RAD

Eingabeformat

RAD

Beschreibung

Das RAD Statement stellt den Winkelmodus auf das Bogenmaß, also die Einheit Radiant ein. Mit der Wahl des RAD Winkelmodus werden alle eingegebenen oder berechneten Winkel im Bogenmaß berechnet. Der RAD Winkelmodus wird auch dadurch eingestellt, daß NEW ALL eingegeben oder das System initialisiert wird.

Querverweis

DEG,GRAD

RANDOMIZE

Eingabeformat

RANDOMIZE *[numerischer Ausdruck]*

Beschreibung

Das RANDOMIZE Statement stellt den Zufallszahlengenerator auf eine Folge nicht voraussagbarer Zahlen ein.

Wenn nach RANDOMIZE ein *numerischer Ausdruck* folgt, wird jedesmal dieselbe Folge von Zufallszahlen erzeugt, wenn das Statement mit diesem Wert ausgeführt wird. Unterschiedliche Werte des *numerischen Ausdruckes* ergeben unterschiedliche Folgen.

Beispiel

Das nachstehende Programm zeigt eine Anwendung des RANDOMIZE Statements. Es nimmt einen Wert für den numerischen Ausdruck entgegen, und druckt die ersten 10 mittels der RANDOMIZE Funktion erzeugten Zufallszahlen. Drücken Sie **[BREAK]**, um das Programm zu unterbrechen.

```
100 INPUT "ANFANGSZAHL:";S
110 RANDOMIZE S
120 FOR A=1 TO 10:PRINT A;RND:PAUSE 1.1
130 NEXT A
140 GOTO 100
```

READ

Eingabeformat

READ *Variablen-Liste*

Beschreibung

Das READ Statement verwendet man zusammen mit dem DATA Statement, um den Variablen Werte zuzuordnen. Die *Variablen-Liste* besteht aus durch Kommata getrennten String- und numerischen Variablen, die indiziert sein dürfen. Der aus dem DATA Statement gelesene Wert, muß von gleicher Art wie die Variable sein, der er mittels READ zugeordnet wird. Achten Sie darauf, daß jede Zahl ein gültiger String-Ausdruck ist. Wenn zwei aufeinanderfolgende Kommata in der Datenliste auftreten, werden diese als Null-String gelesen.

Das READ Statement beginnt mit dem Lesen von Daten aus dem ersten DATA Statement des Programmes oder Unterprogrammes, das gerade verarbeitet wird und schreitet dann zum nächsten DATA Statement fort, wenn die Datenliste des momentan gelesenen DATA Statements zuende gelesen ist. Ein einzelnes READ Statement kann aus mehreren DATA Statements lesen und umgekehrt können mehrere READ Statements ein einzelnes DATA Statement lesen. Wenn ein READ Statement nicht den gesamten Inhalt des momentan verarbeiteten DATA Statements liest, so beginnt das nächste READ Statement, das erste noch nicht gelesene Element, aus der Liste zu lesen. Der Versuch, im momentan verarbeiteten Programm oder Unterprogramm mehr Daten als vorhanden sind zu lesen, erzeugt eine Fehlerbedingung.

Das RESTORE Statement benutzt man, um die Reihenfolge zu ändern, in der die DATA Statements gelesen werden.

Mit READ können nur Daten aus DATA Statements gelesen werden, die sich im gleichen Programm oder Unterprogramm, wie das READ Statement befinden. Jedesmal, wenn ein Unterprogramm aufgerufen wird, werden die Daten aus dem ersten dort enthaltenen DATA Statement gelesen.

Querverweis

DATA, RESTORE

REM

Eingabeformat

REM *[Zeichen-String]*

Beschreibung

Das REM Statement ermöglicht Ihnen, erklärende Bemerkungen in Ihr Programm einzufügen. Die Bemerkungen können Informationen beliebiger Art sein, enthalten aber gewöhnlich Informationen zu Teilen des Programmes. Der *Zeichen-String* darf jedes anzeigbare Zeichen enthalten.

Die Bemerkungen werden nicht wie Programminhalte ausgeführt, sie belegen aber Speicherplatz. Jedes Zeichen das nach REM folgt, einschließlich des Statement-Trennzeichens (:) wird als Teil der Bemerkung interpretiert. Daher muß REM das letzte Statement in einer Zeile sein, wenn diese Zeile mehr als ein Statement enthält.

Das Ausrufungszeichen(!) nennt man Schlußbemerkungssymbol und kann es anstatt des Wortes REM verwenden. Das Ausrufungszeichen kann als erstes Statement in einer Zeile erscheinen, oder als letztes Statement in einer Zeile mit mehreren Statements. Wenn das Ausrufungszeichen nach einem anderen Statement erscheint, so ist das Statement-Trennzeichen (:) nicht erforderlich. Die Verwendung des Schlußbemerkungssymbols erspart Platz in der aufgelisteten Form des Programms.

Beispiel

```
150 REM ANFANG DES TEILPROGRAMMES
```

Bezeichnet einen Programmabschnitt als Anfang eines Teilprogrammes.

```
270 ZWISCHENSUMME=L+B! BERECHNUNG DER ZWISCHENSUMME
```

Bezeichnet Statements, die eine bestimmte Berechnung ausführen.

RENUMBER

Eingabeformat

RENUMBER *[Anfangszeile] [,Inkrement]*

Beschreibung

Der RENUMBER (oder REN) Befehl ändert die Zeilennummern eines Programmes. Wenn keine Anfangszeile angegeben ist, beginnt die neue Numerierung der Zeilen mit Zeilennummer 100. Wenn kein Inkrement angegeben ist, werden die neuen Zeilennummern automatisch mit einem Abstand von 10 Zeilen gebildet.

