HEWLETT-PACKARD

HP-UX Technical BASIC
Reference Manual
Volume 1

HP-UX Technical BASIC Reference Manual

Volume 1

HEWLETT
PACKARD

Edition 1 October 1984

Notice

Portable Computer Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

Printing History **Edition 1** October 1984 Volume 1 Mfg. No. 82978-90002

Volume 1

Contents

Chapter 1

Chapter 2

Volume 2

Chapter 3 — Glossary

Chapter 4 — Reference Tables

Chapter 5 — I/O Registers

Chapter 6 — Errors

Chapter 7

Keyword Summary

Manual Update

Manual Identification

HP-UX Technical BASIC Reference Manual, Volume 1.
Part Number: 82860-90002
Edition 1: October, 1984

Update Identification

Update Number: 1
Part Number: 82860-90008
Update Date: February 1985

This update includes new information for volume 1 of the *HP-UX Technical BASIC Reference Manual*. Copies of pages 2-21, 2-22, and 2-95 are enclosed with vertical lines (change bars) to indicate the new or changed text on each page. In addition, this update provides some information specific to Release 1.0.0 of HP-UX Technical BASIC for the Integral PC.

Entering Data From the HP-IB Interface

If you are using Release 1.0.0 of HP-UX Technical BASIC on an Integral PC equipped with Release 1.0.0 or Release 1.0.1 of the operating system, you may experience a problem when ententering data from an HP-IB peripheral. You can avoid the problem by selecting the correct HP-IB handshake mode.

The HP-IB controller provides two handshake modes for entering data. The default mode is intended for mass-storage operations, and becomes the active mode when the HP-IB interface is assigned with the `ASSIGN` statement. However, if you use this mode when entering data from an HP-IB peripheral with the `ENTER` statement, you may lose the first byte of the incoming data. To avoid this, use the second handshake mode, which includes a "ready for data" hold-off. You can select the desired handshake mode by setting bit 0 of HP-IB Status/Control Register 7. Set bit 0 equal to "1" to select the "ready for data" hold-off mode. Set bit 0 equal to "0" to return to the default mode. Use the `STATUS` statement to determine the current mode. For example:

```
10 ASSIGN 7 TO "hpib"     ! Assign select code 7 to the HP-IB.
20 STATUS 7,7 ; A         ! Check the mode.
30 DISP "A=";A            ! A=0 indicates default mode.
40 CONTROL 7,7 ; 1        ! Change to "ready for data" hold-off mode.
50 STATUS 7,7 ; A         ! Check the mode.
60 DISP "A=";A            ! A should be = 1
70 ENTER 705 ; B,C        ! Read the data.
80 CONTROL 7,7 ; 0        ! Change back to default mode.
90 STATUS 7,7 ; A         ! Check mode again.
100 DISP "A=";A           ! A should be = 0
```

If you do select the "ready for data" hold-off mode, be sure to return to the default mode after your last `ENTER` operation. Mass-storage devices on the HP-IB cannot be accessed in the hold-off mode. An HP-IB lock up may result. (This does not apply to the Integral PC internal disc drive since that drive is not on the HP-IB.)

You can also avoid the loss of the first byte of data by using an input TRANSFER statement (FHS or INTR). In this case ENTER is used to input the data from an I/O buffer, rather than the HP-IB interface.

Non-Controller Operations

If you are using Release 1.0.0 of HP-UX Technical BASIC on the Integral PC, you cannot use the PASS CONTROL statement (refer to pages 2-295 and 2-296 in the *HP-UX Technical BASIC Reference Manual, Volume 1*). Also, the Integral PC is always the System Controller. You cannot make the Integral PC a non-controller device. An error will result if you attempt to do so.

The address of the Integral PC built-in HP-IB interface is always set to 30, and Status Register 4 always indicates this address. The bits of the other HP-IB status registers are consistent with the Integral PC being the Controller Active (CA) and System Controller (SC).

Hewlett-Packard Co.
Portable Computer Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330

HEWLETT PACKARD

Printed in U.S.A. 4/85
82860-90008 English

1 Introduction

How to Use This Manual

The HP-UX Technical BASIC Reference Manual is designed to provide reference information to experienced BASIC programmers. The manual is divided into seven sections:

- The **Introduction** provides general information that applies to all BASIC keywords. The introduction also explains how to interpret tables and syntax diagrams in the keyword dictionary.
- The **Keyword Dictionary** contains an alphabetical listing of all the operators, functions and statements provided with the language. Each entry contains syntax information, examples, and a description of how the keyword interacts with other related BASIC statements.
- The **Glossary** defines many of the technical terms used repeatedly in the keyword dictionary. Certain terms include a syntax diagram to help you understand the definition.
- **Reference Tables** contains a variety of useful tables, including the character set, system reset conditions, and various graphics conditions.
- The **I/O Registers** section contains tables of buffer and interface status and control registers.
- **Error Messages** lists all error messages and probable causes for the errors.
- The **Keyword Summary** groups all the BASIC keywords by function, allowing you to quickly locate the proper keyword for a particular task.

Using the Keyword Dictionary

The keyword dictionary contains an alphabetical listing of all the HP-UX Technical BASIC keywords. Each keyword entry consists of a legal usage table, a definition of the keyword, a syntax diagram, a table of parameters, usage examples, and some additional descriptive information on the use of the keyword within programs.

Legal Usage Table

The legal usage table describes in general terms the conditions under which the keyword can be used.

- If a keyword is *Keyboard Executable,* a properly constructed statement can be typed into the current alphanumeric *(alpha)* display input line and executed by pressing (RETURN).* This type of immediate execution is sometimes referred to as execution "from the keyboard" or execution in "edit mode."
- If a keyword is *Programmable,* a properly constructed statement can be placed after a valid line number and stored in memory as part of a BASIC program. Many keywords are both keyboard executable and programmable. Nonprogrammable keywords are referred to as commands.
- If a keyword can be included *in an IF...THEN,* a statement containing the keyword can be placed after THEN or ELSE in an IF...THEN...ELSE statement.

The Syntax Diagram

The syntax diagram describes pictorially how to construct a proper expression, statement, or command using that keyword. The items enclosed in ovals, circles, and rectangles are the various elements of the statement:

- The elements enclosed in ovals and circles are keywords and punctuation that must be typed in exactly as shown, except that lowercase letters can be substituted for uppercase letters.

* (RETURN) is used throughout this manual to represent the key generating a carriage return character (CR), decimal value 13.

- The elements enclosed in rectangles are parameters. Each parameter is described in the table of parameters underneath the syntax diagram. In most cases, uppercase and lowercase letters are *not* interchangeable.

The elements are connected by lines and arrows that illustrate how they fit together. Each line segment has only one arrow, meaning that the line can be followed in only one direction. Starting with the left side of the diagram, you can use any combination of elements generated by following the lines in the indicated direction. If an element is optional, a path exists around it. Many optional elements have default values listed in the table of parameters or description section. Whatever path you choose, it must terminate at the right side of the diagram.

The syntax diagram does not show line numbers or line labels.

Table of Parameters

The Table of Parameters describes each parameter in the syntax diagram. Where proper syntax or practical semantics requires a parameter to evaluate within a certain range, that range is given. A dash ("—") indicates no range restrictions. For example, in the case of numeric expressions, the parameter can be any REAL number.

Spaces

The syntax diagrams do not fully describe the use of spaces. In general, when two elements are connected by a line and arrow, any number of spaces can be inserted between the elements. In some cases, spaces are optional. For example, when a syntax diagram shows parameters separated by commas, spaces between the commas and the parameters are optional.

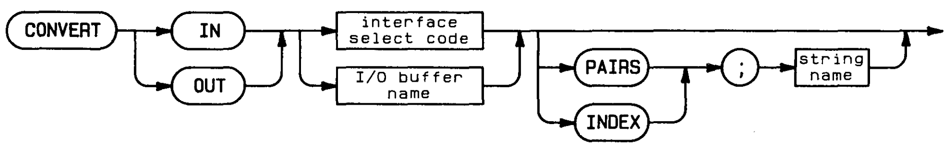
When two elements are drawn adjacent to one another, there must be no spaces between them.

The syntax requires spaces between BASIC keywords, variable names, statement labels, and numeric constants. Valid sequences of letters and digits not recognized as BASIC keywords are interpreted as variable names.

Spaces are not required between keywords or variable names and arithmetic and relational operators. However, logical operators must be separated from keywords and variable names by spaces.

Spaces should not be inserted within keywords unless explicitly shown.

Example: Examine the syntax diagram for the `CONVERT`statement:



To construct a valid statement, type the keyword `CONVERT`, followed by one or more spaces. Then, type the keyword `IN` or `OUT`, followed by one or more spaces. You must then type a valid interface select code (defined in the glossary) or I/O buffer name. The rest of the statement is optional. After leaving one or more spaces, you may type the word `PAIR` or `INDEX`, followed by a semicolon and the name of a string variable.

Line Length

The maximum number of characters that can be entered as a BASIC line is 160. This includes the line number, any embedded blank spaces, and a carriage return (`CHR$(13)`) placed at the end of the line when it is entered into system memory.

Variables

BASIC uses the following variable types:

- Simple numeric:

  Precisions: REAL, SHORT, or INTEGER (default=REAL)
- Numeric array:

  Precisions: REAL, SHORT, or INTEGER (default=REAL)

  Dimensions: one or two

  Lower bound (option base): 0 or 1 (default=0)

  Maximum upper bound: 65,530
- Simple string:

  Maximum string length: 65,530 (default=18)
- String array:

  Maximum string length: 65,530 (default=18)

  Dimensions: one or two

  Lower bound (option base): 0 or 1 (default=0)

  Maximum upper bound: 65,530

String variables are differentiated from numeric variables by using a dollar sign ($) as the final character in all string variable names. Variable names can be up to 32 characters long. Any sequence of letters, numbers, and the underscore character can be used, except that the first character must be a letter. Combinations of uppercase and lowercase letters forming BASIC keywords cannot be used.

Uppercase and lowercase letters are not interchangeable in variable names.

Line Numbers and Line Labels

Every line in a program must be preceded by a unique line number—an integer in the range 1 through 65,535. The line number can be followed by an optional line label. A line label consists of a sequence of up to 32 letters, digits, and the underscore character; the first character must be a letter. The label is followed by a colon in the labeled line; the colon is not used when the line is referenced.

Example:

`300 IF X<5 THEN Finished`	Referencing line.
⋮	⋮
`800 Finished: END`	Labeled line.

Comments

Comments can be added to any program line except a `DATA` statement. A comment is created by placing an exclamation point after the last character in the statement. Comments can also be created using the `REM` statement. Comments can contain any sequence and number of characters up to the maximum allowable line length.

If a comment is added to a multistatement line, it must be placed at the end of the line.

Multistatement Lines

A multistatement line contains two or more BASIC statements joined by the character "@". Multistatement lines can be executed both within programs and from the keyboard. The `DATA` and `REM` statements are not allowed in multistatement lines. If GOTO branching occurs in the middle of a line, the remaining statements on the line are not executed.

Like single-statement lines, multistatement lines are limited to 159 characters (plus a carriage return).

Hardware Dependencies

Certain features of the language are dependent on how BASIC is implemented on various HP-UX systems. Factors such as the internal precision of numbers, the keyboard, the character set, size of the display, availability of display windows and display graphics, multiuser capabilities, and ability to mount a removable file structure affect the use of certain keywords. When implementation of a keyword is machine-dependent, refer to documentation for your computer for additional information. Certain I/O statements have a separate description for each type interface. However, not all systems support every interface listed.

