Proxa

BEDIENUNGS-
HANDBUCH
PET 2001

---

commodore

Inhaltsverzeichnis

Einführung

Der COMMODORE PET 2001 ist ein freiprogrammierbares BASIC-Tischcomputer-System mit hoher Leistungsfähigkeit. Von einem System kann man sprechen, da bereits alle für den Betrieb notwendigen Ein- und Ausgabeeinheiten eingebaut sind:

Tastatur mit 73 Tasten
Bildschirm für 1000 Zeichen
Kassetteneinheit zur Speicherung von Programmen und Daten

Außer den Möglichkeiten zur kompletten Text- und Stringverarbeitung und zum Verkehr mit Peripheriegeräten ist das komplette Interface für den Anschluß sämtlicher IEEE 488-kompatiblen Geräte bereits in der Grundausführung eingebaut. Der 8 k BASIC-Interpreter ist in ROM's untergebracht und beansprucht daher keinen Platz im Arbeitsspeicher.

Bei dem im PET 2001 verwendeten COMMODORE-BASIC handelt es sich um eine leicht programmierbare und äußerst vielseitig anwendbare Programmiersprache. Obwohl COMMODORE-BASIC weitaus umfangreichere Möglichkeiten bietet als die meisten BASIC-Versionen, kann jedes vorliegende BASIC-Programm praktisch ohne Änderung eingegeben werden.

Um mit dem PET 2001 arbeiten zu können, genügt es, ihn an eine normale 220-Volt Schuco-Steckdose anzuschließen und den Netzschalter — auf der Rückseite, links — einzuschalten. Nach einigen Sekunden erscheint auf dem Bildschirm:

Das blinkende Quadrat unter READY ist der sogenannte Cursor. Der Cursor zeigt an, daß auf dem Bildschirm geschrieben werden kann und wo geschrieben wird. Wird irgendeine Taste gedrückt, so erscheint das betreffende Zeichen auf dem Bildschirm an der Stelle des Cursors, der Cursor bewegt sich um einen Schritt nach rechts.

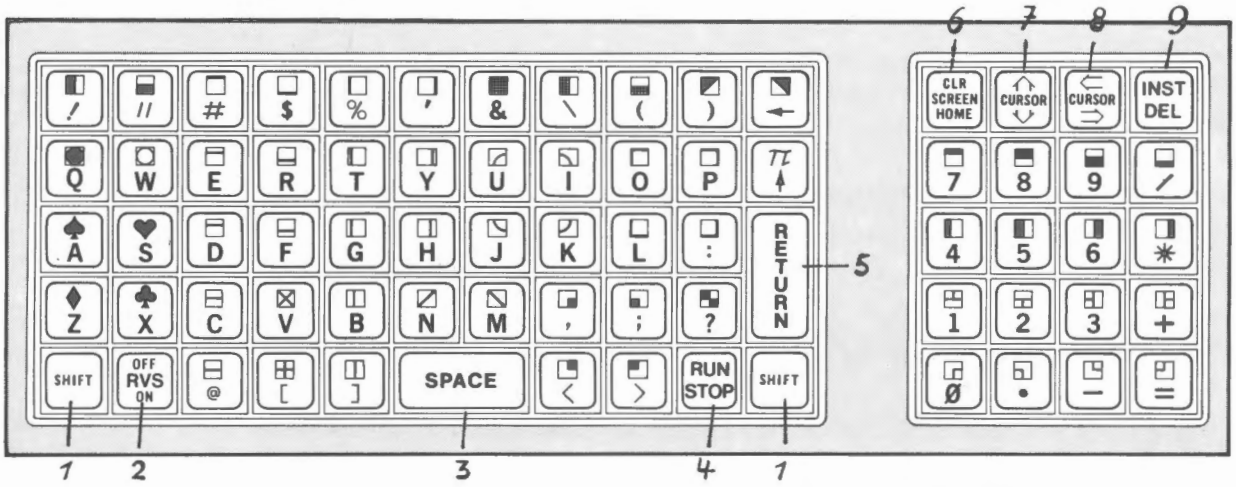
Da auf dem Bildschirm gleichzeitig bis zu 25 Zeilen dargestellt werden können, ist bei Programmierung, Programmausführung und bei allen sonstigen Operationen eine vorzügliche Übersichtlichkeit gewährleistet.

Tastenfeldoperationen

Um mit einem Computer arbeiten zu können, muß man ihm Mitteilungen (z.B. Daten, Arbeitsanweisungen, ...) zukommen lassen, der Computer seinerseits muß der Außenwelt Mitteilungen (Ergebnisse, Fragen, ...) geben können. Dieser Verkehr mit der Außenwelt kann auf mannigfaltige Weise vonstatten gehen. Die Eingabe kann von Meßgeräten, Kassetten, Telefonleitung, Band, etc. erfolgen. Zur Ausgabe bieten sich Drucker, Plotter, Kassette, Telefonleitung, etc. an. Die wichtigsten und meistgebrauchten Mittel im Verkehr mit dem Computer sind das Tastenfeld zur Eingabe und der Bildschirm zur Ausgabe.

Über das Tastenfeld kann wie mit einer normalen Rechenmaschine gerechnet werden, es können Programme und es können Daten in ein laufendes Programm eingegeben werden. Im Zusammenwirken mit dem Bildschirm kann ein echter Dialogverkehr mit dem Rechner stattfinden.

Das Tastenfeld besteht aus 73 Tasten, die in zwei Felder aufgeteilt sind. Der linke Teil entspricht etwa einer normalen Schreibmaschinentastatur, der rechte Teil der Tastatur einer Rechenmaschine.



Auf beiden Tastenfeldern befinden sich spezielle Tasten, die farblich hervorgehoben sind.

Die beiden Tasten SHIFT (1) bewirken die Umschaltung auf die jeweils oben auf der Taste befindlichen Zeichen.

Die Taste SPACE (3) liefert — wie bei einer Schreibmaschine — einen Zwischenraum.

RVS (2) bewirkt die Negativdarstellung (schwarz auf weiß) der Zeichen auf dem Bildschirm. OFF stellt den Normalzustand wieder her.

STOP (4) beendet ein laufendes Programm. RUN hat nicht die Funktion des Buchstabe für Buchstabe eingegebenen RUN, sondern entspricht dem Befehl LOAD mit anschließendem RUN.

RETURN (5) wird zur Beendigung jeglicher Eingabe gebraucht. Es bewirkt Wagenrücklauf (carriage return), bzw. auf dem Bildschirm das Ende einer Zeile und den Beginn der nächsten. Beim direkten Rechnen wirkt es als Ausführungsbefehl. Fängt eine auf den Bildschirm geschriebene Zeile mit einer Zahl zwischen 0 und 63999 an, so wird die Zeile unter dieser Nummer in den Programmspeicher geschrieben, wenn RETURN gedrückt wird. Nach einer Dateneingabe im Anschluß an einen INPUT-Befehl bewirkt RETURN die Fortsetzung des Programms. Wird jedoch in einem INPUT-Befehl die Taste RETURN gedrückt, ohne daß Daten eingegeben wurden, so wird das Programm beendet.

HOME (6) bringt den blinkenden Cursor in die linke obere Ecke des Bildschirms (in die Home-Position). CLR löscht den gesamten Bildschirm und bringt ebenfalls den Cursor in Home-Position.

CRSR (7 und 8) sind Cursor-Kontrolltasten. Mit ihrer Hilfe kann der Cusor an jede beliebige Stelle des Bildschirms geführt werden. Werden die Cursor-Tasten während eines Textes (Zeichen zwischen Anführungszeichen) gedrückt, so erfolgt keine Cursorbewegung auf dem Bildschirm,

sondern die Bewegung wird in das Programm geschrieben. Auf diese Weise kann der Cursor voll programmiert werden.

DEL (9) löscht das links vom Cursor stehende Zeichen. INST schafft einen Zwischenraum zum Einfügen eines Zeichens.

Mittels SHIFT (1) lassen sich die auf den Tasten zusätzlich dargestellten graphischen Zeichen erreichen.

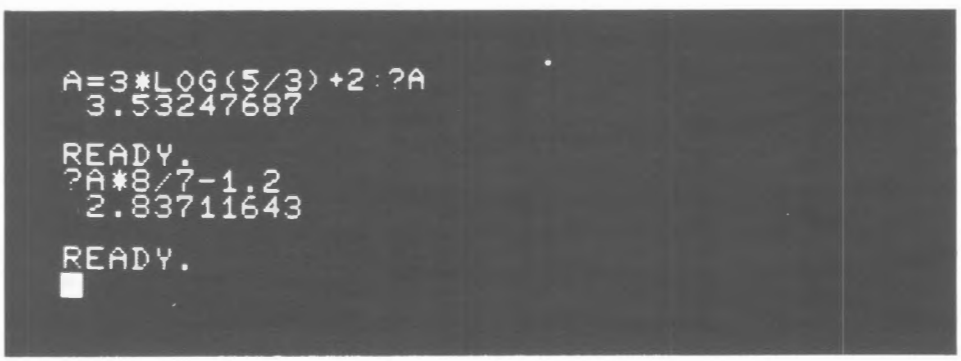
Die über den Buchstaben befindlichen graphischen Zeichen können mittels POKE (siehe Maschinensprache) durch entsprechende Kleinbuchstaben ersetzt werden.

Direktes Rechnen

Wie mit jeder normalen Rechenmaschine können auch mit dem PET 2001 alle üblichen Rechnungen direkt (ohne Programm) durchgeführt werden. Bei der Eingabe ist folgendes zu beachten:

Beginnt eine Eingabe mit einer Zahl, so wird diese Zahl vom PET als Zeilennummer interpretiert und alles was hinter dieser Zahl steht in den Programmspeicher geschrieben. Die vom Taschenrechner her gewohnte Rechnung 2*3= führt also zur Speicherung der (sinnlosen) Programmzeile 2*3=. Die Eingabe muß in diesem Fall lauten ?2*3 (bzw. PRINT 2*3). Nach Drücken der RETURN-Taste erscheint das Ergebnis 6 auf dem Bildschirm. Selbstverständlich können auch kompliziertere Rechnungen und auch mehrere Rechnungen, durch Doppelpunkt getrennt, auf einmal eingegeben werden.
Beispiel:

Für die Durchführung einer Rechnung können maximal 80 Zeichen verwendet werden. Reicht dies nicht aus, so kann das Ergebnis der ersten Zeile in einer Variablen zwischengespeichert und in der nächsten Zeile weiterverwendet werden.
Beispiel:

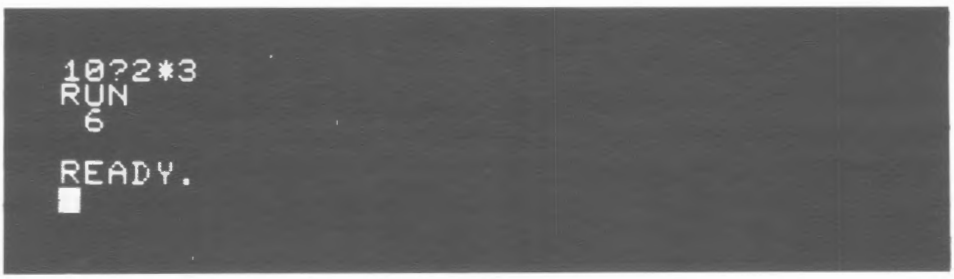


Außer dem direkten Rechnen über das Tastenfeld, können auch Befehle auf dem Tastenfeld eingegeben werden. Lediglich INPUT, GET und DEFFN werden nur innerhalb eines Programms ausgeführt.

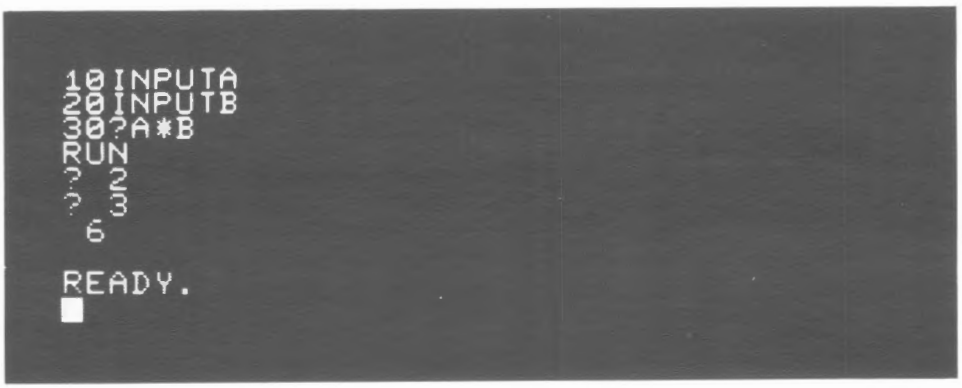
Programmierung

Erst in Verbindung mit einem Programm wird die außerordentliche Leistungsfähigkeit des PET 2001 voll ausgeschöpft. Ein Programm ist die Arbeitsanweisung für den Computer. In ihm stehen sämtliche Operationen, die er mit gegebenen Ausgangswerten auszuführen hat, um möglichst schnell die gewünschten Ergebnisse zu erzielen. Das Programm bestimmt auch die Art der Eingabe und die Form der Ausgabe (Dialogverkehr). Diese oft als nebensächlich betrachteten Teile des Programms bedürfen der meisten Mühe und Aufmerksamkeit, denn von ihnen hängt letztlich die praktische Brauchbarkeit des Programms ab. Eine bequeme Eingabe und eine leichte Korrekturmöglichkeit, wie sie der PET 2001 bietet, sind daher von großer Wichtigkeit. Am Beispiel eines Primitivst-Programmes (Multiplikation zweier Zahlen) soll gezeigt werden, wie ein Programm benutzerfreundlich aufgebaut sein kann.