REN ändert auch alle im Programm enthaltenen Zeilennummern-Angaben in GOTO, GOSUB etc., so daß sich diese wieder auf dieselben Programm-Segmente beziehen wie zuvor. Wenn sich ein Statement auf eine Zeilennummer bezieht, die nicht existiert, wird eine Warnung angezeigt und diese Zeilennummer durch 32767 ersetzt, was jedoch keine gültige Zeilennummer darstellt.

Wenn für die Anfangszeile oder das Inkrement Werte eingegeben werden, die zu Zeilennummern führen, die größer als 32766 sind, wird die Fehlernachricht `E11 Line number error` angezeigt und das Programm bleibt unverändert.

Beispiel

```
REN
```

Alle Zeilennummern werden geändert, beginnend mit Zeile 100 und mit einem Zeilenabstand von 10.

RESTORE

Eingabeformat

RESTORE { *[Zeilennummer]* | *[# Dateinummer [,REC numerischer Ausdruck]*] }

Beschreibung

Das RESTORE Statement benutzt man zur Steuerung der Reihenfolge, mit der Daten aus DATA Statements oder aus einer Datei gelesen werden.

RESTORE bestimmt, daß das als nächstes ausgeführte READ Statement Zugriff zur ersten Dateneinheit in dem DATA Statement enthält, das durch die *Zeilennummer* vorgegeben wurde. Die vorgegebene *Zeilennummer* muß zum gleichen Programm oder Unterprogramm gehören, wie das RESTORE Statement. Wenn die Angabe einer *Zeilennummer* nicht erfolgte, wird das DATA Statement mit der niedrigsten Zeilennummer im momentan ablaufenden Programm oder Unterprogramm gelesen. Wenn bei der angegebenen *Zeilennummer* kein DATA Statement gefunden wird, so wird das nächstfolgende DATA Statement gelesen.

RESTORE # *Dateinummer* bringt eine Datei wieder an ihren Anfang. Das nächste Eingabe/Ausgabe Statement, das sich auf die Datei mit dieser Nummer bezieht, erlangt Zugriff zum ersten Datensatz in dieser Datei. Existiert eine schwebende PRINT-Bedingung, so werden die Daten in die Datei übertragen, bevor das RESTORE Statement ausgeführt wird. Schwebende Eingabe-Daten werden ignoriert. *Dateinummer* 0 bedeutet ein DATA Statement entsprechend der obigen Beschreibung.

REC kann mit Geräten benutzt werden, welche die Verwendung von Dateien mit relativen Datensätzen (wahlfreier Zugriff) zulassen. Der numerische Ausdruck bestimmt, welcher Datensatz der Datei bei wahlfreiem Zugriff gelesen wird. Das nächste Eingabe/Ausgabe Statement, das sich auf diese Datei bezieht, erlangt Zugriff zu diesem Datensatz. Informationen über wahlfreien Zugriff finden Sie in den Peripherie-Gerätebedienungsanleitungen.

Beachten Sie: Der erste Datensatz einer Datei hat die Nummer Null.

RESTORE

Querverweis

DATA, INPUT, LINPUT, PRINT, READ

Beispiele

```
150 RESTORE
```

Das nächste DATA Statement das gelesen wird, ist das erste im diesem Programm.

```
200 RESTORE 130
```

Das nächste DATA Statement, das gelesen wird, befindet sich in Zeile 130. Wenn diese Zeile kein DATA Statement enthält, wird das nächstfolgende DATA Statement nach Zeile 130 gelesen.

```
230 RESTORE #1
```

Bringt die Datei Nr. 1 zum Anfangsdatensatz zurück, welcher der Datensatz mit der Nummer Null ist.

MIT GOSUB

RETURN

Eingabeformat

RETURN

Beschreibung

RETURN in Verbindung mit GOSUB überträgt die Steuerung des Programmablaufes zu dem Statement zurück, das dem GOSUB oder ON GOSUB Statement folgt, das zuletzt ausgeführt wurde. Ein Teilprogramm darf mehrere RETURN Statements enthalten.

Querverweis

GOSUB, ON GOSUB

RETURN MIT ON ERROR

Eingabeformat

RETURN { [NEXT] / *[Zeilennummer]* }

Beschreibung

RETURN schließt ein fehlerverarbeitendes Teilprogramm ab. Ein solches Teilprogramm zur Fehlerverarbeitung wird aufgerufen, nachdem ein Fehler auftritt, wenn zuvor ein ON ERROR *Zeilennummer* Statement ausgeführt worden war. Das Teilprogramm zur Fehlerverarbeitung darf beliebige BASIC Statements enthalten, einschließlich weiterer ON ERROR Statements.

RETURN ohne nachfolgende Angabe überträgt die Steuerung des Programmablaufes zu der Zeile, in welcher der Fehler auftrat. Das den Fehler hervorufende Statement wird nochmals ausgeführt.

RETURN NEXT überträgt den Programmablauf zu dem auf das fehlerhafte Statement folgende Statement.

RETURN *Zeilennummer* überträgt die Steuerung des Programmablaufes zu der angegebenen Zeile. Diese Zeile muß sich im selben Programm oder Unterprogramm wie das fehlerverarbeitende Teilprogramm befinden, auch wenn die Fehlerursache selbst in einem anderen Unterprogramm entstanden ist.

