MICRONIC COMPUTER SYSTEME GMBH

MCS-BASIC

Handbuch

Version 2.0

Zentralflughafen Tempelhof
Tempelhofer Damm 45
1000 Berlin 42

Telefon: 030/690 94 94/95
Telex: 0184 187 vocod

INHALTSVERZEICHNIS

	Seite
Einleitung	1
Einfacher Dialog + Basic	
Direktes Rechnen	2
Variablenarten	3
Dateneingabe	6
Log. Vergleichsoperationen	7
SYNTAX	10
Datenausgabe	11
Sprünge, Verzweigungen, Unterprogramme	11
Programmschleifen	13
Mathematische Funktionen	15
Fehlersuche	16
Fehlermeldungen	16
Liste der BASIC-Befehle und Anweisungen	20
INPUT/Output-Anweisungen	28
OPEN	
CLOSE	
LOAD, SAVE, VERIFIY	
INPUT#, GET#, PRINT#	
ANHANG I	36
ANHANG II	37

E i n l e i t u n g

Dieses Handbuch soll dem Anwender als 'BASIC-Kurzanleitung' und als 'Nachschlagelektüre' über den BASIC-Interpreter sowie des Plattenbetriebssystems BDOS dienen.

Entsprechend dem funktionellen Aufbau des Gesamtsystems, zergliedert sich dieses Buch in zwei Teile:

- Beschreibung des BASIC-Interpreters
- Input/Output-Anweisungen für Floppy Disk

Im Anhang wird das Benutzen des Plattenbetriebssystems ohne Verwendung des BASIC-Interpreters beschrieben. Eine genaue Kenntnis des Monitors MONA ist jedoch für denjenigen Benutzer, der ausschieGlich mit BASIC arbeiten will, nicht erforderlich!

Dieses Handbuch erhebt keinen Anspruch auf Vollständigkeit, was Einzelheiten des Interpreters betrifft. Jedoch sind sämtliche Sprachelemente sowie Ein/Ausgabeanweisungen beschrieben und anhand von einsichtigen Beispielen erläutert.

Berlin im Juli 1979

Direktes Rechnen

Wie mit jeder normalen Rechenmaschine können auch mit dem Interpreter 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 Interpreter 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 auf dem "Ausgabegerät". Selbstverständlich können auch kompliziertere Rechnungen und auch mehrere Rechnungen, durch Doppelpunkt getrennt, auf einmal eingegeben werden.
Beispiel:

```
?COS(0)
 1
READY.

?SQR(1-.75)
 .5
READY.
```

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:

```
U=25.21:U=COS(SQR(1+U*U)):?U
 .995291012
READY.
```

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.

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 verden. "MCS-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 jeden Zeilennummer benötigt unabhängig von ihrer Größe 2 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

Zahlenverte - eingegebene oder berechnete - können auf Speicherplätzen, die mit einem Namen bezeichnet verden, 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 verden:

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

"MCS-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 6662 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.

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 derselbe 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 der Konsole 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.

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

```
10 INPUT"KOMMISSIONSNR. ";K
20 IF K>0 AND K<10 THEN 40
30 PRINT"FALSCHER EINGABEWERT !!!":GOTO10
40 REM USW. .....
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. 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:

```
10 DATA 100,100,1100,2.14
20 FORI=1TO4
30 READ X(I) : NEXT :RESTORE
READY.
```

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

5. 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

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

```
10 PH=7:WE=2:SO=2
20 IFPH>0 OR PH<10 AND WETTER=SOMMER THEN 40
30 PRINT"REGEN ODER FALSCHE EINGABE":GOTO70
40 REM WEITER
70 END
READY.
```

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<>0 AND B<>0 THEN ... Der Vorteil dieser bitweisen Operation ist, daß manche Operationen hierdurch sehr einfach durchgeführt werden können.

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:

```
10 INPUTA$:INPUTB$
20 IF A$<B$ THENPRINTA$+" IST KLEINER ALS "+B$
30 IF A$>B$ THENPRINTA$+" IST GROESSER ALS "+B$
40 IF A$=B$ THENPRINTA$+" IST GLEICH "+B$
50 GOTO10
READY.

