelt, wobei gerade auf mögliche Vielfalt von Datentypen Rücksicht genommen

wurde. Im Endeffekt stellte C dann ein mächtiges Werkzeug zur Programmierung dar. Wie alle jüngeren Programmiersprachen arbeitet C mit einem Compiler, der einen maschinennahen Code erzeugt. Mittlerweile steht dem Joyce-User auch dieser Renner unter den Programmiersprachen zur Verfügung, der bisher in erster Linie MS-Dos oder UNIX Rechnern vorbehalten war.
Zwar gab es schon seit einiger Zeit einige C-Versionen, die sich als lauffähig auf dem Joyce ausgaben, doch entweder waren sie nicht kompatibel zu anderen CP/M Systemen (Arnor) oder verfügten nicht über den Standard nach Kernighan & Ritchie. Diesem Mißstand trat eine deutsche(!) Entwicklung entgegen, die sich auf ihrem Sektor absolut durchgesetzt hat und auch in den verschiedensten Fachzeitschriften schon vor Anpassung an den Joyce (CHIP 7/84; c t 2/87; CPC 5/86, 2/87) lobenswerte Erwähnung fand. Für Entwicklung und Vertrieb dieses MI-C Compilers zeichnet die Fa.:

H.Rose EDV
Buersche Str.43
4390 Gladbeck

Hier soll auch gleich einer der ersten Vorteile dieses Compilers Erwähnung finden: Entwicklung und Vertrieb in Deutschland gewährleisten ein verständliches (und zudem umfangreiches) Handbuch in deutscher Sprache und eventuell auftretende Schwierigkeiten können durch ein kurzes Telefonat kompetent und rasch beseitigt werden. Der Preis (ca. 450,- DM; je nach Ausstattung) ist dem Lieferumfang angemessen. Der Anwender erwirbt hier ein Werkzeug des professionellen Bereichs, daß als einziges u.a. den vollständigen Standard-C Sprachumfang und Bibliotheksfunktionen nach Kernighan & Ritchie sowie Fließkommadatentypen und entsprechende Arithmetikfunktionen bietet. Erwähnenswert ist hier auch das Fehlerprotokoll, das als Datei angelegt oder über Monitor ausgegeben werden kann. Selbstverständlich sind die Meldungen in deutscher Sprache gehalten (wahlw. engl.) und der Fehler wird genau bezeichnet.
Der MI-C kann auch in gewissem Maße Fehler selbst

berichtigen, was sonst nur von Compilern für Großcomputer geboten wird. Eine Tracefunktion, die die meisten Sprachen vermissen lassen (vergl. allerdings trace unter Logo) zeigt nach Aktivierung alle Funktionsaufrufe und Werte der ablaufsteuernden Variablen. Der Anwender kann so den Programmablauf lückenlos nachvollziehen.
Die Grenzen des Compilers sind für den "normalen" Anwender kaum zu erreichen. So können auf jeder Programmzeile 159 Zeichen verwendet werden, in jedem Programmteil können mindestens 300 verschiedene externe, im kompletten Programm unbegrenzt viele Symbole Verwendung finden und bei einem Funktionsaufruf verkraftet der Compiler 40 Parameter.
Die Geschwindigkeit des Compilers ist enorm. Kleinere Programme werden in Sekundenschnelle bearbeitet. Da Assembler-Quellcode erzeugt wird, können kundige Anwender zusätzlich noch Maschinencode-Routinen in ein C-Programm einbinden.
Mit Erwerb des C-Compilers eröffnen sich den Joyce-Usern zusätzliche Möglichkeiten, da dieser "Romfähig" ist, was heißt, daß er 'tools' zur Eprom-Erzeugung enthält. Der Joycer kann jetzt Programme in C schreiben, die via Compiler an einen "Eprombrenner" (etwa den mc-Allprog) geschickt werden, der sie eingesteckten PALs, EPROMs oder PROMs zur Steuerung eines Z80 oder 8051 mit auf den Weg gibt.
(Wer sich näher über den mc-Allprog informieren möchte, sei auf die "mc" 6 & 7 1987 verwiesen. Dort finden sich auf 20 Seiten Bauanleitung und Beschreibung der Hard- und Software). Der Vollständigkeit halber nun nochmal das Dreiecksprogramm in C. Die Vorteile dieser Sprache gegenüber BASIC sind ähnlich wie bei Pascal und können an diesem kleinen Programm kaum demonstriert werden.
(Das Handbuch zum MI-C Compiler gibt übrigens Hilfen zu C)

```
double a, b;
main ()
{printf ("Geben Sie Seite a ein: "); (scanf "% lf", & a);
printf ("\n Geben Sie Seite b ein: "); (scanf "% lf" , & b);
printf ("\n die Seite c = % 8.3f", sqrt(a*a+b*b));}
```

Mit **double** wird zunächst die Gleitkommaverarbeitung signalisiert, danach die Variablen definiert. Die nächsten zwei

Zeilen fordern über Monitor die Eingabe des Wertes an und ordnen ihn der Variablen zu. Der Backslash \n bringt die Ausgabe auf die nächste Zeile. Das l von lf läßt mit doppelter Genauigkeit arbeiten und f signalisiert das Gleitkomma. (e wäre exponential) 8.3 Gibt die Länge und Anzahl der Nachkommastellen an.

Wer sich weiter über C informieren möchte, sei auf folgendes Buch verwiesen:

"Programmieren in C"
Kernighan Ritchie
München/Wien 1983 (Hanser Verlag)

Programmierhilfen und Interna des Joyce

Auf den folgenden Seiten werden sämtliche Codes zur Steuerung des Bildschirms und des Druckers ausführlich erklärt.
Dem Anwender sollen damit entscheidende Hilfen bei der Programmierung an die Hand gegeben werden, um die Bedienung der Peripherie zu vereinfachen.
Dem etwas versierteren Programmierer werden außerdem kleine Routinen gezeigt, um die Möglichkeiten des XBIOS-Systems (eXtended Basic Input/Output System) auszuschöpfen, sowie gezielte Einstellungen des Systems über den SCB (System-Control-Block) vorzunehmen.
Weiterhin wird der Aufbau des Directory erklärt und individuelle Manipulationsmöglichkeiten des Inhaltsverzeichnisses der Diskette über einen Diskettenmonitor (z.B. DU.COM) aufgezeigt.

Steuercode-Tabelle für den Bildschirm

CHR$(7) läßt den eingebauten Signalgeber einen Ton von sich geben.

CHR$(27)+"0" Hiermit wird die Statuszeile ausgeschaltet, so daß sie für andere Zwecke nutzbar wird. Die Statuszeile ist die unterste Zeile des Bildschirms, in der beim JOYCE mit einem Laufwerk das Laufwerk angezeigt wird und die Auskunft über den aktuellen Druckerstatus gibt (aufgerufen durch die [PTR]-Taste). Außerdem werden bei Bedarf CP/M-Fehlermeldungen in der Statuszeile angezeigt (z.B. **B: Laufwerk nicht bereit - Wiederholen, Ignorieren oder Abbrechen**). Wenn nun diese Zeile ausgeschaltet wird, kann sie auch durch normale **PRINT**-Kommandos genutzt werden. Die CP/M-Fehlermeldungen erscheinen dann mit der übrigen Bildschirmausgabe (an der aktuellen Stelle des Cursors), wo sie jedoch nach eventueller Eingabe (**W**, I oder **A**) nicht wieder gelöscht werden.

CHR$(27)+"1"** schaltet die Statuszeile wieder ein und macht alle Einstellungen von **CHR$(27)+"0" rückgängig. Ein weiterer Effekt ist, daß ein eventuell bestehendes Fenster (s. **CHR$(27)+"X"**) aufgehoben wird.

CHR$(27)+"2"+CHR$(zeichensatz) schaltet den eingestellten Zeichensatz auf den durch **zeichensatz** gekennzeichneten um. Die Möglichkeiten sind folgende:

zeichensatz =

0 Amerikanisch
1 Französisch
2 Deutsch (Normaleinstellung)
3 Englisch
4 Dänisch
5 Schwedisch
6 Italienisch
7 Spanisch
8 Japanisch

Die Unterschiede liegen (vom deutschen ausgehend) bei folgenden Zeichen: #, $, §, Ä, Ö, Ü, †, `, æ, Ø.

CHR$(27)+"A" bewegt den Cursor eine Zeile nach oben. Wenn er sich schon in der obersten Zeile befindet, hat diese Escape-Folge keine Bedeutung.

CHR$(27)+"B" bewegt den Cursor eine Zeile nach unten. Wenn er sich schon in der untersten Zeile befindet, hat diese Escape-Folge keine Bedeutung.

CHR$(27)+"C" bewegt den Cursor eine Spalte nach rechts. Wenn er sich schon in der ersten Spalte befindet, hat diese Escape-Folge keine Bedeutung.

CHR$(27)+"D" bewegt den Cursor eine Spalte nach links. Wenn er sich schon in Spalte 90 befindet, bleibt die Escape-Folge ohne Wirkung.

CHR$(27)+"E" löscht das aktuelle Fenster. Der Cursor bleibt an seiner aktuellen Position.

CHR$(27)+"H" rückt den Cursor in die linke obere Fensterecke.

CHR$(27)+"I" setzt den Cursor eine Zeile höher. Ist der Cursor schon in der obersten Zeile, rollt das Bild eine Zeile nach unten.

CHR$(27)+"J" löscht den Bildschirm ab der aktuellen Position des Cursors.

CHR$(27)+"K" löscht den Rest der aktuellen Zeile ab der Position des Cursors.

CHR$(27)+"L" fügt eine Zeile ein. Die aktuelle und alle darunter befindlichen Zeilen rücken eine Zeile nach unten, so daß eine Leerzeile entsteht. Der Cursor bleibt an seiner aktuellen Position.

CHR$(27)+"M" löscht die aktuelle Zeile. Alle darunter befindlichen rücken eine Zeile nach oben. Der Cursor bleibt an seiner aktuellen Position.

CHR$(27)+"N" löscht das Zeichen, das sich an der Stelle des Cursors befindet. Alle rechtseitigen Zeichen rücken ein Zeichen nach links. Die Wirkung dieser Escape-Folge ähnelt also der [DEL→]-Taste.

CHR$(27)+"X"+CHR$(32+zeile)+CHR$(32+spalte)+CHR$(31+hoehe)+CHR$(31+breite) grenzt ein Fenster auf dem Bildschirm ein. Das Fenster ist sozusagen ein zusätzlicher verkleinerter Bildschirm des restlichen Monitors, da um das Fenster herum der Bildschirminhalt erhalten bleibt, solange nicht gescrollt wird. **zeile** gibt die obere Begrenzungszeile des Fensters an. **spalte** gibt die linke Begrenzungsspalte an, **hoehe** wie viele Zeilen das Fenster haben soll und **breite** nennt die Fensterbreite in Zeichen. Der Cursor wird an eine Stelle innerhalb des Fensters gesetzt. Um das Fenster ohne große Probleme wieder auf den ganzen Bildschirm auszuweiten, benutzt man am besten **CHR$(27)+"1"**, was diesen zusätzlichen Effekt hervorruft.

CHR$(27)+"Y"+CHR$(32+zeile)+CHR$(32+spalte) rückt den Cursor an eine bestimmte Position auf dem Bildschirm. **zeile** gibt die Zeile an, **spalte** die Spalte der Position, auf die der Cursor gesetzt werden soll. Wenn die bezeichnete Position außerhalb des Bildschirms liegt, wird der Cursor an die nächste innerhalb des Bildschirms liegende Position gesetzt.

CHR$(27)+"b"+CHR$(farbe) tauscht die Hintergrund- und Schriftfarbe. Für **farbe** nimmt man am besten entweder 0 oder 9. 0 setzt den Bildschirmhintergrund dunkel und die Schriftfarbe hell, 9 den Bildschirmhintergrund hell und die Schriftfarbe dunkel. Wenn die Schriftfarbe dunkel ist, werden z.B. Hi-Res Grafiken schärfer, da sich zwei Pixel nebeneinander nicht verstärken.

CHR$(27)+"c"+CHR$(farbe) wie **CHR$(27)+"b"+CHR$(farbe)**

CHR$(27)+"d" löscht das aktuelle Fenster vom Beginn bis zur aktuellen Position des Cursors. Der Cursor bleibt dabei an seiner Position.

CHR$(27)+"e" macht den Cursor wieder sichtbar.

CHR$(27)+"f" macht den Cursor unsichtbar.

CHR$(27)+"j" speichert die aktuelle Cursorposition.

CHR$(27)+"k"** setzt den Cursor auf die zuletzt durch **CHR$(27)+"j" gespeicherte Position.

CHR$(27)+"l"** löscht die aktuelle Zeile. Im Gegensatz zu **CHR$(27)+"M" rücken die darunter liegenden Zeilen nicht nach oben nach, so daß eine Leerzeile entsteht.

CHR$(27)+"o" löscht die aktuelle Zeile bis zur Position des Cursors. Der Cursor bleibt danach an seiner Position

CHR$(27)+"p" stellt die inverse Darstellungsart ein. Dabei werden bei einem Zeichen jeweils die Pixelfarben vertauscht, das Zeichen wird also dunkel auf hellem Hintergrund geschrieben.

CHR$(27)+"q"** stellt die durch **CHR$(27)+"p" eingestellte inverse Darstellungsart wieder ab.

CHR$(27)+"r" stellt den Unterstreichungsmodus ein.

CHR$(27)+"u" stellt den Unterstreichungsmodus wieder ab.

CHR$(27)+"v" schaltet den Bildumlaufmodus ein, bei dem automatisch eine neue Zeile begonnen wird, wenn der Cursor am Ende einer Zeile angekommen ist. Die Zeichen, die nicht mehr auf die Zeile passen, werden an den Anfang der nächsten Zeile gesetzt.

CHR$(27)+"w" schaltet den Bildumlaufmodus aus, so daß der Cursor, wenn er am Ende einer Zeile angekommen ist, nicht in die nächste Zeile springt.

CHR$(27)+"x" verkleinert den Bildschirm auf 24 Zeilen und 80 Spalten.

CHR$(27)+"y" vergrößert den Bildschirm wieder auf Normalmaß (32 x 90).

Da jeder dieser Codes einen String darstellt, muß er auch entsprechend angewendet werden, z.B. durch `PRINT CHR$(27)+"f"` (Cursor abstellen). Auch kann man einen String bilden, der aus mehreren Codes zusammengesetzt ist, z.B. `cls$=CHR$(27)+"E"+CHR$(27)+"H":PRINT cls$` (Bildschirm löschen und Cursor nach links oben).

Steuercode-Tabelle für den Drucker

CHR$(8) Dieser Code bewirkt das Rücksetzen des Druckkopfs um ein Zeichen der eingestellen Schriftart. Bei Proportionalschrift rückt der Druckkopf um 1/12 Zoll nach links.

CHR$(10) Das Papier wird um eine Zeile mit dem eingestellten Zeilenabstand vorgeschoben.

CHR$(12) Der Drucker führt einen Seitenvorschub aus, d.h. das Papier wird bis zur ersten Zeile der folgenden Seite vorgeschoben.

CHR$(13) Der Druckkopf wird zum linken Rand geführt (Wagenrücklauf).

CHR$(24) Alle im Druckerpuffer befindlichen Daten werden gelöscht (Reset).

CHR$(27)+CHR$(10) Der automatische Zeilenvorschub wird eingestellt. Jeder Wagenrücklauf wird als Wagenrücklauf plus Zeilenvorschub ausgeführt.

CHR$(27)+CHR$(13) Der automatische Zeilenvorschub wird aufgehoben.

CHR$(27)+CHR$(15) 17 Zeichen / Zoll wird eingestellt.

CHR$(27)+CHR$(18) Rückkehr von 17 Zeichen / Zoll zu 10 Zeichen / Zoll.

CHR$(27)+"$" Als Papier wird Einzelblatt verwendet.

CHR$(27)+"-"+CHR$(0) Die Unterstreichung der Schrift wird aufgehoben.

CHR$(27)+"-"+CHR$(1) Die gegenwärtige Schrift wird unterstrichen gedruckt.

CHR$(27)+"4" Die gegenwärtige Schrift wird kursiv gedruckt.

CHR$(27)+"5" Rückkehr von Kursivschrift zu Normalschrift.

CHR$(27)+"8" Das Papierende wird ignoriert.

CHR$(27)+"9" Das Papierende wird angezeigt.

CHR$(27)+"§" Der Drucker wird auf seine Standardeinstellung zurückgesetzt.

CHR$(27)+"C"+CHR$(n) Die Seitenlänge wird auf n Zeilen des aktuellen Zeilenabstandes eingestellt.

CHR$(27)+"C"+CHR$(0)+CHR$(n) Die Seitenlänge wird auf n Zoll gesetzt.

CHR$(27)+"E" Die gegenwärtige Schrift wird fett gedruckt.

CHR$(27)+"H" Aufhebung der durch Fettdruck verstärkten Schrift.

CHR$(27)+"G" Die gegenwärtige Schrift wird durch Doppelanschlag verstärkt.

CHR$(27)+"H" Aufhebung der durch Doppelanschlag verstärkten Schrift.

CHR$(27)+"J"+CHR$(n) Das Papier wird um **n**/216 Zoll vorgeschoben (**n**=1..255).

CHR$(27)+"K"+CHR$(a)+CHR$(b) Es können Bit-Bilder mit normaler Dichte (niedrige Auflösung) gedruckt werden. a und b berechnen sich aus der Anzahl Bit-Bilder, die in eine Zeile gedruckt werden sollen:

```
a=bit_bild_anzahl MOD 256
b=INT(bit_bild_anzahl / 256)
```

Wenn der Drucker in den Grafik-Modus versetzt worden ist, können die Bit-Bilder jeweils mit **LPRINT CHR$(wert)** zu Papier gebracht werden (**wert**=0..255).
Eine Zeile mit normaler Dichte kann maximal 480 Bilder aufnehmen. Ist die Zeile fertig gedruckt, wird der Graphik-Modus wieder abgeschaltet. Vorher **WIDTH LPRINT 255** einstellen.

CHR$(27)+"L"+CHR$(a)+CHR$(b) Es können Bit-Bilder mit doppelter Dichte (hohe Auflösung) gedruckt werden. a und b berechnen sich aus der Anzahl Bit-Bilder, die in eine Zeile gedruckt werden sollen:

```
a=bit_bild_anzahl MOD 256
b=INT(bit_bild_anzahl / 256)
```

Wenn der Drucker in den Grafik-Modus versetzt worden ist, können die Bit-Bilder jeweils mit **LPRINT CHR$(wert)** zu Papier gebracht werden (**wert**=0..255).
Eine Zeile mit doppelter Dichte kann maximal 960 Bilder aufnehmen. Ist die Zeile fertig gedruckt, wird der Graphik-Modus wieder abgeschaltet. Vorher **WIDTH LPRINT 255** einstellen.

CHR$(27)+"M" 12 Zeichen / Zoll wird eingestellt.

CHR$(27)+"N"+CHR$(n) Am Ende des Papieres werden **n** Zeilen freigelassen (n=1..127).

CHR$(27)+"O"** Hebt **CHR$(27)+"N" auf.

CHR$(27)+"P" Rückkehr von 12 Zeichen / Zoll zu 10 Zeichen / Zoll.

CHR$(27)+"R"+CHR$(n) schaltet den eingestellten Zeichensatz auf den durch **n** gekennzeichneten um. Die Möglichkeiten sind folgende:

n =	0	Amerikanisch
	1	Französisch
	2	Deutsch (Normaleinstellung)
	3	Englisch
	4	Dänisch
	5	Schwedisch
	6	Italienisch
	7	Spanisch
	8	Japanisch

Die Unterschiede liegen (vom deutschen ausgehend) bei folgenden Zeichen: #, $, §, Ä, Ö, Ü, ↑, ` , æ, 0.

CHR$(27)+"S"+CHR$(0) Die gegenwärtige Schrift wird hochgestellt gedruckt.

CHR$(27)+"S"+CHR$(1) Die gegenwärtige Schrift wird tiefgestellt gedruckt.

CHR$(27)+"T" Aufhebung von Hoch- oder Tiefstellung der Schrift.

CHR$(27)+"W"+CHR$(0) Rückkehr von doppelter Schriftgröße zur normalen Größe.

CHR$(27)+"W"+CHR$(1) Die gegenwärtige Schrift wird doppelt breit gedruckt.

CHR$(27)+"X" Die Null wird mit Schrägstrich gedruckt (Ø).

CHR$(27)+"c" Als Papier wird Endlospapier verwendet.

CHR$(27)+"d" Alle gegenwärtigen Einstellungen für den Drucker werden als Standardeinstellung festgesetzt.

CHR$(27)+"o" Die Null wird ohne Schrägstrich gedruckt (0).