In seiner einfachsten Form sieht das Programm folgendermaßen aus:
Beispiel:



Nach Eingabe von RUN und Drücken der RETURN-Taste erscheint das richtige Ergebnis: 6. Dies ist zweifellos ein Programm, jedoch ein kaum brauchbares. Sollen andere Zahlen als 2 und 3 multipliziert werden, so müssen diese Zahlen jedesmal neu ins Programm geschrieben werden. Die beiden Faktoren sollten jeweils ohne Umstände eingegeben werden können. Man kann das Programm daher folgendermaßen abändern:
Beispiel:



Der PET 2001 fragt jetzt jeweils mit einem Fragezeichen nach den Faktoren, so daß wir beliebige Zahlen multiplizieren können. Aus dem Fragezeichen geht jedoch nur hervor, daß PET etwas wissen will, jedoch nicht, was er wissen will. Eine weitere kleine Änderung des Programms macht die Fragestellung deutlich.
Beispiel:

```
10 INPUT"FAKTOR 1";A
20 INPUT"FAKTOR 2";B
30 ?A*B
RUN
FAKTOR 1? 2
FAKTOR 2? 3
 6

READY.
```

Es ist jedoch nicht ersichtlich, welche Bewandtnis es mit der Zahl hat, die der Rechner unter die beiden Fragen schreibt. Nur wer das Programm kennt, weiß, daß diese Zahl das Ergebnis darstellt. Mit ein wenig Text in der Zeile 30 kann Abhilfe geleistet werden.
Beispiel:

```
 10 INPUT"FAKTOR 1";A
 20 INPUT"FAKTOR 2";B
 30 PRINT"ERGEBNIS";A*B
READY.
RUN
FAKTOR 1? 2
FAKTOR 2? 3
ERGEBNIS 6

READY.
```

Es ist noch störend, daß nach jeder ausgeführten Rechnung erneut RUN eingegeben werden muß, um die nächste Berechnung vornehmen zu können. Wir befehlen dem Rechner nun, nach Beendigung sofort wieder mit der Frage nach dem Faktor 1 zu beginnen.
Beispiel:

```
 10 INPUT"FAKTOR 1";A
 20 INPUT"FAKTOR 2";B
 30 PRINT"ERGEBNIS";A*B
 40 GOTO10
RUN
FAKTOR 1? 2
FAKTOR 2? 3
ERGEBNIS 6
FAKTOR 1? 4
FAKTOR 2? 7
ERGEBNIS 28
FAKTOR 1?
```

Ein letzter Schönheitsfehler besteht noch darin, daß zwischen den einzelnen Rechnungen kein Zwischenraum vorhanden ist. Mit der Zeile 35 wird eine Leerzeile erzeugt.
Beispiel:

```
 10 INPUT"FAKTOR 1";A
 20 INPUT"FAKTOR 2";B
 30 PRINT"ERGEBNIS";A*B
 35 PRINT
 40 GOTO10
READY.
RUN
FAKTOR 1? 2.23
FAKTOR 2? 6.8
ERGEBNIS 15.164

FAKTOR 1? 9
FAKTOR 2? 3
ERGEBNIS 27

FAKTOR 1? ■
```

Erst in dieser Form kann man von einem benutzerfreundlichen Programm sprechen.

Dateneingabe

In dem vorangegangenen Beispiel wurden Werte mittels INPUT-Befehl in das laufende Programm eingegeben. Dies ist die gebräuchlichste Methode der Dateneingabe. INPUT ist nicht nur auf numerische Werte beschränkt, sondern es kann sich auch um Texte, alphanumerische oder graphische Zeichen handeln, wenn sie in einer Stringvariable eingegeben werden. Mit einer INPUT-Anweisung können mehrere Daten (numerische oder alphanumerische, aber auch gemischt) eingegeben werden.
Beispiel:

```
 100 INPUTA,A$,B,C%
 120 PRINTA,A$,B,C%
READY.
RUN
? 3.57,KELLER,623.85,20147
 3.57         KELLER         623.85        20147

READY.
■
```

Text zur Abfrage im Dialogverkehr kann direkt im INPUT-Befehl stehen. Er muß durch ein Semicolon von den Variablen getrennt sein.
Beispiel:

```
 40 INPUT"PREIS";P
 50 INPUT"NAME";N$
 60 PRINTN$,P
READY.
RUN
PREIS? 122.68
NAME? KASSETTENRECORDER
KASSETTENRECORDER     122.68

READY.
```

Eine weitere Möglichkeit der Eingabe von Zeichen in ein laufendes Programm bietet der Befehl GET. Im Gegensatz zu INPUT (hier können viele Zeichen in einem Befehl eingegeben werden, das Programm hält bis zum Drücken der RETURN-Taste an) wird mit GET ein einzelnes Zeichen übernommen, das Programm hält dabei nicht an. Wurde jedoch vorher eine Taste gedrückt, so bleibt dieses Zeichen gespeichert und wird später durch GET übernommen. GET wird weniger zur Dateneingabe in ein Programm verwendet, als um beliebige Tasten der Tastatur zu Spezialfunktionstasten zu machen.
Beispiel:

```
 10 GETG$
 20 IFG$="A"THEN230
 30 IFG$="B"THENPRINT"B GEDRUECKT":GOTO2
50
 40 IFG$="X"THENX=π:GOTO400
READY.
```

Soll ein Programm beim GET-Befehl anhalten, so läßt sich dies leicht mit einer kleinen „Warteschleife" verwirklichen. Das Programm wird dann sofort nach Druck auf eine beliebige bzw. eine bestimmte Taste fortgesetzt, ohne daß RETURN gedrückt werden muß.
Beispiel:

```
 10 GETX$:IFX$=""THEN10
 20 IFX$="J"THEN100
 30 IFX$="N"THEN200
 40 GOTO10
 100 PRINT"JA":GOTO10
 200 PRINT"NEIN":GOTO10
READY.
RUN
JA
NEIN
```

Basic-Programmierung

1. Zeilennummer

Ein Programm besteht aus einer Reihe von Operationen, die nacheinander ausgeführt werden. Welche Operation auszuführen ist, wird dem Rechner jeweils durch einen entsprechenden Befehl mitgeteilt. Die Befehle werden in numerierten Zeilen im Arbeitsspeicher des Rechners gespeichert. Bei den meisten bisherigen BASIC-Versionen konnte nur ein Befehl in eine Zeile geschrieben werden. COMMODORE-BASIC läßt mehrere Befehle innerhalb einer Zeile zu. Dies ist besonders wertvoll bei logischen Vergleichsoperationen (auf Vergleichsoperationen wird später eingegangen).

Zugelassen sind alle Zeilennummern von 0 bis 63999. Die Abarbeitung der Zeilen erfolgt in der Reihenfolge der Zeilennummern. Zweckmäßigerweise beginnt man die Zeilennumerierung nicht bei 0 oder 1 und auch die Wahl der Abstände der Zeilennummern sollte nicht in 1-er Schritten erfolgen. Besser geeignet ist beispielsweise der Start mit Zeilennummer 10 und die Weiterführung in 10-er Abständen (10,20,30,...). Der Grund hierfür ist, bei später eventuell notwendigen Programmkorrekturen oder -erweiterungen ohne Schwierigkeiten neue Zeilen einschieben zu können. Es besteht keine Veranlassung, kleine Zeilennummern zu verwenden, um Speicherplatz zu sparen, denn jede Zeilennummer benötigt unabhängig von ihrer Größe 4 Byte.

Gleichfalls kann es zweckmäßig sein, einzelne Programmteile bereits durch die Zeilennummer unterscheidbar zu machen. Beispielsweise kann ein Teil des Programms die Zeilennummer 10 bis 130 belegen, der zweite Teil die Nummern 1000 bis 1220 für sich beanspruchen und der dritte Teil bei 2000 beginnen. Große Abstände zwischen den Zeilennummern schaden weder in Bezug auf Speicherplatzbedarf noch auf Ausführungsgeschwindigkeit.

2. Variablen

Zahlenwerte — eingegebene oder berechnete — können auf Speicherplätzen, die mit einem Namen bezeichnet werden, gespeichert werden. Um eine Zahl zurückzurufen, genügt es, diesen Namen zu benutzen. Einen derart mit Namen bezeichneten Zahlenwert nennt man eine Variable (eine Variable entspricht also ziemlich genau dem von kleineren Rechnern her bekannten Register).

2.1 Namen

Als Namen können verwendet werden:

- 26 Buchstaben von A - Z (Großbuchstaben)
- ein Buchstabe und eine Zahl z.B. A0, A3, C7, X9
- zwei Buchstaben z.B. AA, AC, DE, FG ... ZZ

Dies sind insgesamt 962 Namen. Zwei dieser Namen können jedoch nicht verwendet werden, und zwar TI und ST. TI ist belegt durch die laufende Zeit in Einheiten von 1/60-stel Sekunden. ST zeigt den Status von Peripherie-Geräten an. Außerdem können IF, TO, FN, ON und OR nicht verwendet werden, da sie vom Rechner als BASIC-Befehle verstanden werden.

Während diese Variablen Zahlenwerte mit der vollen Genauigkeit und dem gesamten Rechenbereich des Rechners speichern können, kann eine andere Variablenart nur ganze Zahlen im Bereich von -32768 bis 32767 aufnehmen. Hierbei benötigt man zum Abspeichern einer Zahl nur zwei statt sieben Byte. Auch für diese Variablenart können die oben aufgeführten Namen verwendet werden, mit dem einen Unterschied, daß zusätzlich das Prozentzeichen (%) zu verwenden ist.
Z.B.: C%, C7%, LD%

Es können auch längere Namen als Variable verwendet werden
Beispiel: KASSEL
Hiervon werden nur die beiden Anfangsbuchstaben, also KA zur Kennzeichnung benutzt. Dies bedeutet, daß die gleiche Variable auch mit KATZE oder nur mit KA zurückgerufen werden kann.

Außerdem darf in einem solchen Namen an keiner Stelle eine Zeichenfolge enthalten sein, die vom Rechner als Befehl verstanden werden könnte.
Beispiel: UNSINN

Die Variable UNSINN ist unzulässig, da das in diesem Wort enthaltene SIN als Sinus interpretiert wird.

2.2 Variablenfelder (Arrays)

Alle zulässigen Namen lassen sich für Felder verwenden. Unter einem Feld versteht man eine Gruppe von Variablen, die den selben Namen, jedoch verschiedene Indices besitzen. Die Anzahl der verwendeten Indices bezeichnet man auch als Dimension des Feldes. Beispielsweise handelt es sich bei A(2,4,3) um ein Element eines drei-dimensionalen Feldes. Die Indices können Variable oder mathematische Ausdrücke sein — z.B. A(I,J,K) oder C(4,Z+2). Auf diese Weise ist es möglich, diese Variablengruppen indirekt zu adressieren um mit einem einzigen Befehl in einer Schleife viele Variable zuzuordnen oder zurückzurufen. Die Anzahl der Dimensionen und der Elemente eines solchen Feldes ist nur begrenzt durch die verfügbare Speicherkapazität.

Außer diesen Feldern aus Variablen mit voller Genauigkeit können auch Felder aus Ganzzahl-Variablen gebildet werden.
Beispiel: A% (3,2)

Soll ein Feld in irgendeiner Dimension mehr als 11 Elemente (0-10) umfassen, so muß es dimensioniert werden.
Beispiel: DIM B (4,20)

COMMODORE-BASIC erlaubt nicht nur Dimensionierung mit Konstanten, sondern auch mit Variablen oder mathematischen Ausdrücken.
Beispiel: DIMA(M,N), DIMB(I+3,J*2)

Bei kleineren Feldern ist eine Dimensionierung überflüssig. Speziell bei mehrdimensionalen Feldern kann jedoch eine Dimensionierung notwendig sein, um Speicherplatz einzusparen. Wird beispielsweise die Variable A(2,2,2) verwendet, ohne daß sie dimensioniert wurde, so werden automatisch 11x11x11=1331 Variable und damit 6655 Byte des Arbeitsspeichers belegt.

2.3 String (Zeichenketten-Variable)

Während in normale Variablen nur Zahlenwerte eingegeben werden können, erlauben String-Variablen die Eingabe beliebiger Zeichen — Buchstaben, Zahlen, Satzzeichen, graphische Zeichen. Mit den zur Verfügung stehenden String-Operatoren können aus derartigen Zeichenketten beliebige Zeichen oder Gruppen von Zeichen herausgenommen, mit anderen Zeichen verglichen und — falls es sich um Zahlen handelt — in normale Variablen umgewandelt werden. Ganze Wörter oder Sätze, die häufig gebraucht werden, können jeweils mit dem Namen der String-Variablen aufgerufen werden. Ein einzelner String kann bis zu 255 Zeichen umfassen.

Zu beachten:
Wie bereits TI in Kapitel 2.1, darf auch TI$ nicht verwendet werden. Die Spezialvariable TI$ ist belegt für die Tageszeit in Stunden, Minuten und Sekunden.

2.4 String-Arrays

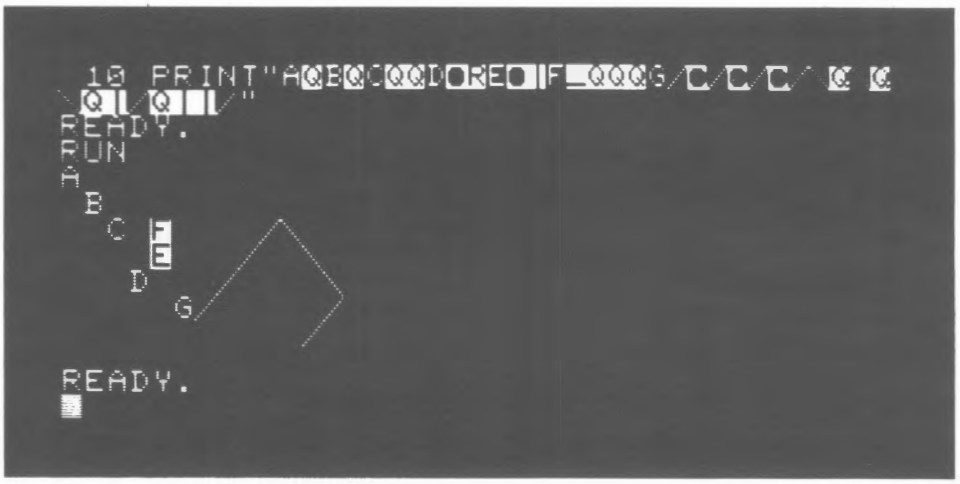
Wie bei gewöhnlichen Variablen können auch hier ein- oder mehrdimensionale Felder gebildet werden. Es gelten die gleichen Regeln wie bei den gewöhnlichen Feldern, das heißt, auch hier brauchen nur String-Arrays mit mehr als 11 Elementen in einer Dimension dimensioniert zu werden. Es ist nicht — wie bei manch anderen BASIC-Versionen — notwendig, einzelne Strings zu dimensionieren. Der Befehl DIM A $ (30) bedeutet *nicht* die Dimensionierung eines Strings mit 30 Zeichen, sondern die Dimensionierung eines Arrays mit 30 Strings!