RUN

? ROTKAEPPCHEN

? FRANKENSTEIN

ROTKAEPPCHEN IST GROESSER ALS FRANKENSTEIN
```

6. 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 "MCS-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 möglilch.

7. 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.

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.

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 ";U
 20 IF U<17 THEN 33
 30 PRINT"NICHT VORHANDEN !!!":GOTO 10
 33 REM WEITER IM PROGRAMM
READY.
```

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 DEF FNA(X)=-ATN(X/SQR(-X*X+1))+1.5708
 20 Y=-0.72:GOSUB500:G(1)=Z
 30 Y=.31:GOSUB500:G(2)=Z
 40 END
```

```
500 REM BERECHNUNG VON ARC COS
510 IF Y=1 THEN Z=0:GOTO540
520 IF Y=-1 THEN Z=3.1416:GOTO540
530 Z=FNA(Y)
540 RETURN
READY.
```

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:

```
10 I=0
20 I=I+1
30 PRINTI,SQR(I)
40 IFI<100THEN20
50 END
READY.
```

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:

```
10 FOR K=1 TO 100
20 PRINTK,SQR(K)
30 NEXT
40 END
READY.
```

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

```
10 PI=3.1416
20 FOR T=0 TO PI/2 STEP PI/20
30 X=5*COS(T)+4.9
40 Y=5*SIN(T)-7.3
50 PRINTX,Y
60 NEXT
70 END
READY.
```

Nach "NEXT" kann der Variablenname entfallen. Nur in Ausnahmefällen - mehrere verschachtelte Loops - ist es erforderlich, den Variablennamen nach diesem Befehl auszuführen. 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 Cosinus
- TAN Tangens
- ATN Arcus Tangens

  Winkel in Bogenmaß

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

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

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

  Winkelangaben in Grad werden ebenfalls mit einer kleinen Hilfsroutine auf das Bogenmaß umgerechnet.

- 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.

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 der Konsole 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.

- Dasselbe 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 vergeseen 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 überschnitten (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 Programms 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 Fehler 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. Es bieten sich 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.

Maschinensprache

Mittels PEEK und POKE ist der Inhalt jedes einzelnen Bytes des Arbeitsspeichers zugänglich. Mit dem Befehl PEEK wird der Inhalt der angegebenen Adresse 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 der Inhalt der Speicherzelle.

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

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 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 mit 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.	
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. List wird durch Drücken der BREAK Taste unterbrochen.
	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 bis einschließlich der zweiten angegebenen Zeilennummer gelistet.

LIST ist programmierbar. Nach Ausführung dieses Befehls 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 einschließlich 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.
LET	LETG=4	Bei früheren BASIC-Versionen war LET zur Zuordnung notwendig.Es kann wahlweise 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 Zahlenverte als auch um alphanumerische Texte handeln. Daten, die in einem DATA-Befehl stehen, können mittels mehrerer READ-Anbweisungen 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=0THEN200	Logischer Vergleich. Im Beispiel wird ein Sprung zur Zeile 200 durchgeführt, 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 derselben Zeile auf THEN folgen, wird jeweils der gesamte folgende Zeileninhalt bei "wahrem" Vergleich ausgeführt, bei "falschem" Vergleich weggelassen.
FOR..NEXT	FORI=1TO20 ...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 entfallen.
STEP	FORK=5TO2STEP -.5	Mittels STEP kann eine beliebige Schrittweite (positiv oder negativ) vorgegeben werden. Im Beispiel wird die Schleife nach einander mit den Werten 5,4.5,..., 2.5,2 durchlaufen.

	FORI=3TO6 FORJ=2TO8 ... NEXTJ,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 mehrfach-geschachtelt in eine Zeile geschrieben werden.
GOTO	GOTO245	Unbedingter Sprung zu der angegebenen Zeile. GOT 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-Befehls 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 Programms 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 letzen 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-Befehls 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$,8	Der PRINT-Befehl bewirkt die Darstellung der angegebenen Variablen auf der Konsole, und zwar in dem Format, das durch die Verwendung von Komma bzw. Semicolon vorgegeben ist (Näheres siehe Kapitel Programmierung - 7. Ausgabeformat). Statt PRINT kann "?" benutzt werden. Wird das Programm aufgelistet, sieht man, daß "?" durch PRINT ersetzt wurde.
	PRINT	PRINT ohne Variablenangabe bewirkt einen Zeilenvorschub.
DIM	DIMA (6,6) DIMC$(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.

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 umgewandet. Beginnt der String mit nichtnumerischen 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-Verknüpfungen jeweils gleichwertiger Bits ergibt.

Beispiel:

	binär	dezimal
Argument1	1011001	89
Argument2	1001101	77
AND	1001001	73
OR	1011101	93
Argument	1011001	89
NOT	1111111110100110	-90

Dieser bitweise Vergleich führt einerseits dazu, daß bei einer Entscheidung, die wahr sein soll, wenn sowohl A als auch B von 0 verschieden sind, nicht geschrieben werden darf: IFANDBTHEN...(2AND4=0!). Die Schreibweise muß sein: IFA<>0ANDB<>0THEN... Der Vorteil dieser Methode des bitweisen 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="ORB=4THEN......
N=NOTA
```