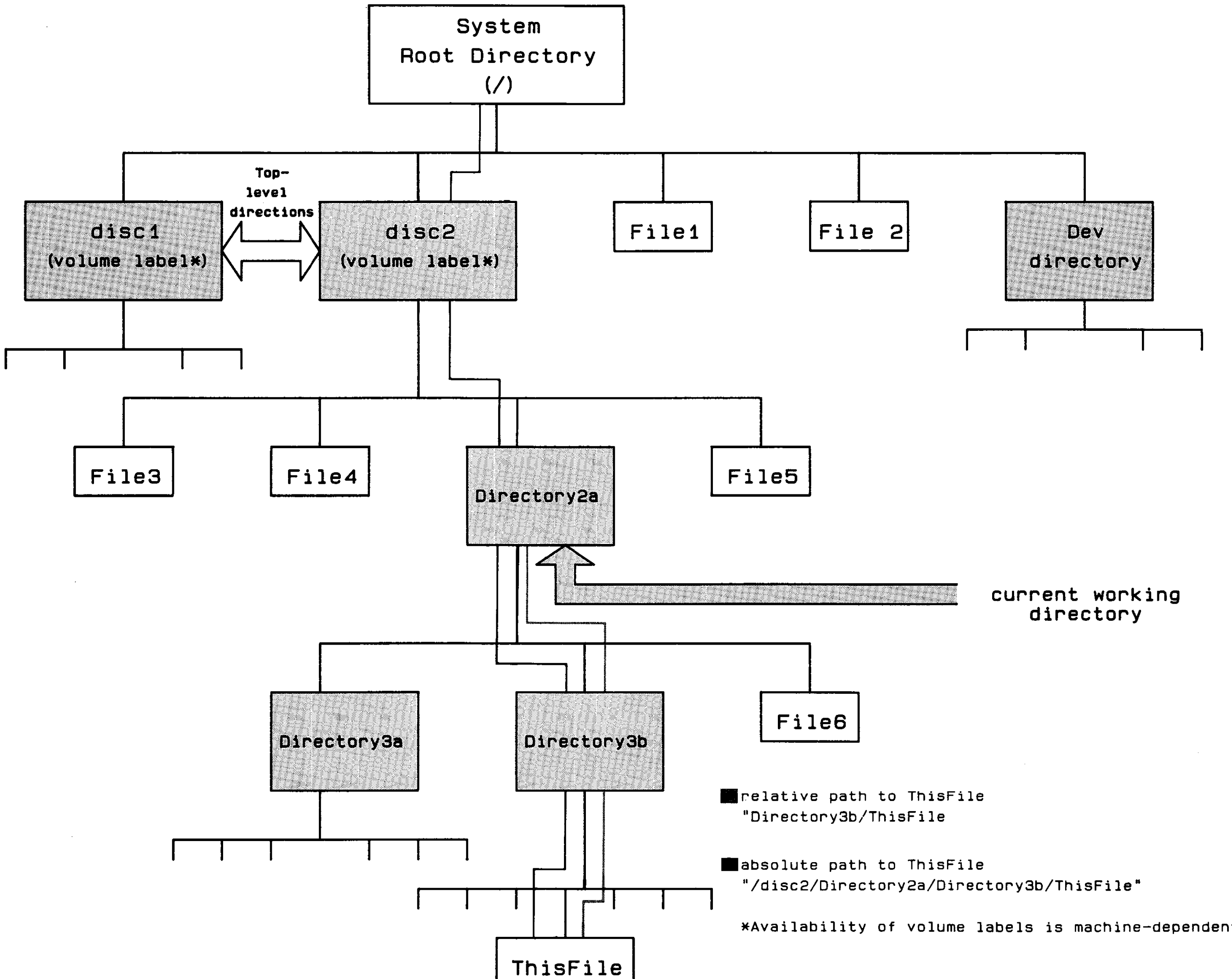
Since the range of numbers varies for different devices, no range for numeric expressions is provided in the Keyword Dictionary.

BASIC Files

BASIC creates its own file environment within the HP-UX file structure. This environment includes certain file types and file security. BASIC-type files can be created, accessed, copied, and purged within BASIC.

File Structure

BASIC uses the HP-UX hierarchical file structure. The structure takes the form of an upside down tree, as shown on page 1-8. Each box is a file. To help with this discussion, file names are drawn inside some of the boxes. Files with shaded boxes are directory files containing cataloging information for the files branching from them. The other boxes represent non-directory files. There are many different types of non-directory files, and they are utilized by the system in a number of different ways. For the purpose of this discussion, however, it is enough to distinguish between directory and non-directory types.


System
Root Directory
(/)
Top-
level
directions
disc1
(volume label*)
disc2
(volume label*)
File1
File 2
Dev
directory
File3
File4
Directory2a
File5
current working
directory
Directory3a
Directory3b
File6
ThisFile
relative path to ThisFile
"Directory3b/ThisFile
absolute path to ThisFile
"/disc2/Directory2a/Directory3b/ThisFile"
*Availability of volume labels is machine-dependent

The file at the apex of the structure is called the root directory. Underneath the root directory are a variety of files, including the *dev* (device) directory and the top-level directory of each mounted disc. In single-user systems with removable file structures, the file name of the top level directory is the volume name of the disc. Every directory file can have underneath it both directory and non-directory type files. Thus, a branching structure is produced. Within this structure, a path exists between any file and every other file in the system. This path is defined by a *path name,* which lists the route to a file from directory to directory.

At any given time, the user is working in a particular directory, called the *current working directory*. Files within this directory can be accessed by file name alone. Files outside the current working directory must be accessed by an HP-UX path name. Two types of path names exist:

- An *absolute path name* describes the path beginning at the root directory and moving downward to the file.
- The *relative path name* describes the path to the file beginning at the current working directory. As it winds its way through the file structure, a relative path can move both upward and downward.

Figure 1 illustrates a current working directory (Directory2a) and the absolute and relative paths to the file ThisFile. The path names to ThisFile are:

Absolute pathname: /disc2/Directory2a/Directory3b/ThisFile
Relative pathname: Directory3b/ThisFile

Files Types

The following file types can be created within BASIC:

BASIC Files

File Type	Contents	Statement Creating the File	Statement(s) Accessing the File
BASIC/PROG	BASIC program	STORE	LOAD
BASIC/SUBP	Subprogram	STORE	FINDPROG, CALL
BASIC/DATA	Data	CREATE	ASSIGN#, PRINT#, READ#
BASIC/GRAF	Graphics	GSTORE	GLOAD
text/data*	ASCII data	SAVE	GET

* Not a BASIC-type file.

Certain BASIC statements and commands (e.g., `GET`, `SAVE`, `CALLBIN`) access other type files. These files must be created outside BASIC. Outside BASIC, all files created in BASIC are regarded as text files.

File Security

BASIC files are created with complete user read/write permission. File security is provided by the `SECURE` statement. In general, user permission status of BASIC files should not be changed outside of BASIC.

The Configuration File

The configuration file is an ASCII text file (created outside BASIC) containing a list of the compiled binary programs to be automatically loaded whenever BASIC is invoked. The file must be named ".bconfig", and must be located in the same directory as BASIC itself.

The ".bconfig" file contains the file name or path name of each binary program to be loaded. Each file must be on a separate line. The file/path names are not enclosed in quotes.

Binary programs can also be loaded individually using the `LOADBIN` statement.

The BASIC Metacharacter

A metacharacter sequence is used to enter non-displayable characters and quotation marks into quoted strings. The sequence consists of the BASIC metacharacter character "~" (decimal code 126) followed by one through three digit characters or by a quotation mark. The metacharacter itself is ignored, in that it is not output by `PRINT`, `DISP`, and `OUTPUT`, it does not occupy a character position (`POS`), and it is not counted in the computation of the string length (`LEN`). However, the metacharacter is output by `LIST` and `PLIST`.

When the metacharacter is followed by one, two, or three digits in the range 0 through 255, that number is interpreted as a character decimal code. For example, "~7" is equivalent to CHR$(7), and "~2558A" is equivalent to CHR$(255)&"8A". If a number is in the range 256 through 999, it is moduloed 256. Thus, "~580" is equivalent to CHR$(68). A minus sign is treated like any other non-digit character.

When the metacharacter is followed a quotation mark, that quotation mark is not interpreted as a string delimiter. For example, the statement:

```
DISP "Type ~"beginner~" or ~"advanced~""
```

displays:

```
Type "beginner" or "advanced"
```

Elsewhere, the metacharater is ignored. For example, `DISP "~abc"` displays `abc`.

To include the character ~ in a string, preface it by a metacharacter; i.e., "~~".

BASIC Function Keys

Where possible, BASIC makes the following typing aid assignments to the function keys. Immediate-execute keys include a terminating carriage return; pressing the key is equivalent to typing the command and pressing (RETURN).

BASIC Function Keys

Key	Key Label	Typing Aid	Immediate Execute?
(f1)	LIST	LIST (without parameters)	Yes
(f2)	RUN	RUN (without parameter)	Yes
(f3)	STEP	SINGLESTEP	Yes
(f4)	CONT	CONT (without parameter)	Yes
(f5)	SCRATCH	SCRATCH	No
(f6)	PRT. IS	PRINTER IS	No
(f7)	PLIST	PLIST	Yes
(f8)	KEY LAB.	KEY LABEL	Yes
(f9)	MS. IS	MASS STORAGE IS	No
(f10)	DELETE	DELETE	No
(f11)	LOAD	LOAD	No
(f12)	STORE	STORE	No
(f13)	TR. ALL	TRACE ALL	Yes
(f14)	CRT IS	CRT IS	No
(f15)	INIT	INIT	Yes
(f16)	EXIT	Exits BASIC	Yes

2 Keyword Dictionary

A
B
C
D
E
F
G
H
I
K
L
M
N
O
P
R
S
T
U
V
W
X
Y

ABORTIO

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ABORTIO` statement terminates any interrupt transfer in process and resets the interface control lines. For certain interfaces, the data lines are reset.

Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10

Examples

```
ABORTIO 7
ABORTIO Isc
```

Description

Performing `ABORTIO` on an interface with an active transfer and EOT branching enabled causes the branch to be taken.

Interface-dependent action:

- HP-IB:

  System Controller—sends Interface Clear (IFC) and Remote Enable (REN).

  Active Controller (but not System Controller)—sends Attention (ATN) and My Talk Address (MTA).

  Non controller—stops handshaking data and becomes ready for next operation.

- Serial: Turns off all modem control lines (control register 2).

...ABORTIO

- BCD: Stops handshaking data, sets CTL line false, and places external data lines in high-impedance state.
- GPIO: Stops handshaking data, sets control lines false, places ports A and B in high-impedance state, and sets lines from ports C and D to false state.
- HP-IL:

  System or Active Controller—sends Interface Clear.

  Non controller—stops current operation and becomes ready for next operation.