Querverweis

ON ERROR

MIT ON ERROR RETURN

Beispiel

Das nachstehende Programm verdeutlicht den Gebrauch von RETURN mit ON ERROR.

```
100 ON ERROR 130
```
Überträgt die bei Auftreten eines Fehlers den Programmablauf zur Zeile 130.

```
110 X=VAL(D)
```
Erzeugt einen Fehler.

```
120 PRINT "weiter im Programm":PAUSE:STOP
```
Dies ist die Stelle, bei der das Programm nach RETURN NEXT fortfährt.

```
130 REM Fehlerverarbeitung
140 A=A+1
150 IF A=4 THEN PRINT "Letzter Fehler": PAUSE 2:
    RETURN NEXT
```
Prüft ob der Fehler vier mal aufgetreten ist. Ist dies der Fall, so wird Letzter Fehler angezeigt und der Programmablauf mit dem auf das fehlerhafte Statement folgende Statement fortgesetzt.

```
160 PRINT A;".er Fehler":PAUSE 2
170 ON ERROR 130
```
Bestimmt den Programmablauf beim nächsten Fehler.

```
180 RETURN
```

RND

Eingabeformat

RND

Beschreibung

Die RND Funktion erzeugt die nächste pseudo-Zufallszahl aus der momentanen Folge von Pseudo-Zufallszahlen. Die erzeugte Zahl ist größer oder gleich Null und kleiner als Eins. Wenn nicht das RANDOMIZE Statement verwendet wird, um eine nicht bestimmbare Zufallszahl zu erzeugen, erzeugt das RND Statement jedesmal die gleiche Zahlenfolge, wenn das Programm abläuft.

Querverweis

RANDOMIZE

Beispiel

```
100 PRINT 10 * RND:PAUSE
```

Zeigt eine Zufallszahl zwischen 0 und 10 an.

RPT$

Eingabeformat

RPT$*(String-Ausdruck, numerischer Ausdruck)*

Beschreibung

Die RPT$ Funktion ergibt einen String, der den ursprünglichen *String-Ausdruck* so oft wiederholt enthält, wie es der *numerische Ausdruck* angibt. Wenn man mit RPT$ einen String erzeugt, der länger als 255 Zeichen ist, so werden die überzähligen Zeichen gelöscht und es wird der Warnhinweis `W28 Truncation` angezeigt.

Beispiele

```
100 M$=RPT$("ABCD",4)
```

setzt M$ gleich "ABCDABCDABCDABCD".

```
100 PRINT USING RPT$("#",40); X$:PAUSE
```

Druckt den Wert von X$ und verwendet dabei ein Format, das 40 Ziffernzeichen enthält.

RUN

Eingabeformat

RUN { *[Zeilennummer]* / *["Programm-Name"]* / *["Gerät.Dateiname"]* }

Beschreibung

Das RUN Statement bewirkt, daß der Ablauf des gespeicherten Programmes beginnt. Die Eingabe des RUN Statements ohne nachfolgende Angaben läßt die Ausführung des momentan im Speicher enthaltenen Programmes, von der Zeile mit der niedrigsten *Zeilennummer* an, beginnen.

RUN *Zeilennummer* läßt den Ablauf des Programmes im Speicher ab der angegebenen Zeilennummer beginnen.

RUN *"Programm-Name"* sucht ein Programm mit diesem Namen in einem Solid *State Software*™ Modul und beginnt mit der Ausführung des Programmes, sobald es gefunden wurde. Wenn ein Programm dieses Namens nicht gefunden wird oder sich der Name auf ein Unterprogramm bezieht, tritt ein Fehler auf. Zur Angabe des *Programm-Namens* kann ein String-Ausdruck verwendet werden.

RUN *"Gerät.Dateiname"* löscht das momentan im Speicher befindliche Programm, liest die Inhalte der Datei des angegebenen *Dateinamens* von dem angegebenen Gerät und führt dieses Programm aus. Zur Angabe von Gerät und Dateiname kann ein String-Ausdruck verwendet werden. **Beachten Sie:** Wenn der *Dateiname* eine Datei bezeichnet, die Daten enthält, anstatt einer Programm-Datei, kann es erforderlich sein, daß Sie die Resettaste drücken müssen.

Bevor ein Programm ausgeführt wird, finden folgende Vorbereitungen statt.

- Variablen werden initialisiert. Numerische Variablen werden gleich Null gesetzt und String-Variablen werden einem Null-String gleichgesetzt.
- Einige Fehlerbedingungen werden überprüft, wie etwa, wenn auf ein FOR Statement kein NEXT Statement folgt oder wenn eine Zeilenangabe außerhalb des zulässigen Bereiches liegt.
- Alle offenen Dateien werden abgeschlossen.
- ON BREAK STOP, ON WARNING PRINT und ON ERROR STOP werden aktiviert.
- Der eingestellte Winkelmodus bleibt unverändert.

RUN

Beispiele

`RUN`
Läßt den Computer mit der Ausführung des im Speicher enthaltenen Programmes, von der niedrigsten Zeilennummer an, beginnen.

`RUN 200`
Läßt den Computer mit der Ausführung des im Speicher enthaltenen Programmes ab Zeile 200 beginnen.

`RUN "1.PRG3"`
Läßt den Computer das Programm aus der Datei PRG3 von Gerät Nr.1 eingelesen und mit dessen Ausführung beginnen.

`RUN "HIST"`
Führt das Programm HIST aus dem *Solid State Software*™ Modul aus.

Das nachstehende Programm verdeutlicht die Anwendung des RUN Statement, um damit innerhalb eines Programmes andere Programme ausführen zu lassen. Eine Wahlliste entsteht, um dem Anwender des Programmes die Möglichkeit zu geben, daß er ein anderes Programm zur Durchführung wählt. Die gewählten Programme sollten dann statt des üblichen Programmabschlusses das Wahllisten-Programm starten, um so nach dem Ablauf eines jeden dieser Programme die Wahlliste erneut anzuzeigen.

```
100 PRINT "Zur Programmwahl 1,2 oder 3 eingeben":PAUSE 2
110 PRINT "Beenden mit 4":PAUSE 2
120 INPUT "IHRE WAHL:";C
130 IF C=1 THEN RUN "1.PRG1"
140 IF C=2 THEN RUN "1.PRG2"
150 IF C=3 THEN RUN "1.PRG3"
160 IF C=4 THEN STOP
170 GOTO 100
```

SAVE

Der SAVE Befehl speichert ein BASIC Programm auf ein externes Speichergerät.