Für sämtliche Variablenarten können alle zulässigen Namen verwendet werden. Wird der selbe Name gleichzeitig für verschiedene Variablenarten verwendet, so erfolgt keine gegenseitige Beeinflussung. Es können somit in einem Programm gleichzeitig die Variablen F4, F4%, F4$, F4(..), F4%(..) und F4$(..) benutzt werden.

3. Text

Während normale Rechner nur Zahleneingabe zulassen und auch nur Zahlen verarbeiten können, ist es bei BASIC möglich, ebenso Buchstaben und Zeichen einzugeben. Werden Buchstaben eingegeben, so kann es sich hierbei um Befehle handeln, die der Rechner erkennen und ausführen muß oder um Text, der auf dem Bildschirm dargestellt oder von einem Drucker geschrieben werden soll. Um diese beiden Funktionen der Buchstaben voneinander unterscheiden zu können, wird Text grundsätzlich zwischen Anführungszeichen ("...") gesetzt.

Text, der zwischen Anführungszeichen steht, beeinflußt den Rechner nicht unmittelbar, das heißt, wenn Befehle wie SIN, PRINT etc. im Text stehen, werden diese Befehle nicht ausgeführt. Stehen im Text nach einer PRINT-Anweisung Befehle für den Rechner selbst — z.B. Bildschirm löschen, Cursor bewegen — so werden diese im Text stehenden Anweisungen vom Rechner befolgt.
(Im Text, das heißt, nach einem Anführungszeichen wird die normale Funktion der Lösch- und Cursorkontrolltasten beim Drücken dieser Tasten nicht ausgeführt, sondern der entsprechende Befehl wird in den Text geschrieben)
Beispiel:



Auch sämtliche graphische Zeichen, einschließlich Negativdarstellung, können in den Text geschrieben werden. Zur Speicherung von Text können selbstverständlich nur Stringvariablen verwendet werden.

4. Eingabe

Bei den meisten Programmen ist es notwendig, an bestimmten Stellen des Programms numerische oder alphanumerische Daten einzugeben. Dies kann mit einer der drei folgenden Anweisungen erfolgen:

— INPUT
— GET
— READ

4.1 INPUT

Steht im Programm der Befehl INPUT und dahinter eine oder mehrere Variablen — mehrere Variablen müssen durch Kommata getrennt werden — so bleibt das Programm an dieser Stelle stehen und auf dem Bildschirm erscheint ein Fragezeichen (?) als Aufforderung zur Dateneingabe. Sollen Daten für mehrere Variablen eingegeben werden, so erfolgt die Eingabe ebenfalls mit Trennung durch Kommata. Zum Abschluß jeder Eingabe ist die Taste RETURN zu betätigen. Werden zu wenige Daten eingegeben, so erscheinen am Bildschirm zwei Fragezeichen (??) als Aufforderung, die restlichen Daten einzugeben. Werden jedoch zu viele Daten eingegeben, läuft das Programm ebenfalls weiter, die überzähligen Daten werden ignoriert und es erfolgt die Meldung: ?EXTRA IGNORED

Soll auf dem Bildschirm eine Frage erscheinen, welche Daten jetzt eingegeben werden sollen, so kann diese Frage als Text vor die Variablen gestellt werden (Dialogverkehr, „Prompting"). Text und Variable müssen durch ein Semicolon getrennt werden.
Beispiel:

```
 10 INPUT"TAG  ";T$
 20 INPUT"MONAT";M$
 30 INPUT"JAHR ";J$
 40 PRINT:PRINT"HEUTE IST DER ";T$;".";M
$;".";J$
READY.
RUN
TAG  ? 3
MONAT? 5
JAHR ? 1978

HEUTE IST DER 3.5.1978

READY.
```

Werden falsche Daten eingegeben — alphanumerische Daten, wenn in eine numerische Variable eingegeben werden soll — so erfolgt die Meldung REDO FROM START. Die Eingabe kann dann mit den richtigen Daten wiederholt werden.

4.2. GET

Im Gegensatz zu INPUT bleibt das Programm nicht stehen, wenn es zu einem GET-Befehl kommt. GET übernimmt nur ein einziges Zeichen. Da bis zu 10 Tastenbetätigungen im Rechner gespeichert werden, wird jedes Zeichen übernommen, das während des Programmlaufes vor Erreichen des GET-Befehles gedrückt wurde. Soll bei einem GET-Befehl der Rechner auf Eingabe eines Zeichens warten, so ist dies leicht durch eine kleine Schleife zu erreichen.
Beispiel:
Mit dem GET-Befehl kann jede Taste zur Spezialfunktionstaste gemacht werden.

4.3. READ

READ war bei den ersten BASIC-Versionen die einzige Möglichkeit, Daten im Programm in Variablen einzugeben. Die Daten werden hierzu an beliebiger Stelle des Programms unter DATA ins Programm geschrieben. Der Befehl READ liest die Daten in der vorgegebenen Reihenfolge. Sollen Daten mehrfach gelesen werden, kann dies durch den Befehl RESTORE erreicht werden.
Beispiel:

Die READ-Anweisung wird heute nur selten angewendet, da Daten von der Kassette eingelesen oder mit INPUT bzw. GET eingegeben werden können.

5. Syntax

Der Rechner versteht nur Befehle, die in seinem Befehlsvorrat enthalten und in einer genau festgelegten Form gegeben sind. Es ist daher wichtig, diese Befehle auch genau in dieser Form ins Programm zu schreiben mit allen jeweils erforderlichen Satzzeichen. Werden Befehle falsch geschrieben, so daß sie vom Rechner nicht verstanden werden, erfolgt die Fehlermeldung: UNDEF'D STATEMENT. Bei der Verwendung falscher Zeichen (z.B. Komma statt Dezimalpunkt) lautet die Fehlermeldung: SYNTAX ERROR. Erscheint eine solche Fehlermeldung, wird gleichzeitig angegeben, in welcher Zeile der Fehler auftrat. In diesem Fall listet man die betreffende Zeile auf und untersucht, ob die darin enthaltenen Befehle mit den in der Befehlsliste angegebenen übereinstimmen.

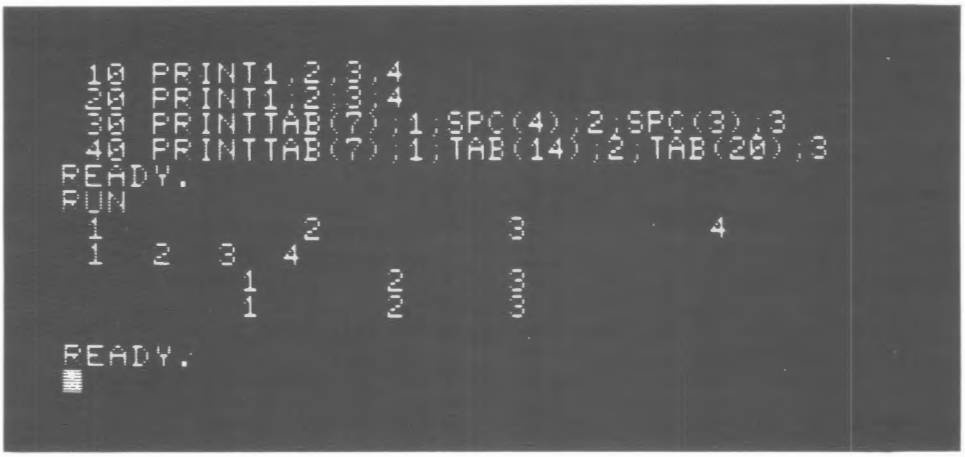
Mehrere Befehle pro Zeile
Im Gegensatz zu den meisten anderen BASIC-Versionen ist es bei COMMODORE-BASIC nicht notwendig, jedem Befehl eine neue Zeilen-Nummer zuzuordnen. Um einen Befehl ins Programm zu schreiben, genügt es, die Befehle durch einen Doppelpunkt (:) voneinander zu trennen. Bis zu einer Maximallänge von 72 Zeichen pro Zeile können auf diese Weise beliebig viele Befehle in eine Zeile geschrieben werden. Neben einer Ersparnis an Speicherplatz bewirkt dies auch eine schnellere Programmausführung. Besonders vorteilhaft wirkt sich das Schreiben mehrerer Befehle in eine Zeile bei logischen Vergleichen aus.

Werden mehrere Befehle in eine Zeile geschrieben, ist jedoch zu beachten:

1. nach Vergleichen wird, wenn der Vergleich „falsch" ergibt, der ganze Rest der Zeile übergangen; die Programmausführung wird bei der folgenden Zeile fortgesetzt.
2. Sprünge sind nur auf den Zeilenanfang, nicht zu einem Befehl innerhalb der Zeile möglich.

6. Ausgabeformat

Nach einem PRINT-Befehl erfolgt normalerweise stets ein „CR/LF" (Wagenrücklauf und Zeilenvorschub), so daß jeder PRINT-Befehl auf einer neuen Zeile ausgeführt werden muß. Steht jedoch hinter dem PRINT-Befehl ein Komma, so wird der nächste PRINT-Befehl auf der nächsten freien vortabulierten Position ausgeführt. Vortabulierte Positionen sind die Kolonnen 0, 10, 20, 30. Wird statt des Kommas ein Semicolon verwendet, so bleiben zwischen aufeinanderfolgenden Zahlenwerten jeweils 2 Leerstellen frei, von denen die zweite für das Vorzeichen reserviert ist. Handelt es sich um Strings, so werden die PRINT-Befehle ohne Zwischenraum aneinandergefügt. Mit SPC kann ein Zwischenraum, mit SPC(n) können n Zwischenräume eingefügt werden. SPC bestimmt immer den Abstand zur vorangegangenen Darstellung, TAB bezieht sich jedoch immer auf den linken Bildschirmrand.
Beispiele:



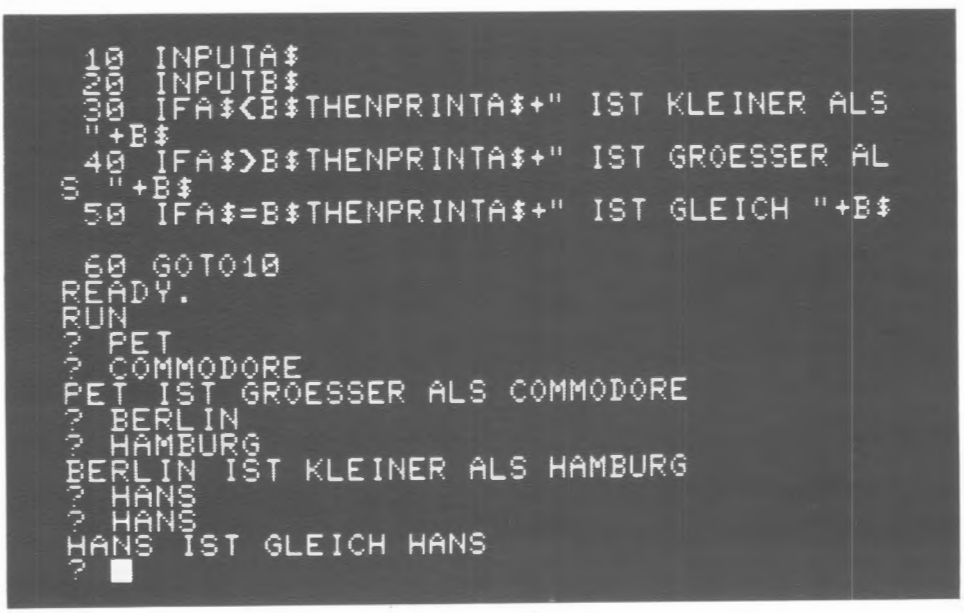
Es ist zu beachten, daß bei Verwendung von TAB und SPC die Trennung stets mit einem Semicolon zu erfolgen hat. Die Verwendung des Kommas führt stets zu vortabulierten Positionen.

7. Logische Vergleichsoperationen

Mit der Befehlsfolge IF .. THEN (wenn .. dann) können Werte verglichen werden. Alle 6 Vergleichsoperationen können durchgeführt werden.

Symbole	Bedeutung
=	gleich
>	größer als
<	kleiner als
=>oder<=	größer oder gleich
=<oder<=	kleiner oder gleich
<>oder><	ungleich

Verglichen werden können Konstanten, Variablen oder mathematische Ausdrücke. Vergleichsoperationen sind nicht nur zwischen Zahlenwerten möglich, sondern auch zwischen Zeichen bzw. Zeichenketten. Hierbei richtet sich die Größe nach dem ASCII-Code des betreffenden Zeichens. Zeichenketten werden in ihrer ganzen Länge verglichen. Da die Buchstaben im ASCII-Code in alphabetischer Reihenfolge angeordnet sind, ist auf diese Weise ebenfalls eine alphabetische Sortierung möglich.
Beispiel:



Die 6 Vergleichsoperatoren können über die logischen Operatoren AND, OR und NOT miteinander verknüpft werden.
Beispiel:

```
 10 INPUTA,B,C
 20 IFA=BANDC=3THENPRINT"RICHTIG":GOTO10
 30 PRINT"FALSCH":GOTO10
READY.
RUN
? 7,7,3
RICHTIG
? 8,6,3
FALSCH
? 5,5,2
FALSCH
? 1,1,1
FALSCH
? 1,1,3
RICHTIG
?
```

Ergibt ein Vergleich — oder eine Verknüpfung von Vergleichen — die Aussage „wahr", so wird der Rest der Zeile nach dem THEN ausgeführt. Bei der Aussage „falsch" wird die Programmausführung mit der nächsten Zeile fortgesetzt. Soll lediglich ein Sprung zu einer anderen Zeile erfolgen, so genügt es, nach THEN die betreffende Zeilennummer einzugeben.