Related Keywords

HALT, ON EOT, RESET

ABS

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ABS` function returns the absolute value of the numeric argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
PositiveValue=ABS(Value)
DISP ABS(Variable)
```

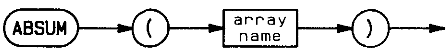
Related Keywords

`SGN`

ABSUM

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ABSUM function returns the sum of the absolute values of the elements in an array.



Item	Description	Range
array name	name of a one- or two-dimensional numeric array	any valid name

Examples

```
IF ABSUM(Array1) > 1 THEN 200
Arraysum=ABSUM(A)
```

Related Keywords

AMAX, AMIN, CNORM

ACS

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ACS function returns the arccosine of a numeric argument.

Item	Description	Range
numeric argument	numeric expression	−1 through +1

Examples

```
Theta=ACS(Y)
DISP ACS(.5)
```

Description

The function returns a REAL number. The value returned depends on the current trigonometric mode. In RAD (the default) mode, the value returned is in the range 0 through π radians. In DEG mode, the value returned is in the range 0 through 180 degrees. In GRAD mode, the value returned is in the range 0 through 200 grads.

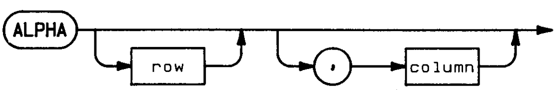
Related Keywords

COS, DEG, GRAD, RAD

ALPHA

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ALPHA statement displays the alpha display. When the optional parameters are included, the cursor is moved to the specified position.



Item	Description	Range
row	numeric expression, rounded to an integer and moduloed to a value in the range 1 through the number of rows in alpha display memory*	≥0
column	numeric expression, rounded to an integer and moduloed to a value in the range 1 through the number of columns in alpha display memory*	≥0

* The number of rows and columns in alpha display memory is machine-dependent.

Examples

```
ALPHA
ALPHA 5,7
ALPHA ,50
```

Description

When ALPHA is executed without parameters, the cursor remains in its previous position. The lines displayed are the same as were displayed most recently.

When non-zero parameters are included, the cursor is moved to the specified position. The row parameter specifies the row to which the cursor is moved. If necessary, alphanumeric display memory scrolls up or down to display the specified row on the bottom or top row of the display.

When ALPHA is executed with a row parameter and no column parameter, the cursor remains in the current column. If you designate only the column parameter, the cursor moves to the specified column and remains in the current row.

When either or both parameters is (are) 0, the cursor moves to the upper left corner of the current screen.

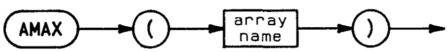
Related Keywords

CURSCOL, CURSROW, GRAPHICS

AMAX

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The AMAX function returns the value of the largest element in the specified array.



Item	Description	Range
array name	name of a one- or two-dimensional numeric array	any valid name

Examples

```
N = AMAX(Array)/10
IF AMAX(Array1) = AMAX(Array2) THEN 500
```

Related Keywords

AMAXROW, AMAXCOL, AMIN

AMAXCOL

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `AMAXCOL` function returns the column number of the largest element in the array specified most recently in an `AMAX` function.

Examples

```
YSubscript = AMAXCOL
MAT B = A(,1:AMAXCOL)
```

Description

If two or more elements in different columns have the largest value, the lowest column number is returned.

Related Keywords

`AMAX`, `AMAXROW`, `AMINCOL`, `AMINROW`

AMAXROW

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `AMAXROW` function returns the row number of the largest element in the array specified most recently in an `AMAX` function.

Examples

```
XSubscript = AMAXROW
MAT B = A(1:AMAXROW,)
```

Description

If two or more elements in different rows have the largest value, the lowest row number is returned.

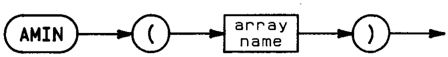
Related Keywords

`AMAX`, `AMAXCOL`, `AMINCOL`, `AMINROW`

AMIN

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The AMIN function returns the value of the smallest element in the specified array.



Item	Description	Range
array name	name of a one- or two-dimensional numeric array	any valid name

Examples

```
Y = AMIN(Array2)
IF AMIN(Array2)=0 THEN 400
```

Related Keywords

AMAX, AMINROW, AMINCOL

AMINCOL

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The AMINCOL function returns the column number of the smallest element in the array specified most recently in an AMIN function.

Examples

```
YSubscript = AMINCOL
MAT B = A(,AMINCOL:5)
```

Description

If two or more elements in different columns have the smallest value, the lowest column number is returned.

Related Keywords

AMAXCOL, AMAXROW, AMIN, AMINROW

AMINROW

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `AMINROW` function returns the row number of the smallest element in the array specified most recently in an `AMIN` function.

Examples

```
XSubscript = AMINROW
MAT B = A(AMINROW:3,)
```

Description

If two or more elements in different rows have the smallest value, the lowest row number is returned.

Related Keywords

`AMAXCOL`, `AMAXROW`, `AMIN`, `AMINCOL`

AND

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The AND operator returns a 1 or 0 based on the logical AND of the operands.

Item	Description	Range
operand	numeric expression	—

Examples

```
S=J(1) AND J(2)
IF S AND P THEN GOSUB 400
```

Description

A non-zero operand (positive or negative) is interpreted as a logical 1; an operand of zero is interpreted as a logical 0. The following table describes the result of performing a logical AND.

Logical AND

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

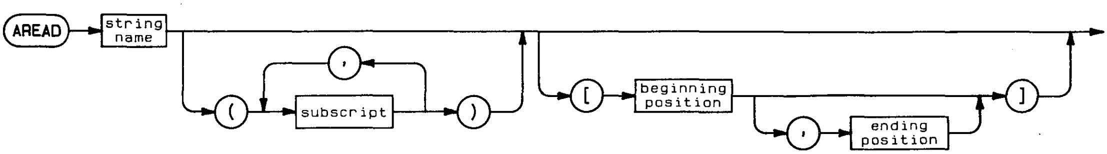
Related Keywords

EXOR, NOT, OR

AREAD

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The AREAD statement reads characters from the alpha display and copies the characters into the specified string variable.



Item	Description	Range
string name	string variable name	any valid name
subscript	numeric expression, rounded to an integer	1 through 65,530; maximum of two allowed
beginning position	numeric expression, rounded to an integer	1 through 65,530
ending position	numeric expression, rounded to an integer	1 through 65,530

...AREAD

Examples

```
AREAD Screen$
AREAD Screen$(3)
AREAD Screen$[5]
```

Description

AREAD begins copying characters at the current cursor location. The number of characters copied equals the size of the explicitly or implicitly dimensioned string variable, or the number of characters in the specified substring.

Copying preserves the characters just as they appear on the display, including leading and trailing blanks. The cursor is not copied.

If the dimensioned size of the AREAD string is larger than the number of characters following the cursor in display memory, the string is filled with trailing blanks.

Related Keywords

ALPHA, AWRIT, OFF CURSOR, ON CURSOR

ASN

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ASN` function returns the arcsine of the numeric argument.

Item	Description	Range
numeric argument	numeric expression	−1 through +1

Examples

```
Theta=ASN(.5)
DISP ASN(X*Y)
```

Description

The function returns a REAL number. The value returned depends on the current trigonometric mode. In RAD (the default) mode, the value returned is in the range $-\pi/2$ through $+\pi/2$ radians. In DEG mode, the value returned is in the range −90 through +90 degrees. In GRAD mode, the value returned is in the range −100 through +100 grads.

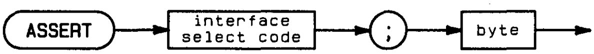
Related Keywords

`DEG, GRAD, RAD, SIN`

ASSERT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ASSERT statement sets and/or clears control lines of the specified interface.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
byte	numeric expression, truncated to an integer and moduloed 256	—

Examples

```
ASSERT 7;12
ASSERT Isc;X
```

Description

The binary value of the byte sets or clears the control lines. The action taken is interface dependent:

- HP-IB—immediately writes the value of the byte to control register 2, regardless of whether an I/O operation is in progress. IFC bit (decimal value 128) is ignored.
- Serial, BCD, GPIO—immediately writes the value of the byte to control register 2, regardless of whether an I/O operation is in progress.
- HP-IL—interrupts the interface and sends a frame using the specified byte and the most recent control bits written to register 2. Loop status is not checked before the frame is sent.

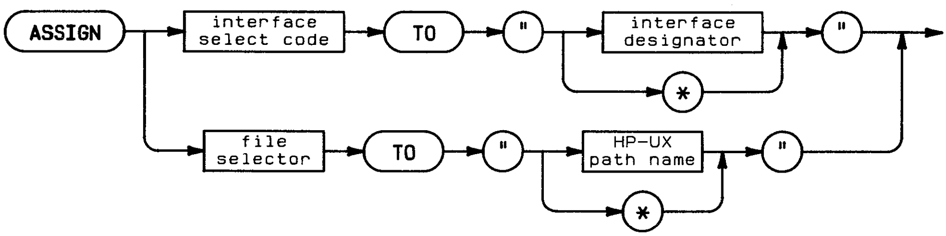
Related Keywords

ABORTIO, CONTROL

ASSIGN

Keyboard Executable	■
Programmable	■
In an IF...THEN	■

The ASSIGN statement assigns an I/O path to an interface or file, or closes an I/O path.



Item	Description	Range
interface select code	numeric expression, rounded to an integer (see glossary)	3 through 10
interface designator	the interface name (hpib, serial, bcd, gpio, hpil) combined with a port designator	
file selector	numeric expression, rounded to an integer	11 through 20
HP-UX path name	path name of an interface, device, or file (see glossary)	—

Examples

```
ASSIGN 7 TO "hpib"
ASSIGN 9 TO "serial.a2"
ASSIGN 15 TO "/vol1/dir1/dir2/myfile"
ASSIGN 7 TO "*"
```

Description

ASSIGN assigns a numeric I/O path to an interface or file. The I/O path assigned to an interface is called the *interface select code*. The I/O path assigned to a file is called the *file selector*.

Assigning an Interface Select Code. Each interface in the system must be assigned a unique interface select code in the range 3 through 10. If the interface can support only one device at a time, the *device selector* of that device is equivalent to the interface select code. For example, if interface select code 8 is assigned to a serial interface, a printer connected to that interface has a device selector of 8.

If the interface supports multiple devices, each device on the interface must have a unique primary address. Then, the device selector of each device consists of a combination of the assigned interface select code and the primary address of the device:

$$\underbrace{[d]d}_{\substack{\text{interface}\\ \text{select}\\ \text{code}\\ (3\text{-}10)}}\quad \underbrace{dd}_{\substack{\text{primary}\\ \text{address}\\ (00\text{-}31)}}$$

For example, if interface select code 7 is assigned to the HP-IB interface, a plotter with a primary address of 05 has a device selector equal to 705.

Once an interface select code is assigned to an interface, that assignment must be cancelled before a new interface select code is assigned to the interface. To cancel an assignment, assign the interface select code to "*". For example, the statement `ASSIGN 7 TO "*"` cancels the current interface select code 7 assignment.

Assigning a File Selector. File selectors in the range 11 through 20 may be assigned to BASIC files. More than one file selector may be assigned to a given file. To cancel a file selector assignment, assign the file selector to "*". `ASSIGN` should not be used to assign an I/O path to BASIC-type files.

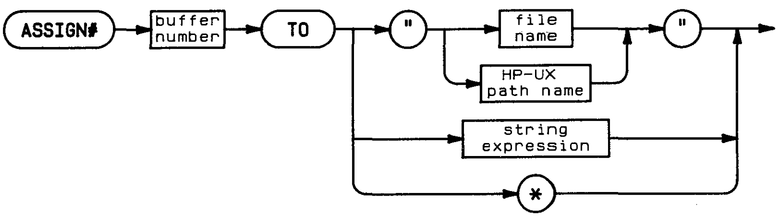
Related Keywords

`CRT IS`, `PLOTTER IS`, `PRINTER IS`

ASSIGN#

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ASSIGN#` statement opens a BASIC/DATA file by assigning to it a mass storage buffer.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—
buffer number	numeric expression, rounded to an integer	1 through 10

Examples

```
ASSIGN# 1 TO "myfile"
ASSIGN# 10 TO "system/accounting/may"
ASSIGN# 3 TO A$
```

...ASSIGN#

Description

If the file name is used alone (rather than as part of an HP-UX path name), the file must be in the current working directory.

A data file must be opened before it can be accessed. Once a buffer is assigned to a file, it remains associated with that file until the file is closed. When a file is opened, the file pointer is placed at the beginning of the file.

A file can be closed by:

- Executing `ASSIGN#` *buffer number* `TO *`.
- Assigning its buffer to another file.

The following operations cause data to be transferred from the buffer to the disc:

- The buffer becomes full.
- The file is closed.
- Program execution is halted.
- A new logical record located in a new disc block is accessed using a random access `PRINT#`.
- A `PRINT#` statement is executed from the keyboard.

Related Keywords

`PRINT#`, `READ#`

ATN

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ATN function returns the arctangent of the numeric argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Theta=ATN(1)
DISP ATN(A)
```

Description

The function returns a REAL number. The value returned depends on the current trigonometric mode. In RAD (the default) mode, the value returned is in the range $-\pi/2$ through $\pi/2$ radians. In DEG mode, the value returned is in the range -90 through 90 degrees. In GRAD mode, the value returned is in the range -100 through 100.