Eingabeformat

SAVE *"Gerät.Dateiname"* [,*PROTECTED*]

Beschreibung

Mit dem SAVE Befehl schicken Sie eine Kopie des BASIC Programms in den Speicher eines externen Speichergerätes. Mit dem OLD Befehl können Sie später jederzeit das Programm wieder in den Speicher des Computers einlesen.

Vor Aufzeichnung des Programms löscht SAVE alle Variablen aus dem System, die vom Programm nicht benutzt werden.

Gerät.Dateiname identifiziert das Gerät, in dem das Programm gespeichert werden soll und gibt den Dateinamen an. **Gerät** bezeichnet die Nummer, die dem physischen Gerät zugeordnet ist; sie kann zwischen 1 und 255 liegen. **Dateiname** bezeichnet die Datei, in der das Programm enthalten ist.

Wählt man die Angabe PROTECTED, bleibt zwar das Programm im Speicher ungeschützt, jedoch ist die in das externe Gerät überspielte Kopie geschützt. Ein geschütztes Programm kann weder aufgelistet noch editiert oder weiter kopiert werden.

Anmerkung: Mit dem SAVE Befehl können Sie keine Informationen in ein Modul abspeichern.

SAVE

Beispiele

SAVE "1.PRG1"

Speichert das im Speicher des Computers enthaltene Programm im Gerät 1 mit Namen PRG1 ab.

SAVE "1.PRG2",PROTECTED

Speichert das im Speicher des Computers enthaltene Programm im Gerät 1 mit Namen PRG2 ab. Man kann zwar das Programm in den Speicher einlesen und ablaufen lassen, es kann aber nicht editiert, aufgelistet oder erneut gespeichert werden.

Querverweis

GET, OLD, PRINT (mit Dateien), PUT, VERIFY

SEG$

Eingabeformat

SEG$*(String-Ausdruck, Position, Länge)*

Beschreibung

Die SEG$ Funktion gibt einen Teil-String aus einem String wieder. Der wiedergegebene Teil-String beginnt ab der angegebenen Position im *String-Ausdruck* und erstreckt sich über die angegebene *Länge* (= Anzahl) von Zeichen. Falls eine *Position* angegeben wird, die über das Ende des ursprünglichen *String-Ausdruckes* hinausgeht, wird ein Null-String ("") wiedergegeben. Wenn die angegebene *Länge* über das Ende des *String-Ausdruckes* hinausreicht, werden nur die Zeichen bis zum Ende wiedergegeben.

Beispiele

```
100 X$=SEG$("VORNAME NACHNAME",1,7)
```
Setzt X$ gleich "VORNAME".

```
200 Y$=SEG$("VORNAME NACHNAME",9,8)
```
Setzt Y$ gleich "NACHNAME".

```
240 Z$=SEG$("VORNAME NACHNAME",8,1)
```
Setzt Z$ gleich " ", d.h. einem Leerzeichen

```
280 PRINT SEG$(A$,B,C):PAUSE
```
Druckt den Teil-String aus A$ beginnend ab Bter Position, C Zeichen lang.

SGN

Eingabeformat

SGN*(numerischer Ausdruck)*

Beschreibung

Die SGN Funktion stellt die mathematische Signumfunktion dar. Für positive Werte des *numerischen Ausdruckes* lautet das Ergebnis +1. Ist der *numerische Ausdruck* Null, so ist auch der Wert der SGN Funktion 0 und bei negativem Wert des numerischen Ausdruckes ist der Wert für die SGN Funktion −1.

Beispiele

```
140 IF SGN(A)=1 THEN 300 ELSE 400
```
Überträgt die Steuerung des Programmablaufes zu Zeile 300, wenn A positiv ist und zu Zeile 400, wenn A gleich Null oder negativ ist.

```
790 ON SGN(X)+2 GOTO 200,300,400
```
Überträgt die Steuerung des Programmablaufes zu Zeile 200, wenn X negativ ist, zu Zeile 300, wenn X gleich Null ist und zu Zeile 400, wenn X positiv ist.

SIN

Eingabeformat

SIN*(numerischer Ausdruck)*

Beschreibung

Die SIN Funktion berechnet den SINUS des *numerischen Ausdrucks.*

Das Ergebnis wird in dem zuletzt vor Benutzen dieser Funktion eingestellten Winkelmodus (RAD, DEG oder GRAD) berechnet. Schlagen Sie in Anhang B nach, um sich über Grenzwerte für den *numerischen Ausdruck* zu informieren.

Beispiele

```
150 DEG
160 PRINT SIN(3 * 21,5+4):PAUSE
```

Zeigt .930417568. an

SINH

Die SINH Funktion berechnet den hyperbolischen Sinus eines Ausdrucks.

Format

SINH (*numerischer Ausdruck*)

Beschreibung

Die Funktion SINH (hyperbolischer Sinus) berechnet den hyperbolischen Sinus eines **numerischen Ausdrucks**. Die Definition des hyperbolischen Sinus lautet:

$$SINH(X) = .5^{*}(e^{x} - e^{-x})$$

Beispiele

100 PRINT SINH(0):PAUSE

Druckt 0 aus.

230 T=SINH(0.75)

Setzt T gleich .8223167319.