BASIC- Ein-/Ausgabeanweisungen für Floppy Disk

Allgemeines:

Das Basic Plattenbetriebssystem (BDOS) stellt eine leistungsfähige, transparente und leicht zu handhabende Schnittstelle von BASIC zur Floppy-Disk-Einheit DFDS1 her. Wichtige Funktionen dieser Schnittstelle sind:

- Laden, Retten, Verifizieren von Programmen
- Schreiben, Lesen von sequentiellen Datenfiles.
- Direktzugriff auf jeden beliebigen Record einer Diskette.
- Aufruf der Dateien durch Namen (bis zu 6 ASCII-Zeichen).
- Löschen, Umbenennen, Reservieren sowie Kopieren von Dateien.
- Bis zu 4 Dateien können sich gleichzeitig im direkten Zugriff befinden.

BDOS gliedert sich in 2 Teile:

1. Der Befehlsentschlüßler erkennt Basic Befehle wie LOAD/ SAVE, OPEN, CLOSE usw. sowie die dazugehörigen Parameter, prüft diese auf syntaktische Richtigkeit und generiert gegebenfalls den Befehlscode für

2. das 'Physical File Executive' (PFE). Das Übergabeprotokoll von Informationen zwischen Befehlsdekoder und PFE geschieht mit Hilfe eines Vektors, der die Übergabeparameter enthält, und eines Unterprogrammaufrufs des PFE. Nach Ausführen des Befehlescodes meldet sich das PFE mit Statusinformation und eventuellen Fehlermeldungen im Übergabevektor zurück. Diese Struktur gestattet es, das Plattenbetriebssystem von MONA aus auf unterer Ebene zu betreiben. Ferner hat der Benutzer hiermit die Möglichkeit zusätzlich Softwarepakete (Editor/Assembler, PASCAL usw.) auf einfache Weise an das PFE anzuschließen.

   Die Basic Ein/Ausgabe ist über Datenkanäle organisiert. Der Befehlsablauf lautet:

```
OPEN X, Y, Z, "NAME"                ; Eröffnen eines
                                      Datenkanales

PRINT #X,--
INPUT #X,--                         ; Lesen/ Schreiben
GET #X,--                             von Daten

CLOSE X                             ; Schließen des Datenkanales
```

   Eine Ausnahme hierzu bilden die Befehle LOAD/SAVE/VERIFY, die die obigen OPEN/CLOSE-Sequenzen automatisch generieren (siehe Befehlesliste).