Related Keywords

DEG, GRAD, RAD, TAN

ATN2

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ATN2` function returns the arctangent of Y/X in the proper quadrant.

Item	Description	Range
argument Y	numeric expression	—
argument X	numeric expression	—

Examples

```
Theta=ATN2(4/3)
DISP ATN2(PointY,PointX)
```

Description

The function returns a REAL number. The value returned depends on the current trigonometric mode. In RAD (the default) mode, the value returned is in the range $-\pi$ through π radians. In DEG mode, the value returned is in the range -180 through 180 degrees. In GRAD mode, the value returned is in the range -200 through 200.

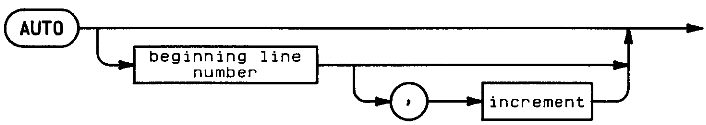
Related Keywords

`ATN, DEG, GRAD, RAD, TAN`

AUTO

Keyboard Executable ■
Programmable □
In an IF...THEN □

The AUTO command provides automatic line numbering during program entry.



Item	Description	Range
beginning line number	integer constant (default=10)	1 through 65,535
increment	integer constant (default=10)	1 through 65,535

Examples

```
AUTO 50
AUTO 100,2
```

...AUTO

Description

Executing AUTO displays the specified beginning line number. When that line has been entered, a new line number, computed by increasing the current line number by the increment, is displayed.

Automatic line numbering is halted by pressing (RETURN) in response to a new line number.*

Related Keywords

NORMAL

* Pressing (RETURN) immediately after a line number does not delete that line. For example, typing 100 (RETURN) does not delete line 100.

AWRIT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The AWRIT statement displays the specified string at the current cursor location on the alpha display.

Item	Description	Range
string expression	—	—

Examples

```
AWRIT String$
AWRIT String$(3,5)[1,10]&"________"
```

Description

The string copied to the display by AWRIT can be up to 65,530 characters in length. If necessary, the alphanumeric display scrolls to display the string as it is being copied. If the string is shorter than the size of display memory, AWRIT has no effect on cursor position—the cursor remains positioned at the first character of the AWRIT string. If the string is longer than the size of alphanumeric display memory, lines are lost from the top of display memory and the cursor is moved to home position (1,1).

Related Keywords

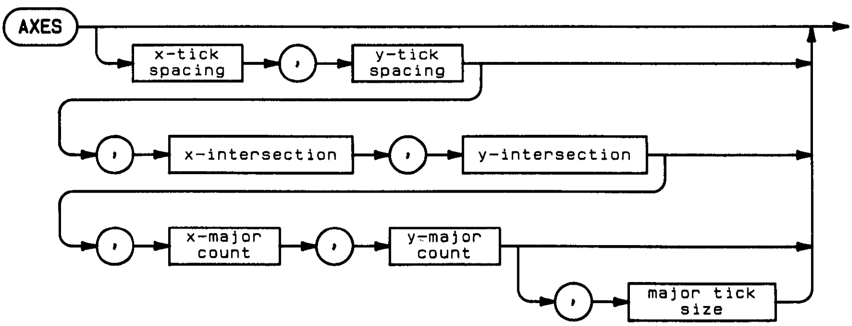
ALPHA, AREAD

A

AXES

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `AXES` statement draws a pair of axes with optional major and minor ticks.



Item	Description	Range
x-tick spacing	numeric expression, interpreted in the current units ~~(default=0, no ticks)~~ default — 10 ticks per axis	—
y-tick spacing	numeric expression, interpreted in the current units (default=0, no ticks)	—
x-intersection	numeric expression, interpreted in the current units (default=lower-left corner)	—
y-intersection	numeric expression, interpreted in the current units (default=lower-left corner)	—
x major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks on the x axis (default=1)	—
y major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks on the y axis (default=1)	—
major tick size	length of a major tick, in graphics units (default=2)	—

Examples

```
AXES 1,2
AXES 1,2,X(I),Y(I)
AXES 1,2,40,20,3,6
```

Description

The axes are drawn across the entire plotting area using the current line type. Tick marks are drawn symmetrically from the intersection of the two axes such that a major tick mark on each axis corresponds with the origin.

The x- and y- tick spacing parameters specify the distance between tick marks on each axis. Negative numbers are interpreted as positive values by taking the absolute value. The default value of 0 draws no ticks.

The x-intersection parameter specifies, in current x-axis units, the point where the x-axis intersects the y-axis. The y-intersection parameter sepcifies, in current y-axis units, the point where the y-axis intersects the x-axis.

The x- and y- major count parameters specify the number of intervals between major ticks. For example, a major count of 4 means that every fourth tick is a major tick. The default value of one draws each tick as a major tick.

The major tick size parameter specifies the length of the major ticks in graphics units. The default length is 2 GU's. Minor ticks are always ½ the size of major ticks.

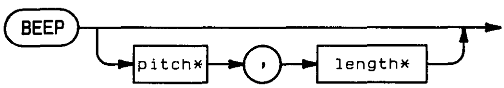
Related Keywords

GRID, LINE TYPE, LAXES, LGRID, XAXIS, YAXIS

BEEP

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `BEEP` statement produces an audible tone. The pitch and length parameters are ignored if the system is not capable of producing variable tones.



Item	Description	Range
pitch*	numeric expression, rounded to an integer	—
length*	numeric expression, rounded to an integer	—
* The ability to change frequency and duration of the tone and the interpretation of BEEP parameters are machine-dependent.		

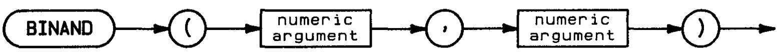
Examples

```
BEEP
BEEP 100,120
```

BINAND

Keyboard Executable ■
B **Programmable** ■
In an IF...THEN ■

The BINAND function returns the bit-by-bit AND of the binary representation of two integer arguments.



Item	Description	Range
numeric argument	numeric expression, rounded to an integer	range of integers

Examples

```
X = BINAND(A(1),31)
DISP BINAND(4X*2,Y)
```

Description

The arguments are represented as two's complement integers. The results of each bit-by-bit AND are used to construct the integer returned by the function. Each bit is computed according the following truth table.

Logical AND Used in BINAND

Argument #1	Argument #2	Result
0	0	0
0	1	0
1	0	0
1	1	1

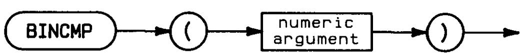
Related Keywords

BINCMP, BINEOR, BINIOR, BIT

BINCMP

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The BINCMP function returns the binary complement of an integer.



Item	Description	Range
numeric argument	numeric expression, rounded to an integer	range of integers

Examples

```
Tflag=BINCMP(Z)
DISP BINCMP(2X+4)
```

Description

The argument is represented as a two's complement integer. Each bit of the result is the inverse of the corresponding bit in the argument. If the argument is smaller than the number of bits per integer, leading zeros are assumed.

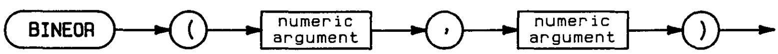
Related Keywords

BINAND, BINEOR, BINIOR, BIT

BINEOR

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The BINEOR function returns the bit-by-bit exclusive OR of the binary representation of two integer arguments.



Item	Description	Range
argument	numeric expression, rounded to an integer	range of integers

Examples

```
A=BINEOR(S(1),S(2))
DISP BINEOR(2X,6)
```

Description

The arguments are represented as two's complement integers. The result of each bit-by-bit exclusive OR is used to construct the integer returned by the function. Each bit is computed according the the following truth table.

Exclusive OR Used in BINEOR

Argument #1	Argument #2	Result
0	0	0
0	1	1
1	0	1
1	1	0

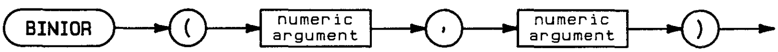
Related Keywords

BINAND, BINCMP, BINIOR, BIT

BINIOR

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The BINIOR function returns the bit-by-bit inclusive OR of the binary representation of two integer arguments.



Item	Description	Range
numeric argument	numeric expression, rounded to an integer	range of integers

Examples

```
DISP BINIOR(X(1),C(1))
IF BINIOR(B,1)=8 THEN 200
```

B

Description

The arguments are represented as two's complement integers. The result of each bit-by-bit inclusive OR is used to construct the integer returned by the function. Each bit is computed according the the following truth table.

Inclusive OR Used in BINIOR

Argument #1	Argument #2	Result
0	0	0
0	1	1
1	0	1
1	1	1

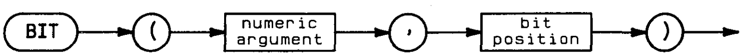
Related Keywords

BINAND, BINCMP, BINEOR, BIT

BIT

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `BIT` function returns the value (0 or 1) of the specified bit of the argument.



Item	Description	Range
numeric argument	numeric expression, rounded to an integer	range of integers
bit position	numeric expression, rounded to an integer, indicating which bit is returned	1 through the number of bits per integer

Examples

```
Flag1=BIT(A(1),0)
IF BIT(R1,15)=1 THEN R1$="ON"
```

Description

The argument is represented as a two's complement integer. Bit 0 is the least significant bit.

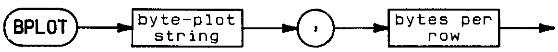
Related Keywords

`BINAND`, `BINCMP`, `BINEOR`

BPLOT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The BPLOT (*byte-plot*) statement plots groups of dots onto the graphics display.



Item	Description	Range
byte-plot string	string expression	non-displayable characters must be specified using CHR$ or a metacharacter sequence
bytes per row	numeric expression, rounded to an integer	even numbers only, cannot exceed width of the display*

* The number of raster dots per row is machine-dependent. The maximum bytes per row equals the number of raster dots per row divided by 8.

Examples

```
BPLOT "Q!w@",2
BPLOT A$&"11(■"&CHR$(122), N
```

Description

The display must be the current PLOTTER IS device; byte-plotting cannot output to a peripheral plotter. An error occurs if any portion of the byte-plot is out-of-bounds.