Bei den logischen Operatoren ist zu beachten, daß die Verknüpfung Bit für Bit erfolgt. Es sollte also nicht geschrieben werden: IF A AND B THEN ....., wenn eine Operation ausgeführt werden soll, falls beide Werte von Null verschieden sind (4 AND 2 = 0)! Die Schreibweise sollte in diesem Fall lauten: IF A«»O AND B«»O THEN . . . Der Vorteil dieser bitweisen Operationen ist, daß manche Operationen hierdurch sehr einfach durchgeführt werden können.

8. Sprünge, Verzweigungen, Subroutinen

Nicht in jedem Fall ist es erwünscht oder möglich, ein Programm jeweils mit der nächsten Zeilennummer fortzuführen. Es kann notwendig sein, von einer bestimmten Programmzeile aus stets zu einer an einer anderen Stelle befindlichen Zeile des Programms zu springen. Ebenso kann die Forderung vorliegen, nur unter bestimmten Bedingungen zu dieser anderen Zeile zu gehen. In diesen Fällen spricht man von einem unbedingten bzw. einem bedingten Sprung. Ein solcher Sprung wird durch den Befehl GOTO bewirkt. GOTO kann auch bei einem bedingten Sprung (nach THEN) Verwendung finden.
Beispiel:

```
 10 INPUT"WELCHES PROGRAMM";P
 20 IFP=2THEN200
 30 IFP=3THENGOTO300
 40 IFP=4GOTO400
 50 PRINT"PROGRAMM NICHT VORHANDEN":GOTO
10
 200 PRINT"PROGRAMM 2":GOTO10
 300 PRINT"PROGRAMM 3":GOTO10
 400 PRINT"PROGRAMM 4":GOTO10
READY.
RUN
WELCHES PROGRAMM? 1
PROGRAMM NICHT VORHANDEN
WELCHES PROGRAMM? 2
PROGRAMM 2
WELCHES PROGRAMM? 3
PROGRAMM 3
WELCHES PROGRAMM? 4
PROGRAMM 4
WELCHES PROGRAMM? 5
PROGRAMM NICHT VORHANDEN
WELCHES PROGRAMM? ■
```

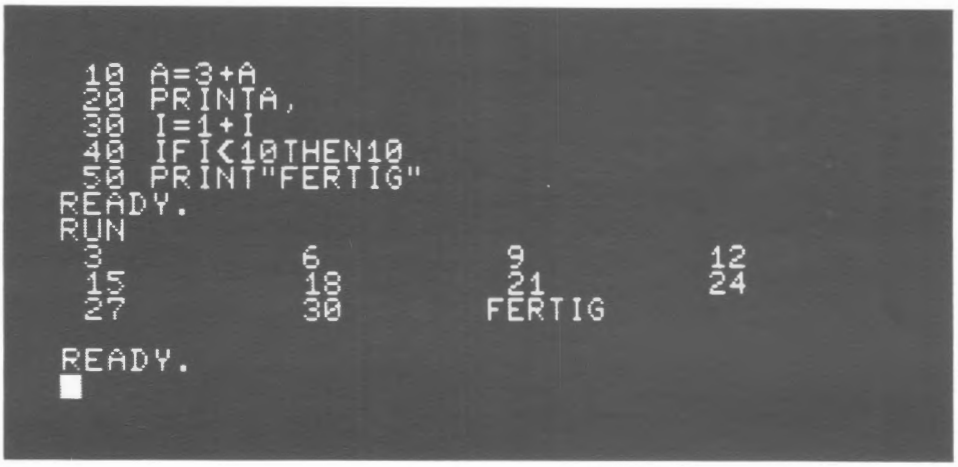
Eine Subroutine unterscheidet sich von einem einfachen Sprung dadurch, daß das Programm nach Ausführung des Sprunges und Abarbeitung eines Programmstückes, das durch den Befehl RETURN abgeschlossen ist, mit dem auf den GOSUB-Befehl folgenden Befehl weitergeführt wird. Der Befehl RETURN muß über die Tastatur Buchstabe für Buchstabe und nicht mit der Taste „RETURN" eingegeben werden. Die Taste „RETURN" hat die Funktion von carriage return (Wagenrücklauf) und schließt eine Eingabe ab, während das RETURN in der Subroutine den Rücksprung ins Hauptprogramm veranlaßt. Subroutinen können auf jeder beliebigen Stelle eines Programmes aufgerufen werden. Sind daher in einem Programm mehrmals die gleichen Befehlsfolgen erforderlich, so brauchen diese nur einmal geschrieben zu werden und können als Subroutine jedesmal dann aufgerufen werden, wenn sie im Programm benötigt werden.
Beispiel:

```
 10 PRINT"BERECHNUNG VON SINUS UND COSIN
US"
 20 INPUT"WINKELMASS";W$
 30 INPUT"WINKEL";W
 40 IFW$="GRAD"THENGOSUB100
 50 PRINT"SINUS";SIN(W),"COSINUS";COS(W)

 60 GOTO20
 100 W=W*π/180:RETURN
READY.
RUN
BERECHNUNG VON SINUS UND COSINUS
WINKELMASS? BOGEN
WINKEL? 1
SINUS  .841470985     COSINUS .540302306
WINKELMASS? GRAD
WINKEL? 40
SINUS  .642787609     COSINUS .766044443
WINKELMASS? GRAD
WINKEL? 60
SINUS  .866025404     COSINUS .5
WINKELMASS? ■
```

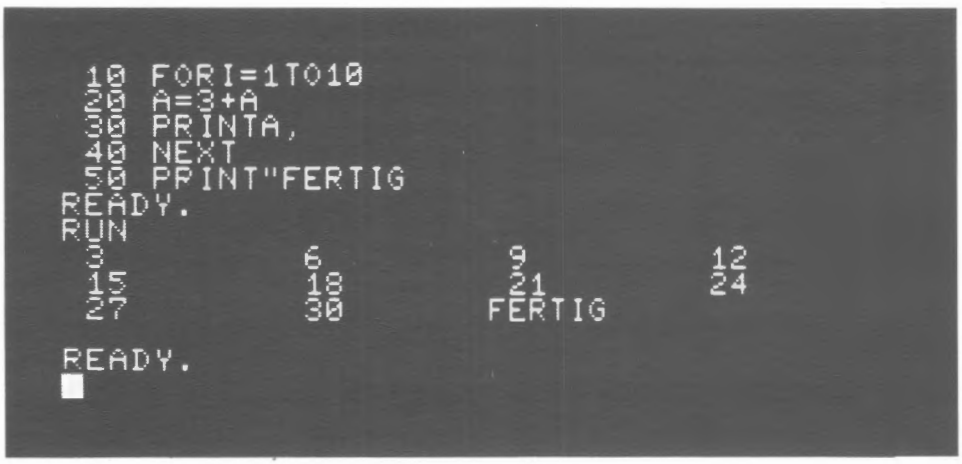
9. Programmschleifen (Loops)

Häufig ist es notwendig, daß Teile von Programmen mehrfach durchlaufen werden. Dies ist möglich, indem man mittels eines GOTO Befehles auf eine niedrigere Zeilennummer zurückspringt. Für eine bestimmte Zahl von Schleifendurchläufen kann dann mit einer Variablen ein Zähler aufgebaut werden. Der Zählerstand wird in einer Vergleichsoperation mit der vorgegebenen Anzahl von Durchläufen verglichen.
Beispiel:



Eine weitaus elegantere Methode ist der „FOR-NEXT-LOOP". Die für eine Schleife notwendigen Operationen — Sprung, Zählen, Vergleich — werden durch die beiden Befehle „FOR" am Schleifenbeginn und „NEXT" am Schleifenende ausgeführt.

Beispiel:



Die Schrittweite des Zählers kann beliebig gewählt werden. Wird sie nicht vorgeschrieben, so wird vom Rechner die Schrittweite +1 ausgeführt.
Beispiel:

Nach „NEXT" kann der Variablenname entfallen. Nur in Ausnahmefällen — mehrere verschachtelte Loops — ist es erforderlich, den Variablennamen nach diesem Befehl aufzuführen.
Bei mehreren verschachtelten Schleifen genügt es auch, den Befehl NEXT ein einziges Mal zu schreiben und dahinter die Variablen in der richtigen Reihenfolge, durch Kommata getrennt, aufzuführen.

10. Mathematische Funktionen

Außer den 4 arithmetischen Funktionen (+,-,*,:) stehen folgende mathematische Funktionen zur Verfügung:

— ↑ Potenzieren
— SQR Quadratwurzel
— LOG natürlicher Logarithmus (zur Basis e). Der dekadische (Brigg'sche Logarithmus) kann erreicht werden durch Division mit LOG (10)
— EXP Exponentialfunktion zur Basis e
— SIN Sinus
— COS Kosinus
— TAN Tangens
— ATN Arcus Tangens

} Winkel in Bogenmaß

Die anderen Arcus-Funktionen (Arcus-Sinus, Arcus-Kosinus) werden mit einer einfachen Routine aus dem Arcus Tangens errechnet.

$$\text{ARCSIN } W = \text{ATN } \frac{W}{\sqrt{1 - W^2}}$$

$$\text{ARCCOS } W = \text{ATN } \frac{\sqrt{1 - W^2}}{W}$$

Winkelangaben in Grad werden ebenfalls mit einer kleinen Hilfsroutine auf das Bogenmaß umgerechnet (Der Wert von π ist fest gespeichert).

— INT ergibt den ganzzahligen Anteil einer Zahl, die Nachkommastellen werden ignoriert. Auch bei negativen Zahlen wird die nächstkleinere ganze Zahl genommen, das heißt, INT (-2,3) ergibt -3.

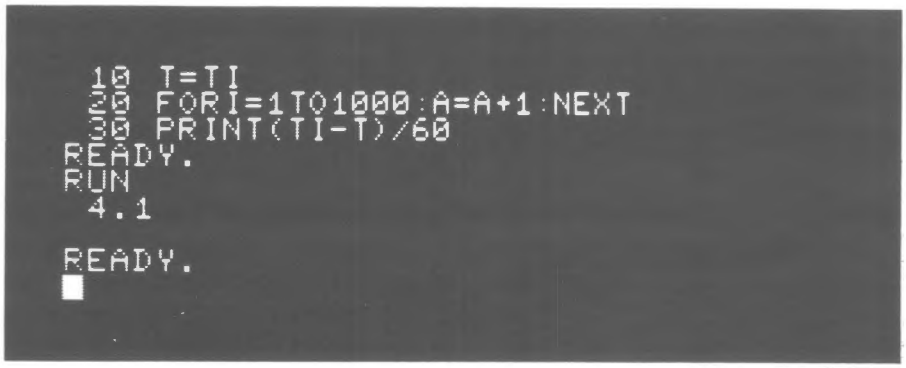
— ABS Absoluter Wert
Positive Zahlen bleiben unverändert, bei negativen Zahlen wird das Vorzeichen geändert.

— SGN Ermittelt das Vorzeichen einer Zahl und liefert bei positiven Zahlen den Wert 1, bei Null 0, bei negativen Zahlen den Wert -1.

11. Uhr

Der PET 2001 besitzt eine eingebaute Uhr, die durch einen Quarz gesteuert wird. Die Zeit steht in zwei verschiedenen Formaten zur Verfügung:

— TI Die Variable TI enthält eine laufende Zeit in Einheiten von 1/60-stel Sekunden. TI wird vorteilhaft zur Erzeugung genauer Pausen, zur Zeitmessung von Vorgängen und dergleichen verwendet.
Beispiel:



— TI $ TI $ enthält die Tageszeit von 0 bis 24 Uhr im Format HHMMSS (Stunden, Minuten, Sekunden). Voraussetzung für die richtige Zeitangabe durch diese Uhr ist — wie bei jeder anderen Uhr — daß sie vorher auf die richtige Zeit gestellt wurde. Dies erfolgt wie bei einer normalen Eingabe in eine String-Variable. Es müssen stets sämtliche 6 Zahlen (auch Nullen) eingegeben werden:

TI $="HHMMSS"

Die Tageszeit kann jederzeit im Programm abgerufen werden und beispielsweise zu bestimmten Zeiten bestimmte Vorgänge auslösen.

12. Fehlersuche

Es gehört zu den ganz seltenen Ereignissen, wenn ein Programm, speziell ein etwas umfangreiches, sofort fehlerlos arbeitet. Es ist beinahe unvermeidlich, daß beim Programmieren gelegentlich eine Klammer vergessen, ein falsches Satzzeichen verwendet , ein Feld falsch dimensioniert ist, ..., die Anzahl der möglichen Fehler ist praktisch unbegrenzt. Kann ein Programm wegen eines Fehlers nicht weitergeführt werden, so wird die Ausführung unterbrochen und auf dem Bildschirm erscheint eine Meldung über die Art des Fehlers und die Zeilennummer, in der der Fehler auftrat. Die betreffende Zeile kann dann durch den Befehl LIST mit der Zeilennummer (Beispiel: LIST 360) in den Bildschirm geholt und auf Fehler untersucht werden.

Folgende Fehlermeldungen . . . ERROR IN . . . können auftreten:

SYNTAX ERROR
Diese Fehlermeldung erfolgt stets, wenn die „SYNTAX" (die für BASIC geltenden Regeln des Satzbaues) nicht eingehalten wurde, das heißt, beispielsweise die Anzahl der geöffneten und der

geschlossenen Klammern stimmt nicht überein, es wurde ein falsches Satzzeichen verwendet, ein nach BASIC-Regeln in Klammer zu setzender Wert wurde ohne Klammer eingegeben und ähnliche Fehler.

ILLEGAL QUANTITY ERROR
Diese Fehlermeldung besagt, daß ein Zahlenwert verwendet wurde, der bei der betreffenden Operation nicht zulässig ist. Beispielsweise können in die Ganzzahl-Variable A% nur Werte zwischen -32768 und 32767 eingegeben werden. Der Versuch, einen außerhalb dieser Grenzen liegenden Zahlenwert einzugeben, führt zu dieser Fehlermeldung.