Querverweis

ACOSH, ASINH, ATANH, COSH, TANH

SQR

Eingabeformat

SQR*(numerischer Ausdruck)*

Beschreibung

Die SQR Funktion ermittelt die positive Quadratwurzel des *numerischen Ausdruckes*. SQR(X) ist äquivalent mit dem Ausdruck X ^ (1/2). Der numerische Ausdruck darf nicht negativ sein.

Beispiele

```
150 PRINT SQR(4):PAUSE
```
Zeigt 2 an.

```
780 X=SQR(2.57E5)
```
Setzt X der Quadratwurzel aus 257000 gleich, wofür sich 506.9516742 ergibt.

STOP

Eingabeformat

STOP

Beschreibung

Das STOP Statement hält den Programmablauf an. Es kann wahlweise anstatt des END Statements verwendet werden, mit der Ausnahme, daß STOP nicht am Ende eines Unterprogrammes stehen darf.

Querverweis

END

Beispiel

Das nachstehende Programm verdeutlicht den Gebrauch des STOP Statements. Das Programm addiert die Zahlen von 1 - 100.

```
100 TOT=0
110 NUMB=1
120 TOT=TOT+NUMB
130 NUMB=NUMB+1
140 IF NUMB > 100 THEN PRINT TOT:PAUSE 2:STOP
150 GOTO 120
```

Wenn die Bedingung in Zeile 140 erfüllt ist hält das Programm an, obwohl es aus einer unendlichen Schleife besteht.

STR$

Eingabeformat

STR$*(numerischer Ausdruck)*

Beschreibung

Die STR$ Funktion gibt die String-Darstellung eines Zahlenwertes oder eines *numerischen Ausdrucks* wieder. Führende und abschließende Leerstellen werden dabei nicht mit einbezogen. Die STR$ Funktion ist die Umkehrung der VAL Funktion.

Querverweis

LEN, VAL

Beispiele

```
150 NUM$=STR$(78.6)
```
Setzt den NUM$ gleich "78.6".

```
220 LL$=STR$(3E15)
```
Setzt LL$ gleich "3E+15".

```
330 J$=STR$(A * 4)
```
Setzt J$ einem String gleich, der dem Wert entspricht, der sich ergibt, wenn man A mit 4 multipliziert. Wenn A zum Beispiel -8 ist, so wird der J$ gleich "-32" gesetzt.

SUB

Eingabeformat

SUB *Unterprogramm-Name* [*(Parameter-Liste)*]

Beschreibung

Das SUB Statement ist das erste Statement in einem Unterprogramm, und muß auch das erste Statement in der Zeile sein. Ein Unterprogramm ist eine vom Hauptprogramm getrennte Gruppe von Statements. Ein Unterprogramm dient dazu, ein und dieselbe Aufgabe an verschiedenen Stellen im Programm zu erfüllen, ohne die betreffenden Statements an diesen Stellen zu wiederholen.

Unterprogramme sind mittels CALL *Unterprogramm-Name (Argumenten-Liste)* zugänglich. Der Unterprogramm-Name darf zwischen 1 und 15 Zeichen enthalten. Das erste Zeichen muß ein alphabetisches Zeichen oder ein Unterstreichungszeichen sein. Die restlichen Zeichen können alphanumerische oder Unterstreichungszeichen sein. Das CALL Statement sucht nach den Unterprogrammen in einer besonderen Reihenfolge (siehe CALL bezüglich der Reihenfolge) und führt das erste aufgefundene Unterprogramm mit dem angegebenen Unterprogramm-Namen aus. Wenn der Name eines Ihrer Unterprogramme mit dem eines eingebauten Unterprogrammes übereinstimmt, so wird das eingebaute Unterprogramm ausgeführt.

Die *Parameter-Liste* bestimmt die Informationen, welche an das Unterprogramm übergeben werden. Ein Parameter kann eine einfache String-Variable, eine einfache numerische Variable oder ein Datenfeld sein. Ein Datenfeld wird in der Parameter-Liste erfaßt, indem man den Namen des Datenfeldes gefolgt von Klammern angibt. Ein eindimensionales Datenfeld gibt man an mit A( ), ein zweidimensionales Datenfeld mit A(,), und ein dreidimensionales Datenfeld mit A(,,), wobei A der Datenfeldname ist.

Informationen werden an das Unterprogramm durch die *Argumenten-Liste* des CALL Statements übergeben. Die in der *Argumenten-Liste* enthaltenen Argumente und die in der Parameter-Liste enthaltenen Parameter müssen nicht die gleichen Bezeichnungen haben. Jedoch müssen Zahl und Art der Argumente in der Argumenten Liste mit der Zahl der Parameter in der Parameter-Liste des SUB Statements übereinstimmen.

Informationen können an ein Unterprogramm über Referenzen, d.h. abhängig übergeben werden oder als Wert, d.h. nicht abhängig. Wenn ein Argument als Referenz übergeben wird, übernimmt das Unterprogramm die Variable aus dem aufrufenden Programm. Wenn der entsprechende Paramenter im Unterprogramm geändert wird, so wird auch das Argument im aufrufenden Programm geändert. Einfache Variablen, Elemente von Datenfeldern, oder ganze Datenfelder werden normalerweise abhängig übergeben. Datenfelder können nur abhängig übergeben werden.

Wenn ein Argument als Wert übergeben wird, so wird der Wert in der entsprechenden Unterprogramm-Variablen übernommen. Wird diese Variable im Unterprogramm geändert, so ändert sich damit nicht der Wert des entsprechenden Argumentes im aufrufenden Programm. Mathematische und andere Ausdrücke werden ausgewertet und als Wert an das Unterprogramm übergeben. Einfache Variablen können als Wert übergeben werden, indem man sie in Klammern setzt.