BPLOT starts plotting at the current pen position and plots across rows of dots from left to right. Each character (byte) in the byte-plot string specifies the pattern of eight display dots. The binary equivalent of each character determines which of the plotted dots are on (1) and off (0).

The bytes per row determines how many characters are plotted on a row. When the specified number of characters are plotted, the pen moves to the next row, and plotting continues with the next character in the byte-plot string. The sign of the bytes per row parameter determines how BPLOT dots interact with existing dots. If the parameter is positive, BPLOT performs an exclusive OR with existing dots. When the parameter is negative, new BPLOT dots overwrite existing dots.

BPLOT writes to the display two bytes at a time. The bytes per row parameter must be an even number. No error is returned for odd numbers, but the byte-plot string is not interpreted properly.

At the conclusion of the byte-plot, the pen is moved to the next row of dots, directly beneath the left-most dot just plotted.

BPLOT Interaction With Exisiting Dots

BPLOT **Dot (as specified in byte-plot string)**	**Existing Dot (on the display)**	**New Dot on the Display**	
		Positive Bytes Per Row Parameter (EXOR)	**Negative Bytes Per Row**
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

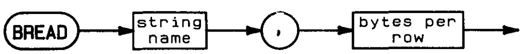
Related Keywords

BREAD

BREAD

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `BREAD` (*byte-read*) statement reads groups of eight dots from the graphics display, converts each group of eight dots to a character, and stores the characters in the specified string variable.



Item	Description	Range
string name	name of a simple string variable or string array element	any valid name
bytes per row	numeric expression, rounded to an integer	even numbers only, cannot exceed width of the display*

* The number of raster dots per row is machine-dependent. The maximum bytes per row equals the number of raster dots per row divided by 8.

Examples

```
BREAD A$,3
BREAD String$(3), N
```

Description

The display must be the current PLOTTER IS device. Byte-reading cannot address a peripheral plotter.

BREAD starts reading at the current pen position, and reads across the row of dots from left to right. Each group of eight dots is converted to a character based on the binary equivalent of each character in the computer character set. When a dot is on, it is read as 1.

The bytes per row determines how many characters are read on a row. When the specified number of characters has been read, the pen moves to the next row, and byte-reading continues. At the conclusion of the byte-read, the pen is moved to the next row of dots, underneath the left-most dot read most recently.

BREAD reads the display two bytes at a time. The bytes per row parameter must be an even number. No error is returned for odd numbers, but the byte-read string will contain incorrect data.

The BREAD string can contain characters with decimal codes in the range 0 through 255. If the contents of the string are to be displayed (on the alpha display) or printed, the characters with decimal codes in the range 0 through 31 should be converted to their decimal codes (using NUM) to avoid unpredictable display or printer activity.

Related Keywords

BPLOT

BTD

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `BTD` (*binary-to-decimal*) function interprets the string argument as the binary representation of a number and returns the numeric decimal equivalent.

Item	Description	Range
string argument	string expression containing the base 2 representation of an integer	0's and 1's only; cannot exceed the range of integers

Examples

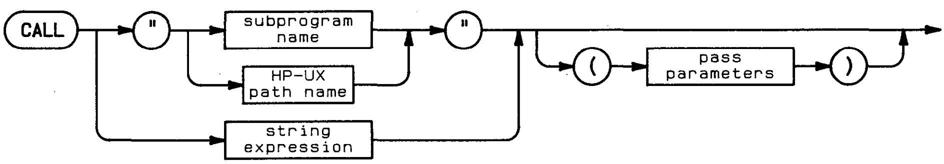
```
Y=BTD(H$)+X
DISP BTD("11010000")
```

Related Keywords

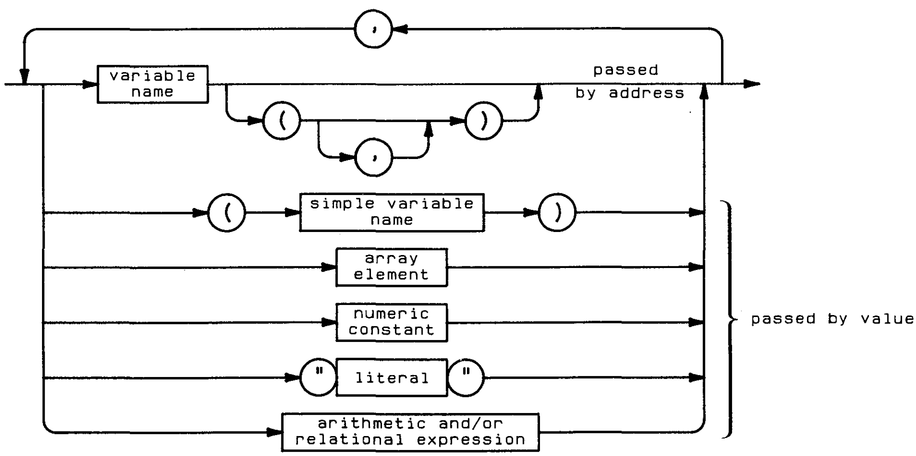
`DTB$`, `DTH$`, `DTO$`, `HTD`, `OTD`

Keyboard Executable	□
Programmable	■
In an IF...THEN	■

The CALL statement transfers program execution to the specified subprogram and, optionally, passes parameters into the subprogram.



pass parameters



c

Item	Description	Range
subprogram name	literal; name of a BASIC/SUBP file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a subprogram name or HP-UX path name	—
simple variable name	name of a simple numeric or string variable	any valid name
array element	element of a numeric or string array	—
numeric constant	none	—
literal	string constant containing keyboard characters, the CHR$ function, and/or metacharacter sequences	—
arithmetic or relational expression	expression containing variables and/or constants, along with arithmetic or relational operators	—

Examples

```
CALL Solstice"
CALL "SUB#1"(Number,String$,Array$(),Element$(3,7)[4,9],A*B/2)
CALL "/D1/D2/SUB1"((a))
```

Description

The CALL statement searches system memory and, if necessary, the current working directory or specified mass storage location, for the designated subprogram. The HP-UX path name must be used if the subprogram file is located elsewhere than computer memory or the current working directory. When the subprogram is found (and, if necessary, loaded), execution begins.

There are three ways to pass parameters between the calling (sub)program and the called subprogram:

- The variables can be included in COM statements in the main program and one or more subprograms. Changes in the values assigned to these variables are returned to the calling program. Numeric and string constants cannot be transferred this way.
- Parameters can be passed by address. The declared precision of numeric variables accompanies them into the subprogram. Changes in the values assigned to the variables are returned to the calling program. Entire arrays can be passed this way; individual elements of arrays cannot be passed. When an array is passed to a subprogram, the option base of the program and subprogram must agree.
- Parameters can be passed by value. Changes in the values assigned to the variables are *local* to the subprogram; they are not transferred back to the calling program. Individual elements of arrays can be passed this way; entire arrays cannot be passed unless they are specified element by element. Numeric and string expressions can be passed by value.

Parameters are passed in the order in which they appear, left to right. The CALL statement can contain fewer parameters than the SUB statement of the subprogram it calls. Optional parameters are listed following the required parameters. The number of parameters passed into the subprogram is returned by the NPAR function. At the beginning of subprogram execution, unfilled numeric parameters are set to 0, type REAL; unfilled string parameters are set to the null string.

Certain system properties are global; they are passed between the main program and subprograms. Other properties are local—known only to the program or subprogram in which they are set or enabled. The following declarations are local; all others are global:

Local Properties

```
OFF EOT ←→ ON EOT
OFF ERROR ←→ ON ERROR
OFF INTR ←→ ON INTR
OFF KEY# ←→ ON KEY#
OFF KYBD ←→ ON KYBD
OFF TIMEOUT ←→ ON TIMEOUT
OFF TIMER# ←→ ON TIMER#
```

When `SUBEND` or `SUBEXIT` is executed, program execution returns to the statement immediately following `CALL`. Subprograms cannot be invoked by event-initiated branching.

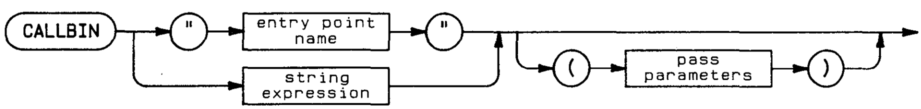
Refer to the table of Reset Conditions for additional information.

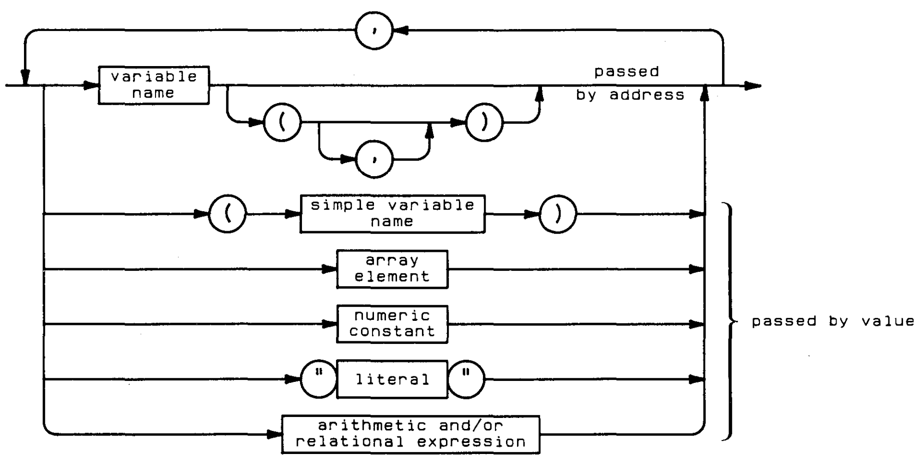
Related Keywords

`COM`, `FINDPROG`, `NPAR`, `STORE`, `SUB`

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The CALLBIN statement calls a binary entry point.





C

Item	Description	Range
entry point name	literal; name of an entry point	⩽16 characters
string expression	expression evaluating to an entry point name	—
pass parameters	(see diagram)	—
simple variable name	name of a simple numeric or string variable	any valid name
array element	element of a numeric or string array	—
numeric constant	none	—
literal	string constant containing keyboard characters, the CHR$ function, and/or metacharacter sequences	—
arithmetic or relational expression	expression containing variables and/or constants, along with arithmetic or relational operators	—

Examples

```
CALLBIN A$
CALLBIN "drawpattern" (A(),(B),C$,D(4,5),2E12,"abcde",4*A)
```

Description

The binary program must have been previously loaded with `LOADBIN` or by invoking the configuration file .bconfig. If duplicate entry point names are present, the first entry pointed retrieved into memory is executed.

Execution of BASIC resumes at the statement following `CALLBIN`.

The following rules and procedures apply only to simple binary programs. Simple programs are diffentiated from more complex binaries in that they make no references to BASIC—all communication with BASIC is through pass parameters listed in CALLBIN.

Passing BASIC Variables Into Binaries. The following rules apply to CALLBIN pass parameters:

- Variables can be passed from BASIC by address or by value (see syntax diagram). The routine to which the BASIC variables are passed must provide the proper formal parameters.
- The precision of BASIC variables should match binary declarations. In "C", for example, INTEGER variables and integer constants become type int; REAL and SHORT variables, and all numeric expressions except integer constants, becomes type double.
- Strings must be dimensioned in BASIC before they are passed to the formal parameters. A character (null string is permissible) should be assigned to the last character position in the string. This ensures that the entire dimensioned length of the string will be recognized by BASIC when parameters are passed back to it. For string arrays, the array must be dimensioned, and a character should be assigned to the last position of each element in the array.
- Array dimensions must be dimensioned in the binary to be the same as in the BASIC program. The BASIC option base should be 0.