BAD SUBSCRIPT ERROR
Diese Fehlermeldung bedeutet, daß ein falscher bzw. unzulässiger Index verwendet wurde. Dies bedeutet im einzelnen:

- Wird ein Feld benutzt, das nicht dimensioniert wurde und der Index bzw. einer der Indices ist größer als 10, erfolgt Fehlermeldung.
- Wurde ein Feld dimensioniert und die Anzahl der Dimensionen stimmt bei Dimensionierung und Benutzung nicht überein, erfolgt Fehlermeldung.
- Das selbe Feld wird ohne Dimensionierung mit verschiedenen Dimensionszahlen benutzt.

DIVISION BY ZERO ERROR
Division durch Null (Es ist zu beachten, daß Zahlen kleiner 10^{-38} zu Null gerundet werden).

TYPE MISMATCH ERROR
Einzugebende Daten passen nicht zum Variablentyp (z.B. Zahlenwert in String-Variable oder Text in einfache Variable).

UNDEF'D STATEMENT ERROR
UNDEF'D STATEMENT ERROR wird angezeigt, wenn ein Sprung zu einer nicht vorhandenen Zeilennummer erfolgen soll.

UNDEF'D FUNCTION ERROR
Im Programm wird eine Funktion aufgerufen, die nicht definiert ist. Es ist zu beachten, daß Funktionen vor ihrem ersten Aufruf im Programm definiert sein müssen. Die Definition von Funktionen sollte immer am Programmanfang erfolgen.

STRING TOO LONG ERROR
Die Länge eines Strings ist begrenzt auf 255 Zeichen. Werden Strings zusammengesetzt, so kann es eintreten, daß der zusammengesetzte String mehr als 255 Zeichen umfassen müßte. Es erfolgt dann die Fehlermeldung.

NEXT WITHOUT FOR ERROR
Ein NEXT-Befehl kann vom Programm nur ausgeführt werden, wenn vorher ein FOR . . . . . durchlaufen wurde. Diese Fehlermeldung erscheint, wenn entweder der FOR-Befehl vergessen wurde oder wenn eine Verzweigung auf eine Zeilennummer innerhalb einer FOR . . . NEXT-Schleife erfolgt (Sprünge innerhalb eines FOR-NEXT-Loops sind möglich).

OUT OF DATA ERROR
- In einer READ-Anweisung werden mehr Daten verlangt als unter DATA vorhanden waren.
- Die vorhandenen Daten sollten noch einmal gelesen werden. Es wurde jedoch keine RESTORE-Anweisung gegeben.

OUT OF MEMORY ERROR
- Die Anzahl der für Programm und Variablen benötigten Bytes übersteigt die Speicherkapazität des Rechners.
- Die Kapazität eines Teils des Arbeitsspeichers wurde überschritten (mehr als 26 GOSUB-Rücksprungadressen).

OVERFLOW ERROR
Diese Fehlermeldung tritt auf, wenn ein Zahlenwert größer als $1{,}5 \times 10^{38}$ ist.

RETURN WITHOUT GOSUB ERROR
Ein RETURN-Befehl kann vom Programm nur ausgeführt werden, wenn vorher ein GOSUB

durchlaufen wurde. Diese Fehlermeldung erscheint, wenn eine Verzweigung auf eine Zeilennummer innerhalb einer Subroutine erfolgt (Sprünge innerhalb einer Subroutine sind möglich).

REDIM'D ARRAY ERROR
Ein bereits dimensioniertes Feld darf nicht nochmals dimensioniert werden. Eine erneute Dimensionierung kann erst nach dem Befehl CLR (CLR löscht alle Variablen, einschließlich Dimensionierung) erfolgen. RUN schließt CLR mit ein.

BAD DATA ERROR
Diese Fehlermeldung erfolgt, wenn von einem Peripheriegerät falsche Daten (z.B. Alphazeichen statt Zahlen) an den Rechner gegeben werden.

Außerdem gibt es noch zwei Fehlermeldungen, die jedoch nicht im Programm, sondern bei Fehlbedienung auftreten.

ILLEGAL DIRECT ERROR
INPUT und DEFFN können nicht über die Tastatur, sondern nur über das Programm ausgeführt werden.

CAN'T CONTINUE ERROR
Das Programm kann nicht weitergeführt werden, weil
- kein Programm existiert, das zu Ende geführt werden könnte oder
- eine Programmkorrektur vorgenommen wurde oder
- das Programm durch eine ERROR-Meldung gestoppt wurde.

In den meisten Fällen ist ein Fehler bei genauer Untersuchung der angezeigten Fehlerzeile sehr schnell aufzufinden. Es kann jedoch vorkommen, daß in der betreffenden Zeile kein Fehler vorhanden ist. Der Rechner zeigt nur die Zeile an, in der sich ein Fehler bemerkbar macht. Beispielsweise könnte durch einen Rechenfehler in einer Zeile, die lange zuvor ausgeführt wurde, erst in der angezeigten Zeile eine Zahl auftreten, die zum OVERFLOW ERROR führt. In solchen Fällen muß der vorher abgearbeitete Teil des Programmes untersucht werden. Oft ist es zweckmäßig, zur Fehlersuche an verschiedenen Stellen des Programms die Variablen, die zu dem Fehler führen könnten, durch einen eingefügten PRINT-Befehl auf dem Bildschirm darzustellen. Treten hierbei unerwartete Werte auf, läßt sich der Fehler meist sehr schnell einkreisen.

Ein häufiger Fehler ist vergessene Dimensionierung. Da die normalerweise benötigten relativ kleinen Felder nicht dimensioniert werden müssen, kann im Normalfall auf jegliche Dimensionierung verzichtet werden. Dies verleitet dazu, auch in den Fällen keine Dimensionierung vorzunehmen, wo es erforderlich wäre. Die Fehlermeldung BAD SUBSCRIPT ERROR tritt dann in irgendeiner hohen Zeilennummer auf, obwohl diese Zeile fehlerfrei ist. Der eigentliche Fehler ist das Fehlen der Dimensionierung ganz am Anfang des Programms. Dies soll nur ein Beispiel sein, daß Fehlerort und Anzeige, wo Fehler zu suchen ist, nicht übereinstimmen müssen. Dasselbe kann selbstverständlich auch bei vielen anderen Operationen auftreten.

13. Korrektur

Ist ein Fehler im Programm gefunden worden, so muß es entsprechend abgeändert werden. Der PET 2001 bietet folgende Korrekturmöglichkeiten:

13.1. Zeilen können gelöscht werden, indem die zu löschende Zeilennummer eingetastet und anschließend die RETURN-Taste gedrückt wird.

13.2. Zeilen können überschrieben werden, indem die Zeile mit der betreffenden Zeilennummer neu eingegeben wird.

13.3. Zeilen können eingeschoben werden, indem eine Zeile mit einer Zeilennummer zwischen den Zeilennummern der vorhergehenden und der nachfolgenden Zeile eingetastet wird.

Alle diese Operationen können an jeder beliebigen Stelle des Bildschirmes durchgeführt werden. Bedingung ist lediglich, daß auf der benutzten Zeile des Bildschirms keine weiteren unerwünschten Zeichen vorhanden sind.

Außer diesen Korrekturmöglichkeiten, die sich auf eine ganze Zeile beziehen, können auch einzelne Zeichen in vorhandenen Zeilen gelöscht, überschrieben oder eingefügt werden. Dies ist besonders wertvoll, wenn sehr lange Zeilen zu korrigieren sind.
Um Korrekturen innerhalb einer Zeile vorzunehmen, wird die Zeile zuerst mittels LIST-Befehl auf dem Bildschirm dargestellt. Anschließend wird der Cursor mit Hilfe der beiden Cursor-Kontrolltasten an die Stelle der Zeile gebracht, an der die Korrektur stattfinden soll. Um ein Zeichen zu ändern, muß der Cursor genau auf diesem Zeichen stehen. Das neue Zeichen wird eingetastet und überschreibt so das bisherige. Um ein Zeichen zu löschen, muß sich der Cursor rechts neben dem zu löschenden Zeichen befinden. Sollen mehrere aufeinanderfolgende Zeichen gelöscht werden, so setzt man den Cursor rechts neben das letzte zu löschende Zeichen. Durch wiederholtes Drücken von DEL wird nun von rechts nach links ein Zeichen nach dem anderen gelöscht. Sollen Zeichen eingeschoben werden, so ist der Cursor über das Zeichen zu bringen, vor dem ein Zwischenraum einzufügen ist. Bei jedem Druck auf INST wird der Teil der Zeile, der mit dem unter dem Cursor befindlichen Zeichen beginnt, um eine Position nach rechts verschoben, während der Cursor an der gleichen Stelle bleibt. Man kann daher unmittelbar nach dem Schaffen des nötigen Zwischenraumes mit dem Schreiben beginnen.
Nach der Korrektur wird RETURN gedrückt. Der Cursor kann sich hierbei an irgend einer beliebigen Stelle in der Zeile befinden.

Etwas schwieriger als die normale Korrektur ist die Programmkorrektur von Text, das heißt, von Zeichen zwischen Anführungszeichen. Wird eine Zeile geschrieben, so kann der Cursor nach einem Anführungszeichen durch die Cursor-Kontrolltasten nicht mehr bewegt werden, sondern die entsprechenden Befehle werden ins Programm geschrieben. Es ist also nicht möglich, ein falsch geschriebenes Zeichen dadurch zu korrigieren, daß man wie sonst üblich den Cursor zurücksetzt und das Zeichen neu schreibt, sondern man muß DEL benutzen. Eine andere Möglichkeit besteht darin, daß man die RETURN-Taste betätigt und den Cursor dann mit den Cursor-Kontrolltasten wieder in die Zeile führt. Jetzt kann der Cursor auch im Text beliebig bewegt werden und es können Zeichen überschrieben werden. Andererseits können jetzt natürlich keine Cursorbewegungen ins Programm geschrieben werden. Sollen also jetzt solche Cursorbewegungen geändert werden, so geht dies nicht durch Überschreiben, sondern nur durch Löschen des alten Zeichens mit DEL und anschließendem INST. Nach INST werden Cursorbewegungen stets programmiert. Die selben Regeln wie für Cursorbewegungen gelten für HOME, CLR (Bildschirm löschen), RVS und OFF.

Der sicherste Weg Text zu korrigieren, ist speziell für Anfänger das Neuschreiben der Zeile. Man sollte sich jedoch die andere Methode der Textkorrektur baldmöglichst aneignen, da sie speziell bei längeren Texten sehr viel weniger Mühe macht.

Maschinensprache

Mittels PEEK und POKE ist der Inhalt jedes einzelnen Bytes des Arbeitsspeichers zugänglich. Mit dem Befehl PEEK wird der Inhalt der angegebenen Adressen gelesen.
Beispiel:

Der Inhalt der Adresse 59468 soll auf dem Bildschirm dargestellt werden. Dies erfolgt mittels

?PEEK (59468)

Nach Bestätigen der RETURN-Taste erscheint auf dem Bildschirm die Zahl 12 (diese Adresse ist für die Umschaltung des Zeichengenerators zuständig; 12 bedeutet, daß auf graphische Zeichen geschaltet ist).

Mit dem Befehl POKE kann das adressierte Byte mit einem anderen Inhalt überschrieben werden.
Beispiel:

PET soll nun auf Kleinschreibung umgeschaltet werden. Der hierfür notwendige Befehl lautet

POKE 59468,14

Über einen neuen PEEK-Befehl kann man sich vergewissern, daß der neue Inhalt dieser Adresse 14 ist. Durch Schreiben von Buchstaben bei gedrückter SHIFT-Taste sieht man, daß die Umschaltung erfolgt ist.

Sowohl die Adressen als auch (bei POKE) der zu schreibende Wert können Variable sein.

Der zulässige Bereich für die Adressen ist 0 — 65535, der für die Inhalte 0 — 255.

BASIC-Befehle und Anweisungen

Systemkommandos

Systemkommandos sind Anweisungen an den Rechner, die normalerweise nicht programmiert werden. Zwar lassen sie sich auf dem PET 2001 zum großen Teil programmieren, dies ist jedoch nur in einzelnen Fällen sinnvoll.

RUN	RUN	Führt das Programm aus, beginnend mit der niedrigsten Zeilennummer.
	RUN 120	Programmausführung beginnt bei der angegebenen Zeilennummer.
	RUN löscht sämtliche Variablen einschließlich Dimensionierung und bewirkt ein RESTORE. Der Befehl RUN muß mit den Buchstaben R U N eingegeben werden. Er darf nicht mit der Taste "RUN" verwechselt werden!	
CONT	CONT	Setzt die Programmausführung von der momentanen Position aus fort. Man kann ein laufendes Programm durch Betätigen der Taste STOP anhalten, von Hand Variablen auf den Bildschirm bringen, Variablen neu zuordnen, etc. und dann die Programmausführung mittels CONT fortsetzen. Nach Programmkorrekturen oder nach Anhalten des Programms mit einer Fehlermeldung ist CONT nicht möglich.
GOTO	GOTO 160	GOTO ist kein eigentliches Systemkommando, sondern steht normalerweise im Programm. Es kann jedoch als Systemkommando verwendet werden. Die Programmausführung beginnt dann bei der angegebenen Zeile. Es werden jedoch keine Variablen gelöscht, wie es bei RUN 160 der Fall wäre.
LIST	LIST	Das gesamte Programm wird aufgelistet. Wird bei längeren Programmen während des Listens die Taste RVS gedrückt, so wird die Geschwindigkeit der Auflistung verringert, sobald der untere Bildschirmrand erreicht ist.
	LIST 40	Es wird die angegebene Zeile dargestellt.
	LIST —120	Das Programm wird ab der niedrigsten Zeilennummer bis einschließlich der angegebenen Zeile aufgelistet.
	LIST 120—	Das Listen erfolgt ab der angegebenen Zeile bis zum Ende des Programms.
	LIST 40—80	Es wird nur der Programmteil von der ersten angegebenen Zeilennummer bis einschließlich der zweiten angegebenen Zeilennummer gelistet.
	LIST ist programmierbar. Nach Ausführung dieses Befehles wird die Programmausführung unterbrochen.	
NEW	NEW	Löscht den gesamten Programmspeicher einschließlich der Variablen. NEW ist programmierbar. Das Programm löscht sich dann selbst.
CLR	CLR	Löscht sämtliche Variablen einschließliche Dimensionierung sowie sämtliche Rücksprungadressen. CLR ist programmierbar. Dies ist beispielsweise zweckmäßig, wenn für eine neue Aufgabe ein Feld neu dimensioniert werden muß und nicht mit RUN frisch begonnen werden soll. CLR muß mit den Buchstaben CLR eingegeben werden und darf nicht mit der Taste "CLR" verwechselt werden!