CHR$(27)+"p"+CHR$(0) Rückkehr von Proportionalschrift zu 10 Zeichen / Zoll.

CHR$(27)+"p"+CHR$(1) Proportionalschrift wird eingestellt.

CHR$(27)+"x"+CHR$(0) Die Zeichen werden in Entwurfs-Qualität gedruckt.

CHR$(27)+"x"+CHR$(1) Die Zeichen werden in Korrespondenz-Qualität gedruckt.

Da jeder dieser Codes einen String darstellt, muß er auch entsprechend angewendet werden, z.B. durch `LPRINT CHR$(27)+"E"` (Text fett drucken). Auch kann man einen String bilden, der aus mehreren Stuercodes zusammengesetzt ist, z.B. `breit$= CHR$(27)+"M"+CHR$(27)+"W"+CHR$(1):LPRINT breit$` (12 Zeichen / Zoll doppelt breit drucken).

N ü t z l i c h e X B I O S - R o u t i n e n

Bildschirm zurücksetzen

Die folgende Routine ist die einfachste der XBIOS-Routinen, da keinerlei Parameter übergeben werden. Sie löscht den Bildschirm, setzt den Cursor in die linke obere Ecke des Bildschirms und schaltet den Cursor an.

```
10 MEMORY &HF4FF
20 FOR I=&HF500 TO &HF505
30   READ a$
40   POKE I,VAL("&H"+a$)
50 NEXT
60 adr=&HF500
70 CALL adr
80 END
90 DATA CD,5A,FC          :' CALL XBIOS
100 DATA C2,00            :' DW 00C2H
110 DATA C9               :' RET
```

Statuszeile ermitteln

Die folgende Routine ermittelt, ob die Statuszeile ein- oder ausgeschaltet ist.

```
10 MEMORY &HF4FF
20 FOR I=&HF500 TO &HF510
30   READ a$
40   POKE I,VAL("&H"+a$)
50 NEXT
60 adr=&HF500
90 CALL adr
91 PRINT "Die Statuszeile ist ";
93 IF PEEK(&HF511)=255 THEN PRINT "aus"; ELSE PRINT "ein";
94 PRINT "geschaltet."
100 DATA 3E,00            :' LD A,00H         ; Speicherplatz
110 DATA 32,11,F5         :' LD (0F511H),A    ; initialisieren
120 DATA CD,5A,FC         :' CALL XBIOS       ; und prüfen...
130 DATA C5,00            :' DW 00C5H
140 DATA C0               :' RET NZ           ; Statuszeile ist an
150 DATA 3E,FF            :' LD A,0FFH        ; Speicherplatz
160 DATA 32,11,F5         :' LD (0F511H),A    ; mit 0FFH belegen
170 DATA C9               :' RET              ; und zurück
```

Tastatur umbelegen

Die folgende Routine weist einer Taste der Tastur ein neues Zeichen zu.

```
10 OPTION RUN
20 DIM a(6)
30 MEMORY &HF54F
40 adr=&HF550
50 FOR i=&HF550 TO &HF55B
60   READ a$
70   POKE I,VAL("&H"+a$)
```



```
80 NEXT
90 CALL adr
100 DATA 06,FF            :' LD B,0FFH   ; FF = neuen ASCII-Code
110 DATA 0E,07            :' LD C,07H    ; 07 = Taste
120 DATA 16,02            :' LD D,02H    ; 02 = Codierung (siehe unten)
130 DATA CD,5A,FC         :' CALL XBIOS
140 DATA D7,00            :' DW 00D7H
150 DATA C9               :' RET
```

Die Codierung:

Bit 0	Normale Tastenebene
Bit 1	SHIFT
Bit 2	ALT
Bit 3	SHIFT+ALT
Bit 4	EXTRA

Die Tastennumerierung entnehmen Sie bitte dem CP/M Plus Benutzer-Handbuch zum Joyce Anhang 1 Seite 7.

Tastatur abfangen

Mit dieser Routine kann man ein Zeichen von der Tastatur abfangen, so daß auch z.B. die [SHIFT]-Taste als solche erkannt werden kann. Das Programm läuft solange, bis die SPACE-Taste gedrückt wird.

```
10 MEMORY &HF54F
20 adr=&HF550
30 b=&HF55E                    ' Speicherstelle der er- : b=Status
40 c=&HF55D                    ' mittelten Parameter    : c=Tastennummer
45 cls$=CHR$(27)+"E"+CHR$(27)+"H":PRINT cls$
50 FOR i=&HF550 TO &HF55C
60   READ a$
70   POKE I,VAL("&H"+a$)
80 NEXT
90 DATA CD,5A,FC        :' loop: CALL XBIOS   ; warten, bis Zeichen
100 DATA DA,00          :' DW 00DAH           ; von Tastatur anliegt
110 DATA D2,50,F5       :' JP NC,loop         ; kein Zeichen, dann loop
120 DATA ED,43,5D,F5    :' LD (0F55DH),BC     ; Tastencode abspeichern
130 DATA C9             :' RET
140 CALL adr
150 PRINT CHR$(27)+"HTaste:";PEEK (c)
160 PRINT "Status: ";
170 z=PEEK(b)
180 IF (z AND 128)=128 THEN PRINT "ALT  ";:'          Hier erfolgt die
190 IF (z AND 64)=64 THEN PRINT "SHIFT LOCK  ";:'     bitweise Dekodierung
200 IF (z AND 32)=32 THEN PRINT "SHIFT  ";:'          des in Register B
210 IF (z AND 16)=16 THEN PRINT "NUM LOCK  ";:'       übergebenen Status
220 IF (z AND 8)=8 THEN PRINT "Tastenwiederholung  ";
230 IF (z AND 4)=4 THEN PRINT "CAPS LOCK  ";
240 IF (z AND 2)=2 THEN PRINT "EXTRA  ";
250 PRINT SPC(50);:PRINT
260 IF PEEK(c)=47 THEN END
270 GOTO 140
```

Tastaturwiederholung einstellen

Mit dieser Routine kann man die Wiederholgeschwindigkeit der Tastatur festlegen. Sie besagt, wann das Wiederholen einer gedrückten Taste einsetzt und wie schnell sie danach wiederholt wird.

```
10 MEMORY &HF4FF
20 FOR i=&HF500 TO &HF509
30   READ a$
40   POKE i,VAL("&H"+a$)
50 NEXT
51 adr=&HF500
55 CALL adr
60 DATA 26,19            :' LD H,19H      ; Verzögerung für Einsetzen
70 DATA 2E,01            :' LD L,01H      ; Verzögerung bei Wiederholung
80 DATA CD,5A,FC         :' CALL XBIOS    ; Werte einstellen
90 DATA E0,00            :' DW 00E0H
100 DATA C9              :' RET           ; und fertig
```

Computertyp ermitteln

Die nächste Routine ermittelt, ob dieses Programm auf einem JOYCE oder CPC 6128 abläuft. Liefert das A-Register als Ergebnis 1, so läuft das Programm auf einem JOYCE, bei A=0 handelt es sich um einen CPC 6128.

```
10 MEMORY &HF4FF
20 FOR i=&HF500 TO &HF508
30   READ a$
40   POKE i,VAL("&H"+a$)
50 NEXT
60 adr=&HF500
70 CALL adr
80 PRINT "Dieses Programm läuft auf dem SCHNEIDER ";
90 IF PEEK(&HF509)=0 THEN PRINT "CPC-6128" ELSE PRINT "JOYCE"
100 DATA CD,5A,FC      :' CALL XBIOS     ; ermitteln
110 DATA E3,00         :' DW 00E3H
120 DATA 32,09,F5      :' LD (0F509H),A  ; Ergebnis sichern
130 DATA C9            :' RET            ; und fertig
```

Hardware-Erweiterungen ermitteln

Diese Routine informiert, wieviel Laufwerke angeschlossen sind, wieviele Speicherblöcke (zu je 16 Kbyte) zur Verfügung stehen und ob eine Schnittstelle angeschlossen ist. Das A-Register informiert über die angeschlossenen Laufwerke (A=0 ein Laufwerk, A=255 zwei Laufwerke), das B-Register enthält die Zahl der verfügbaren Speicherblocks zu je 16 KByte, das C-Register belegt, ob eine serielle Schnittstelle angeschlossen ist (C=0 keine Schnittstelle, C=255 Schnittstelle angeschlossen).

```
10 MEMORY &WF4FF
20 FOR i=&HF500 TO &HF50C
30   READ a$
40   POKE i,VAL("&H"+a$)
```



```
50 NEXT
60 adr=&HF500
70 CALL adr
80 PRINT "Angeschlossene Laufwerke: ";
90 IF PEEK(&HF50D)=0 THEN PRINT "eins" ELSE PRINT "zwei"
100 PRINT "Joyce Version: ";
110 PRINT PEEK(&HF50F)*16"K"
120 PRINT "Serielle Schnittstelle: ";
130 IF PEEK(&HF50E)=255 THEN PRINT "angeschlossen" ELSE PRINT "nicht vorhanden"
140 DATA CD,5A,FC      :' CALL XBIOS          ; Hardware-Erweiterungen
150 DATA E6,00         :' DW 00E6H              ; ermitteln
160 DATA 32,0D,F5      :' LD (0F50DH),A       ; und Ergebnisse ab-
170 DATA ED,43,0E,F5   :' LD (0F50EH),BC   ; speichern
180 DATA C9            :' RET                   ; und fertig
```

System-Control-Block (SCB)

Im SCB stehen manche für den Anwender nützliche Informationen. Sie können durch einen entsprechenden POKE geändert oder mit einem PEEK ausgelesen werden.

Adr.	Offset	Inhalt Hex	Inhalt Dez	Erklärung	
FBA1	05	31	49	BDOS-Versionsnummer	
FBA2	06	00	0	User Flags	
FBA3	07	00	0		
FBA4	08	00	0		
FBA5	09	00	0		
FBAC	10	00	0	Programm Error Return Code	
FBAD	11	00	0		
FBB6	1A	58	88	Anzahl der Bildschirmspalten	
FBB7	1B	00	0	Augenblickliche Bildschirmzeile	
FBB8	1C	1E	30	Seitenlänge Bildschirm	
FBBE	22	00	0	Console Input-Flag	für die folgenden Flags
FBBF	23	80	128		gilt:
FBC0	24	00	0	Console Output-Flag	Bit gesetzt:
FBC1	25	80	128		4: Centronics (CEN)
					5: Seriel Input Output (SIO)
FBC2	26	00	0	Auxiliary Input-Flag	6: Drucker (LPT)
FBC3	27	00	0		7: Console (CRT)
FBC4	28	00	0	Auxiliary Output-Flag	
FBC5	29	00	0		
FBC6	2A	00	0	Printer Output-Flag	
FBC7	2B	40	64		
FBC8	2C	00	0	Page Modus	

Adr.	Offset	Inhalt Hex	Inhalt Dez	Erklärung
FBCA	2E	00	0	Flag für CTRL-H = DEL
FBCB	2F	FF	255	Flag für DEL = CTRL-H
FBCF	33	00	0	Konsolen-Modus (s. BDOS-Aufruf 109)
FBD0	34	00	0	
FBD3	37	24	36	Ausgabe-Delimiter (s. BDOS-Aufruf 110)
FBD4	38	00	0	Flag für parallele Druckerausgabe (z.B. mit CTRL-P unter CP/M Plus): 0=aus, 1=ein
FBD8	3C	DE	222	Derzeitige DMA-Adresse
FBD9	3D	76	118	
FBDA	3E	01	1	Derzeitiges Bezugslaufwerk
FBED	44	00	0	Derzeitige Benutzernummer
FBE6	4A	01	1	BDOS Multi-Sector Count (s. BDOS-Aufruf 44)
FBE7	4B	00	0	BDOS Error Mode (s. BDOS-Aufruf 45) z.B. 255: wenn ein BDOS-Error auftritt, wird das laufende Programm normalerweise abgebrochen, bei 255 wird jedoch zum laufenden Programm zurückgekehrt (unter BASIC bei Diskettenarbeit sehr nützlich)
FBE8	4C	00	0	Suchpfad für max. 4 Laufwerke (s. SETDEF)
FBE9	4D	FF	255	(0=A, 1=B, 2=C,..,15=P)
FBEA	4E	FF	255	
FBEB	4F	FF	255	
FBEC	50	00	0	Laufwerk für temporäre Dateien
FBED	51	00	0	Laufwerk, an dem zuletzt ein Fehler auftrat
FBF3	57	80	128	Flag für die Ausgabeart der BDOS-Fehlermeldungen 0=kurz, 128 (80H)=ausführlich
FBF4	58	3E	62	Tage seit dem 1. Januar 1978
FBF5	59	0E	14	
FBF6	5A	15	21	Stunden im BCD-Format (Hier: 21 Uhr,
FBF7	5B	14	20	Minuten im BCD-Format 20 Minuten,
FBF8	5C	32	50	Sekunden im BCD-Format und 50 Sekunden)
FBF9	5D	00	0	Basisadresse des Common-Speicherbereiches
FBFA	5E	C0	192	Hier: 0C000H

O U T ' e r

```
OUT 245,x   'verwandelt Bildschirm in Bitmuster
    245,91  'wandelt Bildschirm wieder zurück...
    246,x   'legt Punkt 0,0 auf Punkt 0,x
```

Beispiel:

```
10 PRINT CHR$(27)+"E"+CHR$(27)+"H"+CHR$(27)+"f"
20 PRINT SPC(35)"JOYCE-BILDSCHIRMTEST"
30 FOR I=0 TO 255:OUT 246,1:FOR U=1 TO 20:NEXT
40 NEXT:GOTO 20
```



```
OUT 248,1   'Kaltstart
      ,7    'Bildschirm ein
      ,8    'Bildschirm aus
      ,9    'Diskettenmotor ein
      ,10   'Diskettenmotor aus
      ,11   'Dauerpieps ein
      ,12   'Dauerpieps aus
```

P O K E ' s (u n t e r B A S I C)

POKE	28000,0	(63)	BASIC-Befehle werden im Direktmodus nicht angenommen	
	8793,234	(239)	Bildschirmausgabe wird auf Drucker umgeleitet	
	8793,241	(239)	Bildschirmausgabe wird unterdrückt	
	&HFBF6,VAL("&H"+"STUNDEN")		Stellt die Stunden der JOYCE-UHR	Adressen
	&HFBF7,VAL("&H"+"MINUTEN")		Stellt die Minuten der JOYCE-UHR	im
	&HFBF8,VAL("&H"+"SEKUNDEN")		Stellt die Sekunden der JOYCE-UHR	SCB

Entsprechend kann mit `PRINT HEX$(PEEK(adr))` die Uhrzeit ausgelesen werden, ein spezielles Maschinenprogramm entfällt also.

S o n s t i g e s

Verfahren, um **listgeschützte** BASIC-Programme **listbar** zu machen:

```
LOAD "DATEI                'Datei laden
OPEN "O",#1,"M:PASS.OFF"   'In M: Datei einrichten
MERGE "M:PASS.OFF"         'Datei aus M: dazuladen

LIST                       'Nun ist das Programm ungeschützt
```

Bildschirmfenster können mit **PRINT CHR$(27)+"1"** unter BASIC und mit ↑Ä1 (↑Ä=[EXIT]-Taste) unter CP/M aufgehoben werden.

PRINT CHR$(27)+"b"+CHR$(9) tauscht Hintergrund- und Schriftfarbe.

? ist gleichwertig mit **PRINT**

```
a=0:CALL a      'gleichwertig mit SYSTEM
a=&H100:CALL a  'Neustart BASIC
```

Da die BASIC-Funktion **FIND$** mit steigender Zahl an Dateien sehr langsam wird, ist es angebracht, sich selbst eine eigene Routine in Assembler zu schreiben. Ein mögliches Beispiel sieht folgendermaßen aus:

```
10 MEMORY &HF54F:RESTORE 360:FOR i=&HF550 TO &HF5DD:READ a$:POKE i,VAL("&H"+a$):NEXT
20 erst%=&HF550:dann%=&HF56C
30 DIM z(260),datei$(260):OPTION NOT TAB
40 DEF FNlocate$(x,y)=CHR$(27)+"Y"+CHR$(32+x)+CHR$(32+y)
50 PRINT CHR$(27)+"E"
60 j=0
70 PRINT CHR$(27)+"f"+FNlocate$(3,67);"Gefundene Dateien: ";
```

```
80 CALL erst%
90 WHILE PEEK(&H57)<>&HFF
100 j=j+1
110 datei$(j)=""
120 PRINT FNlocate$(3,85) USING "####";j
130 FOR i=&H3B TO &H42+3:datei$(j)=datei$(j)+CHR$(PEEK(i)):NEXT
140 CALL dann%
150 WEND
160 PRINT FNlocate$(4,67);"Sortiere Dateien ":PRINT:PRINT
170 p=1:s(1)=0:s(2)=j                                    'QUICKSORT-Verfahren
180 l=s(p):r=s(p+1):p=p-2
190 i=l:j1=r
200 d=(l+r)/2:g$=datei$(d)
210 IF datei$(i)>=g$ THEN 230
220 i=i+1:GOTO 290
230 IF datei$(j1)<=g$ THEN 250
240 j1=j1-1:GOTO 290
250 IF i>j1 THEN 290
260 SWAP datei$(i),datei$(j1)
270 i=i+1
280 j1=j1-1
290 IF i<=j1 THEN 210
300 IF i>=r THEN 320
310 p=p+2:s(p)=i:s(p+1)=r
320 r=j1
330 IF l<r THEN 190
340 IF p<>-1 THEN 180
350 FOR i=1 TO j:PRINT LEFT$(datei$(i),8)+"."+RIGHT$(datei$(i),3);" ";:NEXT:PRINT CHR$(27
)+"e":END
360 DATA 3E,FE,32,E7,FB,0E,1A,11,80,00,CD,05,00,0E,11,11
370 DATA BE,F5,CD,05,00,32,57,00,CD,7E,F5,C9,3A,57,80,FE
380 DATA FF,C8,0E,12,CD,05,00,32,57,00,CD,7E,F5,C9,3A,57
390 DATA 00,FE,00,C2,8C,F5,11,81,00,C3,83,F5,3A,57,00,FE
400 DATA 01,C2,9A,F5,11,A1,00,C3,83,F5,3A,57,00,FE,02,C2
410 DATA AB,F5,11,C1,00,C3,83,F5,3A,57,00,FE,03,C2,80,F5
420 DATA 11,E1,00,62,6B,11,3B,00,01,1a,00,ED,80,C9,00,3F
430 DATA 3F,3F,3F,3F,3F,3F,3F,3F,3F,00,00,00,00,00,00,00
440 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00
```

Dieses Beispielprogramm liest alle auf der Diskette befindlichen Dateinamen ein und gibt sie sortiert auf dem Bildschirm aus. Kernstück des Programm ist dabei die Assembler-Routine, die das heraussuchen der Dateinamen übernimmt. Wenn Sie nur bestimmt Dateien herausgesucht haben möchten, müssen Sie die 11 ?-Wildcards (3F's in Zeile 420/430) gegen Ihre Dateispezifikation austauschen.
Der dokumentierte Assembler-Quellcode befindet sich auf der zu dem Buch erhältlichen Diskette.

Aufbau des Directories

Jede angefangenen 16 KByte einer Datei haben ihren eigenen Directoryeintrag auf der Diskette. Beim Standard JOYCE-Format befinden sich die Directoryeinträge auf Spur 1 Sektor 0-9. Der Eintrag ist wie folgt aufgebaut:

```
00 44 49 53 43 4B 49 54 20 43 4F 4D 00 00 00 38  .DISCKIT COM...8
A5 A6 A7 A8 A9 AA AB 00 00 00 00 00 00 00 00 00  %&'()*+.........
```

Die Bedeutung der einzelnen Bytes:

0	Kennzeichnung für den Modus der Datei. Für die Werte 00H..0FH kennzeichnet es den entsprechenden USER-Bereich der Datei (0..15), für den Wert 0E5H kennzeichnet es die Datei als gelöscht.
1..11	Hier ist der Dateiname abgelegt. Die Bytes können beliebige Werte annehmen, es ist jedoch für den normalen Gebrauch ratsam, nur Großbuchstaben zu verwenden, da CP/M bei anderen Werten die Datei nicht laden (finden) kann.
12	Dieses Byte kennzeichnet die Nummer des Eintrages (Extent). Jede angefangenen 16 KByte benötigen ihren eigenen Directoryeintrag. Um diese Einträge auseinanderzuhalten, wird jeder Eintrag durch das 12. Byte nummeriert.
13..14	Für interne Zwecke reserviert
15	Anzahl der abgelegten Records im Extent (Byte 16..31). Byte 15 kann nur Werte zwischen 00H und 7FH annehmen.
16..31	Hier wird die Stelle auf der Diskette gekennzeichnet, an der die Datei abgelegt ist. Jedes einzelne Byte kennzeichnet einen Block, in dem ein Teil der Datei abgelegt ist.