Compiling and Loading Simple Binaries. When the binary is written, the following procedure compiles it:

1. Compile the binary program into and object file. Refer to your language documentation for further information. (For "C", compile using the "-c" option. This creates a compiled ".o" file.)
2. Execute the HP-UX ld command, using the object file as the input file. Use the "-r" and "-d" options to retain relocation information and space allocated for common variables. If the binary uses any library routines, include the appropriate ld parameter to load them. (Refer to the HP-UX Reference documentation.)
3. The output file specified in the ld command now contains the complete, compiled binary. The binary can be loaded using LOADBIN or by using the .bconfig (BASIC configuration) file.

The .bconfig file contains a list of binary programs to be loaded whenever BASIC is invoked. The binary programs are listed one per line. BASIC and .bconfig must be in the same directory.

Related Keywords

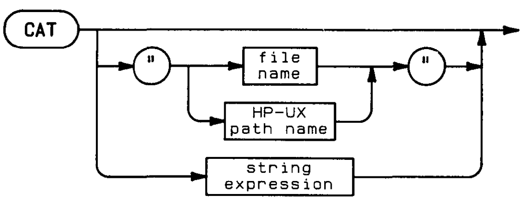
`CALL, LOADBIN, SCRATCHBIN`

CAT

C

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The CAT statement displays the contents of the specified directory or the directory information associated with a particular file.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
CAT "/vol1"
CAT "vol1/dir2"
```

Description

The output from CAT depends on whether the file is a directory or non-directory file, and whether the non-directory file was created in BASIC or elsewhere.

Cataloging Directory Files. When the specified file is a directory, `CAT` displays the path to the specified directory (as specified in the CAT parameter) and a list of the directory contents. The directory entry for each file contains the following information:

- **name**—the file name.
- **size**—the size of the file in bytes.
- **type**—directory, text/data, fifo/pipe, or device.
- **permission**—read, write, read/write, or none.
- **date modified**—the date the file was last modified.

When `CAT` is executed without parameters, the contents of the current working directory and the directory's absolute path name are listed. Files with file names beginning with a period will not be listed.

Cataloging BASIC Files. When the specified file is a BASIC file, `CAT` displays the following information about the file:

- **name**—the file name.
- **bytes**—the number of bytes per file record.
- **recs**—the number of records in the file.
- **blocks**—the number of blocks occupied by the file.
- **type**—BASIC/DATA, BASIC/PROG (program), BASIC/SUBP (subprogram), BASIC/GRAF (graphics).
- **date modified**—the date the file was last changed.

If the file name is used alone rather than as part of an HP-UX path name, the file must be located in the current working directory. Files beginning with a period will be listed.

Cataloging Non-BASIC Files

If the specified file is not a directory and is not a BASIC-type file, the catalog consists of a file header followed by one line one line containing the name, size, type, permission, and date modified.

CEIL

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `CEIL` function returns the smallest integer greater than or equal to the numeric argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
T=CEIL (X)
RoundUpX=CEIL(X)
```

Description

The `CEIL` and `IP` functions return the same result for negative arguments.

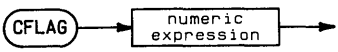
Related Keywords

`IP`, `INT`

CFLAG

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The CFLAG statement clears the specified flag.



Item	Description	Range
flag number	numeric expression, rounded to an integer	+1 through +64

Examples

```
CFLAG 25
IF X#4 then CFLAG 2*I
```

Description

The CFLAG statement clears one flag at a time. SFLAG is used to clear from 1 to the entire 64 flags at once.

All flags are cleared when RUN, INIT, or CHAIN is executed.

Related Keywords

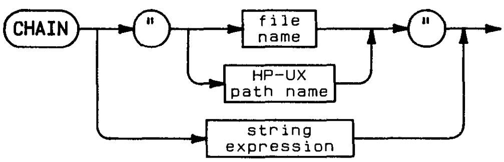
FLAG, FLAG$, SFLAG

CHAIN

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The CHAIN statement scratches the current BASIC program, retrieves the specified BASIC/PROG file, and starts program execution.

C



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
CHAIN "Filename"
CHAIN FILE$
CHAIN "/Dir1/Dir2/filename"
```

Description

If the file name is used alone (rather than as part of an HP-UX path name), the file must be in the current working directory.

When a program is chained into memory:

- All variable assignments are scratched except those declared in common by `COM` statements in the calling program and chained program.
- Assignments made to the user-defined keys by the previous program are scratched.
- Event-initiated branches (`ON ERROR`, `ON TIMER#`, `ON KEY#`, `ON KYBD`, `ON TIMEOUT`, `ON INTR`, `ON EOT`,) are disabled.
- Binary programs in memory remain intact.
- Program flags are cleared.
- All subprograms in memory are scratched.

Refer to the table of Reset Conditions on pages 4-8 and 4-9 for additional information.

Related Keywords

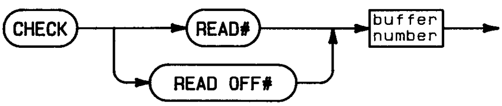
`COM`

CHECK READ#

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The CHECK READ# statement enables and disables verification of data printed to the disc.

C



Item	Description	Range
buffer number	numeric expression, rounded to an integer	1 through 10

Examples

```
CHECK READ# 1
CHECK READ# BufferNumber
CHECK READ OFF# 1
```

Description

When check read is enabled, the system performs an immediate read-after-write whenever data is transferred from the specified buffer to the disc. If a byte-by-byte comparision detects a difference, an error is returned.

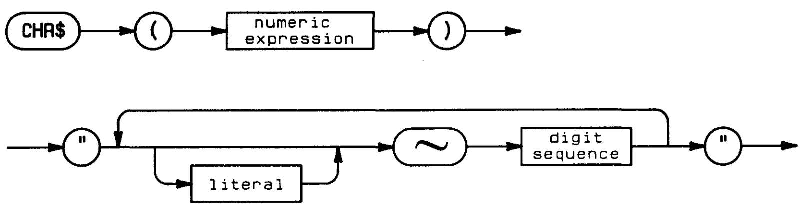
Check read is disabled by executing CHECK READ OFF#.

Related Keywords

PRINT#

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The CHR$ function converts a numeric value into a string character according to the machine character set.



Item	Description	Range
numeric argument	numeric expression, rounded to an integer and moduloed 256 to evaluate within the range 0 through 255	numbers outside the range −32,767 through +32,767 are interpreted as 255.

Examples

```
PRINT A,B,CHR$(13), C
IF A$[X,X]=CHR$(10) THEN 300
```

Description

CHR$ can be used to include non-displayable characters and quotation marks in literals. (The metacharacter, ~, can also be used. Refer to page 1-11).

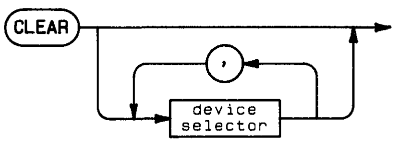
Related Keywords

NUM

CLEAR

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

When executed without a decive selector, the CLEAR statement clears the contents of the alpha display. When a device selector is specified, CLEAR clears the specified interface or resets the peripheral device.



Item	Description	Range
device selector	numeric expression, rounded to an integer (see glossary)	—

Examples

```
CLEAR 705
CLEAR Isc*100+D1, Isc
```

Description

CLEAR Without Parameters. When CLEAR is executed without parameters, it clears all of alpha display memory and moves the cursor to home position (1,1).

CLEAR With Parameters. If two or more device selectors are specified, they must be valid combinations of the interface select code and primary address. The devices must be at the same select code.

The following interface-dependent action is taken:

- HP-IB and HP-IL (must be active controller):

  If the device selector is an interface select code, then Device Clear (DCL) is sent.

  If the device selector contains a primary address, then Unlisten (UNL), Listen Address(es)(LAD), and Selected Device Clear (SDC) are sent.

  HP-IB leaves ATN true; use RESUME to set ATN false.

- GPIO:

  If the device selector is an interface select code, the interface pulses RESA and RESB.

  If the device selector contains an even primary address, the interface pulses RESA.

  If the device selector contains an odd primary address, the interface pulses RESB.

- Serial, BCD: Error.

Related Keywords

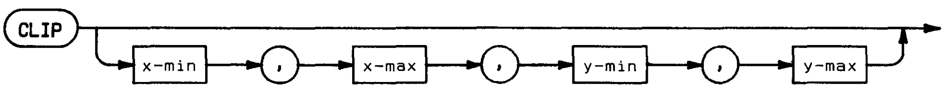
CONTROL, GCLEAR, SEND

CLIP

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The CLIP statement specifies plotting boundaries (the soft clip area) in the current scale units.

C



Item	Description	Range
x min	numeric expression, interpreted in current units	—
x max	numeric expression, interpreted in current units	—
y min	numeric expression, interpreted in current units	—
y max	numeric expression, interpreted in current units	—

Examples

```
CLIP 0,50,0,10
CLIP 10*D, 10*D+50, 0,100
```

Description

The CLIP parameters, expressed in current units, define the boundaries of the plotting area. These boundaries replace any previously established plotting boundaries. No lines can be drawn beyond the plotting boundaries, but labels can be drawn outside the plotting area and within the graphics limits.

Executing `CLIP` without parameters provides for digitizing the plotting boundaries. Program execution halts until two corners of the boundaries are entered from the plotting device.

The plotting area defined by `CLIP` cannot be scaled by `SCALE`, `MSCALE`, or `SHOW`. When a scaling statement is executed after a `CLIP` statement, the new user units are mapped onto the LOCATE plotting area or onto the graphics limits.

Plotting boundaries set by `CLIP` are canceled when `LIMIT`, `PLOTTER IS`, or `UNCLIP` is executed. The `SETGU` statement deactivates the plotting boundaries; they are restored by executing `SETUU`.

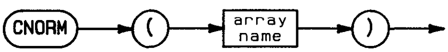
Related Keywords

`LOCATE`, `UNCLIP`

CNORM

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `CNORM` function returns the *column norm* of an array. The column norm is computed by summing the absolute values of the elements in each column of the array and selecting the largest sum.



Item	Description	Range
array name	name of a one- or two-dimensional array	any valid name

Examples

```
SUM=CNORM(Array1)
IF CNORM(A)≤CNORM(B) THEN Y=CNORM(A)
```

Related Keywords

`ABSUM`, `CNORMCOL`, `FNORM`, `RNORM`

CNORMCOL

Keyboard Executable	■
Programmable	■
In an `IF...THEN`	■

The `CNORMCOL` function returns the column number of the column having the largest sum of absolute values, using the array specified in the most recently executed `CNORM` function.

Examples