BASIC-Befehle

LET	LET G=4	Bei früheren BASIC-Versionen war LET zur Zuordnung notwendig. Es kann auch beim PET 2001 verwendet werden, so daß vorhandene Programme ohne Änderung übernommen werden können. Im angeführten Beispiel genügt es jedoch G=4 zu schreiben.
READ	READ A,A$,B,C	Daten, die unter DATA im Programm stehen, werden in die betreffenden Variablen eingelesen. Bei den Daten kann es sich sowohl um Zahlenwerte als auch um alphanumerische Texte handeln. Daten, die in einem DATA-Befehl stehen, können mittels mehrerer READ-Anweisungen zugeordnet werden. Umgekehrt können auch mit Hilfe eines READ-Befehles die Daten mehrerer DATA-Befehle zugeordnet werden.
DATA	DATA3, Text,4,5	Die Daten, die durch READ zugeordnet werden sollen, können in einem oder mehreren DATA-Befehlen an beliebiger Stelle ins Programm geschrieben werden. Die Zuordnung erfolgt in der Reihenfolge, in der sie im Programm stehen. Text braucht nur dann in Anführungszeichen gesetzt zu werden, wenn er Kommata oder Doppelpunkte oder Zwischenräume am Anfang oder Ende enthält.
RESTORE	RESTORE	Bei jeder READ-Anweisung wird jeweils der nächste unter DATA stehende Wert benutzt. Sind sämtliche vorhandenen Daten eingelesen, so erfolgt bei der nächsten READ-Anweisung die Fehlermeldung OUT OF DATA. In vielen Fällen sollen jedoch die vorhandenen Daten mehrfach verwendet werden. Durch den Befehl RESTORE wird erreicht, daß das Lesen wieder beim ersten Wert beginnt.
IF . . THEN	IFA=OTHEN200	Logischer Vergleich. Im Beispiel wird ein Sprung zur Zeile 200durchgeführt, wenn der Vergleich wahr ist. Im anderen Falle wird das Programm mit der nächsten Zeile fortgesetzt.
	IFB=2THEN?C	Außer Sprüngen können nach einem Vergleich auch andere Befehle verwendet werden. Auch bei mehreren Befehlen, die innerhalb der selben Zeile auf THEN folgen, wird jeweils der gesamte folgende Zeileninhalt bei „wahrem" Vergleich ausgeführt, bei „falschem" Vergleich weggelassen.
FOR . . NEXT	FORI=1T020 . . . NEXT	Die zwischen FOR und NEXT befindlichen Befehle werden so oft durchlaufen, wie Start- und Endwert angeben (im Beispiel 20mal) (Schleife, FOR . . NEXT—Loop). Die angegebene Variable (im Beispiel I) muß eine gewöhnliche Variable, keine Ganzzahl-Variable sein. Die Angabe dieser Variablen nach NEXT, die bei vielen BASIC-Versionen notwendig ist, kann im PET 2001 entfallen.
STEP	FORK=5TO2STEP —.5	Mittels STEP kann eine beliebige Schrittweite (positiv oder negativ) vorgegeben werden. Im Beispiel wird die Schleife nacheinander mit den Werten 5, 4.5, . . ., 2.5,2 durchlaufen.
	FORI=3TO6 FORJ=2TOB . . . NEXT J,I	Bei mehreren ineinandergeschachtelten FOR . . NEXT-Loops (mit oder ohne STEP) genügt eine einzige NEXT-Anweisung, wenn die betreffenden Variablen durch Komma getrennt sind. FOR . . NEXT-Loops können selbst mehrfachgeschachtelt in eine Zeile geschrieben werden.

GOTO	GOTO245	Unbedingter Sprung zu der angegebenen Zeile. GOTO kann auch zu bedingten Sprüngen (nach Vergleichen) verwendet werden. Bei mehreren Befehlen pro Zeile muß GOTO der letzte Befehl sein, da Befehle nach GOTO nie ausgeführt werden.
GOSUB	GOSUB320	Sprung zu einer Subroutine. Beim Durchlaufen des GOSUB-Befehles wird eine „Rücksprungadresse" gesetzt. Wird in der Subroutine der Befehl RETURN erreicht, so erfolgt die Fortsetzung der Programmausführung bei der Rücksprungadresse. Die Rücksprungadresse steht unmittelbar beim GOSUB-Befehl, so daß in einer Zeile mehrere GOSUB-Befehle stehen können, die alle ausgeführt werden. Innerhalb einer Subroutine kann eine weitere Subroutine gerufen werden, von der aus wiederum eine Subroutine gerufen werden kann . . . . . Die mögliche „Schachtelungstiefe" beträgt 26 Ebenen. Subroutinen können an beliebiger Stelle des Programmes stehen und beliebig oft, auch von verschiedenen Stellen des Programms aus gerufen werden.
RETURN	RETURN	RETURN ist der letzte Befehl in einer Subroutine und bewirkt den Rücksprung zum letzten durchlaufenen GOSUB-Befehl. Wird RETURN erreicht, ohne daß ein entsprechender GOSUB-Befehl durchlaufen wurde, so erscheint die Fehlermeldung RETURN WITHOUT GOSUB ERROR. Dies erfolgt manchmal dadurch, daß anstelle eines GOSUB-Befehles ein GOTO-Befehl verwendet wurde. Meist ist die Ursache jedoch darin zu suchen, daß Subroutinen häufig ans Ende des Programms geschrieben werden und dann vergessen wird, vor deren Beginn ein STOP oder END zu setzen. Nach Abarbeitung des gesamten Programms „fällt" der Rechner dann in die Subroutine. RETURN muß Buchstabe für Buchstabe eingegeben werden und darf nicht mit der Taste „RETURN" verwechselt werden!
ON . . GOTO	ONF GOTO50, 60,70	Mittels dieser „berechneten" Verzweigung kann vom Wert einer Variablen abhängig gemacht werden, an welcher Stelle das Programm fortgesetzt werden soll. Im Beispiel wird das Programm bei F=1 mit der Zeile 50, bei F=2 mit der Zeile 60 und bei F=3 mit der Zeile 70 fortgesetzt. In allen anderen Fällen wird das Programm ohne Sprung fortgesetzt. Statt einer Variablen kann auch ein mathematischer Ausdruck verwendet werden.
ON . . GOSUB	ONJGOSUB55, 65,75	Genau wie bei der berechneten Verzweigung kann bei Subroutinen verfahren werden.

Sowohl bei berechneten Verzweigungen als auch bei berechneten Subroutinen ist die Anzahl der „Zweige" nur begrenzt durch die verfügbare Zeilenlänge. Reicht diese nicht aus, um alle Zeilennummern unterzubringen, so können diese auf zwei oder mehrere Zeilen verteilt werden. In der ersten Zeile, die 8 Zeilennummern enthält, kann dann stehen: ONK . ., auf der nächsten dann: ONK—8. . . Auf diese Weise kann zu beliebig vielen Punkten verzweigt werden.

PRINT	PRINTA,X$,B	Der PRINT-Befehl bewirkt die Darstellung der angegebenen Variablen auf dem Bildschirm, und zwar in dem Format, das durch die Verwendung von Komma bzw. Semicolon vorgegeben ist (Näheres siehe Kapitel Programmierung — 6. Ausgabeformat). Statt PRINT kann ? benutzt werden. Wird das Programm aufgelistet, sieht man, daß ? durch PRINT ersetzt wurde.
	PRINT	PRINT ohne Variablenangabe bewirkt den Zeilenvorschub.
DIM	DIMA(6,6) DIMC$(30) DIME(N)	Felder, die in einer Dimension mehr als 11 Elemente (0—10) enthalten, müssen vor ihrer ersten Verwendung dimensioniert werden. Nachdem eine automatische Dimensionierung bei der ersten Verwendung eines Elementes des Feldes erfolgt und eine Dimensionierung nur einmal möglich ist, ist es ratsam, die Dimensionierung ganz am Anfang des Programmes vorzunehmen. Die Dimensionierung kann außer mit Konstanten auch mit Variablen oder mathematischen Ausdrücken vorgenommen werden. Die Anzahl der Dimensionen ist nur durch die verfügbare Speicherkapazität begrenzt.
END	END	Im allgemeinen letzte Anweisung eines Programms. Kann weggelassen werden. END kann auch mehrfach innerhalb eines Programmes verwendet werden. Bewirkt Programmhalt ohne Meldung „BREAK IN . . .“ wie bei STOP. Nach END kann das Programm mittels CONT fortgesetzt werden.
REM	REM TEXT	Erlaubt Bemerkungen in das Programm zu schreiben (keine Wirkung im Programm). Wird REM gemeinsam mit anderen Befehlen in einer Zeile verwendet, so muß REM der letzte Befehl der Zeile sein. Befehle nach REM werden nicht ausgeführt!
INPUT	INPUTA INPUTC,F$ INPUT“TEXT“;A	Erlaubt Eingabe von Werten über die Tastatur (siehe Kapitel Programmierung — 4. Eingabe).
DEFFN	DEFFNY(X)=A*B	Oft gebrauchte Funktionen können definiert und dann vom Programm gerufen werden. Im Beispiel ist Y der Name der Funktion (muß einer der 26 Buchstaben sein), „(X)“ ist eine „Füll-Variable“. DEFFN muß im Programm eine niedrigere Zeilennummer besitzen als der erste Ruf nach dieser Funktion. Maximale Länge der definierten Funktion ist eine Zeile.
FN	?FNY(X)	Funktion wird aufgerufen. Im Beispiel wird A*B ausgegeben.
GET	GETV	Eine einzelne Ziffer wird vom Tastenfeld in die Variable V gelesen.
	GETV$	Ein einzelnes Zeichen wird vom Tastenfeld in V$ eingelesen. Nähere Ausführungen zu GET sind in Kapitel Programmierung — 4. Eingabe zu ersehen.
STOP	STOP	Beendet ein laufendes Programm. Bewirkt die Meldung BREAK IN . . . (Zeilennummer). Nach STOP kann das Programm mittels CONT fortgesetzt werden.

Input-Output-Anweisungen

OPEN	OPEN1,1,1	Eröffnung eines logischen File (jedes Peripheriegerät wird als File angesprochen). Der erste Parameter ist die File-Nr. (0 bis 255, max. 10 Files gleichzeitig offen), der zweite die Gerätenummer (0=Tastatur, 1=eingebaute Kasette, 2=2. Kassette, 3=Bildschirm, 4—15=externe Geräte), der dritte bestimmt, ob Ein- oder Ausgabe (0=Lesen, 1=Schreiben, 2=Schreiben mit zusätzlichem End-of-Tape-Zeichen). Im Beispiel wird File 1 zur Aufzeichnung auf die eingebaute Kassette eröffnet.
	OPEN1	Der Ersatzwert für den 2. Parameter ist 1, für den dritten 0. Beispiel: File 1 wird zum Lesen von der eingebauten Kassette eröffnet.
	OPEN1,1,1, "Daten"	Files können mit Namen bezeichnet und dann gezielt geladen werden. Namen dürfen nicht in anderen Namen enthalten sein: Das File ZINSESZINS wird auch geladen, wenn ZINS gesucht wird!
CLOSE	CLOSE1	Schließen eines File. Eröffnete Files müssen stets wieder geschlossen werden!
SAVE	SAVE	Aufzeichnung eines Programms auf die eingebaute Kassette.
	SAVE"NAME"	Aufzeichnung eines Programms auf die eingebaute Kassette mit Namen.
LOAD	LOAD	Laden des nächsten auf der Kassette befindlichen Programms.
	LOAD"NAME"	Laden des mit Namen bezeichneten Programms von der eingebauten Kassette. LOAD ohne zusätzlichen Namen kann ersetzt werden durch Drücken der Taste RUN (SHIFT und STOP). In diesem Falle wird sofort mit der Programmausführung begonnen, das heißt, der so gegebene LOAD-Befehl beinhaltet den Befehl RUN.
VERIFY	VERIFY	Vergleich des im Rechner gespeicherten Programmes mit dem nächsten auf der Kassette befindlichen Programm.
	VERIFY"NAME"	Vergleich des unter diesem Namen auf der Kassette befindlichen Programms mit dem in der Maschine gespeicherten.
	Bei allen Kassettenoperationen mit Namen gilt die selbe Regelung wie bei OPEN: Namen dürfen nicht in anderen Namen enthalten sein.	
PRINT#	PRINT# 1,A,B,C	Die Variablen A, B und C werden im File 1 ausgegeben auf das Gerät, das im OPEN-Befehl dem File 1 zugeordnet wurde. Dieses File muß ein Ausgabefile sein (3. Parameter 1 oder 2)
INPUT#	INPUT# 2,X,Y,Z	Vom File 2 und dem im OPEN-Befehl zugeordneten Gerät werden Daten in X, Y und Z eingelesen. Dieses File muß ein Eingabefile sein (3. Parameter 0). Soll INPUT ausgeführt werden, ohne daß das File eröffnet wurde, erscheint FILE NOT OPEN ERROR.
GET#	GET# 2,A$	Ein einzelnes Zeichen wird vom File 2 in A$ eingelesen. Zu beachten: Im Gegensatz zu INPUT # wird GET # durch das vom Band gelesene Zeichen CR (das nach jeder PRINT-Anweisung auf das Band geschrieben wird) *nicht* beendet! Es muß also gegebenenfalls geprüft werden, ob es sich um das ASCII-Zeichen 13 handelt bzw., ob ST<>0 ist.