Alle Variablen, die in einem Unterprogramm benutzt werden und nicht in der *Parameter-Liste* erfaßt wurden, sind unabhängig, d.h. ausschließlich diesem Unterprogramm zugeordnet. Damit kann derselbe Variablenname im Hauptprogramm und in anderen Unterprogrammen unabhängig voneinander verwendet werden (lokale Variablen). Die Änderung von Werten unabhängiger Variablen in einem Programm oder Unterprogramm beeinflußt also nicht die gleichnamigen unabhängigen Variablen in anderen Unterprogrammen.

Alle unabhängigen Variablen im Unterprogramm werden jedesmal initialisiert, d.h. auf Null gesetzt, wenn das Unterprogramm aufgerufen wird.

Ein Unterprogramm wird durch SUBEND abgeschlossen. Man kann auch mit SUBEXIT aus den Unterprogramm zurückkehren. Die Steuerung des Programmablaufes wird dann an das dem CALL Statement folgende Statement zurück übertragen.

Unterprogramme stehen ausschließlich getrennt nach dem Hauptprogramm. Ein Unterprogramm kann kein anderes Unterprogramm enthalten. Wenn ein SUB Statement in einem Hauptprogramm auftritt, wird das Hauptprogramm abgeschlossen, als ob ein STOP Statement ausgeführt worden wäre. Zwischen dem SUBEND Statement eines Unterprogrammes dürfen nur Bemerkungen und END Statements erscheinen.

Die Optionen, welche für die ON BREAK, ON WARNING, ON ERROR und PAUSE ALL Statements gewählt sind, wenn ein Aufruf eines Unterprogrammes ausgeführt wird, bleiben während des Ablaufes des Unterprogrammes wirksam. Wenn das Unterprogramm eines dieser Statements ändert, so werden diese Änderungen rückgängig gemacht, wenn das Unterprogramm abgeschlossen wird. Mit Ausnahme von Fehlerverarbeitungsprogrammen, dürfen Unterprogramme nicht auf Teilprogramme des Hauptprogramms zurückgreifen.

Querverweis

CALL, ON BREAK, ON ERROR, ON WARNING, RETURN, SUBEND, SUBEXIT.

Beispiele

Das folgende Programm versucht die sehr vielseitige Ausnutzung von Unterprogrammen darzustellen. Jede Variable hat ihren eigenen Zweck:

```
100 REM Hauptprogramm
110 DIM ZWEIDIMFELD(4,12)
120 A=10:B=20:C=30:D=40
130 ZWEIDIMFELD(2,4)=PI
140 TEXT$="ORIGINAL "
200 CALL UNTERPROGRAMM(A,(B),2.22,D*100,TEXT$,
    "WERT",ZWEIDIMFELD(,),R)
220 DISPLAY A;B;C;D;TEXT$;AUCHTEXT$;
    ZWEIDIMFELD(2,4);R:PAUSE
1000 REM Anfang des Unterprogramms
1010 SUB UNTERPROGRAMM(A,B,C,D,TEXT$,AUCHTEXT$,
    FELD(,),RUECKGABE)
1020 DISPLAY A;B;C;D;TEXT$;AUCHTEXT$;FELD(2,4);
    RUECKGABE:PAUSE
1030 A=A/2:B=B+1:C=C*2:D=D+1000
1070 TEXT$="Aenderung "
1080 FELD(2,4)=888:RUECKGABE=1234
1090 DISPLAY A;B;C;D;TEXT$;AUCHTEXT$;FELD(2,4);
    RUECKGABE:PAUSE
1100 SUBEND
2000 REM nachfolgend sind keine Statements des
    HAUPTPROGRAMMS erlaubt
```

Anmerkungen zum Programm:

Die Zeilen 110 – 140 definieren Variablen. Zeile 200 ruft das Unterprogramm namens UNTERPROGRAMM auf. Das CALL Statement übergibt eine Reihe von Daten an das Unterprogramm.

Zeile 1010 ist der Anfang des Unterprogramms und damit das Ende des Hauptprogramms. Das Unterprogramm übernimmt der Reihe nach die spezifizierten Parameter des Hauptprogramms. Dabei muß der Datentyp jeweils exakt übereinstimmen.

Die einzelnen Parameter:

Variable A wird in das Unterprogramm übergeben, dort geändert und bleibt mit dem neuen Wert dem Hauptprogramm zur Verfügung. Variable B wird in das Unterprogramm übergeben, dort geändert, aber nicht in das Hauptprogramm zurückgegeben, da sie in Klammern steht.

Variable C wird vom Hauptprogramm nicht übergeben. Jedoch liest das Unterprogramm den Wert 2,22 in die lokale Unterprogramm-Variable C ein. C wird dann verändert, aber nicht in das Hauptprogramm zurückgegeben. Die beiden gleichnamigen Variablen beeinflussen sich nicht.

Variable D wird nach der Multiplikation im CALL Statement zum Wert und danach wie Variable C behandelt.

Die String-Variable TEXT$ wird ähnlich der numerischen Variablen A im Hauptprogramm definiert, übergeben, im Unterprogramm geändert und steht dann dem Hauptprogramm mit neuem Text zur Verfügung. Der String "WERT" wird ähnlich dem numerischen Wert 2.22 in das Unterprogramm übergeben und steht dort lokal zur Verfügung.

Das gesamte Datenfeld ZWEIDIMFELD wird an das Unterprogramm übergeben, kann dort geändert werden und steht dann geändert im Hauptprogramm zur Verfügung. Die Namen dürfen im Haupt- und Unterprogramm verschieden sein.

Die Variable R ist im Hauptprogramm nicht definiert, wird also als Null übergeben, im Unterprogramm geändert und steht dann im Hauptprogramm zur Verfügung.