```
MAT B = MAT A(,CNORMCOL)
Array1(3,CNORMCOL)=0
```

Related Keywords

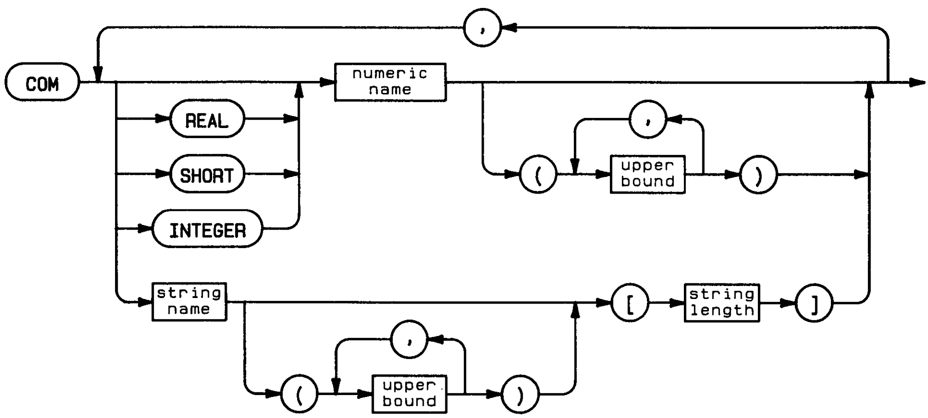
`ABSUM`, `CNORM`

COM

Keyboard Executable ☐
Programmable ■
In an IF...THEN ☐

The COM statement dimensions variables, reserves memory for them, and preserves variable assignments when chaining programs or calling subprograms.

C



Item	Description	Range
numeric name	name of a simple numeric variable or numeric array	any valid name
upper bound	integer constant	1 through 65,530
string name	name of a simple string variable or string array	any valid name
string length	integer constant	1 through 65,530

Examples

```
100 COM Number,Array(3,10),String$,SArray$(12)[30]
300 COM REAL A,B(5),INTEGER I(50),d$
```

Description

COM declares variables to be held "in common" between programs and subprograms. When a variable is held in common, its precision (REAL, SHORT, or INTEGER), properties (array lower and upper bounds, string length), and assigned value are preserved.

COM has two purposes:

- To preserve variables during program chaining. When a program chains another program, all program variables are scratched except those held in common.
- To pass variables between a program and a subprogram.

Common variables are scratched by executing RUN, INIT, or SCRATCH.

When variables are held in common, matching COM statements must appear in the originating program and the (sub)program (accessed by CHAIN or CALL). Variables held in common must agree in type (numeric versus string, simple versus array), precision, option base, upper bound, and maximum string length. When precision is not specified, the variable is assumed to be REAL. All string variables must include an explicitly dimensioned string length.

When COM includes one or more precision declarations, all numeric variables following the declaration have that precision until another declaration is encountered.

A (sub)program can have any number of COM statements. However, the same variable cannot appear in more than one COM statement. The variable names need not match between (sub)programs. Variable assignments and properties are passed based on the order in which they appear in the (sub)program's COM statement(s).

If an OPTION BASE statement is used in a program, it must appear before any COM statements. If one or more arrays are held in common during chaining, the option base of the two programs must agree. Likewise, the option base of a program and subprogram must agree if arrays are passed into the subprogram.

A COM statement cannot be included within a function definition. COM cannot be used to pass numeric and string constants to subprograms.

Related Keywords

DIM, INTEGER, REAL, SHORT

CONT

Keyboard Executable ■
Programmable □
In an `IF...THEN` □

The `CONT` command resumes execution of a program at the specified line after it has been paused.

Item	Description	Range
line number	integer constant identifying a program line (default=next program line)	1 through 65,535

Examples

```
CONT 100
```

Description

Executing `CONT` without a line number causes program execution to resume at the line at which execution was paused. When a line number is specified, execution resumes at that line in the current program or subprogram. If the specified line number does not exist, execution resumes at the next higher-numbered line.

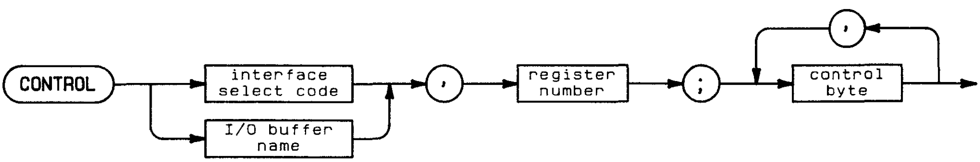
When a program is continued, variables retain their current values. If a program is edited while paused, it cannot be continued. Instead, it must be run.

Related Keywords

`INIT`, `PAUSE`, `RUN`

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `CONTROL` statement writes one or more bytes of data to interface registers or I/O buffer registers.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
I/O buffer name	name of string variable declared an I/O buffer	—
register number	numeric expression, rounded to an integer	0 through 23
control byte	numeric expression, rounded to an integer	0 through 255

Examples

```
CONTROL 7,16; 3
CONTROL I,17; C(1),C(2),C(3)
```

Description

The register number specifies the first register to be used. If more than one control byte is listed, the values are written to consecutive registers. The binary equivalent of each control byte sets and clears bits in the register(s).

Related Keywords

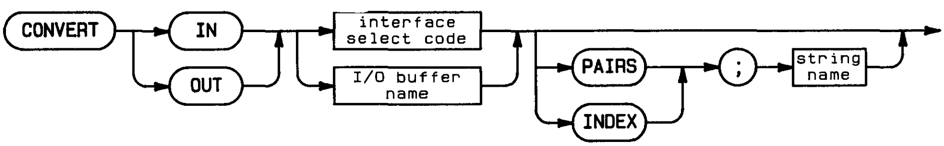
`ASSERT`, `ENABLE INTR`, `STATUS`

CONVERT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The CONVERT statement enables or disables a specified character conversion table to be used during ENTER and OUTPUT operations.

C



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
I/O buffer name	name of a string variable previously declared as an I/O buffer	—
string name	name of string variable containing the conversion table	—

Examples

```
CONVERT IN 7 PAIRS; A$
CONVERT OUT 3 INDEX; B$
```

Description

CONVERT converts incoming or outgoing data exchanged via a specified I/O path—an interface (interface select code) or I/O buffer (I/O buffer name). The OUT option specifies that the conversion is to be used on all OUTPUT data on the specified path; IN specifies that the conversion is to be used on all ENTER data on that path. The conversion is not performed on SEND and TRANSFER operations.

Separate IN and OUT conversions can be specified for the same I/O path.

When the optional parameters are omitted, the previously specified conversion for that I/O path and direction is disabled.

CONVERT by PAIRS. `PAIRS` specifies that the conversion string contains pairs of characters. Each pair consists of the original character and the character to which it is converted. Before each character is moved through the interface or buffer, it is compared to the original characters in the conversion string. If a match is found, it is replaced by the character following the original character.

CONVERT by INDEX. `INDEX` defines a conversion table based on the string variable. The decimal value of each incoming or outgoing character is interpreted as a character position value in the table. For example, an incoming # (decimal value 35) is converted to the 35th character in the string variable.

Related Keywords

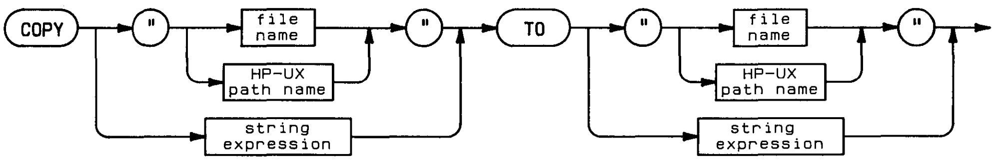
`ENTER`, `OUTPUT`

COPY

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The COPY statement copies an individual file or all the files in a specified directory.

C



Item	Description	Range
file name	literal	14 characters maximum; slash, quotation marks, and leading colon not allowed;
HP-UX path name	literal (see glossary)	—
string expression	expression evaluating to an HP-UX path name	—

Examples

```
COPY "/discA/testI/trial5" TO "/discB/test1a/trial5"
COPY "/mydisc" TO "/yourdisc"
```

Description

Two copying operations are available. *File-to-file* copy copies the contents of a non-directory file to a new file. The new file can be in the same directory or in another directory. *Directory-to-directory* copy copies the contents of all the files in a directory to another directory. The syntax of both operations is the same; the type of copying that occurs depends on whether the file to be copied is a directory file.

Files secured with type 1 security cannot be copied. No error is generated, but the copying operation does not occur.

Attempting to copy a file to a disc with insufficient space for that file causes an error. If the error occurs during a directory-to-directory copy, all files copied before the error remain intact.

File-to-File Copy. When the file to be copied is a non-directory file, file-to-file copying occurs. File-to-file copying creates a new file with the specified name in the directory indicated by the path name of the new file. The contents of the source file is copied into the new file, and the directory in which the new file is located is updated. The source file and the new file can be in the same directory if they have unique file names. If a file name is used alone, that file must be located in, or will be created in, the current working directory.

Directory-to-Directory Copy. When the file to be copied is a directory file, all the files in the source directory are copied to the destination directory, and the destination directory is updated to add the new files. The destination directory must have been previously created. The names of the copied files are not changed. Subdirectories are not copied.

If a duplicate file name or other non-fatal copying error occurs during copying, that file is skipped and copying continues. An error message is displayed when copying is completed. If more than one non-fatal error occurs, only one message corresponding to the first error is displayed.

If a file name alone is used, it must be the name of a directory file in current working directory.

Related Keywords

SECURE, UNSECURE

COS

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `COS` function returns the cosine of the angle argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Y=COS(Angle)
X=R*COS(Theta)
```

Description

The angle argument is interpreted according to the current trigonometric mode—RAD (radians), DEG (degrees), or GRAD (grads). The default mode is RAD.

Related Keywords

`ACS, DEG, GRAD, RAD`

COT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The `COT` function returns the cotangent of the angle numeric argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Y=COT(Theta)
DISP "Cotangent of angle is"; COT(A)
```

Description

The angle argument is interpreted according to the current trigonometric mode—RAD (radians), DEG (degrees), or GRAD (grads). The default mode is RAD.

Related Keywords

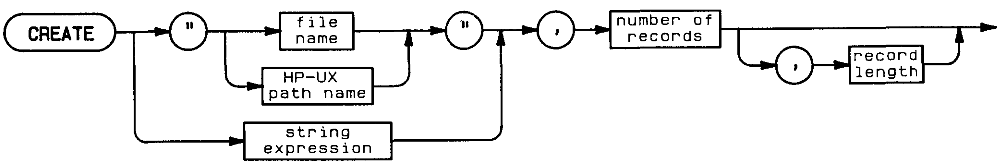
`ATN, ATN2, DEG, GRAD, RAD, TAN`

CREATE

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The CREATE statement creates a BASIC/DATA file on a disc.

C



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—
number of records	numeric expression, rounded to an integer	limited by capacity of medium
record length	numeric expression, rounded to an integer (default=256)	≥4 bytes

Examples

```
CREATE "newfile", 20, 64
CREATE "/disc1/newfile", Recs, Size
```

Description

If the file name is used alone (rather than as part of an HP-UX path name), the file is created in the current working directory. When an HP-UX path name is used, the file is created in the specified directory. An error is returned if the file name already exists.

When the file is created, space is allocated to it on the disc, and a directory entry is made. The file is not opened when it is created.

Regardless of the file size, the first 256 bytes of a BASIC/DATA file is set aside to store file management information, and is unavailable for data storage. Minimum file size is one block—1024 bytes. Files are created in integer number of blocks, and additional logical records of the specified record size are added, if necessary, to fill the file. For example, `CREATE "file", 50,30` creates a 2-block file containing 59, 30-byte records and 256 bytes of ovehead.

Related Keywords

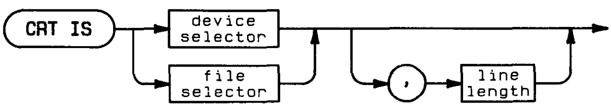
`ASSIGN#`

CRT IS

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The CRT IS statement selects the destination device for output ordinarily displayed.

C



Item	Description	Range
device selector	numeric expression, rounded to an integer (see glossary)	—
file selector	numeric expresison, rounded to an integer	11 through 20
line length	numeric expression, rounded to an integer (default=80)	1 through 220

Examples

```
CRT IS 1
CRT IS P1
```

Description

Error messages and output from DISP (USING), LIST, and CAT, are sent to the CRT IS device or to the specified file.

The line length specifies the maximum number of characters sent to the CRT IS device before an end-of-line (EOL) sequence is sent. EOL character(s) are not counted as part of the line length. When a DISP USING format string specifies output that exceeds the CRT IS line length, the line is broken at the line length and the format is continued at the beginning of the next line.

Related Keywords

ASSIGN, DISP, IMAGE

CSC

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `CSC` function returns the cosecant of the angle argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Cosecant=CSC(Angle)
DISP CSC(Theta)
```

Description

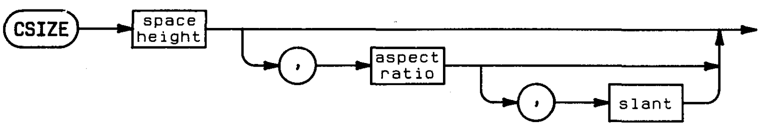
The angle argument is interpreted according to the current trigonometric mode—RAD (radians), DEG (degrees), or GRAD (grads). The default mode is RAD.