Tips zur Verwendung:

Wenn man eine Datei versehentlich gelöscht hat, muß man nur den entsprechenden Eintrag im Directory suchen und Byte 0 (vor dem Dateinamen) von E5 (gelöscht) auf 00H oder den entsprechenden USER-Wert setzen.

Durch die Verwendung von Escape-Sequenzen im Dateinamen kann man beim Auflisten des Directories zum Beispiel den Bildschirm löschen oder den Cursor an/aus schalten, wie es der Simulator TOMAHAWK tut.

Aufbau des Directory-Labels

Das Directory-Label, das Informationen über die verwendete Diskette enthält, befindet sich wie die Directory-Einträge auf Spur 1. Pro Diskette kann nur ein Label verwendet werden, das folgendes Format hat:

```
20 42 45 52 4E 48 41 52 44 47 52 41 81 24 00 00   BERNHARDGRA.$..
6C 63 62 61 60 67 66 65 00 00 00 06 12 07 00 30   lcba gfe.......0
```

Die Bedeutung der einzelnen Bytes:

0	Beim JOYCE ist dieses Byte zur Erkennung des Directory-Labels immer auf 20H gesetzt.
1..11	Hier ist der Diskettenname eingetragen. Die Bytes können beliebige Werte annehmen. Es ist jedoch nicht ratsam, Bytes zu verwenden, deren Wert kleiner als 32 ist, da dies den Bildschirmaufbau unter CP/M stören könnte.
12	Dieses Byte behinhaltet weitere Informationen über die Dateien. Bit gesetzt: 0 Label existiert 4 Create-Timestamps aktiviert 5 Update-Timestamps aktiviert 6 Access-Timestamps aktiviert 7 Password-Schutz aktiviert
13..15	Reserviert
16..23	Diese Bytes beinhalten das Password für das Label. Da das Password verschlüsselt abgelegt ist, läßt es sich nicht einsehen. Im obigen Beispiel lautet das Password: ABCDEFGH.
24..27	Create- oder Access-Timestamps für das Label.
28..31	Update-Timestamps für das Label.

Tips zur Verwendung: Wenn man ein durch Password geschütztes Disketten-Label ändern möchte, braucht man nur die entsprechenden Bytes des Eintrages mit den neuen Buchstaben zu vertauschen, wobei man sich um das Password nicht zu kümmern braucht.
Um das Label komplett zu löschen, muß man Byte 0 des Eintrages auf den Wert 0E5H setzen.

Aufbau des Datei-Labels

Im Datei-Label, das sich ebenfalls im Directory auf Spur 1 der Diskette befindet, sind weitere Informationen über eine Datei abgelegt (vollständiger Schutz gegen Zugriffe, Schutz gegen Schreiben und Schutz gegen Löschen durch Passwordabfrage). Das Datei-Label ist wie folgt aufgebaut:

```
10 42 41 53 49 43 20 20 20 43 4F 40 80 DA 00 00   .BASIC   COM.Z..
FA FA FA 9F 99 83 95 98 00 00 00 00 00 00 00 00   zzz............
```

Die Bedeutung der einzelnen Bytes:

0	Beim JOYCE ist dieses Byte zur Erkennung des Datei-Labels immer auf 10H gesetzt.
1..11	Dateiname der zu diesem Label gehörigen Datei
12	Kennzeichnung des Password-Schutzes Bit gesetzt: 4 Create-Timestamping aktiviert 5 Delete-Modus (Schutz nur gegen Löschen) 6 Write-Modus (Schutz nur gegen Schreiben) 7 Read-Modus (vollständiger Schutz gegen Zugriffe)
13..15	Für interne Zwecke reserviert
16..23	Hier befindet sich das verschlüsselte Password. Im Beispiel wurde als Password **JOYCE** benutzt.
24..31	Für interne Zwecke reserviert

Tips zur Verwendung: Wenn man den durch Password geschützten Modus einer Datei ohne Wissen des Passwords aufheben möchte, muß man Byte 0 des Eintrages auf den Wert 0E5H setzen.

Um die genannten Einträge auf Spur 1 zu ändern, benutze man am besten das Public Domain Programm **DU.COM** (Disk Utility). Es beinhaltet einen kompletten Diskettenmonitor, mit dem man bequem die Einträge der Diskette einsehen und manipulieren kann.

D i e " h a r t e J o y c e - W a r e "

Wer effektiv mit seinem Joyce arbeiten will, stößt früher oder später an Grenzen, die sich softwaremäßig kaum umgehen lassen. Den Besitzern des PCW 8256 geht es ein wenig schneller so, weil ihnen ein großzügiger RAM-Speicher fehlt. Dabei gehört gerade dieser Speicher zu den Dingen, die den Joyce bei seinen Anwendern so beliebt gemacht haben. Hätte dem Joyce von Anfang an eine derartig große Palette von Hardwareerweiterungen zur Verfügung gestanden, wie es im Moment der Fall ist, hätten sich die PCW's nicht nur bei Usern beliebt gemacht, die sie in erster Linie als Textsystem benutzen wollten. Mit den folgenden Kapiteln sollen nun die Hardware-Basteleien und -Erweiterungen erklärt werden, die ein effektiveres Arbeiten mit dem Joyce ermöglichen und ein völlig neues Spektrum von Einsatzmöglichkeiten der PCW eröffnen.

Die Erweiterung des Speichers

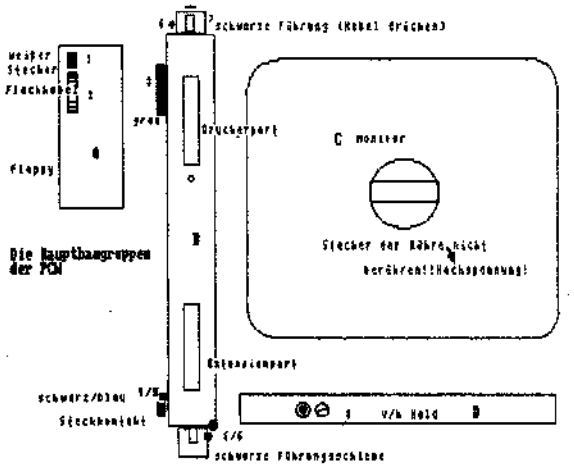
Anwender, die den PCW 8256 benutzen, können eigentlich gar nicht besser ca. 200,- DM in Hardware investieren, wenn sie sich die Speichererweiterung für den Joyce zulegen. Sie besteht aus 8 dynamischen NMOS Bausteinen, vorzugsweise mit der Typenbezeichnung 257. Bei Toshiba z.B. lautet die vollständige Typenbezeichnung: 41257C-15. (Der Autor arbeitet mit den billigeren D41256C-15 von Nec, wobei keine Probleme auftreten.)
Die Vorteile eines größeren Speichers liegen auf der Hand, und dies trifft erst Recht beim Joyce zu, denn Dateien, die in seinem Speicher (oder Laufwerk M:) abgelegt wurden, können wie von einer Diskette abgerufen werden. Das Laufwerk M: wird einfach als Standardlaufwerk angesprochen. (Für Neulinge: hinter das A> wird M: geschrieben und [RETURN] gedrückt.) Der erste Erfolg dieser Aktion ist, daß sich die Zugriffszeit beim Aufruf der Programme wesentlich verringert. (Bei 200 Einträgen unter Jetsam von 3.47 auf

3.00 Minuten) Größere Programme, die in Laufwerk A: einen Diskettenwechsel bei der Abarbeitung notwendig machen, können komplett in M: geholt und aufgerufen werden, ohne den Anwender als "Disk-Jockey" in Anspruch zu nehmen.

Wer keine Angst davor hat, einen Schraubenzieher in die Hand zu nehmen und derartig bewaffnet seinem Joyce entgegenzutreten, kann leicht seinen RAM-Speicher mit den oben erwähnten Bausteinen aufrüsten, wenn er sich an die nachstehende Anleitung hält. Zu diesem Thema wurden zwar schon einige Informationen veröffentlicht, aber die waren teils so widersprüchlich, daß sie eher verwirrten als weiterhalfen. Einige Anwender hatten dadurch Nachteile in Kauf zu nehmen, die sich zwar gegenüber dem erweiterten Speicher gering ausnahmen, aber es war doch ärgerlich, daß ein in den Speicher gelegtes Betriebssystem durch einen Neustart mit [SHIFT]+[EXTRA]+[EXIT] nicht gelöscht werden konnte. Nun aber zur Anleitung:
Netzstecker ziehen (!) und das Tastaturkabel trennen. Nun kann Joyce mit dem "Gesicht" nach unten auf eine weiche Unterlage (Sessel, Couch etc.) gelegt werden, so daß das Standbein dem Betrachter zugekehrt und die schöne Rückfront sichtbar ist. Auf die Rückfront sind sechs Pfeile gestanzt, die auf sechs Schrauben zeigen. Zwei befinden sich am oberen Rand (lange Schrauben), je eine unter dem Printer- und Expansionsport (kleine Schrauben) und zwei am unteren Rand des Monitors (dicke Schrauben). Zum Herausdrehen dieser Schrauben erweist sich ein langer Schraubenzieher als vorteilhaft. Nach erfolgreicher Entfernung läßt sich die rückwärtige Monitorabdeckung leicht nach oben abziehen und der Blick kann ungehindert auf die Hauptbaugruppen des Joyce fallen, (s. Skizze folgende Seite) dem Laufwerk (A), der Hauptplatine und Speicherträger (B), dem Monitor (C) und seiner Steuerplatine (D).
Im folgenden müssen nun die Kabel, die die Blechdose (B) mit den anderen Baugruppen verbindet, abgezogen werden.

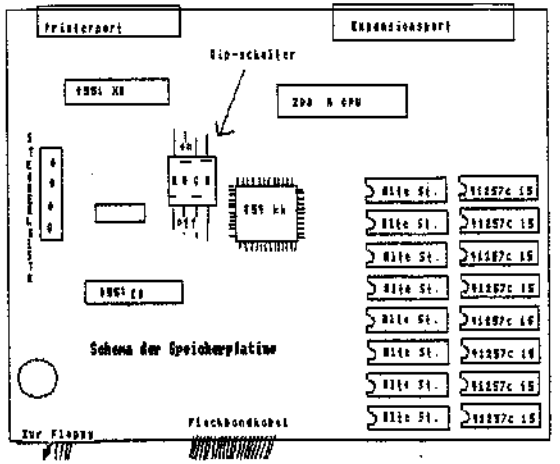
Die Stecker 1,2,4,5 sind in eine Plastikführung geschoben, die eine kleine Wölbung an den Steckern in sie einrasten läßt. Es ist ganz hilfreich, wenn die Plastikführung bei Zug

am Stecker mit einem spitzen Gegenstand leicht abgebogen wird. Der graue Stecker (3) sitzt erfahrungsgemäß sehr fest.



Es hilft hier weiter, wenn man immer im Wechsel die rechte und linke Längsseite ein wenig abhebelt. Die mit 6 gekennzeichneten drei schwarzen Massekabel werden am besten an der Mantelung des Blechkastens abgeschraubt. (Merken, in welche Löcher die Schrauben beim Einbau wieder hineinkommen!) Wurden alle Verbindungen gelöst, kann der Platinenträger nach Druck auf den schwarzen Hebel (7) an der Führung nach oben herausgezogen werden. Rund um den Rand der Blechumhüllung sitzen Schrauben, die jetzt herausgedreht werden müssen, um den Deckel abheben zu können. Ein kurzer Blick aufs Innere läßt gleich die acht in Reihe stehenden Speicherbausteine erkennen. (s. Skizze) Rechts daneben sind noch acht Steckplätze frei. Hier werden die neuerworbenen dyn. NMOS Steine eingesteckt. Die Aufschrift der Steine muß nach dem Einbau aus der gleichen Blickrichtung zu lesen sein, wie die der alten Steine. Ihre runde Einkerbung muß in die gleiche Richtung zeigen. (s. Skizze folgende Seite)
Eventuell müssen die Beinchen der neuen Bausteine noch ein

wenig zurechtgebogen werden, bevor sie in ihre Führungen passen. Auch wenn es beim Reindrücken ein wenig knirscht - sind die Beinchen einmal richtig in ihren Führungen, sollten die Bausteine auch fest angedrückt werden. (Nur nicht die Platine durchbrechen). Jetzt muß der Rechner noch so

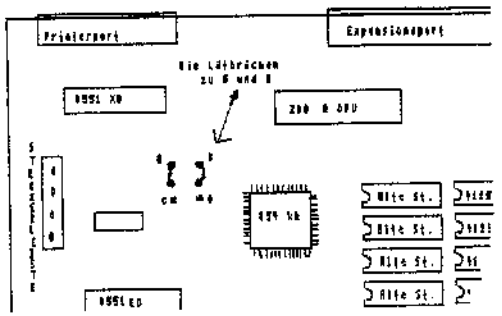


eingestellt werden, daß er auch weiß, daß ihm ein größerer Speicher zur Verfügung steht. Ungefähr in der Mitte der Platine (s.↑ auf der Skizze) befindet sich ein kleiner Schalter mit 4 Knöpfen, bei Rechnern älterer Baureihen eine Lötbrücke. Oben auf dem Schalter ist "on" zu lesen. Wird der Schlitz im Schaltknopf dorthin bewegt, ist die entsprechende Leiterbahn durchgeschaltet. Wurden alle Bausteine eingesetzt, so muß die Schalterstellung A=off B=on C=off D=on eingestellt sein, damit sich der Rechner mit 512 KB meldet.

A	B	C	D	
off	on	off	on	= 512 KB
on	off	off	on	= 256 KB
off	on	on	off	= 128 KB
on	off	on	off	= 128 KB

Diese Angaben nur zur Information, was bei anderen Schalterstellungen geschehen würde. Also, wenn schon Erweiterung,

dann auf 512 KB! Bei den Lötbrücken läuft es analog. Hier sind 4 Lötpunkte mit den Buchstaben A-D gekennzeichnet. Dazwischen liegen zwei Punkte, die Brücken zu B und C schlagen. Die Verbindung vom Mittelpunkt zu B wird jetzt getrennt (durchschneiden) und dafür nach A gelegt und angelötet (s. Skizze).



Nach erfolgter Operation kann der Umbau in umgekehrter Reihenfolge vonstatten gehen. Beim Zusammenbau der Blechkiste sollte man darauf achten, welche Löcher für die Masseanschlüsse frei bleiben sollten.
Wer jetzt allerdings noch ein wenig weiter an seiner Hardware basteln will, der kann die Rückwand seines Joyce gleich auflassen. Es wäre zum Beispiel nützlich, wenn man über den Helligkeitsregeler den Monitor sowohl ganz hell (gut für Lightpen) oder ganz dunkel (gut für Lebensdauer und Augen) schalten könnte. Zum entsprechenden Einbau eines 10K Helligkeitspotis mit 1,8K Vorwiderstand (auswechseln) sei auf den Artikel des Joyce Sonderheftes Nr. 2 (DMV-Verlag) verwiesen.

Das Zweitlaufwerk

Ein flüchtiger Blick auf die "Innereien" des Joyce zeigt, daß der Einbau des Zweitlaufwerks keine Schwierigkeit darstellt. Die Kabel, die zum Standardlaufwerk führen, sind zweifach vorhanden, wobei die zweite Ausführung nirgends angeschlossen ist. Diese "toten" Leitungen werden an ein Zweitlaufwerk gesteckt, wobei die Steckerformung keine falschen Anschlüsse zuläßt. An dieser Stelle gilt es jedoch zu erwägen, für welche Art von Zweitlaufwerk man sich entscheidet. Ästheten unter den Joycern werden wohl dazu tendieren, daß dem Design des Joyce angepaßte Original-Zweitlaufwerk einzubauen. Nach Entfernung der Abdeckklappe unter dem Standardlaufwerk kann es leicht von hinten in den freigewordenen Schacht gesetzt werden.
Praktiker haben die Qual der Wahl. Sie können zwischen einem 5¼- und einem 3½-Zoll Laufwerk wählen. Hier müssen jedoch die Kabel aus dem Joyce nach außen geführt werden, woran die Laufwerke dann im Freien hängend "brummen". Entscheidungen für das ein oder andere Laufwerk sind schwer zu fällen. Hier können die Zahlen für sich sprechen:
Original FD-2 3"-Laufwerk: 2x80 Spuren; 1MB unformatiert, 706 KB formatiert. Preis: ca. 350,-DM.
TEAK 3½" 1MB unformatiert, 706KB formatiert,
Preis: ca. 340,- DM.
TEAK 5¼" 1MB unformatiert, 706KB formatiert,
Preis: ca. 435,- DM.
Wahlweise kann bei diesem Laufwerk zwischen 40 und 80 Tracks umgeschaltet werden. Außerdem gibt es ein Programm dazu, welches beliebige Datenfiles (z.B. ASCII, Turbo Pascal, DBase, Wordstar) **von CP/M- auf MsDos-Rechnerformat** (und umgekehrt) überträgt. Bei der Zusammenstellung dürfen die Diskettenpreise nicht außer Acht gelassen werden! So kosten:

10 St. im 3 Zoll Format mindestens 59,- DM;
10 St. im 3½ " " " 28,- DM;
10 St. im 5¼ " " " 10,- DM (no name).

Anhand dieser knappen Zusammenstellung mag sich jeder Anwender selbst ein Bild machen, ob überhaupt und wenn ja, welche Anschaffung eines Laufwerks sich lohnt.

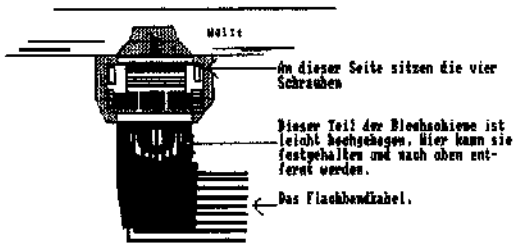
Die Reinigung des Druckkopfes

Eines der empfindlichsten Teile des Joyce ist der Drucker. Dies liegt daran, daß er, ebenso wie Tastatur und Laufwerk, mechanischen Beanspruchungen unterworfen ist. Wenn er trotzdem klaglos seine Arbeit verrichtet, liegt das an seiner grundsoliden Konstruktion. Dort, wo es darauf ankommt, wurde nicht am Material gespart. (z.B. Messinghülsen an der Laufschiene des Druckkopfes etc.) Bis auf die Tatsache, daß er ab und an - je nach Beanspruchung - ein neues Farbband braucht, arbeitet er recht wartungsfrei.
Dennoch, nach einer Zeit von etwa hundert Betriebsstunden kann es sein, daß bei Grafikausdrucken mit dunklen Flächen schwarze Streifen auf dem Papier erscheinen, wo sie nichts zu suchen haben. In Extremfällen kann dies auch beim normalen Schreiben geschehen. Dann hilft alles nichts mehr, entweder man gibt den Drucker zur Wartung in eine Spezialwerkstatt, oder man macht sich selbst die Finger schmutzig, denn dann ist der Druckkopf verunreinigt und muß auseinander genommen werden. Dazu folgende Anleitung:

Im Druckkopf sitzen neun kleine Nadeln, die von Elektromagneten ans Farbband gedrückt werden und einen Punkt aufs Papier schlagen. Diese Nadeln können von Farb- und Farbbandresten so verkleben, daß sie nicht mehr schnell genug in ihre Ruheposition zurückkehren können und beim Weiterlauf des Kopfes Schlieren aufs Papier ziehen. Entfernt man Abdeckklappen und Farbband, so erkennt man unter der Stelle, wo das Farbband gesessen hat, ein Flachbandkabel, das in den Druckkopf läuft. Bevor es in diesen hineingelangt, muß es unter einer Blechklammer hindurch. Diese Blechklammer (sie hält den Druckkopf) hat eine kleine, nach oben gebogene Nase. An dieser Nase wird die Klammer mit einer entsprechend kleinen Zange nach oben hin abgezogen. Der Druckkopf kann jetzt ein wenig von der Papierrolle weggezogen und nach oben aus seiner Halterung geholt werden.
Um das Flachbandkabel nicht zu beschädigen - es bleibt am Druckkopf - muß desweiteren entsprechend vorsichtig mit dem

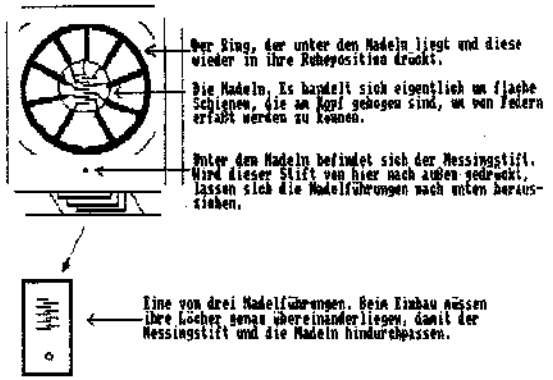
Kopf umgegangen werden. Von der Papierrollenseite (vorne) werden jetzt vier Schrauben am Kopf sichtbar. Diese Schrauben werden (ohne rohe Gewaltanwendung - sonst verdreht sich die Magnetanordnung und muß durch Probieren neu justiert werden) herausgedreht, wobei im Endstadium die Schrauben nach unten zeigen sollten, damit die Nadeln nicht aus dem Druckkopf fallen. Jetzt kann die hintere Platte mit der Aufschrift "HOT" entfernt werden, und das schwarze vordere Plastikteil vom inneren Eisenteil getrennt werden. Im Endeffekt hat man den magnetischen Kern, in den das Flachbandkabel hineinläuft, und den Nadelträger in der Hand. Das Teil am Kabel kann am Drucker bleiben, der Rest muß gesäubert werden.
Man sollte sich jetzt genau die Lage der Zwischenringe merken, sie müssen in der gleichen Reihenfolge wieder hinein (am besten der Reihe nach auf ein Blatt Papier in eine sichere Ecke legen). Einer dieser Ringe kann nicht entfernt werden, ohne die Nadeln aus der Führung zu ziehen. Dieser Ring sorgt dafür, daß die Nadeln wieder in ihre Ruhestellung zurückgedrückt werden. Also müssen zuerst die Nadeln raus.
Mit einer Pinzette kann man die Nadeln ganz leicht von innen aus ihren Führungen herausziehen. Dabei muß man sich unbedingt ihre Lage merken, denn sie müssen genauso wieder hinein. Am besten man bereitet ein Stück Styropor oder Karton mit Zahlen von 1-9 vor, in das die Nadeln ihrer Lage entsprechend hineingesteckt werden. Danach kann auch der Ring, der sie zurückdrückt, abgenommen werden. Im unteren Teil der Plastikhalterung befindet sich ein kleiner Stift aus Messing. Dieser kann von innen nach außen herausgedrückt werden. Dieser Stift hält drei Führungen für die Nadeln. Sie lassen sich danach nach unten herausziehen. (Auch ihre Lage muß gemerkt werden!)
Alle Teile können nun mit Benzin gereinigt werden. Für die drei Nadelführungen eignet sich am besten eine in Benzin getauchte Zahnbürste.
Nach der Reinigung werden die Teile in umgekehrter Reihenfolge wieder zusammengesetzt. Die vier Schrauben des Kopfes sollten zwar fest, jedoch nicht allzu stramm

angezogen werden. Vor Anbringen des Druckkopfes sollte die Walze gründlich mit einem dafür vorgesehenen Reinigungsmittel gesäubert werden (sonst Spiritus). Danach steht einem Probedruck nichts mehr im Wege.