String-(Zeichenketten)-Befehle

LEN	A=LEN(A$)	Die tatsächliche Länge des Strings (Anzahl der Zeichen einschließlich der Leerstellen) wird festgestellt und im Beispiel in A gespeichert.
VAL	H=VAL(B$)	Beginnt der String mit Zahlen, Leerstellen, +, — oder einem Dezimalpunkt, so wird der Zahlenwert in eine gewöhnliche Variable umgewandelt. Beginnt der String mit nicht-numerischen Zeichen, so ist VAL=0.
STR$	N$=STR$(X)	Eine normale Zahl wird in einen String umgewandelt.
ASC	?ASC(D$)	Der ASCII-Code des ersten Zeichens im String wird ermittelt und im Beispiel auf dem Bildschirm dargestellt.
CHR$	A$=CHR$(77)	Das zum ASCII-Code 77 gehörende Zeichen (M) wird ermittelt und in A$ gespeichert.
LEFT$	L$=LEFT$(A$,3)	Die ersten (hier 3) Zeichen von A$ werden in L$ gespeichert. Hat A$ weniger Zeichen als spezifiziert, so wird der gesamte String dargestellt.
RIGHT$	?RIGHT$(B$,5)	Die letzten (hier 5) Zeichen von B$ werden auf den Bildschirm gebracht. Hat B$ weniger Zeichen als spezifiziert, so wird der gesamte String dargestellt.
MID$	M$=MID$(C$,4)	Vom 4. Zeichen ab wird C$ in M$ gespeichert.
	M$=MID$(C$4,2)	2 Zeichen, beginnend mit dem 4. Zeichen, aus C$, werden in M$ gespeichert.
+	G$=A$+B$	Aus A$ und B$ wird ein neuer String (G$) erzeugt. A$ und B$ bleiben unverändert. Gegebenenfalls ist sicherzustellen, daß der erzeugte String nicht länger wird als 255 Zeichen.

Logische Operatoren: AND, OR und NOT

Mit diesen 3 Operatoren können Boole'sche Operationen durchgeführt werden. Sie wandeln die Argumente zunächst in 16—Bit Binärzahlen um (dezimaler Bereich von —32768 bis 32767). Anschließend werden die jeweiligen Operationen Bit für Bit vorgenommen. Im Falle NOT ist das Ergebnis das binäre Komplement, dezimal bedeutet dies: NOT X=—(X+1). Bei AND und OR ist das Ergebnis jeweils die Zahl, die sich bei AND- bzw. OR-Verknüpfung jeweils gleichwertiger Bits ergibt.

Beispiel:	binär	dezimal
Argument 1	I O I I O O I	89
Argument 2	I O O I I O I	77
AND	I O O I O O I	73
OR	I O I I I O I	93
Argument	I O I I O O I	89
NOT	I I I I I I I I O I O O I I O	-90

Dieser bitweise Vergleich führt einerseits dazu, daß bei einer Entscheidung, die wahr sein soll, wenn sowohl A als auch B von O verschieden sind, nicht geschrieben werden darf: IFAANDBTHEN . . . (2AND4=O!). Die Schreibweise muß sein: IFA«»OANDB«»OTHEN . . . Der Vorteil dieser Methode des Bit-weisen Vergleiches ist jedoch darin zu suchen, daß diese Operatoren nicht nur in logischen Vergleichen verwendet werden können, sondern daß auch Vergleiche oder Operationen mit einzelnen Bits durchgeführt werden können.
Beispiele:
?IFA=4ANDB=5THEN . . . . .
?IFA=2ORB=4THEN . . . . . .
N=NOTA

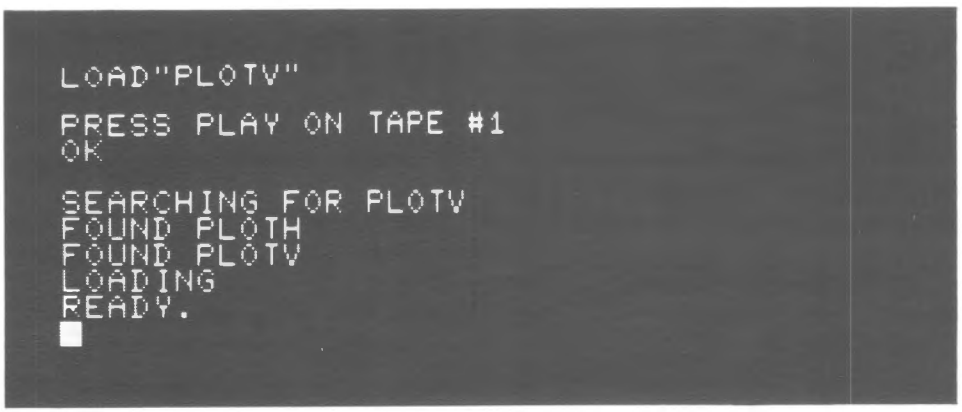
Kassettenoperationen

Es ist nicht notwendig, besonders teure HIFI- oder spezielle EDV-Kassetten zu verwenden. Man sollte, um übermäßige Suchzeiten zu vermeiden, C 90- oder C 120-Kassetten nicht benützen. Es lassen sich auf diesen zwar sehr viel mehr Programme oder Daten speichern, die Praxis zeigt jedoch, daß es viel zweckmäßiger ist, mehrere kurze Bänder statt eines langen zu verwenden. Selbst C 60-Kassetten bieten reichlich Speicherplatz. Um nicht Gefahr zu laufen, daß öfter als notwendig die Meldung READ ERROR auftritt, sollte man vermeiden, die billigsten Kassetten zu verwenden.

Mittels der eingebauten Kassette können Programme und Daten aufgezeichnet und in den Speicher eingelesen werden.

Programme

Um ein Programm auf der eingebauten Kassette aufzuzeichnen, muß die Kassette, auf die das Programm geschrieben werden soll, eingelegt werden. Es muß REW (rewind) gedrückt werden, um die Kassette zurückzuspulen. Danach ist SAVE einzugeben und die RETURN-Taste zu drücken (soll ein Programm mit einem Namen bezeichnet werden, um es später gezielt laden zu können, so muß SAVE "NAME" eingegeben werden). Auf dem Bildschirm erscheint die Aufforderung: PRESS PLAY & RECORD ON TAPE # 1. Nach dem Drücken dieser beiden Tasten bringt der Bildschirm die Mitteilung: OK WRITING (bei einem mit Namen gekennzeichneten Programm ist auf dem Bildschirm zu sehen: OK WRITING NAME). Nach der Beendigung der Aufzeichnung erscheint READY und die Kassette hält an. Zum Laden eines Programmes von der Kassette gibt es zwei Möglichkeiten. Wird die Taste RUN (SHIFT und STOP) gedrückt, so erscheint auf dem Bildschirm: LOAD, PRESS PLAY ON TAPE # 1. Bestätigt man die PLAY-Taste, kann man folgende Mitteilung auf dem Bildschirm lesen: OK SEARCHING. Sobald der Rechner den ersten Programmanfang nach der augenblicklichen Bandposition gefunden hat, zeigt der Bildschirm: FOUND (bzw. FOUND NAME), LOADING. Nach Beendigung des Ladens wird das Programm automatisch gestartet. Soll ein Programm gezielt eingeladen werden, so ist es erforderlich LOAD "NAME" einzugeben, der Rechner antwortet wieder mit PRESS PLAY ON TAPE 1. Anschließend erfolgen die Meldungen: OK, SEARCHING FOR NAME, FOUND NAME und LOADING. Werden auf der Suche nach dem aufgerufenen Programm andere Programme gefunden, so werden diese auf dem Bildschirm registriert, jedoch nicht geladen. Erst wenn das gesuchte Programm erreicht ist, zeigt der Bildschirm LOADING und es wird geladen. In diesem Fall muß das Programm jedoch durch RUN gestartet werden.
Beispiele:



Treten beim Ladevorgang Fehler auf (schlechtes oder verschmutztes Band), so erfolgt die Meldung LOAD ERROR. Das Laden muß dann wiederholt werden.

Bei der Namensgebung eines Programmes ist zu beachten, daß ein Name nicht bereits im Namen eines anderen Programmes enthalten sein darf. So wird beispielsweise das Programm ZINSES-ZINS auch geladen, wenn nach dem Programm ZINS gesucht wird.

Soll ein Programm auf eine Kassette (Zielkassette), auf der sich bereits Programme befinden, gespeichert werden, so ist darauf zu achten, daß sich das Band bei Beginn der Aufzeichnung in einer Position nach dem letzten der vorangegangenen Programme befinden muß. Am einfachsten geht man hier folgendermaßen vor: Das aufzuzeichnende Programm wird zunächst auf einer Hilfskassette aufgezeichnet. Um auf die richtige Bandposition der Zielkassette zu gelangen, wird nun das letzte der vorhandenen Programme in den Rechner geladen und die Kassette herausgenommen, ohne die Bandposition zu verändern. Anschließend wird das Programm von der Hilfskassette wieder geladen und auf die Zielkassette aufgezeichnet.

Mit dem Befehl VERIFY kann geprüft werden, ob das im Rechner vorhandene Programm mit einem auf der Kassette aufgezeichneten identisch ist. Vorzugehen ist hier genau wie bei LOAD. Sind die Programme identisch, so antwortet der Rechner mit OK, im anderen Falle bringt er die Mitteilung: VERIFY ERROR.

Daten

Zur Eingabe von Daten muß ein logisches File mit seiner File-Nummer eröffnet werden. Der Befehl PRINT # hat nun auf dieses File die gleiche Wirkung wie der Befehl PRINT für den Bildschirm. Der Befehl INPUT # entspricht dem Befehl INPUT (der Befehl INPUT # bezieht sich auf das File in der gleichen Weise wie der Befehl INPUT auf das Tastenfeld). Zu beachten ist folgendes:

1. PRINT muß immer Buchstabe für Buchstabe eingegeben werden. PRINT darf in diesem Fall nicht durch ? ersetzt werden. ? # n führt zur Fehlermeldung SYNTAX ERROR.
2. Für PRINT -Operation muß das logische File ein Ausgabefile sein, das heißt, der dritte Parameter muß 1 oder 2 lauten. Entsprechend muß für INPUT -Operationen ein Eingabefile mit 0 als dritten Parameter definiert sein.
3. Die File-Nummer kann eine beliebige Zahl im Bereich von 1 bis 255 sein. Der zweite Parameter muß für die eingebaute Kassette 1 lauten.
4. Wird nur die File-Nummer angegeben, so wird als zweiter Parameter 1 (eingebaute Kassette) und als dritter Parameter 0 angenommen, das heißt, OPEN 4 ist gleichbedeutend mit OPEN 4,1,0.
5. Eröffnete File müssen stets wieder geschlossen werden. Der entsprechende Befehl ist CLOSE n. Der Befehl, das File 4 zu schließen, nachdem es mit dem Befehl OPEN 4,1,0 eröffnet wurde, lautet: CLOSE 4.

Die zur Aufzeichnung von Daten auf die Kassette und das Wiedereinlesen notwendigen Operationen werden am besten durch das folgende Programm erklärt:

```
 20 PRINT"{DOWN}DATEN EINGEBEN"
 30 FORI=1TO5:INPUTA$(I):NEXT
 40 PRINT"DATENKASSETTE EINLEGEN UND RUE
CKSPULEN"
 50 PRINT"DANN 'S' DRUECKEN"
 60 GETA$:IFA$<>"S"THEN60
 70 OPEN1,1,1
 80 FORI=1TO5:PRINT#1,A$(I):NEXT
 100 CLOSE1
 200 CLR
 210 PRINT"DATENKASSETTE RUECKSPULEN, DA
NN 'L'"
 220 GETA$:IFA$<>"L"THEN220
 230 OPEN3
 240 FORI=1TO5:INPUT#3,A$(I):NEXT
 260 CLOSE3
 270 FORI=1TO5:PRINTA$(I):NEXT
READY.
```

Zeile 20 löscht den Bildschirm und fordert zur Dateneingabe auf. In Zeile 30 werden 5 Werte in A(1) bis A(5) eingegeben. Die Zeilen 40 und 50 fordern zum Einlegen der Datenkassette und Rückspulen sowie zum Drücken der Taste „S" (Schreiben) auf. In Zeile 60 wartet der Rechner bis die Taste „S" gedrückt wurde. In Zeile 70 wird das File 1 zum Schreiben auf der Kassette 1 eröffnet. In Zeile 80 werden die Werte aus A(1) bis A(5) auf der Kassette aufgezeichnet. Zeile 100 schließt das File 1. Zeile 200 löscht sämtliche Variablen und stellt damit sicher, daß die anschließend gezeigten Werte nur von der Kassette stammen können. Nachdem die in Zeile 210 gegebene Anweisung zum Rückspulen ausgeführt wurde, kann nach Drücken von „L" (Lesen) die Warteschleife in Zeile 220 verlassen werden und es erfolgt die Eröffnung des Eingabefile 3. Zeile 240 liest die Werte von der Kassette wieder in A(1) bis A(5), die nach Schließen des File 3 in Zeile 260 dann durch die Zeile 270 auf dem Bildschirm dargestellt werden.
Beispiel:

```
DATEN EINGEBEN
? 2.2
? 6.75
? 3.123
? 63.47
? 3.14159
DATENKASSETTE EINLEGEN UND RUECKSPULEN
DANN 'S' DRUECKEN

PRESS PLAY & RECORD ON TAPE #1
OK
DATENKASSETTE RUECKSPULEN, DANN 'L'

PRESS PLAY ON TAPE #1
OK
2.2
6.75
3.123
63.47
3.14159

READY.
```