Wie Sie sehen, ist nicht der Variablenname zwischen Haupt- und Unterprogramm maßgebend, sondern die Übergabestelle in CALL bzw. SUB, also der Datentyp und bei Datenfeldern auch die Dimension. Die obigen Gegebenheiten können sehr leicht in der Anzeige verfolgt werden. Die DISPLAY Statements zeigen folgende Werte:

Zeile 1020 (Daten bei Übernahme in das Unterprogramm)

```
 10 20 2.22 4000 ORIGINAL WERT 3.141592654 0
```

Zeile 1090 (Daten nach Änderungen im Unterprogramm)

```
 5 21 4.44 5000 Aenderung WERT 888 1234
```

Zeile 220 (Daten nach Rückkehr in das Hauptprogramm)

```
 5 20 30 40 Aenderung 888 1234
```

SUBEND

Eingabeformat

SUBEND

Beschreibung

Das SUBEND Statement markiert das Ende eines Unterprogrammes. Wenn das SUBEND Statement ausgeführt wurde, wird die Steuerung des Programmablaufes zu dem Statement übertragen, das nach dem Statement folgt, welches das Unterprogramm aufgerufen hat. Das SUBEND Statement muß immer das letzte Statement in einem Unterprogramm sein und darf nicht in einem IF THEN ELSE Statement enthalten sein. Zwischen dem SUBEND Statement eines Unterprogrammes und dem SUB Statement des nächsten Unterprogrammes dürfen nur Bemerkungen und END Statements erscheinen.

Querverweis

SUB, SUBEXIT

SUBEXIT

Eingabeformat

SUBEXIT

Beschreibung

Das SUBEXIT Statement schließt die Ausführung eines Unterprogrammes ab. Nach Ausführung von SUBEXIT wird die Steuerung des Programmablaufes zu dem Statement übertragen, das nach dem Statement folgt, welches das Unterprogramm aufgerufen hat.
Das SUBEXIT Statement darf beliebig oft in einem Unterprogramm erscheinen.

Querverweis

SUB, SUBEND

TAB

Eingabeformat

TAB*(numerischer Ausdruck)*

Beschreibung

Die TAB Funktion wird in PRINT oder DISPLAY Statements verwendet, um für Druckelemente eine besondere Spaltenposition anzugeben. Für *numerische Ausdrücke,* die kleiner oder gleich Null sind, wird die Position immer gleich 1 gesetzt. Wenn der *numerische Ausdruck* größer als die beim jeweiligen Peripherie-Gerät zulässige Datensatzlänge ist, wird der *numerische Ausdruck* so oft um die Datensatzlänge vermindert, bis das Ergebnis kleiner als die Datensatzlänge ist.

Wenn die momentane Position kleiner als die angegebene, oder ihr gleich ist, schreitet die TAB Funktion zur angegebenen Position fort. Wenn die momentane Position größer ist, als die angegebene Position, geht die TAB Funktion zum nächsten Datensatz über, und schreitet dort bis zur angegebenen Position fort.

Die TAB Funktion wird wie ein *Druckelement* behandelt und muß von anderen *Druckelementen* durch einen Druckseparator getrennt werden. Der Druckseparator der vor TAB steht, wird vor der TAB Funktion ausgewertet und der Druckseparator der nach TAB folgt, wird nach der TAB Funktion ausgewertet. Normalerweise werden Strichpunkte vor und nach TAB gesetzt.

In einem DISPLAY Statement wird die TAB Funktion als relativ zum Anfang des Anzeigefeldes interpretiert. Wenn AT verwendet wird, ist die TAB Funktion relativ zur angegebenen Spaltenposition. Wenn mehr als eine Zeile angezeigt wird, beginnen die folgenden Zeilen in Spalte 1. Jede TAB Funktion ist dann relativ zu Spalte 1.

Bei Verwendung von SIZE, ist der spezifizierte Wert die absolute Grenze für die angezeigte Zahl von Zeichen. Diese Grenze ist die Datensatzlänge, die für die Auswertung von TAB-Funktionen herangezogen wird.

TAB

Querverweis

DISPLAY, PRINT

Beispiele

```
100 PRINT TAB(12);35:PAUSE
```

Druckt die Zahl 35 beginnend in Spalte 12. Da Spalte 12 vom Vorzeichen belegt ist, beginnt die eigentliche Zahl ab Spalte 13.

```
190 PRINT 356;TAB(18);"NAME":PAUSE
```

Druckt 356 am Anfang der Zeile und das Wort NAME beginnend in Spalte 18.

```
710 DISPLAY AT(10)SIZE(20),"MGB";TAB(10);"ADR.":
    PAUSE
```

Druckt MGB beginnend in Spalte 10 und ADR. beginnend in Spalte 19. Die Spalte 10 zählt als die erste Spalte für TAB.

TAN

Eingabeformat

TAN*(numerischer Ausdruck)*

Beschreibung

Die TAN Funktion berechnet den Tangens des *numerischen Ausdrucks*. Das Ergebnis wird in dem zuletzt vor Benutzen dieser Funktion eingestellten Winkelmodus (RAD, DEG oder GRAD) berechnet. Schlagen Sie in Anhang B nach, um sich über Grenzwerte für den *numerischen Ausdruck* zu informieren.

Beispiel

```
250 RAD
260 PRINT TAN(20):PAUSE
```

Druckt 2.237160944

TANH

Die TANH Funktion berechnet den hyperbolischen Tangens eines Ausdrucks.

Eingabeformat

TANH (*numerischer Ausdruck*)

Beschreibung

Die TANH Funktion gibt den hyperbolischen Tangens eines **numerischen Ausdrucks** wieder. Die Definition des hyperbolischen Tangens lautet:

$$\text{TANH}(X) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Beispiele

100 PRINT TANH(0):PAUSE

Druckt 0 aus.