Der Druckkopf von oben gesehen

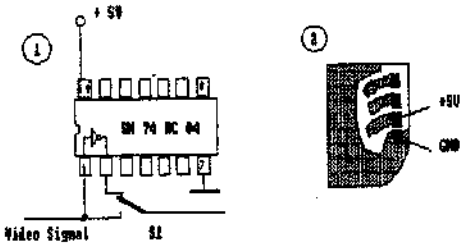
Schematische Darstellung des Druckkopfes von hinten/innen



Ein Bildschirminverter

Nach Einbau entsprechender Teile ist es möglich, den Bildschirm unabhängig vom gerade laufenden Programm durch Umlegen eines Schalters in seinen Darstellungsfarben umzukehren (zu invertieren). Ein Invertieren wäre etwa für das Mica-CAD Programm (Mica ist hervorragend geeignet für wissenschaftliche Zeichnungen) sehr nützlich, da Mica einen softwaremäßig umgeschalteten Bildschirm wieder zurücksetzt. Überhaupt ist der Umschalter bei all den Programmen gut einzusetzen, bei denen es auf Darstellungsschärfe am Bildschirm ankommt. Die Funktionsweise des Inverters ist denkbar einfach:
Ein Bildschirm-(Video-)Signal besteht aus zwei Pegeln: entweder low (0 V) oder high (+5 V). Steht der Pegel auf high, wird ein Punkt auf den Bildschirm gesetzt, andernfalls nicht. Jetzt liegt der Gedanke nahe, die Pegel zu vertauschen. Dies wird durch Verwendung des TTL-Bausteins SN 74 HC 04, der aus sechs Invertern besteht (einer wird nur benutzt), erreicht. Wird ein Umschalter mit eingebaut, (s. Skizze folgende Seite) kann man unabhängig von der gerade laufenden Software zwischen hellem oder dunklem Hintergrund wählen. Der Einbau ist einfach durchzuführen, wenn man nicht gerade mit dem Lötkolben auf Kriegsfuß steht.
Stückliste: 1 SN 74 HC 04 / 1 Sockel 14-polig. / mind. 50 cm Kabel 4-adrig / 1 Schalter 1 * um / Schrumpfschlauch oder Isolierband.
Netzstecker des Gerätes ziehen und das Video-Kabel kappen, um das IC dazwischen zu setzen. Das Viedeo-Kabel verläuft vom Blechkasten B (s. Skizze S. 265) zur Monitorplatine D. Es ist orange und kommt (als 3./vorletztes) Kabel aus der blauen Steckverbindung der Hauptplatine.
Die vom Blechkasten B kommende Video-Leitung wird mit Pin 1 des Sockels verbunden, Pin 2 über den Umschalter mit dem weiterführenden Kabel zur Monitorplatine (D). Der andere Pol des Umschalters wird an Pin 1 des Sockels gelötet. Die Plusleitung der Tastaturplatine (dort wird die Tastatur angeschlossen) (s. Skizze 2) muß an Pin 14, die Minusleitung an

Pin 7 gelötet werden. Wurden Lötstellen und Kontakte ausreichend gegen Kurzschlüsse isoliert, kann das IC in den Sockel gesteckt werden. (Auf richtige Polung achten!)



Zum Schluß kommt es nur noch auf eine entsprechende Anbringung des Umschalters an. Eine besonders dekorative Art ist, ihn in den Standfuß des Monitors zu setzen. Der Aufwand dafür ist jedoch recht hoch, da viel gebohrt werden muß. Einfacher ist es, das Kabel durch irgendeinen Lüftungsschlitz nach außen zu führen. Ist dies geschehen, steht dem Zusammenbau des Gehäuses - oder falls man weiterbasteln will - einem Testlauf nichts mehr im Weg.

Wenn es zu **Fehlfunktionen nach Einbauten** kommt...

Ein besonders häufiger Fehler ist eine sogenannte kalte Lötstelle, bei der das Lötzinn nicht korrekt verlaufen ist. Sämtliche Lötverbindungen sind hierauf zu prüfen (sehen grau, rauh und runzelig aus) und eventuell nachzulöten.

Der Bildschirm bleibt ständig hell:

Überprüfen Sie die Pegel des IC 2001 (SN 74 HC 00) auf der Monitorplatine links hinten und die richtige Anbringung des Umschalters nach dem IC SN 74 HC 04.

Nichts geht mehr:

Überprüfen Sie sämtliche Verbindungen und Lötstellen auf Kurzschluß oder Lötbrücken. Außerdem sind die Spannungen und Sicherungs-IC's P501 und P502 zu überprüfen (+5 V und +12 V). Sollten die Sicherungs-IC's hochohmig sein, reicht eine einfache Überbrückung.

Der Expansions-Port

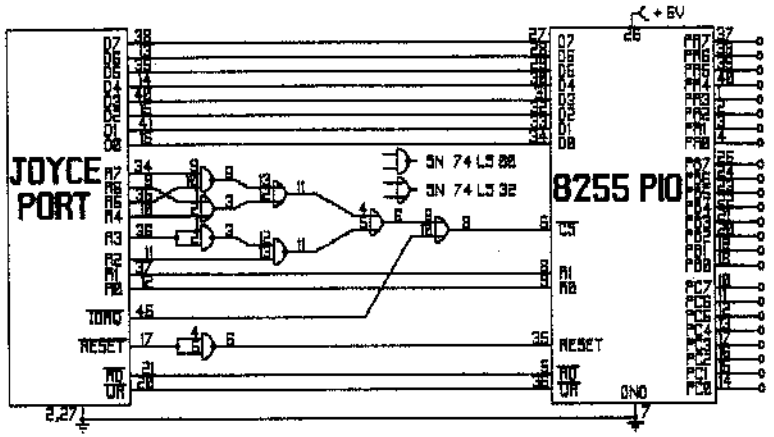
Über den Expansions-Port lassen sich vielfältige Aufgaben bewältigen. Für die meisten Joyce-User ist er allerdings nur als Steckplatz für die CPS 8256 (zur Schnittstelle später mehr) in Erscheinung getreten.

Um den Hardware-Freaks Gelegenheit zu geben, eigene Experimente anzustellen, kann hier die Belegung der einzelnen Ports nachgesehen werden.

Pin	Belegung	Belegung	Pin
1	nc	nc	26
2	GND	GND	27
3	+5 V	+5 V	28
4	nc	+12 V	29
5	A14	A15	30
6	A12	A13	31
7	A10	A11	32
8	A8	A9	33
9	A6	A7	34
10	A4	A5	35
11	A2	A3	36
12	A0	A1	37
13	D6	D7	38
14	D4	D5	39
15	D2	D3	40
16	D0	D1	41
17	/Reset	/M1	42
18	/BUSRQ	/INT	43
19	/BUSAK	/WAIT	44
20	/WR	/MREQ	45
21	/RD	/IORQ	46
22	nc	NSYNC	47
23	/MIDS	VIDEO	48
24	/32 MHZ	/4MHZ (Z80A Takt)	49
25	GND	GND	50

Wem z.B. die Anschaffung der CPS8256 zu teuer erscheint, kann sich mit Hilfe dieses Belegungsplans und der Anleitung auf den folgenden Seiten seine **Schnittstelle** selbst bauen. Hier wird gezeigt, wie drei parallele Ausgänge durchzuschleifen sind.

Kernstück einer aus drei mal acht Bit bestehenden Schnittstelle ist der Ein / Ausgabebaustein 8255. Er verfügt über die angesprochenen 3 E/A-Ports zu je 8 Bit, die in ihrer Funktion auf verschiedene Weise programmiert werden können. Im folgenden soll nur auf die Betriebsart 0 (Standard-Ein/Ausgabe) eingegangen werden.



Wer sich obigen Schaltplan nicht selbst verdrahten will, der kann entweder auf die zur Ätzung der fertigen Platinenlayouts (zweiseitig!!) zurückgreifen, die sich im Anhang des Buches befinden, oder sich, wie bereits besprochen, seine geätzten Platinen bei der Firma Joyce-Platinenservice bestellen. Diese Platinen sind bereits fertig durchkontaktiert, gebohrt, verzinnt und mit Bestückungsaufdruck

versehen.
Diese Schnittstelle wird über den Port 0A7H programmiert. Sie erwartet einen aus 8 Bit bestehenden Wert, der nach folgender Tabelle kodiert wird:

```
D0:     Port C, untere 4 Bits:
        1 = Eingang
        0 = Ausgang

D1:     Port B:
        1 = Eingang
        0 = Ausgang

D2:     Betriebsartdefinition für Port C (untere 4 Bits) und Port B:
        0 = Betriebsart 0 (Standard-Ein/Ausgabe)
        1 = Betriebsart 1 (getaktete Ein/Ausgabe)

D3:     Port C, obere 4 Bits:
        1 = Eingang
        0 = Ausgang

D4:     Port A:
        1 = Eingang
        0 = Ausgang

D5, D6: Betriebsartdefinition für Port C (obere 4 Bits) und Port A:
        00 = Betriebsart 0 (Standard-Ein/Ausgabe)
        01 = Betriebsart 1 (getaktete Ein/Ausgabe)
        1X = Betriebsart 2 (getaktete bidirektionale Bus-Ein/Ausgabe)

D7:     Kennbit für  Betriebsart definieren"
        1 = aktiv
```

Um eine neue Betriebsart einzustellen, muß mindestens Bit 7 (D7) zusammen mit den gewünschten Bits gesetzt werden. Falls die Schnittstelle 3 8-Bit Ausgabekanäle erhalten soll, muß der Wert 128 (80H) an die Adresse 0A7H übergeben werden.

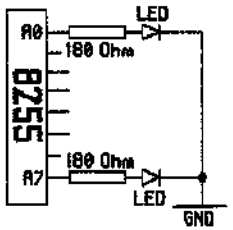
Die Portadressen für die Programmierung sind folgende:

0A4H: Port A
0A5H: Port B
0A6H: Port C
0A7H: Steuerport (zum Definieren)

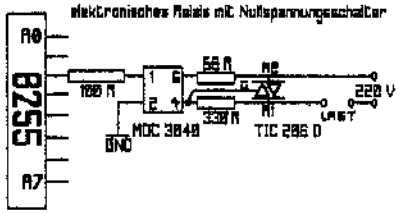
Das Beispiel

```
OUT &HA7,&H80:OUT &HA4,&HFF
```

setzt alle Kanäle auf Ausgabe und gibt an Port A den Wert 255. Um die Datenbits sichtbar zu machen, bieten sich verschiedene Möglichkeiten an:



Beschaltung mit LED



Beschaltung mit Relais

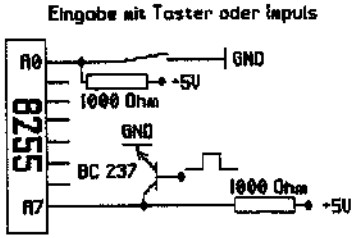
Es ist natürlich auch möglich, über die Schnittstelle Daten einzulesen. Dazu muß jedoch die Schnittstelle als Eingabemedium definiert sein. Ein möglicher Initialisierungscode könnte zum Beispiel lauten:

```
OUT &HA7,&H89
```

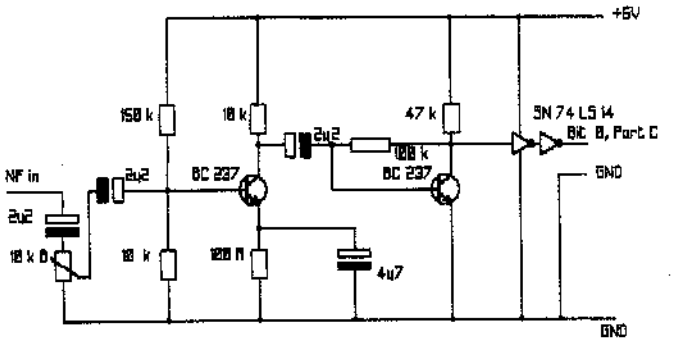
Dieses Beispiel definiert Port A und B als Ausgang, Port C als Eingang.
Mit:

```
PRINT INP(&HA6)
```


wird der aktuelle Wert an Port C auf dem Bildschirm ausgegeben. Es ist dabei zu beachten, daß die Werte über Pull-Up Widerstände angelegt werden (s. Schaltplan), um Datenfehler zu vermeiden:



Ein wirkungsvolles Anwendungsbeispiel für Ein- und Ausgabe über die Schnittstelle ist die Ansteuerung einer musikgesteuerten **8-Kanal Lightshow**. Eine solche Steuerung läßt sich am besten nach folgender Schaltung aufbauen:



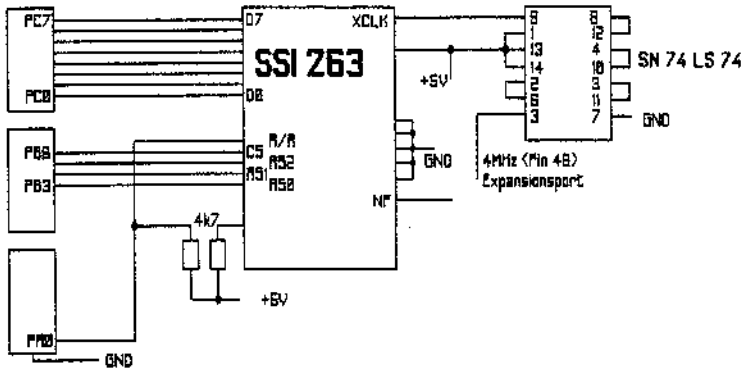
Die Ansteuerung der Lichterkette kann wie auf Seite 272 dargestellt erfolgen.

Das Programm basiert darauf, daß die Schaltung die NF der Musik-Anlage in eine positive Spannung umsetzt. Die Tatsache, daß ein Beat (Tief-Ton) eine größere Spannung als hochfrequente NF erzeugt, wird von dem Logikbaustein SN 74 LS 14 so ausgenutzt, daß er einen Beat in einen Impuls umsetzt, der von dem folgenden Programm verwertet werden kann:

```
10 OUT &HA7,&H89                    'Port A, B: Ausgabe;   Port C: Eingabe
20 PRINT CHR$(27)+"E"+CHR$(27)+"f"
30 RESTORE 300
40 READ wert$                       'Lichtmuster auslesen
50 IF wert$="-1" THEN 30            'Lichtmustertabelle zu Ende
60 wert=VAL("&H"+wert$)
70 IF (INP(&HA6) AND 1)<>1 THEN 70 'Solange Zeile wiederholen, bis Bit 1 gesetzt
80 OUT &HA4,wert                    'Programmiertes Lichtmuster  Port A
90 OUT &HA5,INT (RND*256)           'Zufallslichtmuster          Port B
100 FOR i=0 TO 95:NEXT              'Warteschleife
110 GOTO 40
297 '
298 ' Ab Zeile 300 stehen die programmierten Lichtmuster (Bit gesetzt = Lampe an)
299 '
300 DATA 80,C0,E0,F0,F8,FC,FE,FF,FE,FC,F8,F0,E0,C0,80,00,80,C0,E0,F0,F8,FC,FE,FF,FE
310 DATA FC,F8,F0,E0,C0,80,00,80,C0,E0,F0,F8,FC,FE,FF,FE,FC,F8,F0,E0,C0,80,00,80,C0
320 DATA E0,F0,F8,FC,FE,FF,FE,FC,F8,F0,E0,C0,80,00,80,C0,E0,F0,F8,FC,FE,FF,FE,FC,F8
330 DATA F0,E0,C0,80,00,80,C0,E0,F0,F8,FC,FE,FF,FE,FC,F8,F0,E0,C0,80,01,03,07,0F,1F
340 DATA 3F,7F,FF,7F,3F,1F,0F,07,03,01,00,01,03,07,0F,1F,3F,7F,FF,7F,3F,1F,0F,07,03
350 DATA 01,00,01,03,07,0F,1F,3F,7F,FF,7F,3F,1F,0F,07,03,01,03,07,0F,1F,3F,7F,FF,7F
360 DATA 3F,1F,0F,07,03,01,00,01,03,07,0F,1F,3F,7F,FF,7F,3F,1F,0F,07,03,01,00,01,03
370 DATA 07,0F,1F,3F,7F,FF,7F,3F,1F,0F,07,03,06,0C,18,30,60,C0,81,03,06,0C,18,30,60
380 DATA C0,81,03,06,0C,18,30,60,C0,81,03,06,0C,18,30,60,C0,81,03,06,0C,18,30,60,C0
390 DATA 81,03,06,0C,18,30,60,C0,81,03,06,0C,18,30,60,C0,81,03,06,0C,18,30,60,C0,81
400 DATA 82,84,88,90,A0,C0,C1,C2,C4,C8,D0,E0,E1,E2,E4,E8,F0,F1,F2,F4,F8,F9,FA,FC,FD
410 DATA FE,FF,81,82,84,88,90,A0,C0,C1,C2,C4,C8,D0,E0,E1,E2,E4,E8,F0,F1,F2,F4,F8,F9
420 DATA FA,FC,F0,FE,FF,81,82,84,88,90,A0,C0,C1,C2,C4,C8,D0,E0,E1,E2,E4,E8,F0,F1,F2
430 DATA F4,F8,F9,FA,FC,FD,FE,FF,81,42,24,18,FF,00,FF,18,24,42,81,42,24,18,FF,00,FF
440 DATA 18,24,42,81,42,24,18,FF,00,FF,18,24,42,81,42,24,18,FF,00,FF,18,24,42,81,42
450 DATA 24,18,FF,00,FF,18,24,42,81,42,24,18,FF,00,FF,18,24,42,81,42,24,18,FF,00,FF
460 DATA 18,24,42,81,42,24,18,FF,00,FF,18,24,42,81,AA,55,AA,55,AA,55,AA,55,AA,55,AA
470 DATA 55,AA,55,AA,55,AA,55,AA,55,AA,55,AA,55,AA,55,AA,55,AA,55,CC,33,CC,33
480 DATA CC,33,CC,33,CC,33,CC,33,CC,33,CC,33,CC,33,CC,33,CC,33,CC,33,CC,33,CC,33,CC
490 DATA 33,CC,33,CC,C3,3C,C3,3C,C3,3C,C3,3C,C3,3C,C3,3C,C3,3C,C3,3C,C3,3C,C3,3C,C3
500 DATA 3C,18,3C,7E,FF,7E,3C,18,00,18,3C,7E,FF,7E,3C,18,00,18,3C,7E,FF,7E,3C,18,-1
```