   Mit dem Parameter "Z" im OPEN-Befehl werden BDOS-Befehle wie COPY, KILL, CREATE usw. verschlüsselt (siehe Befehlsliste).

INPUT/ OUTPUT ANWEISUNGEN FÜR FLOPPY DISK

OPENX,Y,Z,'DATEN'	Eröffnung eines logischen Files. X ist die File Nr. (Ø-255), Y die Gerätenummer (0,1= Drive 1-2=Drive 2). Z bestimmt, ob Eingabe oder Ausgabe (Ø=Lesen, 1=Schreiben,2,3,5,6 usw. werden später erläutert). Files müssen mit Namen versehen werden. Maximal sechs Zeichen darf der Name umfassen.
CLOSE X	Schließen eines Files. Eröffnete Files müssen stets wieder geschlossen werden!
PRINT#X,A PRINT#X,A% PRINT#X,A$	Die Variablen A (bzw. A% oder A$) werden im File X auf das Gerät ausgegeben, das im OPEN-Befehl dem File X zugeordnet wurde.
INPUT#X,A,A%,A$	Vom File X und dem im OPEN-Befehl zugeordneten Gerät werden Daten in A, A% und A$ eingelesen.
GET#X,A$	Ein einzelnes Zeichen wird vom File X in A$ eingelesen.

Spezielle OPEN/CLOSE Sequenzen

Bei Benutzung der beiden folgenden Befehle ist ein Direktzugriff (RANDOM ACCESS) auf jeden beliebigen Sektor eines Files möglich. Das bedeutet bei großen Datensätzen eine erheblich kürzere Zugriffszeit.

OPENX,Y,2,Z$:CLOSEX OPENX,Y,2,"Z":CLOSEX	Es kann ein beliebiger Record (Sektor) eines zuvor eröffneten Files zum Lesen positioniert werden.
OPENX,Y,3,Z$:CLOSEX OPENX,Y,3,"Z":CLOSEX	Es kann ein beliebiger Record eines zuvor eröffnete Files zum Schreiben positioniert werden.

Bemerkung:
X = logische Gerätenummer (Ø-255)0,1
Y = physikalische Gerätenummer (Y=Drive1, 2=Drive 2)
Z$= String, der die Nummer des gesuchten Records enthält (Ø=erster Record)
"Z" = Nummer des gesuchten Record (Ø=erster Record)

Beim Schreiben (RANDOM-WRITE) braucht nicht darauf geachtet zu werden, daß 128 - Byte lange Blöcke gesendet werden. Der Buffer wird beim Schließen des Files automatisch auf die Diskette übertragen.

Befehlsfolge:

1. OPENX,Y,0,"NAME"	File wird zum Lesen eröffnet
2. OPENZ,Y,2,A$	Positionierung auf den gesuchten Record
3. INPUT#Z....	LESEN aus dem gesuchten Record
4. CLOSEZ	Schließen des gesuchten Records
5. CLOSEX	Schließen des LESE-FILES
	Bemerkung: Punkt 3-4 kann beliebig oft wiederholt werden, oder sich mit Punkt 7-9 abwechseln.
6. OPENX,Y,0,"NAME"	Siehe Punkt 1
7. OPENZ,Y,3,A$	Positionierung auf den gesuchten Record
8. PRINT#Z....	Schreiben auf dem gesuchten Record
9. CLOSEZ	Siehe Punkt 4
10. CLOSEX	Siehe Punkt 5
	Punkt 7-9 kann beliebig oft wiederholt werden oder sich mit Punkt 2-4 abwechseln.
Allgemeiner Hinweis:	Das letzte Byte auf dem letzten Sektor darf nicht beschrieben werden.

Mit den folgenden beiden Befehlen können gelöschte Files unter einem neuen Namen wieder aktiviert bzw. existente Files umbenannt (RENAME) werden.