Wird statt des normalen Feldes A( ) das Feld A$( ) verwendet, so können beliebige Zeichen aufgezeichnet werden.
Beispiel:

```
 20 PRINT"{CLR}DATEN EINGEBEN"
 30 FORI=1TO5:INPUTA(I):NEXT
 40 PRINT"DATENKASSETTE EINLEGEN UND RUE
CKSPULEN"
 50 PRINT"DANN 'S' DRUECKEN"
 60 GETA$:IFA$<>"S"THEN60
 70 OPEN1,1,1
 80 FORI=1TO5:PRINT#1,A(I):NEXT
 100 CLOSE1
 200 CLR
 210 PRINT"DATENKASSETTE RUECKSPULEN, DA
NN 'L'"
 220 GETA$:IFA$<>"L"THEN220
 230 OPEN3
 240 FORI=1TO5:INPUT#3,A(I):NEXT
 260 CLOSE3
 270 FORI=1TO5:PRINTA(I):NEXT
READY.
```

```
DATEN EINGEBEN
? MAIER
? MUELLER
? SCHMIDT
? SCHULZE
? BAUER
DATENKASSETTE EINLEGEN UND RUECKSPULEN
DANN 'S' DRUECKEN

PRESS PLAY & RECORD ON TAPE #1
OK
DATENKASSETTE RUECKSPULEN, DANN 'L'

PRESS PLAY ON TAPE #1
OK
MAIER
MUELLER
SCHMIDT
SCHULZE
BAUER

READY.
```

Statt INPUT kann GET verwendet werden. Hierbei ist jedoch zu beachten, daß die GET-Anweisung nicht wie INPUT durch ein „carriage return"(ASCII 13) beendet wird. Um zu verhindern, daß auf der Kassette immer weitergelesen wird — was zur Folge haben kann, daß der Rechner auf keine Taste mehr reagiert und nur durch Aus- und Einschalten wieder betriebsfähig gemacht werden kann — muß jedes gelesene Zeichen überprüft werden, ob es sich um einen CR-Befehl handelt. Das abgeänderte Programm sieht folgendermaßen aus:

```
 20 PRINT"{CLR}DATEN EINGEBEN"
 30 FORI=1TO5:INPUTA$(I):NEXT
 40 PRINT"DATENKASSETTE EINLEGEN UND RUE
CKSPULEN"
 50 PRINT"DANN 'S' DRUECKEN"
 60 GETA$:IFA$<>"S"THEN60
 70 OPEN1,1,1
 80 FORI=1TO5:PRINT#1,A$(I):NEXT
 100 CLOSE1
 200 CLR
 210 PRINT"DATENKASSETTE RUECKSPULEN, DA
NN 'L'"
 220 GETA$:IFA$<>"L"THEN220
 230 OPEN3
 240 FORI=1TO5
 242 GET#3,Q$:IFASC(Q$)<>13THENA$(I)=A$(
I)+Q$:GOTO242
 244 NEXT
 260 CLOSE3
 270 FORI=1TO5:PRINTA$(I):NEXT
READY.

■
```

Der FOR . . NEXT-Loop in Zeile 240 wird in 3 Zeilen aufgeteilt. In der neuen Zeile 242 wird durch GET 3 Zeichen für Zeichen in Q$ übernommen und — falls es sich nicht um CR (ASCII) handelt — zum String A$(I) hinzugefügt. Bei CR wird das Programm mit Zeile 244 fortgesetzt und der nächste String A$(I) begonnen.

Peripheriegeräte

IEEE 488-Bus (IEC-Bus)

Das Interface nach IEEE 488 ist bereits eingebaut. Außer den 8 Datenleitungen DI01 bis DI08 sind 3 Handshake-Leitungen und 5 Steuerleitungen vorhanden. Das im PET eingebaute IEEE 488 Interface weist folgende Besonderheiten auf:

1. REN (remote enable) ist immer auf low.
2. IFC (interface clear) wird stets beim Einschalten ausgelöst (Impuls ca. 100 ms).
3. PET erwartet innerhalb von 60 ms Antwort vom Peripheriegerät. Bei langsamen Peripheriegeräten muß durch den Befehl WAIT der Programmablauf bis zur Rückmeldung angehalten werden. Diese Eigenschaft ermöglicht eine Datenübernahme bzw. -übergabe auch mit Geräten, die der IEEE-Norm nicht voll entsprechen.

Der Anschlußstecker entspricht den elektrischen Spezifikationen nach IEEE 488, aber nicht den mechanischen. Dies ist jedoch kein Nachteil, da sich IEEE-Bus und IEC-Bus bei gleichen elektrischen Spezifikationen im Stecker unterscheiden.
Steckerbelegung:

1. DI01	7. NRFD	A. DI05	H. ⏚
2. DI02	8. NDAC	B. DI06	J. ⏚
3. DI03	9. IFC	C. DI07	K. ⏚
4. DI04	10. SRQ	D. DI08	L. ⏚
5. EOI	11. ATN	E. REN ⏚	M. ⏚
6. DAV	12. ⏚	F. ⏚	N. ⏚

Steuerbefehle für den IEEE-Bus

Zusammenfassung der für Input-/Output-Operationen notwendigen BASIC-Befehle:

OPEN
CLOSE
INPUT
GET
PRINT
CMD
WAIT

Peripheriegeräte, die am IEEE-Bus angeschlossen sind, werden als logische Files angesprochen. Bei der Eröffnung des Files gilt folgendes:

OPEN A,B,C,"TEXT"
A ist die File-Nummer (logische Adresse) im Bereich von 1 bis 255. Alle INPUT -, GET -, PRINT -, CMD - und CLOSE-Befehle beziehen sich auf diese logische Adresse.

B ist die Geräteadresse (Primäradresse) und liegt im Bereich von 4 bis 15. C ist die Sekundäradresse und braucht nur bei Bedarf angegeben werden. Der Bereich der Adresse erstreckt sich von 0—31. Die Sekundäradresse wird nur bei OPEN- und CLOSE-Befehlen ausgegeben. Erfolgt die Ausgabe, so werden die Bits 6 und 7 gesetzt. Wird „TEXT" eingegeben, so *muß* die Sekundäradresse angegeben werden. In diesem Falle wird sie folgendermaßen ausgegeben: Bei der Ausführung von OPEN und CLOSE wird Bit 8 gesetzt, bei OPEN zusätzlich noch Bit 5.

Der OPEN-Befehl zieht die ATN-Leitung auf low, das heißt, beim OPEN-Befehl werden Kommandos ausgegeben. CLOSE gibt ebenfalls die Sekundäradresse aus. Eröffnete Files müssen stets geschlossen werden!

PRINT hat folgende Wirkung:

1. Das durch die File-Nummer bestimmte Gerät wird als Listener adressiert.
2. Die spezifizierten Daten werden ausgegeben.
3. Es wird unlisten an alle Geräte ausgegeben.

Bei Geräten, die nur Talker sind und daher nur Befehle empfangen können, besteht die Möglichkeit, mit PRINT auch Befehle auszugeben.

INPUT bewirkt analog die Adressierung als Talker, das Einlesen von Daten und die Ausgabe des Untalk-Befehles.

CMD hat beim PET eine andere Bedeutung als bei manchen anderen Rechnern. CMD dient nicht zur Ausgabe von Kommandos — dies geschieht bei PET im OPEN-Befehl — sondern leitet jegliche Ausgabe auf den IEEE-Bus. So bewirkt der Befehl LIST — wenn vorher CMD ausgeführt wurde — daß das Programm nicht mehr auf dem Bildschirm, sondern auf einem beispielsweise angeschlossenen Drucker gelistet wird. Da CMD den IEEE-Bus aktiv hält, erlaubt es auch Mehrfachadressierung.

WAIT A,B,C wirkt folgendermaßen:
Der Inhalt der in A angegebenen Adresse wird mit B durch die UND-Funktion sowie mit C durch Exclusiv-ODER-Funktion verknüpft (Wird C nicht angegeben, so wird O angenommen). Das Programm wird solange angehalten, bis das Ergebnis dieser Verknüpfung von Null verschieden ist.
Die UND-Verknüpfung gestattet es, einzelne Bits zu überwachen, indem der Wert des betreffenden Bits als Maske unter B angegeben wird.

Beispiel:
Bit 1 (geringstwertiges Bit) der Adresse 59456 ist die Leitung NDAC. WAIT 59456,1 bewirkt nun ein Anhalten des Programmes bis dieses Bit gesetzt wurde. Das Bit 7 der selben Adresse zeigt den Zustand der Leitung NRFD. Mit dem Befehl WAIT 59456,64 wird also NRFD überwacht. Der Parameter C gestattet die Umkehrung der Logik. Wird er zu 1 gesetzt, so wird die Wartezeit beendet, wenn das betreffende Bit 0 wird.

Dies sind die für den IEEE 488-Bus wichtigen Adressen:

Dezimal Adr.	Bits	IEEE-Leitungen	
59424	1—8	DIO 1—8	(INPUT)
59426	1—8	DIO 1—8	(OUTPUT)
59425	4	NDAC	(OUTPUT)
59427	4	DAV	(INPUT)
	5	SRQ	(INPUT)
59408	7	EOI	(INPUT)
59456	1	NDAC	(INPUT)
	2	NRFD	(OUTPUT)
	3	ATN	(OUTPUT)
	7	NRFD	(INPUT)
	8	DAV	(OUTPUT)

(Die in der Aufstellung aufgeführten Leitungen müßten eigentlich überstrichen werden, da es sich um Active Low handelt. Nach der neuen Norm ist dies jedoch nicht mehr erforderlich.)

Bei Datenübertragung von langsamen Peripheriegeräten kann statt der Verwendung von WAIT die Statusvariable ST abgefragt und die übernommenen Werte nur dann weiterverarbeitet werden, wenn ST gleich Null ist.

Die Statusvariable ST ist:

0	für gültige Daten
1	bei überschrittener Zeit — bei Übernahme der Daten ohne Handshake —
64	für EOI und
128	falls Peripheriegeräte über- haupt nicht geantwortet hat — ergibt den DEVICE NOT PRESENT ERROR.

Im übrigen wird auf die Bedienungsanleitungen der jeweils anzuschließenden Peripheriegeräte und auf den IEEE-Standard 488 hingewiesen.

USER PORT

Dieser Stecker enthält neben den zum Test mit dem Diagnosestecker notwendigen Leitungen und Ausgängen für einen externen Bildschirm-Monitor (Video, Horizontal- und Vertikalsynchronisation) noch einen 8-Bit Parallel-Ein-/Ausgang mit 2 Handshake-Leitungen.

Steckerbelegung:

1.	7. Kassette 2 (w)	A. ⏚	H. PA4
2. TV video	8. Kassette Aufn.	B. CA1	J. PA5
3. IEEE SRQ	9. TV vertical	C. PAO	K. PA6
4. IEEE EOI	10. TV horizontal	D. PA1	L. PA7
5. Diagnostik	11. ⏚	E. PA2	M. CB2
6. Kassette 1 (w)	12. ⏚	F. PA3	N. ⏚

Die Ein- und Ausgangsleitungen können mittels PEEK und POKE einzeln gelesen bzw. gesetzt werden. Die Adresse ist 59471. Die Zuordnung der einzelnen Bits zu den Leitungen sieht folgendermaßen aus:

PIN	C	D	E	F	H	J	K	L
Bit	1	2	3	4	5	6	7	8

CA1 ist die Eingang-, DB2 die Ausgang-Handshake-Leitung. CB2 kann auch als serieller Ein- oder Ausgang Verwendung finden.

Beim Einschalten sind die Datenleitungen als Eingangsleitung programmiert. CA1 reagiert auf negative Übergänge. Das Umprogrammieren von CA1 erfolgt mit dem Befehl POKE 59468, PEEK (59468) OR 1. Hierdurch wird auf der angegebenen Adresse zusätzlich Bit 1 gesetzt und damit die Logikrichtung von CA1 umgestellt. Die Adresse des Flag-Register ist 59469. Mittels des Befehles WAIT 59469,2 kann dieses Register das Programm bis zum Empfang des Flag-Signals stoppen. Anschließend wird der übermittelte Wert durch D=PEEK (59457) aus der Adresse 59457 in eine Variable übernommen. Es kann sich hierbei um eine Zahl zwischen 0 und 255 handeln. Durch dieses PEEK wird außerdem das CA1-Register zurückgesetzt, sodaß der Rechner zur Übernahme des nächsten Zeichens bereit ist.
Die 8 Datenleitungen können durch den Befehl: POKE 59459,255 zu Ausgangsleitungen umprogrammiert werden. Die Ausgangs-Handshake-Leitung CB2 wird durch den Befehl:

POKE 59468, PEEK (59468) OR 244

auf 1 gesetzt, bzw. durch den Befehl:

POKE 59468, PEEK (59468) AND 31 OR 192

auf Null gebracht.

Interface für zweite Kassette

An dem dafür vorgesehenen Stecker kann eine zweite Kassetteneinheit angeschlossen werden. Die eingebaute Kassette wird mit 1 angesprochen, die zweite mit 2.
Steckerbelegung:

1. Kein Anschluß	1. ⏚
2. „ „	2. +5
3. „ „	3. Motor
4. „ „	4. Wiedergabe
5. „ „	5. Aufnahme
6. „ „	6. Motorstrom-Überprüfung

Speichererweiterungsanschluß

Durch den Anschluß externer RAM's ist eine Speichererweiterung bis zu 32k möglich. Die notwendigen Leitungen stehen an dem seitlichen Stecker zur Verfügung. Die Stromversorgung des Zusatzspeichers muß extern erfolgen.
Steckerbelegung:

2. A0 Adreß-Bit	42. NS6 Adreß select
4. A1 „	44. NS7 „
6. A2 „	46. NS9 „ „
8. A3 „	48. NSA „ „
10. A4 „	50. NSB „ „
12. A5 „	52. kein Anschluß
14. A6 „	54. Reset
16. A7 „	56. Interrupt
18. A8 „	58. Clock
20. A0 „	60. Read/Write
22. A10 „	62. kein Anschluß
24. A11 „	64. kein Anschluß
26. kein Anschluß	66. BD0 Daten-Bit
28. kein Anschluß	68. BD1 „ „
30. kein Anschluß	70. BD2 „ „
32. NS1 Adreß select	72. BD3 „ „
34. NS2 „ „	74. BD4 „ „
36. NS3 „ „	76. BD5 „ „
38. NS4 „ „	78. BD6 „ „
40. NS5 „ „	80. BD7 „ „

commodore

Commodore Büromaschinen GmbH
Frankfurter Straße 171-175
Postfach 4 26
6078 Neu-Isenburg
Telefon (0 61 02) * 80 03 · Telex 4185663 como d

2,5 K 6/78