Ein Sprachsynthesizer

Mit dem bisherigen Wissen über die Schnittstelle läßt sich dem Joyce das Reden beibringen. Eine entsprechende Schaltung zu diesem Zweck ist weiter nicht schwierig aufzubauen, wenn man die hauseigene HiFi-Anlage endlich einmal sinnbringend als NF-Verstärker einsetzt. (s. Skizze)



An dieser Stelle wollen wir der Phantasie der Anwender keine Grenzen setzen. Lediglich zur Anregung folgendes kleines Listing. Erste Probeschritte können durch Austausch der Dateneinträge in Zeile 140 in Abstimmung mit dem Phonemeinventar vorgenommen werden. Die Datazeile wird durch die Werte 0 (Pause) und -1 (Ende Liste) abgeschlossen.

```
40 OUT &HA7,&H90                'Port A Eingabe, Port B und C Ausgabe
50 dac=&HA6:dab=&HA5
60 c=128:b=24:GOSUB 160         'Initialisierung des SSI 263
70 c=128:b=0:GOSUB 160          '(siehe CPC-International
80 c=95:b=24:GOSUB 160          ' Sonderheft 7-88/89
90 c=50:b=32:GOSUB 150          ' S. 121 ff)
100 c=105:b=16:GOSUB 150        '
110 c=40:b=8:GOSUB 150          '
120 READ a$:IF a$="-1" THEN 130 ELSE c=VAL("&H"+a$):b=0:GOSUB 150:GOTO 120
130 b=0:GOSUB 150:END
140 DATA 25,7,2a,30,0,7,30,28,0,0,30,27,7,7,28,32,2a,7,2a,0,-1 :'Phonemliste / 0,-1=Ende
150 IF INP(&HA4)<>0 THEN 150    'warten, bis SSI 263 fertig
160 OUT dac,c:OUT dab,b OR 64:OUT dab,b AND 56:RETURN
```

Der Sprachchip SSI 263 besitzt ein Phoneminventar, das nach entsprechender Initialisierung ausgegeben wird. Im folgenden nun die Phonemliste mit den dazugehörigen Codes. Bei der Anwendung sollte beachtet werden, daß die Laute der englischen Sprache angepaßt wurden. Bei der Programmierung von Lautfolgen sollte der Anwender in einer Laut- und nicht in einer Schriftsprache konstruieren.: - hier = hia -

Das Phoneminventar:

Hex-Code	Symbol	Beispielwort (in Englisch)	Hex-Code	Symbol	Beispielwort (in Englisch)
00	PA	(Pause)	20	L	lift
01	E	meet	21	L1	play
02	E1	bent	22	LF	fall (End-L)
03	Y	before	23	W	water
04	Y1	year	24	B	bag
05	AY	piggse	25	D	paid
06	IE	any	26	KV	tag
07	I	six	27	P	pencil
08	A	made	28	T	tart
09	AI	care	29	K	kit
0A	EH	nest	2A	HV	(Vokal halten)
0B	EH1	beit	2B	HVC	d(h)ought
0C	AE	dad	2C	HF	heart
0D	AE1	after	2D	HFC	p(h)ound
0E	AH	got	2E	HN	(Nasal halten)
0F	AH1	father	2F	Z	zero
10	AW	office	30	S	same
11	O	store	31	J	pleasure
12	OU	boat	32	SCH	sheep
13	OO	look	33	V	very
14	UU	you	34	F	five
15	UU1	could	35	THV	there
16	U	tune	36	TH	with
17	U1	moon	37	M	most
18	UH	wonder	38	N	nine
19	UH1	love	39	NG	song
1A	UH2	what	3A	:A	Mädchen (*)
1B	UH3	hut	3B	:OU	Möwe (*)
1C	ER	bird	3C	:U	Tür (*)
1D	R	roof	3D	:UH	menu (+)
1E	R1	rug	3E	EZ	Bitte (*)
1F	R2	Kutter.(*)	3F	LB	il (+)

(*) bedeutet, daß das Beispielwort aus dem Deutschen kommt; (+), daß es sich um ein französiches Wort handelt.

Wer sich noch ein wenig mehr über den Sprachsynthesizer informieren möchte, der sei auf den Artikel des CPC-International Sonderheftes 7-88/89 S. 121 ff verwiesen.
Der dort vorgeführte "Laberkasten" wurde an unsere Schnittstelle angepaßt.

Erweiterungen zur Joyce-Hardware

Zur Hardware läßt sich nicht nur das zählen, was die elektrischen Eigenschaften des Joyce unterstützt. So kommt es, daß an dieser Stelle auch drei Dinge Erwähnung finden, die - ohne Produktwerbung betreiben zu wollen - sich als ausgezeichnete Hilfsmittel bei der Arbeit mit dem Joyce erwiesen haben.
Wer lange am Bildschirm arbeiten muß, wird es über kurz oder lang als äußerst lästig empfinden, daß sich helle Gegenstände, Lampen oder Fenster im Bildschirm spiegeln. Speziell für den Joyce wurde ein **Bildschirmfilter** entwickelt, der sich formvollendet dem Design des Gehäuses anpaßt und mit Klettverschlüssen, die unsichtbar unter dem Rahmen des Bildschirmfilters angebracht sind, am Monitor befestigt wird. Dieser Filter - er besteht im wesentlichen aus einer schwarzen Gaze - fängt unerwünschte Reflexionen auf, läßt einen eventuell leicht flackernden Monitor ruhiger erscheinen und die matte, grünliche Aura, die die Zeichen am Monitor begleitet, leuchtet nicht mehr so schnell durch. Bei diesem Bildschirmfilter handelt es sich um ein nützliches Utensil, daß jedem Anwender, der Augen und Nerven schonen möchte, nur empfohlen werden kann.
Bei der zweiten nützlichen Erweiterung handelt es sich um einen **Einzelblatteinzug** für den Drucker, den der DMV-Verlag in Eschwege zum Preis von DM 29.95 anbietet. Der Vorteil gegenüber anderen Einzügen besteht darin, daß stufenlos auf verschiedene Blätter von 45 mm bis 260 mm eingestellt werden kann (bei Rastereinstellungen haken oft die Blätter und fallen nicht von allein auf die Walze), und daß er vom Design her dem Joyce-Drucker angepaßt ist. Der Einzug wird einfach auf den Drucker gesteckt. Die für den Drucker mitgelieferten zwei schwarzen Papierhalter können an den Einzelblatteinzug angeclippt werden. Unter Verwendung einer Einzugsführung wird selbst bei einer größeren Anzahl von Druckseiten die Seitenzahl immer an genau der gleichen Stelle erscheinen und auf Grund der rasterlosen Einstell-

möglichkeit (fast jedes Blatt ist bei geforderter Genauigkeit verschieden) gelingen Doppeldrucke eines Blattes (dazu später mehr) auf den Pixel genau.
Eine weitere wichtige Neuerung auf dem Hardwaremarkt sind **Farbbänder** für den Joyce, wobei "Farb"- jetzt wörtlich zu verstehen ist. Mittlerweile gibt es eine Vielzahl von Firmen, die sich von der Angebotspalette her auf den Joyce konzentrieren und auch immer ein wachsames Auge auf den Englischen Markt werfen, um interessante Neuerungen gleich parat zu haben. Unter anderem bieten sie Farbbänder für den Joyce-Drucker in z.B. rot, blau, grün oder braun an. Der Preis für diese Farbbänder, (die auch leicht selbst nachgefärbt werden können) liegt bei ca. 24,- DM. Textverarbeitungssysteme, die von den Druckbefehlen her eine Druckunterbrechung zulassen (Prowort z.B. könnte während des Druckvorgangs den Drucker stoppen und die Meldung "Farbband wechseln!" auf den Monitor bringen) könnten Texte jetzt sogar farbig markieren. Jedoch lassen sich auch Grafiken produzieren, deren Hauptaussagewert sich durch ihre Farbigkeit ergibt! (s. Hardwareerweiterungen zur Grafik)
Im Folgenden möchte ich kurz auf Erweiterungen eingehen, die die elektrischen Eigenschaften des Joyce in effektivem Maße verbessern. Hier muß zuerst

Die Schnittstelle CPS 8256

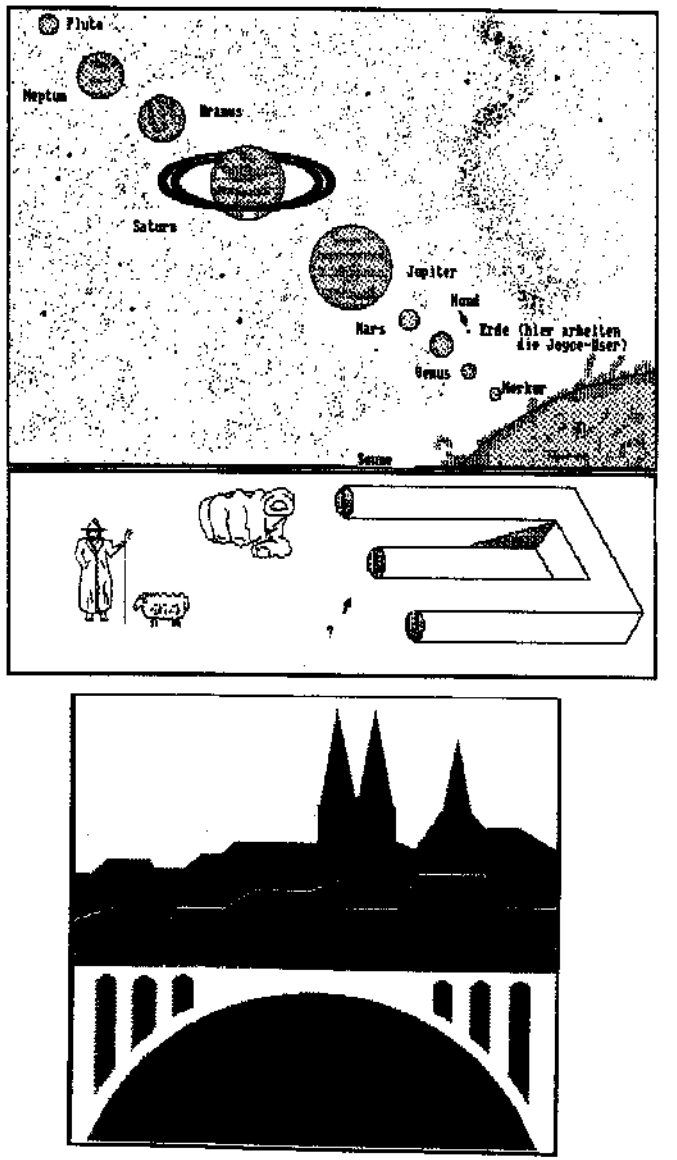
Erwähnung finden, denn am Anfang jeder Hardwareerweiterung steht eine passende Schnittstelle, über die die Vermittlung zwischen Joyce und seiner Außenwelt vonstatten geht. Wer mit Fremddruckern, Akustikkopplern, Modems, anderen Terminals oder dergleichen arbeiten will, ist mit der CPS8256 Schnittstelle von Schneider gut beraten. Sie stellt einen seriellen RS232C und einen parallelen (Centronics) Bus zur Verfügung. Im Handel ist diese Schnittstelle für ca. 180,- DM erhältlich. Der erste Vorteil dieser Schnittstelle besteht darin, daß sie sich optimal dem Design des Joyce anpaßt. Sie wird mit zwei Schrauben solide am Computergehäuse befestigt und führt Anschlüsse nach rechts zur Seite weg. Zum Lieferumfang dieser Schnittstelle gehört ein kleines Handbuch, das allerdings entsprechend der komplexen Technik ein wenig

umfangreicher hätte ausfallen dürfen. Die Schnittstelle kann unter CP/M u.a. mit **device, setsio** oder **pip**, welches ganze Dateien an die Schnittstelle sendet, angesprochen werden.
Auf Seite eins der Systemdisketten befindet sich unter dem Modus "versteckt" (LocoScript) das Programm MAIL232, das unter diesem Namen von CP/M aus gestartet werden kann. Dieses Programm arbeitet klaglos mit der CPS8256 zusammen, und gestattet auch, die Daten-Fernübertragung (DFÜ) via Akustikkoppler oder Modem mit dem Joyce vorzunehmen (s. unten). Über diese Schnittstelle können auch mehrere Rechner im Zusammenhang mit Festplatten und/oder leistungsfähigen Druckern zu einem mächtigen Verbund zusammengeschlossen werden, der selbst Geschäftsabwicklungen größerer Firmen über mehrere Bildschirmarbeitsplätze hinweg koordinieren und durchführen kann. Aber auch der Anwender, der mit seinem PCW nicht so hoch hinaus möchte und ihn im kleineren Rahmen für Grafik und Text benutzen möchte, ist mit der CPS8256 gut beraten. Bis auf die der englischen Steckernorm entsprechende Hardware laufen sämtliche Hardwareerweiterungen über diese Schnittstelle. So auch die:

Gerdes-mouse.

Dieses Eingabegerät wird mit einer ausführlichen Anleitung und einem Softwarepaket ausgeliefert, daß keine Wünsche offen läßt. Soft- und Hardware werden im sogenannten Joyce-MousePack für ca. 180,- DM von der Fa. Reis-Ware angeboten; ein Preis, der in Anbetracht der Fähigkeiten des Systems nicht zu hoch gegriffen ist. Die zur Mouse gelieferte Software erlaubt es, sie in LocoScript und CP/M zum Einsatz zu bringen.
Unter Basic wird zum normalen Basic-Befehlssatz eine vielzahl von mächtigen Befehlen zur Grafikprogrammierung hinzugefügt. Der Anwender braucht sich allerdings um die Programmierung nicht mehr zu kümmern, wenn er unter dem erweiterten Basic das Programm "Centaur" aufruft. Via Menue können mit der Mouse Funktionen angeklickt werden, die alle bekannten Grafikbefehle von horizontal/vertikal spiegeln, drehen, zoomen, vergrößern, kopieren und und und beinhalten.

Pixelweise können unter dem Lupenmenue Punkte auf dem Monitor gesetzt und gelöscht werden. Über das Beschriftungsmenue stehen 24 Schriftarten zur Verfügung, die jeweils in über 20 Größen und 4 Breiten zur Ausführung kommen können.
Neue Schriftarten lassen sich über ein extra Programm ebenso wie die Füllmuster leicht selbst kreieren. Schrift kann auch unter dem "Untagmodus" in vorgegebenen Zeilen- und Spaltenbreiten der Grafik angemessen ausgegeben werden, wobei dann auch schrittweise mit den Cursortasten angesteuert werden kann. Das "Centaur-Programm" simuliert damit einen Desk-Top-Publisher. Faszinierend ist bei alledem die Präzision, mit der der Cursor Punkte ansteuert und festhält.
Was den Programmierern dieses Packs noch zusätzlich Bewunderung abringt, ist die enorme Geschwindigkeit, mit der die Aktionen abgewickelt werden.
Die Option "Ausschnitt löschen" gestattet jetzt auch, Grafik bunt auszudrucken. Nach Erstellung wird ein Bildschirminhalt abgespeichert, danach Teile, die in anderen Farben gedruckt werden sollen, gelöscht. Der noch stehende Bildschirmteil wird unter Zuhilfenahme des Einzelblatteinzugs, in gewünschter Farbe gedruckt. (Der Einzug garantiert, daß die Blätter bei einem erneuten Druckdurchlauf wieder genauso zu liegen kommen, wie beim ersten. Dazu müssen die Blätter auch durch Eigengewicht durch den Einzug auf die Walze fallen können, um gleiche Druckhöhe zu erreichen.)
Jetzt kann die Grafik wieder neu von Diskette geladen, entsprechende Teile gelöscht und der Druck mit einer anderen Farbe nach Wechsel des Farbbandes vorgenommen werden. Dies wird so oft wiederholt, wie Farben in einer Grafik vorhanden sein sollen.
Die Bilder der folgenden Seite entstanden auf dem Joyce-Drucker, und sollen nicht für sich selbst, sondern ebenso wie die technischen Zeichnungen in diesem Buch, die ebenfalls alle mit dem Mousepack erstellt wurden, für dieses Eingabemedium und seine Möglichkeiten sprechen (im Druckermenue hätten auch andere Größen für den Ausdruck - von A4 bis Posterformat - gewählt werden können).



Grafikdemos zum Joyce-MousePack

Neben dem Gerdes MousePack von Reis-Ware gibt es noch andere Eingabemedien, für die aber keine derart gute Grafiksoftware entwickelt wurde. Diese "Mäuse" haben jedoch auch ihre Vorteile. Aufgrund ihrer englischen Herkunft (Kempston, AMX, Electric Studio) besitzen sie entsprechende Schnittstellen, die sie sowohl Hard- als auch Softwaremäßig kompatibel zueinander und zu einer ganzen Reihe von Programmen machen. Gemeinsames Merkmal fast aller Produkte ist die Möglichkeit, das DeskTopPublishing Programm **News Desk** zu steuern, welches sich auch allgemein bei den Joyce-Usern gegen den konkurrierenden Fleet Street Editor durchgesetzt zu haben scheint.

News Desk gestattet dem Anwender über das eingebaute Grafikprogramm Bilder zu erstellen und diese dann mit Text auf einer Din A4 Seite zu arrangieren. Der Text kann über einen Fremdeditor eingegeben werden, und dann nach gutdünken in diverse Spaltenbreiten gebracht werden. Gearbeitet wird bei News Desk immer an Ausschnitten einer A4 Seite. Wurde alles richtig plaziert, kann man sich über ein spezielles Menue die vollständige Seite auf den Monitor holen.

Dem Anwender stehen danach aber immer noch alle Möglichkeiten offen, Text und Grafik zu manipulieren, so daß er sich nach kürzester Zeit als sein eigener Verleger fühlt. Zusätzlich sind zu News Desk allerhand Disketten im Handel, auf denen schon die verschiedensten Schriftarten- und Grafikfonts zur Verfügung stehen. Der Anwender braucht sie nur noch seinen Zwecken entsprechend einzusetzen.

Ohne geeignete Eingabemedien gestaltet sich die Arbeit mit dem Grafikprogramm von News Desk allerdings ein wenig schwierig. Schon der Versuch, eine gebogene Linie mit Hilfe der Cursortasten auf den Monitor zu bringen, scheitert kläglich und hinterläst ein kammartiges Gebilde auf dem Schirm.

Hier könnte nun auch der **Light Pen** von Electric Studio weiterhelfen. (Wird News Desk (ND) gestartet und der Light Pen ist an den PCW angeschlossen, fragt ND automatisch ab, ob mit Cursortasten oder Light Pen gearbeitet werden soll. Dies geschieht auch mit Mäusen des englischen Standards)

Der Light Pen wird gehandhabt wie ein kleiner Zeigestock. In seiner Spitze befindet sich ein Photosensor, der über ein flexibles Kabel die Meldung an den Monitor weitergibt, auf welchen Punkt er gerade zeigt. So können einzelne Punkte eines Menues angewählt werden, die, nachdem der Light Pen auf sie gezeigt hat, invers dargestellt und durch Druck auf die Leertaste angeklickt werden. Befindet man sich im Zeichnenmodus, erscheint ein kleines Kreuz direkt unter dem Light Pen am Monitor, das, je nach gewählter Option, nun Linien hinter sich herzieht, Kreise oder Vielecke zeichnet oder auch dreidimensionale einfache geometrische Gebilde herstellt, von denen jeweils nur die Eckpunkte angegeben werden müssen.

Hat man schon einiges durch Führung des Light Pen über den Monitor gezeichnet und vielleicht schon einige Flächen dunkler gefüllt, wird es schwierig, den Kreuzcursor exakt zu führen. Er vollzieht dann einige Sprünge und ist nicht so leicht zu bewegen, eine gerade Linie über ein dunkleres Feld hinweg zu ziehen. Sollte sich der Cursor dennoch exakt unter der Spitze des Light Pen befinden (wo er ja hingehört) wird es schwierig einen bestimmten Zielpunkt Pixelgenau zu erwischen - man sieht ihn ja nicht, der Lichtgriffel hängt davor. Hat man jetzt eine Technik entwickelt, an der Seite des Griffels vorbeizuschielen, so hat man zumindest nach den ersten Sitzungen einen steifen Arm davongetragen (ist wohl Trainingssache), erst recht dann, wenn der Monitor auf Augenhöhe steht. (Ich habe tatsächlich schon von Anwendern gehört, die sich ihren Monitor auf den Schoß gelegt haben!) Für kleinere Freihandmalereien zwischendurch mag der Light Pen (Preis ca. 280,-DM) seine Berechtigung haben, komplexere Zeichnungen oder Schaltpläne etc. geraten damit allerdings zu einer Qual.