230 T=TANH(0.75)

Setzt T gleich .6351489524.

Querverweis

ACOSH, ASINH, ATANH, COSH, SINH

UNBREAK

Eingabeformat

UNBREAK [*Zeilen-Liste*]

Beschreibung

Das UNBREAK Statement entfernt alle Stoppstellen aus dem Programm. Wenn eine *Zeilen-Liste* angegeben wurde, dann werden nur die Stoppstellen bei den angegebenen Zeilen entfernt.

Querverweis

BREAK

Beispiele

```
UNBREAK
```
Entfernt alle Stoppstellen.

```
400 UNBREAK 100, 130
```
Entfernt die Stoppstellen, die vor den Zeilen 100 und 130 gesetzt sind.

USING

Eingabeformat

USING { *Zeilennummer* / *String-Ausdruck* }

Beschreibung

USING kann in PRINT oder DISPLAY Statements auftreten, um die Ausgabe zu formatieren. Wenn eine *Zeilennummer* angegeben wurde, so wird das Format durch ein in dieser Zeile enthaltenes IMAGE Statement spezifiziert. Die Zeilennummer muß sich auf eine Zeile im momentan verarbeiteten Programm oder Unterprogramm beziehen. Siehe hierzu IMAGE. Wenn ein *String-Ausdruck* angegeben ist, wird das Format mittels USING definiert.

Durch die Anwendung von USING ergeben sich folgende Änderungen für die Auswertung der *Druck-Listen* in PRINT oder DISPLAY Statements.

- Der Druckseparator Komma wird wie ein Strichpunkt behandelt.
- Die TAB Funktion ruft einen Fehler hervor.
- Die Druckelemente werden dementsprechend formatiert, wie es die in der Format-Definition spezifizierten Felder angeben. Wenn die Zahl der Druckelemente in der *Druck-Liste* die Zahl der vorhandenen Felder im Format übersteigt, wird der momentan formatierte Datensatz geschrieben. Die restlichen Werte werden in den nächsten Datensatz geschrieben, wobei die Format-Definition wieder von ihrem Anfang bis zu ihrem Ende angewandt wird. Das Format wird so oft angewendet, wie es erforderlich ist, um die *Druck-Liste* vollständig zu verarbeiten. Jedesmal bei wiederholter Anwendung des Formates, wird ein neuer Datensatz gebildet. Wenn die Zahl der Druckelemente kleiner ist als die Zahl der Felder in der Format-Definition, wird die Ausgabe abgeschlossen, sobald das erste Feld auftritt, für das kein Druckelement existiert.
- Wenn ein formatiertes Druckelement für den Rest des momentan verarbeiteten Datensatzes zu lang ist, so wird es geteilt. Der erste Teil belegt den Rest des momentan verarbeiteten Datensatzes, und die verbleibenden Teile werden in den nächsten Datensatz geschrieben.

Querverweis

DISPLAY, IMAGE, PRINT

VAL

Eingabeformat

VAL*(String-Ausdruck)*

Beschreibung

Die VAL Funktion gibt den numerischen Wert eines *String-Ausdrukkes* an. Führende und abschließende Leerstellen werden unterdrückt. Die VAL Funktion ist die Umkehrung der STR$ Funktion.

Wenn der *String-Ausdruck* keine gültige Darstellung einer Zahl ist, tritt ein Fehler auf. Um diesen Fehler zu vermeiden, kann der String-Ausdruck zuvor mit der NUMERIC Funktion geprüft werden.

Querverweis

NUMERIC, STR$

Beispiele

```
170 NUM=VAL("78.6")
```
Setzt die Variable NUM gleich 78.6.

```
190 LL=VAL("3E15")
```
Setzt die Variable LL gleich 3.E+15.

```
300 PRINT VAL("DM3.50"):PAUSE
```
Verursacht einen Fehler, weil der String keine gültige Darstellung einer numerischen Konstanten ergibt.

VERIFY

Eingabeformat

VERIFY *"Gerät.Dateiname"* [,PROTECTED]

Beschreibung

Der VERIFY Befehl überprüft, ob Programme auf einem externen Speichergerät richtig gespeichert, oder in den Speicher des Computers richtig eingelesen wurden. VERIFY wird nach SAVE oder OLD Befehlen verwendet, um das Programm im Speicher des Computers mit dem Programm im externen Speichergerät zu vergleichen. Wenn eine Abweichung auftritt, wird eine Fehlermeldung angezeigt. Die beiden Eingabe/Ausgabe Fehlerkodes 12 und 24 zeigen einen Überprüfungsfehler an.

Gerät.Dateiname bezeichnet das Gerät und die Datei in der das Programm gespeichert ist. Gerät bezeichnet die dem Gerät zugeordnete Nummer, für die Werte zwischen 1 und 255 zulässig sind. Der *Dateiname* bezeichnet eine Datei.

Wie SAVE entfernt VERIFY alle Variablen, die nicht benutzt werden, aus dem Programm. Wenn das Programm geschützt ist, muß PROTECTED im VERIFY Befehl spezifiziert werden.

Querverweis

OLD,SAVE

Beispiele

```
SAVE"1.MEINPROGRAMM"
```
Speichert die Datei namens MEINPROGRAMM in Gerät Nr. 1.

```
VERIFY"1.MEINPROGRAMM"
```
Überprüft, ob die Datei richtig gespeichert wurde.

```
OLD"1.STAT"
```
Liest die Datei namens STAT von Gerät 1 in den Speicher des Computers.

```
VERIFY"1.STAT"
```
Überprüft, ob die Datei richtig gelesen wurde.