OPENX,Y,5,"ALT":CLOSEX	Aufrufen eines sich auf der Diskette befindlichen Files mit Namen "ALT" von Drive 1, wenn Y=0,1 oder von Drive 2, wenn Y =2.
OPENX,Y,6,"NEU":CLOSEX	Umbenennung des zuvor aufgerufenen Files.
	Bemerkung: Ist ein File öfter als 2mal auf einer Diskette unter gleichem Namen, so wird das erste gelöschte Files

COPY:
Befehlsfolge: OPEN L1, D1, Ø, "Quelle"
OPEN L2, D2, 7, "Zieldatei"
CLOSE L2: CLOSE L1

Bemerkung:
Die auf dem Gerät (Drive) #D1 befindliche Quelldatei wird auf die Zieldatei (Drive#D2) kopiert. D1=D2 ist möglich! Der Benutzer hat zu beachten, daß auf der Diskette, die die Zieldatei aufnehmen soll, genügend Platz vorhanden ist. Ggf. CATLOG-Programm starten!

KILL:
Befehlsfolge: OPENX,Y,8,"NAME"
CLOSEX

Das auf dem Gerät Y befindliche File "NAME" wird mit einer "Delete"-Kennung versehen!

CREATE:
Befehlsfolge: OPEN L1,D1,1 "NAME"
OPEN L2,D1,9,V$
CLOSE L2: CLOSE L1

V$:
String, der die länge (in Sektoren) des zu erzeugenden Files enthält.

Z. B.: V$ = "0123" (führende Nullen werden ignoriert!)

Bemerkung:
Es wird eine Datei "NAME" generiert und ein der Variablen V$ entsprechend großer Bereich auf der Diskette reserviert. Verschiedene Records (Sektoren) einer solchen Datei können dann mit dem RANDOM READ/WRITE-Befehlen angesprochen werden.

1. Initialisieren einer Diskette

Leere Diskette nach Drive #1
$EBØØ : (GO)

Bemerkung:
Die Diskette wird im IBM-Format (128 Byte pro Sektor) soft-sektoriert und kontrollgelesen. Anschließend wird die "File Directory" auf der Grundspur (Track ØØ) initialisiert.

2. Packen einer Diskette

Leere Diskette nach Drive #1
"Quell"-Diskette nach Drive #2
$EBØ2 = (GO)

Bemerkung:
Es wird eine Kopie aller "gültigen" (nicht gelöschten) Files der Diskette im Drive #2 auf die Diskette in Drive #1 geschrieben. Nach Beendigung des Befehls enthält die Diskette in Drive #1 eine "gepackte" Kopie der Quelldiskette. Letztere kann nun neu initialisiert und weiterverwendet werden.

Fehlermeldungen

FILE NOT FOUND	wenn ein aufgerufenes File nicht gefunden wird
#2 R/W ERROR	(Read/Write Error) Lese-/Schreibfehler
#4 NO DISK	Drive-Klappe nicht geschlossen oder Diskette nicht vorhanden
#8 WRITE PROTECT	Versuch, eine geschützte Diskette zu beschreiben (Minidisketten werden durch Entfernen des schwarzen Aufklebers geschützt)
#3	(Seek Error) Suchfehler, z. B. Aufruf eines Sektors, der außerhalb der Disk-Adressen liegt
#8	(Disk full) Diskettenkapazität erschöpft
#9	(Directory full) bereits 150 Files gespeichert
#0	zu viele Files geöffnet
#5	unbekannter Fehlercode

- SYSTEM GENERIERUNG -

BASIC -

1. Urladen:

- Systemdiskette nach Drive # 1
- Adresse $E!2Ø anwählen (GO)
- BASIC Kaltstart: $E7EE (GO)
- BASIC Warmstart: $ C 389 (GO)

2. Speicheranforderungen:

BASIC:	$CØØØ - $ E7FF
Drucker + Disk I/O:	$ E8ØØ - $ EFFF
Disk RAM Puffer:	$BØØØ - $B7FF
Disk Loader:	$ B8ØØ - $BFFF
Basic-Programm + Datenspeicher:	$400 - $AFFF (maximal)

Die obere Grenze des für BASIC frei verfügbaren Speichers wird nach einem Kaltstart automatisch ermittelt!