Die Schnittstelle des Light Pen wird über ein Flachbandkabel an den Joyce Bus angeschlossen und hängt dann recht locker an der Rückseite des Joyce. Bei der mitgelieferten Software handelt es sich um ein ähnliches Malprogramm wie bei ND, nur daß einige Optionen nicht zur Verfügung stehen.

Ob nun irgendwelche Zeichnungen gelingen oder nicht ist

nicht immer eine Frage des Eingabemediums. Oft genug steckt mangelnde Kreativität dahinter. Dieses Manko läßt sich mit dem **Scanner** nebst Software beheben, die als "Master Pack" von der Fa. Database Software im Handel sind (ca.380,- DM). Die Idee bei diesem Scanner ist, daß eine Photozelle, neben der eine kleine Lichtquelle angebracht ist, dicht über ein Blatt geführt wird. Sobald das Licht der Birne nicht im gleichen Maß vom weißen Papier zurückstrahlt, wenn es also auf eine Linie trifft, gibt die Photozelle einen Impuls weiter. Werden Photozelle und Birne auf einen Druckerkopf gesetzt, kann man über den Drucker, der Zeile für Zeile langsam abfährt, genau die Stellung eines Punktes auf einem Papier festhalten. Wie der Light Pen wird der Scanner über ein Flachbandkabel an den Expansionsport des Joyce gesteckt. An diesem Kabel baumelt dann ein kleiner Schaltkasten, der im wesentlichen einen Operationsverstärker (CA 3140), je einen 74133 / 74125 und einen Poti enthält. Über diesen nach außen gelegten Poti läßt sich die Empfindlichkeit des Lesekopfes regulieren. Aus diesem Kästchen führt dann ein flexibles Kabel zu der in Plastik geschweißten Photozelle und der Birne.

Das Plastikteil wurde ordentlich dem Joyce-Druckkopf angemessen und läuft millimetergenau zwischen Farbbandabdeckung und Papierandruckrolle. Will man allerdings via Papiereinzugshebel eine Vorlage in den Drucker einziehen, muß man vorher den Lesekopf vom Druckkopf abziehen. Das Farbband sollte aus dem Drucker entfernt werden.

Dem Scanner liegt ein Softwarepaket bei, das sich in zwei Hauptprogramme gliedert - dem zum Scannen und dem zum Bearbeiten der gescannter Bilder -. Wurde das Scanprogramm eingelegt und eine Vorlage in den Drucker gespannt, kann man über ein Menue die verschiedensten Punkte anwählen. So etwa den zu scannenden Ausschnitt oder die Vergrößerung beim Scannen. Dann kann der Scanvorgang starten. Je nach Vergrößerung fällt nun auch die Schrittweite beim Zeilenvorschub des Druckers aus, wenn der Druckkopf Zeile für Zeile der Vorlage abfährt und dabei "beleuchtet". Ebenso wie der Lesekopf das Blatt abfährt, erscheint nun Zeile für Zeile

das Bild auf dem Monitor. Ist man mit dem Ergebnis zufrieden, kann man das Bild abspeichern, wobei man noch entscheiden kann, für welche weitere Software das Ergebnis abgelegt werden soll. Hier kann man sich für News Desk-, Fleet Street Editor- oder Master Paint-Format entscheiden!

Bei größeren dunklen Flächen, Zeitungsausschnitten oder Fotografien etc. wird kein gutes Ergebnis erzielt. Knappe schwarz-weiß Darstellungen gelingen recht gut. Im Beispiel auf der nächsten Seite (Kippbild junges Mädchen oder alte Frau???) war das größere Bild die Vorlage, das kleinere die Scanner-Reproduktion auf dem Joyce-Drucker, die allerdings noch ein wenig größer hätte ausfallen können, ohne Qualitätseinbußen hinzunehmen. Das gescannte Bild wurde im Nachhinein nicht mit dem Malprogramm überarbeitet.

Wer sich von der vorherigen guten Kritik über das Joyce-Mouse-Pack hat überzeugen lassen, jetzt aber doch nicht auf einen Scanner verzichten möchte, sei getröstet. Wurde ein Bild gescannt und für das Master Paint-Programm mit der Extension .pic abgespeichert, kann es auch mit dem Reis-Ware Paket bearbeitet werden! Es entsteht lediglich eine minimale Rechtsverschiebung, die leicht korrigiert werden kann. (Anm.: Bis auf die Grafik in diesem Kapitel, sind alle anderen ohne Scanner entstanden!) Das dem Scanner beiliegende recht ordentliche Grafikprogramm weicht von der Steuerung her von der ND-Ausführung ab. Am linken und oberen Rand des Bildschirms erscheinen Symbole, die mit einem kleinen Stift angeklickt werden müssen. Die Symbole sind leicht verständlich (z.B. Radiergummi) und schnell zu merken. Umständlich erscheint nur, daß der Stift keine Menuefenster aufrufen kann, die an seiner Position erscheinen, sondern immer wieder an den Rand geführt werden muß. Insgesamt stehen 37 Befehle zur Verfügung, allerdings gilt hier dasselbe wie für ND: Ohne Eingabemedium (englische Mouse) wird es schwer, nur mit den Tasten ein ordentliches Bild zustande zu bringen. Mir persönlich erscheint es sinnvoll, lediglich Bilder unter den besprochenen Umständen zu scannen und diese dann mit der Gerdes-Mouse zu bearbeiten. Auf das Scanner.-Grafikprogramm könnte man dann verzichten.

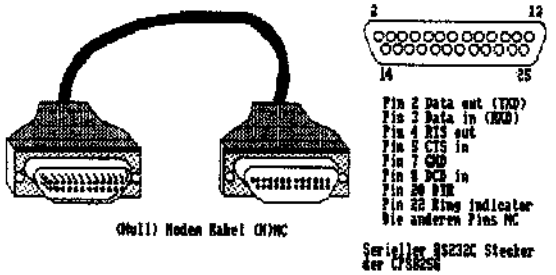
Original gescannt

(Die Vorlage und die gescannte Joyce-Printerausgabe mußten aus drucktechnischen Gründen photomechanisch reproduziert werden. Beide sind in natura ein wenig farbkräftiger.)

Die Datenfernübertragung

In einem kurzen Abriß über sinnvolle Joyce-Hardwareerweiterungen darf ein kleines Kapitel über die Datenfernübertragung (DFÜ) nicht fehlen. Zwar nehmen die Disketten des 3"-Formats nicht viel Platz in Anspruch, so daß Dateien mit diesem Medium leicht via Postversand ausgetauscht werden können, doch gibt es auch Situationen, bei denen es auf ständigen und schnellen oder auf wahlfreien Zugriff aus einer Fülle von Informationen ankommt. Es wäre nicht sinnvoll und auch kaum möglich, den Inhalt ganzer Datenbanken (z.B. Juris) auszutauschen, wenn nur einige bestimmte Auskünfte gewünscht werden.
Die einfachste Möglichkeit, mit dem Joyce in ein fremdes Betriebssystem einzusteigen und mit dessen Daten zu arbeiten, bietet sich mit einem **Akustikkoppler.** Knapp beschrieben wandelt ein Akustikkoppler Signale des Computers in hörbare Signale um, die über ein Telefon an einen anderen Koppler (oder Modem) gelangen und wieder in Signale verwandelt werden, mit denen ein angeschlossener Computer etwas anfangen kann.
Getestet wurde dieser Vorgang am Joyce mit einem Akustikkoppler von Dataphon und zwar mit dem s21/23d (ca. 300,-DM). Für den Joyce werden keine kompletten Pakete mit Kabel und Software ausgeliefert, so daß man sich selbst ein Verbindungskabel zwischen CPS8256 und dem Dataphon beschaffen muß. Bei dem zu verwendenden Kabel handelt es sich um ein "MC" (Modem Kabel); das heißt, daß die Verbindungen gerade durchgeschleift sind also Pin2 an Pin2, Pin3 an Pin3 etc. (Sollen Computer ohne Modem direkt miteinander verbunden werden, so geschieht dies über ein "NMC", dem "Null Modem Kabel". Beim NMC werden die Pins 2 und 3 miteinander vertauscht. Tests mit Atari ST und Joyce Mail232 waren erfolgreich!)
Bei den Steckern wird ein Männchen für das Dataphon und ein Weibchen für die Schnittstelle gebraucht (s. Skizze der folgenden Seite). Am preiswertesten ist es, wenn man sich die Verbindung selbst lötet. Die zwei Stecker und ein halber

Meter achtadriges Datenkabel kosten nur ein paar Mark. Die Pins der Stecker sind durchnummeriert und die Adern des Kabels haben verschiedene Farben, so daß es ein Leichtes ist, die Pins 2-8 der Stecker über das Kabel miteinander zu verbinden.



(Null) Modem Kabel (X)NC

Serieller RS232C Stecker der CPC464

Ein fehlendes besseres DFÜ-Programm als Mail232 läßt sich jetzt bei einem Probelauf leicht beschaffen (Ist aber nicht unbedingt nötig).
Dem Dataphon liegt ein gutes Handbuch bei, so daß ich auf die einzelnen Funktionen nur so weit eingehen möchte, wie es für dieses Beispiel dienlich ist.
Über die RS232 wird das Dataphon mit der Schnittstelle verbunden und für eine ausreichende Stromversorgung (9V Block, Akku oder Netzteil) des Datenföns gesorgt. Nach dem Booten von CP/M wird die Seite 1 der Systemdisketten (Loco-Script) eingelegt und hinter das A> wird MAIL232 eingetippt und so aufgerufen. Die Systemdiskette kann aus dem Laufwerk entnommen und eine formatierte Diskette eingelegt werden. Am oberen Rand des Schirms sind die Menues von Mail232 zu sehen. Durch Druck von [F1] kann man jetzt die Parameter z.B. einer Mailbox seiner Wahl einstellen.
Mailboxen fungieren als Sammelstellen von Nachrichten. Jeder Anwender, der sich dort eingetragen hat (oft kostenlos!), kann unter einem Passwort seiner Wahl Nachrichten für sich hinterlegen lassen oder Nachrichten für andere (auch alle

Mailboxbenutzer - schwarzes Brett -) hinterlegen. In diesen Mailboxen findet man auch Programme der Public Domain, die man sich dort via Akustikkoppler abholen kann.
Für unser Beispiel kann die Mailbox RBBS (0431/336038) in Kiel dienen (viele CP/M Programme). Für diese Box müßten die Parameter von [F1] auf 300 Baud senden, 300 Baud empfangen, 8 Datenbits, keine Parität und 1 Stopbit eingestellt werden. Hardware handshaking kann unterbleiben. Durch Druck auf [EXIT] verschwindet das Menue und die Parameter sind eingestellt. Am Dataphon wird der Bereich Auto gewählt. Danach wird der Telefonhörer in die Gummimuffen des Datenföns gepreßt und das Ganze auf die Seite gelegt (Bei langem Betrieb mit Hörer nach unten können Kohlemembranen älterer Telefone zusammenbacken und so den Schallpegel senken). Jetzt kann die Telefonnummer der Mailbox angewählt werden. Kommt die Verbindung zustande, kann der zweite Schalter des Dataphons von off auf 300 plaziert werden. (Geschieht das vor Zustandekommen der Verbindung, erscheinen oft noch Störzeichen auf den Monitor).
Nach kurzer Zeit meldet sich die Mailbox auf dem Bildschirm und gibt dem Anwender genaue Hinweise, z.B. was er beachten muß, wie er sich einzutragen hat und wie Programme gelesen werden können.
All das, was über den Monitor läuft, kann Joyce speichern. Durch Druck auf [F3] erscheint ein Fenster, in das man einen Dateinamen für das zu Empfangende (oder zu Sendende) einträgt. Bei Druck auf Exit geschieht nichts weiter, als daß Joyce sich diesen Namen merkt. Läuft jetzt ein interessantes Programm über den Monitor, braucht der Anwender nur nochmals [F3] zu drücken und durch [ENTER] zu bestätigen. Jetzt wird eine Datei mit diesem Namen auf Diskette angelegt und alles, was über den Monitor läuft, wandert in den Speicher von Joyce. Ist dieser voll, wird durch Abspeichern in die Diskettendatei neuer Platz geschaffen. Hat der Anwender alle Informationen erhalten, wird durch gleichzeitigen Druck auf [ALT] und [STOP] der restliche Speicherinhalt auf Diskette geschrieben und die Datei geschlossen. Sie steht jetzt dem Anwender zur Verfügung.

Auf diese Weise kann man sich nun über eine Mailbox komplexere Programme zur DFÜ beschaffen (z.B. Kermit) und der Welt der Datenfernübertragung steht nichts mehr im Weg.
An dieser Stelle noch ein wichtiger Tip:
Wird Mail232 von Laufwerk A aus gestartet, muß Joyce seinen Zwischenspeicher immer wieder dahin entleeren. Bei dieser brummigen Arbeit gehen ihm immer einige Zeichen verloren, die der Sender in der Zwischenzeit munter weitergibt. Um dies zu verhindern, ist es zweckmäßig, Mail232 in den das Laufwerk M zu holen, und M als Standardlaufwerk anzusprechen. ("M:" schreiben und RETURN drücken). So werden alle Signale gleich in der Ramdisk abgelegt, wobei kein Zeichen verloren geht. Bei der Einstellung 300 Baud kann unser Joyce dann mehr als eine Stunde seinen erweiterten Speicher vollschreiben.
Zur Erreichung dieses Ziels muß allerdings eine Hürde genommen werden! Mail232 ist als "System-Datei" auf Diskette zu finden. Wenn man sie mit der Dateiverwaltung von Locoscript ansehen will, geht das nur, wenn der Modus "Versteckte Dateien sichtbar" eingeschaltet ist. Solche Systemdateien können nicht mit **Pip** in den M-Speicher geholt werden. Hier hilft uns das CP/M-Programm **Set** weiter. Mit der Dateiverwaltung von Locoscript kopiert man sowohl **Set** als auch **Mail232** auf eine Diskette. Nach dem Neubooten von CP/M wird diese Diskette ins Laufwerk geschoben und eingetippt: **set mail232.com dir** . **Set** wird dann aus der Systemdatei **Mail232** eine normale Datei herstellen, die wie jede andere auch mit **Pip** nach "M" kopiert werden kann.
Zum Experimentieren hier weitere Mailboxen und ihre dazugehörigen Parameter:

```
Data Becker 0211/340071 24h online 300/8/N/1
IBB 030/6818679 24h online 300/7/N/1
MCS 040/2512371  & 2512373 24h online 300/7/E/1
c.l.i.n.c.h 040/6323517 24h online 300/8/N/1
OIS München 089/4606021 24h online 300/8/N/1
```

In den Mailboxen sind Listen über Rufnummern anderer Boxen erhältlich. (In fast jeder größeren Stadt existiert mindestens eine!)

Der Joyce nach Feierabend

Nach der Fülle von trockenem Stoff und zum Abschluß nun zu einem leichteren Gebiet, auf dem der Joyce eigentlich nichts zu suchen hat.
In den Anwenderkreisen, in denen der Joyce zu Hause ist, wurde beim Kauf sicher nicht in Erwägung gezogen, daß zur Entspannung auch Spiele auf ihm gestartet werden können.
Gegenüber einigen Soundmaschinen mit Buntmonitor, Joystick, Trackball und dergleichen mehr, die den Eindruck machen, als seien sie einer Spielhöhle entstiegen, um Kinder von ihrem pädagogisch wertvollen Spielzeug wegzulocken, gibt sich der Joyce mit seinem Grünmonitor und dem kleinen Piepser zugegebenermaßen sehr bieder. Dementsprechend spät wurde auch erst die Software für ihn angepaßt, die in erster Linie der Entspannung (oder auch nicht) dienen sollte. Die Softwareentwickler und -händler dieses Genres ließen sich Zeit, und als sie dann ihre Waren auf den Markt brachten, waren diese auf das Käuferpotential der PCW's abgestimmt.
"Weltraumballerkillspiele" wurden erst gar nicht ins Repertoire aufgenommen. Dafür gab es gleich zu Anfang drei verschiedene Schachspiele für den Joyce, wobei das "Cyrus-Chess" bei den mir bekannten Usern zum Favoriten avancierte.
Bald danach erschienen die ersten Textadventures wie "Silikon Dreams" (mit ein wenig Grafik) oder "Lord of the Rings", bei denen man sich durch einen Roman lesen muß, und durch Anweisungen wie "go, put, take" oder "read" den Held der Geschichte durch ein Gestrüpp von Widernissen lenken muß. Daß diese Adventures in Englisch gehalten waren, diente so manchem User als Ausrede für seinen Spieltrieb, konnte er doch die Aufbesserung seiner nachlassenden Englischkenntnisse vorschieben.
Geschicklichkeitsspiele wie "Bounder", bei dem ein Ball hüpfender Weise über ein Feld voll böser Hindernisse gelenkt werden mußte (für Eingeweihte: cheat-mode: gleichzeitig die Tasten "c"h"e"a"t drücken) oder Flugsimulatoren wie "Tomahawk" (Hubschrauber) zogen nach.
Die absoluten Spielehits für den Joyce kamen aber immer nur

von einer Firma: OCEAN.
Der größte Erfolg dieses Softwarehauses war ein Spiel namens "Batman", wobei diesmal der Spruch vom nomen, welches omen sein soll, nicht zutraf. Zwar war das Titelbild noch arg vom stupiden Vorbild der Comicfigur beeinflußt, doch das Spiel selbst schlug alles Bisherige. Grafik und Sound des Joyce waren voll ausgereizt und das Spiel an sich verlangte dem Joycer soviel Witz und Verstand, Erinnerungsvermögen und spielerisches Geschick ab, daß der Reiz dieses Spieles so manchen eine Masse Zeit gekostet hat. Eigentlich ist der Spieler nur damit beschäftigt, eine kleine quirlige Figur durch ein Labyrinth von 150! Räumen zu schicken und sie dabei 8 Teile für ein Flugzeug sammeln zu lassen, mit dem es dann fliehen kann. Eigentlich ist aber die ganze Aktion eine einzige Flucht vor Fallen, bösen Maschinen oder finsteren Gestalten. Geballert wird nicht, nur weggelaufen, wobei "nur" eine schlimme Untertreibung ist. Wer das Spiel nicht kennt, sollte es ausprobieren. Ich habe keinen Joycer kennengelernt, der nicht begeistert war!
Von diesem Erfolg angespornt lieferte OCEAN dann gleich einen weiteren Hit nach: "Head over Heals". Hier galt es fast doppelt soviele Räume mit zwei Figuren, die man zwecks Erweiterung ihrer Fähigkeiten übereinanderstellen konnte, zu erkunden. Wurde Joyce mit einem Sound Digitizer und einem Joystickkontroller englischer Herkunft ausgestattet, hatte man neben dem bequemen Knüppeleingabemedium sogar noch ständig eine Rockmelodie im Ohr.
Als weiteres Produkt für den Joyce erschien dann "Match-Day II" von OCEAN. Hier wird sehr realistisch ein Fußballspiel simuliert, wobei der User in der Lage ist, zwischen einer Menge von Möglichkeiten wie: Austragung eines Pokals, Computer gegen Computer oder zwei Spieler gegen Computer spielen zu lassen etc. wählen kann. Auch mit diesem Spiel setzte OCEAN die Tradition, hochwertige Spiele für den Joyce zu liefern, fort.
Das jüngste der qualitativ hochwertigen Spiele, war ein Geschicklichkeitsspiel, das schon auf anderen Computern zum Spiel des Jahres ernannt worden war: "Tetris". Hier fallen

geometrische Figuren vom Computerhimmel, und der gequälte Anwender muß versuchen, sie durch geschickte Drehung und Placierung so aufeinander fallen zu lassen, daß sie möglichst wenig Raum einnehmen und immer den Boden bedecken. Nur so können folgende Schichten nachsacken und es verhindern, daß die Türme in den Himmel wachsen, was den Abbruch des Spiels bedeuten würde. Ein simples Spiel mit ungeheurem Spielreiz!
Wer nun bei diesem knappen Review, das vieles aus dem Spielesektor nicht erwähnt hat, auf den Geschmack gekommen ist und bisher nicht daran gedacht hat, mal mit oder gegen seinen Joyce zu spielen, der kann, ohne Geld im nächsten Softwareladen auszugeben, das folgende kleine BASIC-Listing abtippen und mal sehen, wie sich Joyce denn so als Spielpartner macht. Bei dem winzigen Bonbon zum Abschluß handelt es sich um "Kniffel", ein Würfelspiel, daß wohl den meisten vertraut sein dürfte (von einer Karte mit Ergebnisvorlagen muß jeweils nach dreimaligem Würfeln pro Durchgang eine Spalte ausgefüllt werden, wobei man zumindest zu Anfang in der Wahl der Spalten frei ist). Die Qualität des Spiels entspricht zwar nicht gerade dem Standart von Ocean, dafür wurde es aber selbst entwickelt. Wer sich das Abtippen ersparen möchte, findet dieses Programm ready to run auf der Diskette zum Buch. Wir wünschen:

Viel Spaß dabei!

```
10 'Spielprogramm: Kniffel
20 '
30 '
40 stu$=HEX$(PEEK(&HFBF6))
50 min$=HEX$(PEEK(&HFBF7))
60 sec$=HEX$(PEEK(&HFBF8))
70 date=PEEK(&HFBF4)+PEEK(&HFBF5)*256
80 random=date-VAL(stu$)-VAL(min$)-VAL(sec$)
90 RANDOMIZE random
100 '
110 'Copyright 1988 by Bernhard Graßhoff, Heikendorf
120 '
130 'Initialisierung
140 DEFINT a-z
150 DIM z(6,13)
160 GOSUB 3040'Maschinenprogramm INPUT
```

```
170 CALL disable% 'Ctrl-C und Ctrl-S außer Betrieb
180 esc$=CHR$(27):co$=esc$+"e":cf$=esc$+"f":csave$=esc$+"j":cload$=esc$+"k":invon$=esc$+"p
":invoff$=esc$+"q":uon$=esc$+"r":uoff$=esc$+"u":umon$=esc$+"v":umoff$=esc$+"w":home$=esc$+
"H":cls$=esc$+"E"+home$:links$=esc$+"D"+esc$+"D"+" "
190 bell$=CHR$(7):deutsch$=esc$+"2"+CHR$(2)
200 normal$=esc$+"X"+CHR$(33)+CHR$(33)+CHR$(60)+CHR$(119)'Fenster auf ganzen Bildschirm au
sweiten
210 DEF FNfenster$(oz,ls,h,b)=esc$+"X"+CHR$(32+oz)+CHR$(32+ls)+CHR$(31+h)+CHR$(31+b) 'Fens
ter einrichten
220 '
230 PRINT FNfenster$(0,0,31,90) 'Fenster auf ganzen Bildschirm ausweiten
240 PRINT cls$
250 a=3:b=20:c=13:d=50:GOSUB 2870 'Rahmen zeichnen
260 PRINT FNfenster$(4,21,10,49) 'Fenster für 1. Spielmeldung einrichten
270 PRINT "          "CHR$(164)" 1988 by Bernhard Grabhoff":PRINT
280 PRINT "            ";invon$+"  K  N  I  F  F  E  L  "+invoff$
290 PRINT:PRINT:PRINT "    Wv. Spieler (1-6) nahmen am Spiel teil ?":PRINT
300 PRINT "      Bitte Zahl eingeben !   ";
310 OPTION RUN
320 a$=INPUT$(1)
330 spieler=VAL (a$):IF spieler<1 OR spieler>6 THEN 320
340 OPTION STOP
350 PRINT FNfenster$(0,0,31,90) 'Fenster auf ganzen Bildschirm ausweiten
360 PRINT cls$
370 a=0:b=0:c=31:d=88:PRINT cf$;umoff$;:GOSUB 2870 'Rahmen zeichnen
380 '
390 FOR runde=1 TO 13
400 FOR p=1 TO spieler
410 '
420 GOSUB 440:GOTO 480
430 '
440 FOR i=1 TO 5
450 t(i)=INT (RND (1)*6)+1'Würfeln Wert zuweisen
460 NEXT i:RETURN
470 '
480 GOSUB 520:GOTO 610
490 '
500 'Eingefügtes Unterprogramm 'Sortieren der Würfel-Werte'
510 '
520 f=0
530 FOR i=1 TO 4
540 IF t(i+1)>=t(i) THEN 570
550 SWAP t(i),t(i+1)
560 f=1
570 NEXT i
580 IF f=1 THEN 520
590 RETURN
600 '
610 PRINT FNfenster$(1,2,28,87) 'Fenster für Spielfeld einrichten
620 '
630 PRINT cls$
640 PRINT " Spieler";p
650 PRINT:PRINT
660 PRINT " Sie haben folgende Zahlen gewürfelt:":PRINT
670 FOR f=1 TO 5
680 PRINT "          Würfel";f;"=";t(f)
690 NEXT
700 '
```



```
710 PRINT FNfenster$(12,1,21,90) 'Fenster für Spielkartenausgabe einrichten
720 '
730 PRINT "  Hier ist Ihre Kniffelkarte":PRINT
740 PRINT " 1er  =1            ";ei(p);:IF ei(p)=0 AND z(p,1)=0 THEN PRINT links$:ELSE PRI
NT
750 PRINT " 2er  =2            ";zw(p);:IF zw(p)=0 AND z(p,2)=0 THEN PRINT links$:ELSE PRI
NT
760 PRINT " 3er  =3            ";dr(p);:IF dr(p)=0 AND z(p,3)=0 THEN PRINT links$:ELSE PRI
NT
770 PRINT " 4er  =4            ";vi(p);:IF vi(p)=0 AND z(p,4)=0 THEN PRINT links$:ELSE PRI
NT
780 PRINT " 5er  =5            ";fu(p);:IF fu(p)=0 AND z(p,5)=0 THEN PRINT links$:ELSE PRI
NT
790 PRINT " 6er  =6            ";se(p);:IF se(p)=0 AND z(p,6)=0 THEN PRINT links$:ELSE PRI
NT
800 PRINT " Dreierpasch =7     ";dp(p);:IF dp(p)=0 AND z(p,7)=0 THEN PRINT links$:ELSE PRI
NT
810 PRINT " Viererpasch =8     ";vp(p);:IF vp(p)=0 AND z(p,8)=0 THEN PRINT links$:ELSE PRI
NT
820 PRINT " Full-House  =9     ";fh(p);:IF fh(p)=0 AND z(p,9)=0 THEN PRINT links$:ELSE PRI
NT
830 PRINT " Kleine Str. =10    ";ks(p);:IF ks(p)=0 AND z(p,10)=0 THEN PRINT links$:ELSE PR
INT
840 PRINT " Große  Str. =11    ";gr(p);:IF gr(p)=0 AND z(p,11)=0 THEN PRINT links$:ELSE PR
INT
850 PRINT " Kniffel     =12    ";kn(p);:IF kn(p)=0 AND z(p,12)=0 THEN PRINT links$:ELSE PR
INT
860 PRINT " Chance      =13    ";ch(p);:IF ch(p)=0 AND z(p,13)=0 THEN PRINT links$:ELSE PR
INT
870 FOR wn=1 TO 2
880 '
890 'Beginn des Dialoges
900 '
910 PRINT FNfenster$(13,44,5,43):PRINT cls$;"Wv. Würfel sollen neu gesetzt werden ?";csave
$;
920 PRINT co$;cload$;:maxlen=1:GOSUB 2990:PRINT cf$;
930 IF x$="1" THEN n=1:GOTO 1000
940 IF x$="2" THEN n=2:GOTO 1000
950 IF x$="3" THEN n=3:GOTO 1000
960 IF x$="4" THEN n=4:GOTO 1000
970 IF x$="5" THEN n=5:GOTO 1000
980 IF x$="0" THEN 1380
990 PRINT bell$;cload$;"  ";:GOTO 920
1000 PRINT:IF n=5 THEN GOSUB 440:GOTO 1210
1010 PRINT FNfenster$(13,44,10,44):PRINT cls$;
1020 PRINT co$;
1030 '
1040 'Neue Würfel-Wert Zuweisung
1050 '
1060 FOR nn=1 TO n
1070 PRINT "Würfelnummer=";csave$;
1080 maxlen=1:GOSUB 2990:nu(nn)=VAL(x$):IF nu(nn)<1 OR nu(nn)>5 THEN PRINT bell$;cload$;:G
OTO 1080 ELSE PRINT
1090 NEXT nn
1100 PRINT cf$
1110 f=0 'Sortierung der neu zu würfelnden Werte, um Fehlfunktion auszuschließen
1120 FOR nn=1 TO n-1
1130 IF nu(nn+1)>=nu(nn) THEN 1160
```

```
1140 SWAP nu(nn),nu(nn+1)
1150 f=1
1160 NEXT
1170 IF f=1 THEN 1110
1180 FOR nn=1 TO n
1190 t(nu(nn))=INT(RND(1)*6)+1
1200 NEXT nn
1210 GOSUB 520
1220 PRINT
1230 PRINT home$;
1240 '
1250 'Ausgabe der Werte in entspr. Fenstern
1260 '
1270 IF wn=2 THEN PRINT FNfenster$(6,65,7,15):GOTO 1290
1280 PRINT FNfenster$(6,39,7,15)
1290 FOR i=1 TO 5
1300 PRINT "Würfel"i"="t(i)
1310 NEXT i
1320 '
1330 PRINT normal$
1340 NEXT wn
1350 PRINT:PRINT normal$:GOTO 1380
1360 PRINT bell$;
1370 '
1380 PRINT FNfenster$(13,43,5,42):PRINT cls$;"Wo wollen Sie die Augenzahl eintragen?";csav
e$
1390 PRINT co$;cload$;:max(en=2:GOSUB 2990:we=VAL(x$):PRINT cf$;
1400 '
1410 'Prüfen der Eingaben auf Richtigkeit und Setzen einzelner Flags
1420 '
1430 IF we<1 OR we>13 THEN 1450
1440 ON we GOTO 1460,1510,1560,1610,1660,1710,1760,1800,1840,1880,1910,1950,1980
1450 PRINT bell$;:GOTO 1390
1460 n=1:IF z(p,1)=1 THEN 1360 ELSE z(p,1)=1
1470 GOSUB 2030:GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1480 ei(p)=e
1490 e=0
1500 GOTO 2090
1510 n=2:IF z(p,2)=1 THEN 1360:ELSE z(p,2)=1
1520 GOSUB 2030:GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1530 zw(p)=e
1540 e=0
1550 GOTO 2090
1560 n=3:IF z(p,3)=1 THEN 1360:ELSE z(p,3)=1
1570 GOSUB 2030:GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1580 dr(p)=e
1590 e=0
1600 GOTO 2090
1610 n=4:IF z(p,4)=1 THEN 1360:ELSE z(p,4)=1
1620 GOSUB 2030:GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1630 vi(p)=e
1640 e=0
1650 GOTO 2090
1660 n=5:IF z(p,5)=1 THEN 1360:ELSE z(p,5)=1
1670 GOSUB 2030:GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1680 fu(p)=e
1690 e=0
1700 GOTO 2090
```



```
1710 n=6:IF z(p,6)=1 THEN 1360:ELSE z(p,6)=1
1720 GOSUB 2030:GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1730 se(p)=e
1740 e=0
1750 GOTO 2090
1760 zp=3:n=5:IF z(p,7)=1 THEN 1360:ELSE z(p,7)=1:GOSUB 2370
1770 GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1780 dp(p)=g
1790 GOTO 2090
1800 zp=4:n=5:IF z(p,8)=1 THEN 1360:ELSE z(p,8)=1:GOSUB 2370
1810 GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1820 vp(p)=g
1830 GOTO 2090
1840 GOSUB 2450:IF z(p,9)=1 THEN 1360:ELSE z(p,9)=1
1850 GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1860 fh(p)=g
1870 GOTO 2090
1880 GOSUB 2500:IF z(p,10)=1 THEN 1360:ELSE z(p,10)=1
1890 IF y>=3 THEN ks(p)=30:ELSE ks(p)=0
1900 GOTO 2090
1910 GOSUB 2500:IF z(p,11)=1 THEN 1360:ELSE z(p,11)=1
1920 IF y=4 THEN gr(p)=40:ELSE gr(p)=0
1930 GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
1940 GOTO 2090
1950 IF z(p,12)=1 THEN 1360:ELSE z(p,12)=1:GOSUB 2550
1960 IF y=4 THEN kn(p)=50:ELSE kn(p)=0
1970 GOTO 2090
1980 IF z(p,13)=1 THEN 1360:ELSE z(p,13)=1
1990 FOR i=1 TO 5
2000 ch(p)=ch(p)+t(i)
2010 NEXT i:GOSUB 2550:IF y=4 THEN gy(p)=gy(p)+100
2020 GOTO 2090
2030 IF t(1)=n THEN e=e+n
2040 IF t(2)=n THEN e=e+n
2050 IF t(3)=n THEN e=e+n
2060 IF t(4)=n THEN e=e+n
2070 IF t(5)=n THEN e=e+n
2080 RETURN
2090 PRINT:NEXT p 'Höchster Spieler
2100 PRINT:NEXT runde
2110 FOR i=1 TO 30:t$=INKEY$:NEXT 'Leeren des Tastaturpuffers
2120 PRINT normal$:PRINT cls$:PRINT SPC(34);uon$;" A B R E C H N U N G ";uoff$
2130 '
2140 spielermax=0
2150 a=5:b=28:c=17:d=34:GOSUB 2890
2160 PRINT FNfenster$(6,29,14,33)
2170 FOR i=1 TO spieler
2180 PRINT home$;cls$;"Spieler"i"hat:":PRINT
2190 q(i)=ei(i)+zw(i)+dr(i)+vi(i)+fu(i)+se(i)
2200 PRINT "Oberer Teil      : ";q(i)
2210 IF q(i)>=63 THEN q(i)=q(i)+35:bonus=35
2220 PRINT "Bonus            : ";bonus:bonus=0
2230 unten=dp(i)+vp(i)+fh(i)+ks(i)+gr(i)+kn(i)+ch(i)
2240 PRINT "Unterer Teil     : ";unten
2250 IF kn(i)=50 THEN q(i)=q(i)+gy(i) ELSE gy(i)=0
2260 PRINT "Extrakniffel     : ";gy(i)
2270 PRINT "_____________________________"
2280 q(i)=q(i)+unten
```

```
2290 IF q(f)>q(spielermax) THEN spielermax=i
2300 PRINT:PRINT "Gesamt           : "q(i)"Punkte"
2310 t$=INPUT$(1)
2320 NEXT i
2330 PRINT co$:PRINT "Gewonnen hat Spieler "spielermax
2340 PRINT:PRINT "Nochmal Ä¿/NÜ";:maxlen=1:GOSUB 2990:IF UPPER$(x$)="J" THEN PRINT:RUN
2350 PRINT FNfenster$(0,0,31,90)
2360 PRINT cls$:END
2370 GOSUB 520
2380 y=0:FOR i=1 TO 4
2390 IF t(i)=t(i+1) THEN y=y+1
2400 NEXT:ok=0
2410 IF zp=3 THEN GOSUB 2590
2420 IF zp=4 THEN GOSUB 2670
2430 IF ok=1 THEN g=t(1)+t(2)+t(3)+t(4)+t(5)+t(6):ELSE g=0
2440 RETURN
2450 GOSUB 520
2460 GOSUB 2780
2470 IF ok=1 THEN g=25
2480 IF ok=0 THEN g=0
2490 RETURN
2500 GOSUB 520
2510 y=0:FOR i=1 TO 4
2520 IF t(i)=t(i+1)-1 THEN y=y+1
2530 NEXT
2540 RETURN
2550 y=0:FOR i=1 TO 4
2560 IF t(i)=t(i+1) THEN y=y+1
2570 NEXT
2580 RETURN
2590 FOR i=1 TO 3
2600 IF t(i)=t(i+1) THEN bh=1:ELSE bh=0
2610 IF bh=1 THEN 2640
2620 NEXT
2630 ok=0:RETURN
2640 IF t(i)=t(i+2) THEN ok=1:RETURN
2650 GOTO 2620
2660 ok=0:RETURN
2670 FOR i=1 TO 4
2680 IF t(i)=t(i+1) THEN 2710
2690 GOTO 2750
2700 ok=0:RETURN
2710 IF t(i)=t(i+2) THEN 2740
2720 GOTO 2750
2730 ok=0:RETURN
2740 IF t(i)=t(i+3) THEN 2770
2750 NEXT i
2760 ok=0:RETURN
2770 ok=1:RETURN
2780 IF t(1)=t(2) THEN 2800
2790 ok=0:RETURN
2800 IF t(4)=t(5) THEN 2820
2810 ok=0:RETURN
2820 IF t(3)=t(2) THEN 2850
2830 IF t(3)=t(4) THEN 2850
2840 ok=0:RETURN
2850 ok=1:RETURN
2860 '
```

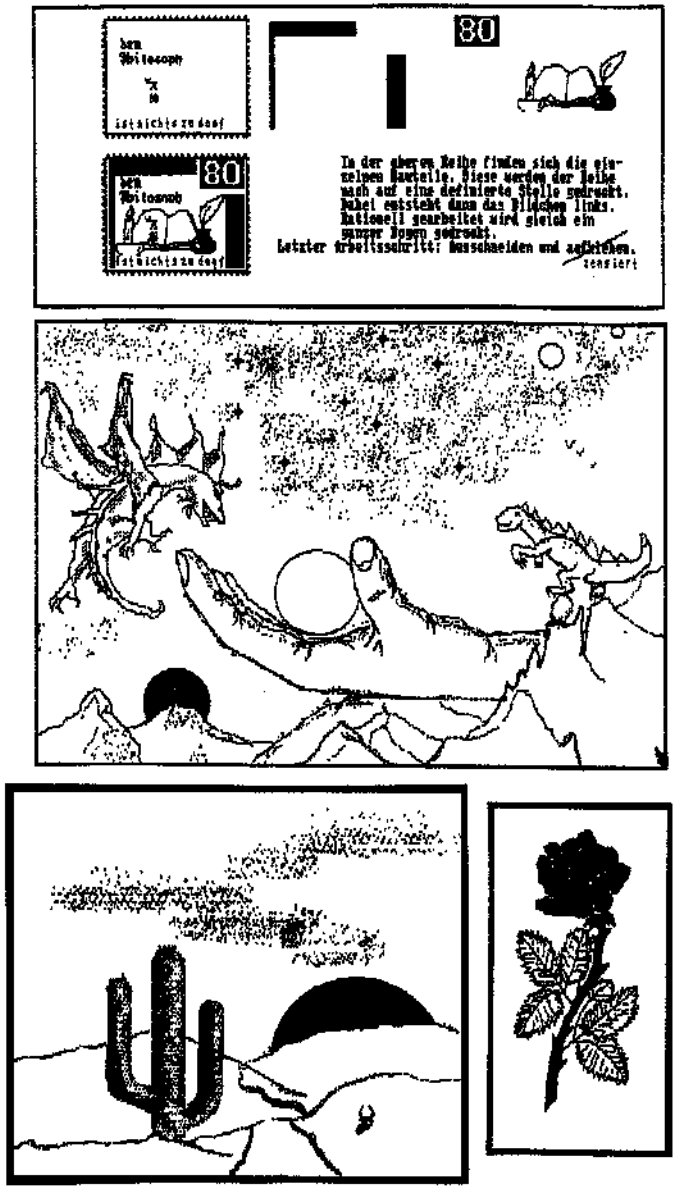


```
2870 'Rahmen zeichnen
2880 '
2890 PRINT FNfenster$(a-1,b-1,c+3,d+3)
2900 PRINT home$;CHR$(134);'Ecke links oben
2910 PRINT STRING$(d,CHR$(138));'Balken waagerecht
2920 PRINT CHR$(140);:PRINT umon$'Ecke rechts oben
2930 FOR i=1 TO c-2:PRINT CHR$(133);STRING$(d," ");CHR$(133):NEXT
2940 PRINT CHR$(131);
2950 PRINT STRING$(d,CHR$(138));
2960 PRINT CHR$(137);
2970 PRINT home$;
2980 PRINT FNfenster$(a,b+2,c+2,d+2):RETURN
2990 x$=""
3000 POKE &H80,maxlen:POKE &H82,0
3010 CALL dskres%:CALL inp%
3020 FOR yyy=&H82 TO &H81+PEEK(&H81):x$=x$+CHR$(PEEK(yyy)):NEXT
3030 RETURN
3040 REM --- DATA ---
3050 DATA 5c,0c,00,00,0e,68,11,a2,f5,cd,05,00,c9
3060 DATA 00,00,00,00,0e,69,11,af,f5,cd,05,00,c9
3070 DATA 59,0e,04,cd,05,00,c9,00
3080 DATA 11,00,00,0e,0a,cd,05,00,c9
3090 DATA 0e,0d,cd,05,00,c9
3100 DATA 11,0a,00,0e,6d,cd,05,00,c9
3110 MEMORY &HF5A1:RESTORE 3050:FOR i=&HF5A2 TO &HF5DB:READ a$:POKE i,VAL("&H"+a$):NEXT
3120 dset%=&HF5A6:dget%=&HF5B3:dmp%=&HF5BC:inp%=&HF5C4:dskres%=&HF5CD
3130 disable%=&HF5D3
3140 RETURN
```