3. Änderung der unteren Grenze des Programm-Daten-Speichers:
(Geht nur in Inkrementen von einer Page!)

Z. B. Start = $1ØØØ
$E153 - $10
$E183 - $10 einzutragen vor Basic-Kaltstart!

- ANHANG -

Benutzen des Plattenbetriebssystems von MONA-Ebene

1. Übertragung eines Speicherbereichs auf Diskette

Startadresse (L,H) nach $F7FØ, $F7F1
Endadresse (L,H) nach $F7F2, $F7F3

a) $F1 nach $B48Ø

b) Ø oder 1nach $B481 ("0" = Drive #1;"1"; = Drive #2)

c) "NAME" nach $B482-$B487
(6 ASCII-Zeichen (siehe Tabelle) aufgefüllt mit Spaces ($20))

d) 1 nach $B488

Adresse $EF7Ø anwählen: (GO)

2. Laden eines Speichersegmentes von der Diskette:

Startadresse (L,H) nach $ØØ, $Ø1
Endadresse (L,H) nach $F7F2, $F7F3

a) $FØ nach $B480

b) 0 oder 1nach $B481 (= Drive $ 1,2)

c) "NAME" nach $B482-$B487

Adresse $EF42 anwählen: (GO)

- A N H A N G II -

Dienstprogramm CATLOG

Nach Einladen und Starten des mitgelieferten Programms 'CATLOG' erhalten Sie ein vollständiges Listing der Directory von der im Drive 1 oder Drive 2 befindlichen Diskette auf dem Bildschirm:

1. Anzahl der freien Sektoren auf der Diskette
   (# OF AVAILABLE RCDS)

2. Anzahl der noch frei zu benennenden Dateien
   (# OF AVAILABLE FCB-ENTRIES)

3. Für jedes auf der Diskette befindliche File ein Ausdruck in der folgenden Form:

Dateiname (NAME)	Dateityp (TYPE)	Länge (SIZE)	Start-Adresse (TRK#)	(SCTR#)	Kennwort (ATTR)

Dateiname:	1-6 ASCII-Zeichen
Dateityp:	P= Programmfile - D=Datenfile
Länge:	in Sektoren zu je 128 Byte
Startadresse:	bestehend aus Spur# und Sektor#
Kennwort:	gibt an, ob die Datei gelöscht (DELETED) ist

Auf Programmdisketten sollte als erstes das 'CATLOG'--Programm stehen, damit man sich jederzeit den Inhalt der Diskette ansehen kann.

```
 10 REM DEMONSTRATIONSPROGRAMM FUER:
 20 REM CREATE / RANDOM READ / RANDOM WRITE
 30 REM
 40 REM *** CREATE ***
 50 OPEN1,1,1,"DATEN"
 60 INPUT"ANZAHL DER ZU RESERVIERENDEN SEKTOREN (MAX. 624)";S$
 70 OPEN2,1,9,S$
 80 CLOSE2
 90 CLOSE1
 100 REM *** RANDOM-WRITE ***
 110 OPEN1,1,0,"DATEN"
 120 FOR I=0 TO VAL(S$)-1
 130 SE$=STR$(I)
 140 OPEN2,1,3,SE$
 150 PRINT#2,I
 160 CLOSE2
 170 NEXTI
 180 CLOSE1
 190 REM *** RANDOM-READ ***
 200 OPEN1,1,0,"DATEN"
 210 FOR I=0 TO VAL(S$)-1
 220 SE$=STR$(I)
 230 OPEN2,1,2,SE$
 290 INPUT#2,X
 300 IF X<>I THEN PRINT"LESEFEHLER IN SEKTOR ";I
 310 CLOSE2
 320 NEXTI
 330 CLOSE1
 340 END
READY.
```