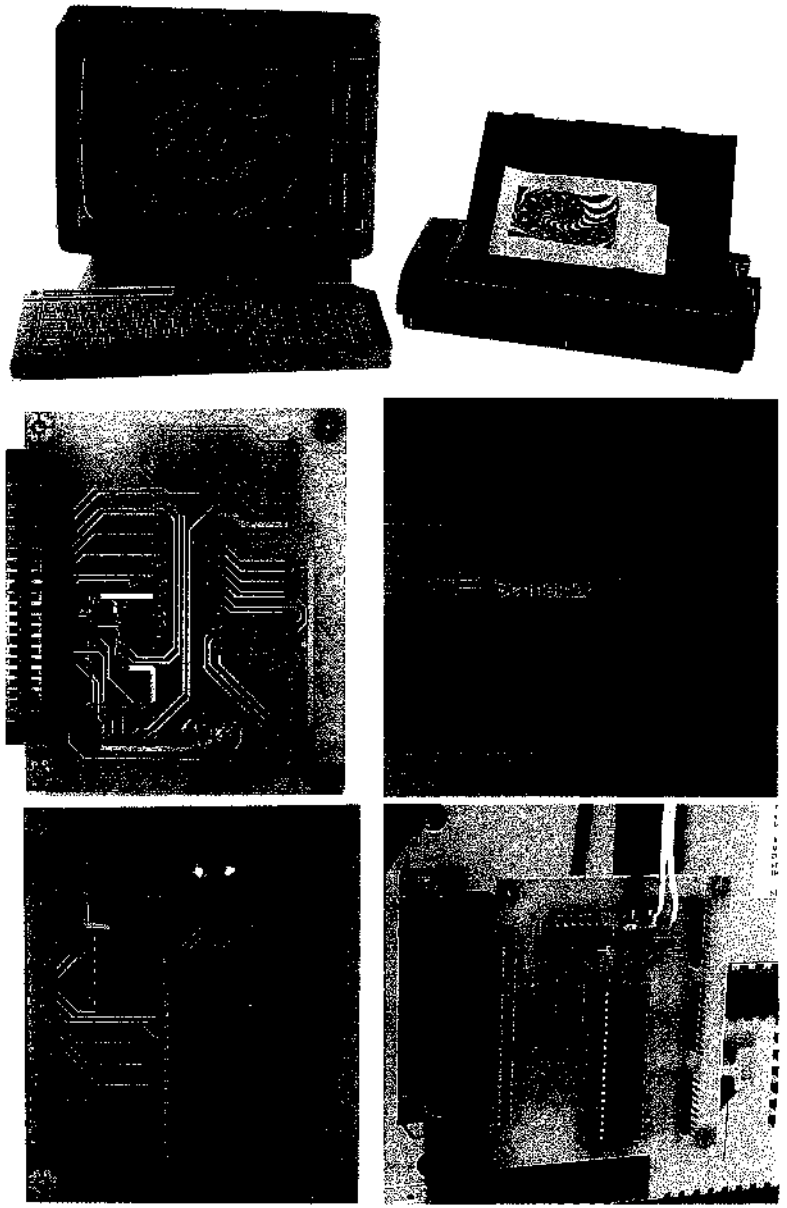
Stichwortverzeichnis

Vollständige ASCII-Code-Tabelle

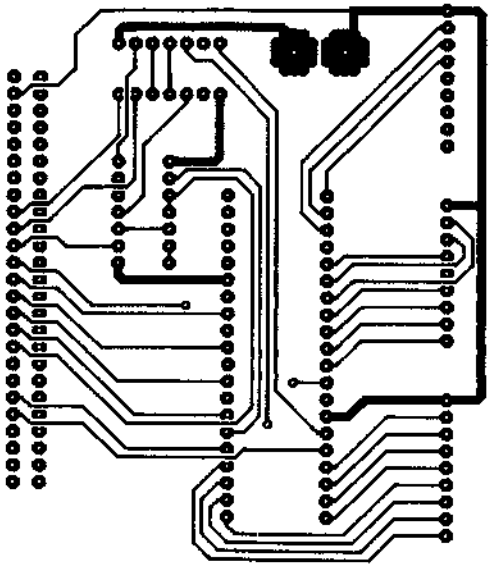
0= ▮	1= ☺	2= ☻	3= ♥	4= ♦	5= ♣	6= ♠	7= •	8= ◘
9= ○	10= ◙	11= ♂	12= ♀	13= ♪	14= ♫	15= ☼	16= ►	17= ◄
18= ↕	19= ‼	20= ¶	21= §	22= ▬	23= ↨	24= ↑	25= ↓	26= →
27= ←	28= ∟	29= ↔	30= ▲	31= ▼	32=	33= !	34= "	35= #
36= $	37= %	38= &	39= '	40= (	41=)	42= *	43= +	44= ,
45= -	46= .	47= /	48= 0	49= 1	50= 2	51= 3	52= 4	53= 5
54= 6	55= 7	56= 8	57= 9	58= :	59= ;	60= <	61= =	62= >
63= ?	64= @	65= A	66= B	67= C	68= D	69= E	70= F	71= G
72= H	73= I	74= J	75= K	76= L	77= M	78= N	79= O	80= P
81= Q	82= R	83= S	84= T	85= U	86= V	87= W	88= X	89= Y
90= Z	91= [	92= \	93=]	94= ↑	95= _	96= `	97= a	98= b
99= c	100= d	101= e	102= f	103= g	104= h	105= i	106= j	107= k
108= l	109= m	110= n	111= o	112= p	113= q	114= r	115= s	116= t
117= u	118= v	119= w	120= x	121= y	122= z	123= {	124= |	125= }
126= ~	127= ⌂	128= Ç	129= ü	130= é	131= â	132= ä	133= à	134= å
135= ç	136= ê	137= ë	138= è	139= ï	140= î	141= ì	142= Ä	143= Å
144= É	145= æ	146= Æ	147= ô	148= ö	149= ò	150= û	151= ù	152= ÿ
153= Ö	154= Ü	155= ø	156= £	157= Ø	158= ×	159= ƒ	160= á	161= í
162= ó	163= ú	164= ñ	165= Ñ	166= ª	167= º	168= ¿	169= ®	170= ¬
171= ½	172= ¼	173= ¡	174= «	175= »	176= ▒	177= ▓	178= │	179= ┤
180= ╡	181= ╢	182= ╖	183= ╕	184= ╣	185= ║	186= ╗	187= ╝	188= ╜
189= ╛	190= ┐	191= └	192= ┴	193= ┬	194= ├	195= ─	196= ┼	197= ╞
198= ╟	199= ╚	200= ╔	201= ╩	202= ╦	203= ╠	204= ═	205= ╬	206= ╧
207= ╨	208= ╤	209= ╥	210= ╙	211= ╘	212= ╒	213= ╓	214= ╫	215= ╪
216= ┘	217= ┌	218= █	219= ▄	220= ▌	221= ▐	222= ▀	223= α	224= ß
225= Γ	226= π	227= Σ	228= σ	229= µ	230= τ	231= Φ	232= Θ	233= Ω
234= δ	235= ∞	236= φ	237= ε	238= ∩	239= ≡	240= ±	241= ≥	242= ≤
243= ⌠	244= ⌡	245= ÷	246= ≈	247= °	248= ∙	249= ·	250= √	251= ⁿ
252= ²	253= ■	254= ▪	255=					



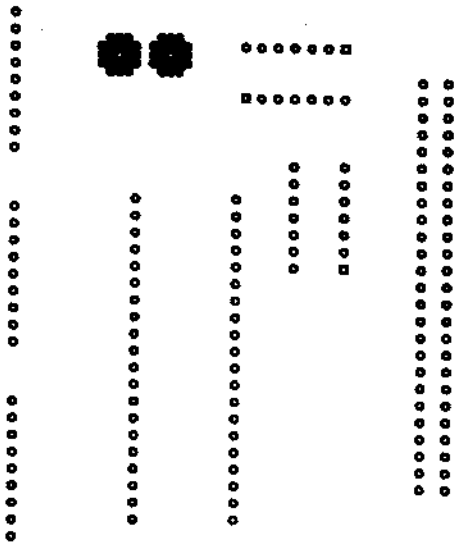
Buntgrafik erstellt mit dem Joycedrucker. (Vgl. S. 280f)



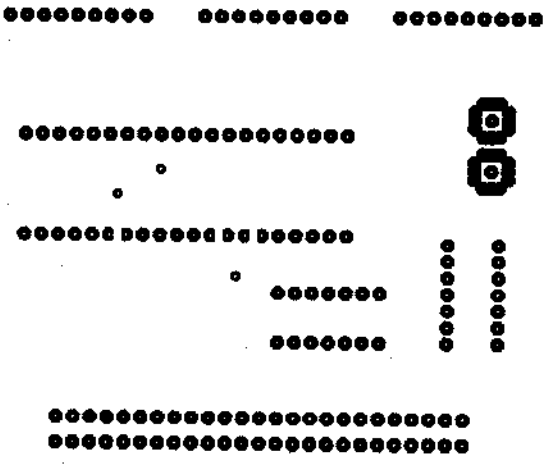
Detailfotos zum Hardware-Kapitel (vgl. Seiten 258 - 276)



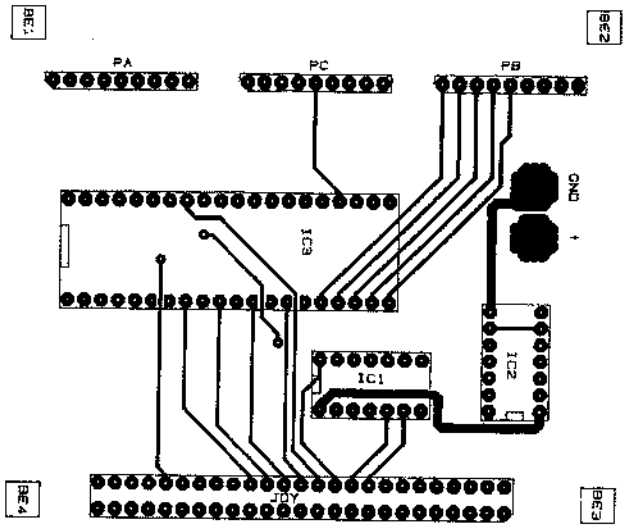
Layout unten
(Vorlagen mit Tuscheseite)
(auf Kupferfolie legen !)



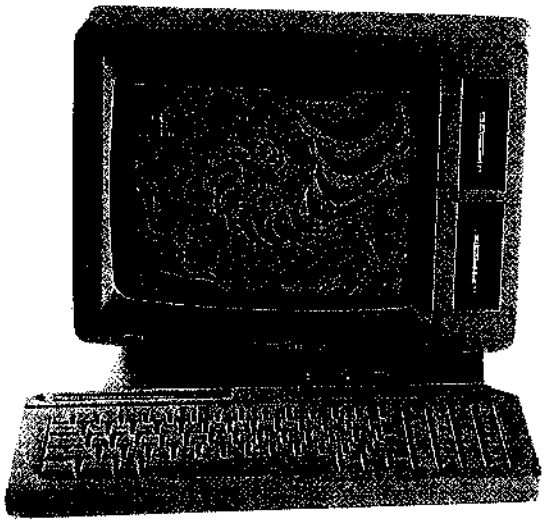
Loetstopmaske

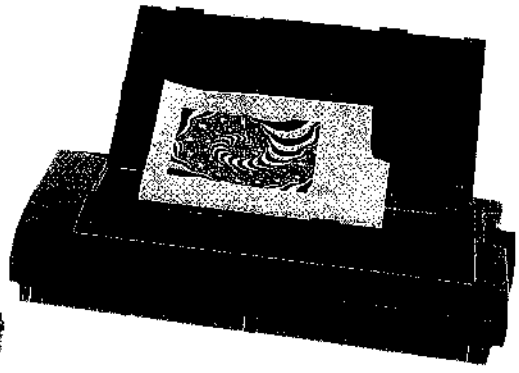


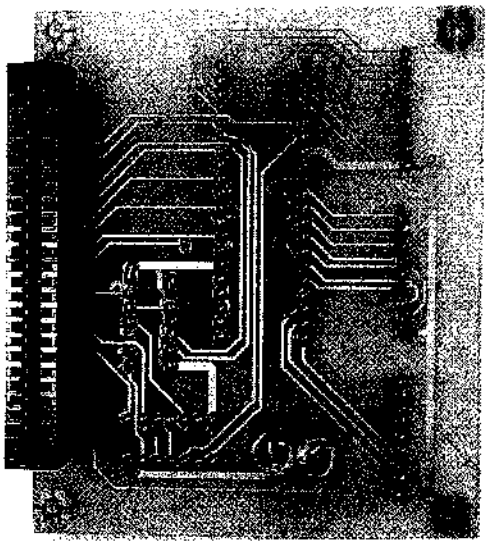
Bohrplan

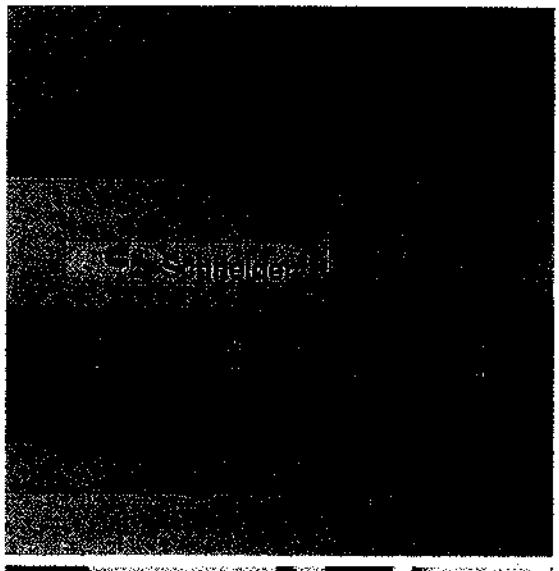


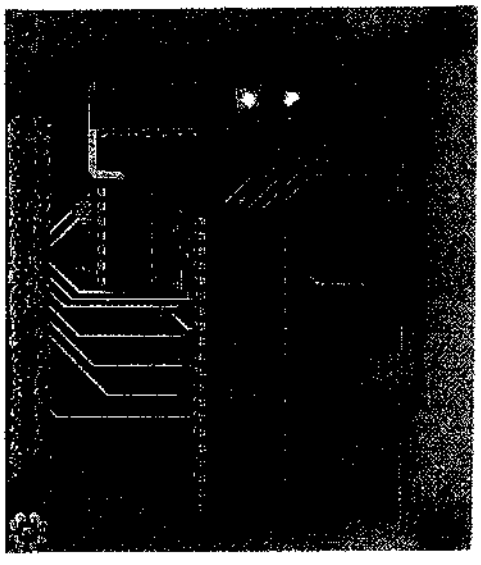
Layout oben
(gespiegelt)

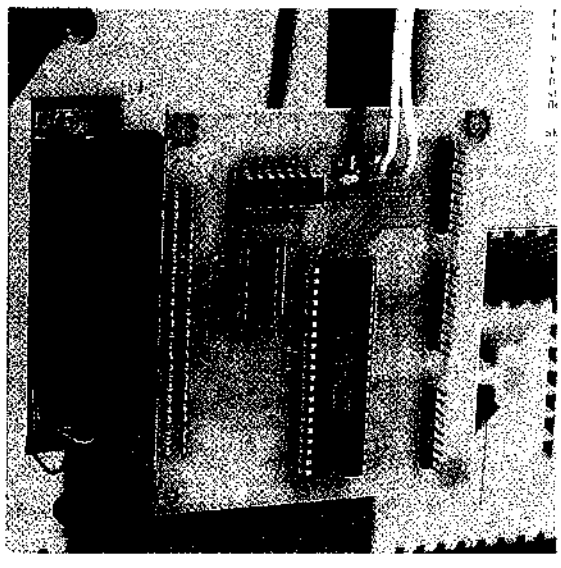












Norbert Kröger
(Jg.1958)
schlug nach seinem Abitur die Laufbahn des Organisationsprogrammierers in der Finanzverwaltung ein. Heute schreibt er Programme, die die Finanzämter bei der Bewältigung anfallender Datenmengen unterstützen. Privat ein PCW-User der ersten Stunde, führte er auf seinem JOYCE u.a. das weiter, was er beruflich an Großrechenanlagen entwickelte. So galt sein Interesse hauptsächlich Jetsam, dem Instrument zur Datenverwaltung und Stiefkind der PCW-Handbücher.

Winfried Kersting
(Jg.1956)
Nutzte den JOYCE während seines Studiums der Philosophie, Germanistik und Literaturwissenschaft zunächst für die Textverarbeitung und die ED-Verwaltung von Buchbeständen. Daraus entwickelte sich sein genereiles Interesse am PCW-System, speziell an der Erweiterung der Hardware und an verschiedenen Programmiersprachen. Die Erkenntnis, daß geeignete "Read-Ware" zum JOYCE fehlte, motivierte ihn, Gelerntes anzuwenden und vorliegendes Buch unter Berücksichtigung didaktischer Aspekte zu konzipieren.

Bernd Graßhoff
(Jg.1971)
ebenfalls JOYCE-User der ersten Stunde, baute schon in den unteren Klassen des Gymnasiums als versierter Elektroniker komplizierte Versuchsaufbauten und Anlagen für den Unterricht und Schulfeiern, die über den PCW gesteuert wurden. Seine Kenntnisse im EDV- und Elektronikbereich konnte er so weit ausbauen, daß sie mittlerweile im kommerziellen EDV-Geräteservice gefragt sind und sich diverse seiner Softwareprodukte auf dem Markt etablieren konnten.

Zum Inhalt dieses Buches

Es ist das erklärte Ziel des vorliegenden Buches, den JOYCE/PCW als das zu präsentieren, was er de facto ist:
Ein vollwertiger Computer.

Das Konzept zur Erreichung dieses Ziels ist im Rahmen der handelsüblichen "Read-Ware" einmalig. Dem Anwender soll die Arbeit mit der zum System gelieferten Software nähergebracht und erleichtert werden, wobei sowohl der fortgeschrittene User als auch der absolute Laie an diesem Werk eine wertvolle Hilfe finden werden. Der erfahrene Programmierer erhält hier effektive Unterstützung durch die Befehlsübersichten und Befehlszusammenstellungen, die alphabetische Ordnung der BASIC-Kommandos, das ausführliche Stichwortverzeichnis und die vielen in die Tiefe gehenden Hinweise rund um die Programmierbeispiele. Anfänger erhalten durch kurze Beispielprogramme Einsicht in die Wirkungsweise der Befehle. Neben den zahlreichen Tips und Tricks in den einzelnen Programmen gestaltet sich das Buch für den erfahrenen User zu einem unentbehrlichen Nachschlagewerk, wobei die Hinweise zur Erweiterung und zum Ausbau der Peripherie noch keine Erwähnung gefunden haben. Mit einem Kapitel über LOGO, das auch dem erfahrenen Programmierer noch einiges Neue bieten kann, wird dem Anfänger ein leichter und verständlicher Einstieg in die Welt der Computersprachen inklusive Grafikausgabe geboten. Alle programmierbaren Vorgänge wurden zur Erleichterung für die Anwender auf eine Diskette gebracht, die zum Buch über den Verlag zu beziehen ist. Neben der Arbeit mit den Sprachen des JOYCE wird auch der Ausbau und die Verbesserung der Hardware gezeigt. Stichworte wie Bildschirminverter, Datenfernübertragung, Druckkopfreinigung, Buntdruck oder Sprachsynthesizer werden auch für den JOYCE-User zu vertrauten Begriffen. Hierauf aufbauend stellt eine Anleitung zum Selbstbau einer leistungsfähigen Schnittstelle eine attraktive Erweiterungsmöglichkeit dar.
Fazit: Das Konzept des informativen Nachschlagewerks plus leicht verständlicher Anleitung macht dieses Buch zu einem unentbehrlichen Helfer für jeden JOYCE-Besitzer.

Zum Buch erhältlich:
- Diskette 3" mit den abgedruckten Programmen
- Schnittstellenplatine zum problemlosen Aufbau

Bestellkarten im Buch

3440 Eschwege
ISBN 3-926177-02-0
Preis: 69,- DM