Related Keywords

`DEG, GRAD, RAD`

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `CSIZE` statement specifies the height, aspect ratio (width/height), and slant of label characters.



Item	Description	Range
space height	numeric expression, interpreted in graphics units (default=3 GUs)	—
aspect ratio	numeric expression (default=.6 for pen plotters; machine dependent for the display)	—
character slant	numeric expression, interpreted according to the current trigonometric mode (default=0)	$-\pi/2$ < character slant < $\pi/2$ (RAD mode) -90 < character slant < +90 (DEG mode) -100 < character slant < +100 (GRAD mode)

Examples

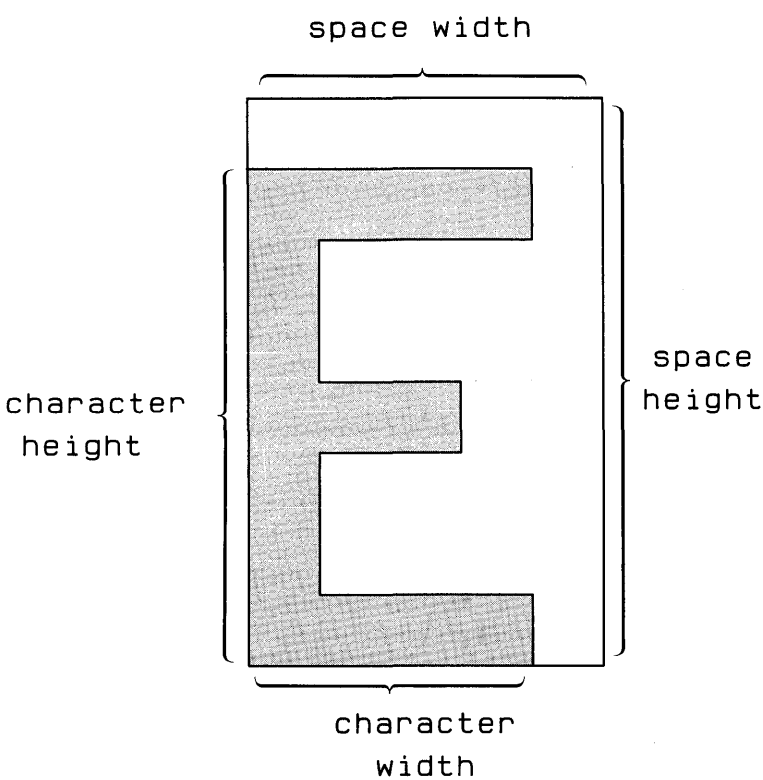
```
CSIZE 12
CSIZE 12,,8
CSIZE Height,Shape,Slant
```

Description

The space height parameter is the height, in graphics units, of the character space (see glossary). The aspect ratio is the ratio of the width of the character to its height.

The slant parameter specifies, in the current trigonometric mode, the clockwise slant of the character from vertical. If the slant parameter is out of range, the character slant defaults to 0.

The following diagram and table describes how pen plotters position characters in the character space.*



* Character dimensions on the graphics display are machine-dependent.

CSIZE Character Dimensions

Character Dimension	Description
Space height	CSIZE space height parameter
Symbol height	½ × the space height
Space width	¾ × aspect ratio parameter × height parameter
Symbol width	⅔ × space width

Labels can be reflected by changing the sign of the CSIZE parameters:

Reflecting Labels

Sign of Height	Sign of Aspect Ratio	Effect
positive	positive	unreflected
positive	negative	reflected across y-axis
negative	negative	reflected across x-axis
negative	positive	reflected across origin

Related Keywords

DEG, GRAD, RAD

CURSCOL

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

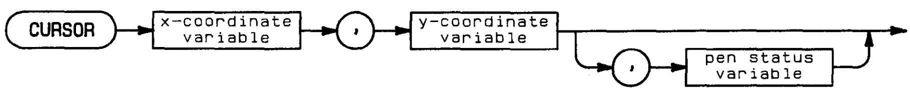
The `CURSCOL` function returns the current column location of the cursor in alpha display memory.

Related Keywords

`CURSROW`

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `CURSOR` statement reads the current location and status of the physical pen from the plotting device and assigns those values to numeric variables.



Item	Description	Range
x-coordinate variable	simple numeric variable or array element	any valid name
y-coordinate variable	simple numeric variable or array element	any valid name
pen status variable	simple numeric variable or array element	any valid name

Examples

```
CURSOR Xposition, YPosition, Penstatus
CURSOR x(I), y(I)
```

Description

The pen x- and y- coordinates are interpreted according to the current units. The pen status variable is assigned the value 0 if the pen is up, 1 if the pen is down.

Related Keywords

`DIGITIZE`, `WHERE`

CURSROW

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `CURSROW` function returns the current row location of the cursor in alpha display memory.

Description

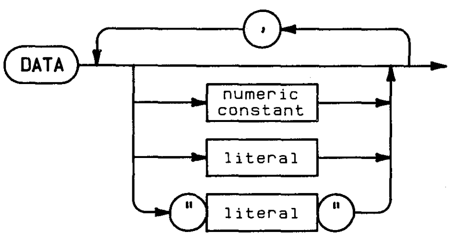
The row number returned by `CURSROW` corresponds to the cursor position on the screen when row 1 of display memory is at the top of the screen.

Related Keywords

`CURSCOL`

Keyboard Executable	☐
Programmable	■
In an IF...THEN	☐

The DATA statement contains numeric and/or string data which is assigned to program variables listed in one or more READ statements. (For information about using DATA as a secondary keyword, see **SEND**.)



Item	Description	Range
numeric constant	numeric quantity consisting of digits 0 through 9 with optional decimal point, sign, and exponential notation	—
literal	string constant consisting of characters entered from the keyboard	—

Examples

```
DATA 2,4,6,8
DATA ABC,2.5E20,DEF,3,"   leading spaces"
```

Description

A program can contain any number of DATA statements. The statement is declaratory, and extra data is ignored if there are no corresponding READ variables. A *data pointer* is used to access data items. A (sub)program's READ operations start with the first item in the lowest numbered DATA statement. When all data items in a DATA statement have been read, the pointer moves to the next-higher numbered DATA statement.

When a READ statement accesses a DATA statement for a numeric variable assignment, the data constant must be a numeric value. When the READ statement is assigning a value to a string variable, the DATA statement can contain a numeric value, an unquoted string, or a quoted string; a numeric value is interpreted as a literal containing digits. Quotation marks are regarded as string delimiters, and are not part of the string. Strings delimited by quotation marks, however, can contain commas and leading and trailing blanks.

Quotation marks around literals are optional and are not part of the assignment; the quotation marks make it possible to include leading and trailing blanks in literals.

If the keyword is not followed by a numeric constant or literal, the statement is interpreted as DATA "" (null string).

Subprograms maintain their own data pointers. When a subprogram is being executed, READ statements access DATA statements within the subprogram, starting with the lowest numbered DATA statement in the subprogram. When program execution returns to the calling program, READ operations resume where they left off when the subprogram was called.

DATA statements cannot be included in multistatement lines. Comments (using the comment delimiter !) cannot be added to DATA statements.

Related Keywords

INPUT, READ, RESTORE

DATE

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `DATE` function returns the current value of the system clock date counter.

Description

The date counter is in the form YYDDD where YY is the year and DDD is the day number in the range 1 through 366.

Related Keywords

`DATE$`, `TIME`

DATE$

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The DATE$ function returns the current value of the system clock date counter in YY/MM/DD format.

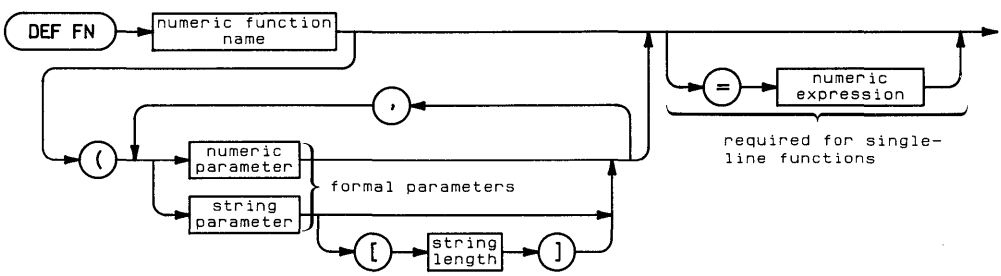
Related Keywords

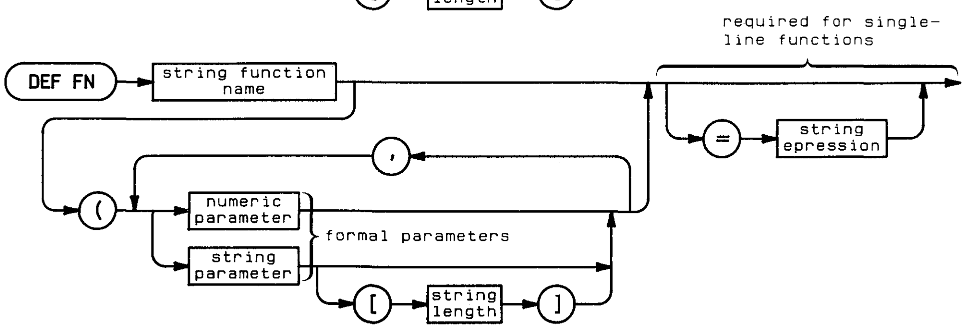
DATE, TIME

DEF FN

Keyboard Executable	☐
Programmable	■
In an IF...THEN	☐

The DEF FN statement defines a single-line user-defined function and its formal parameters. For multiple-line functions, DEF FN defines the beginning of the function and the formal parameters used within the function.





D

Item	Description	Range
numeric function name	name of the user-defined function	any valid numeric variable name
numeric parameter	numeric variable name	subscripted variables not allowed
string parameter	string variable name	subscripted variables not allowed
numeric expression	(see glossary)	—
string function name	name of the user-defined function	any valid string variable name
string expression	(see glossary)	—
string length	numeric expression, rounded to an integer	1 through 65,530

Examples

```
DEF FNCube(Number)=Number^3
DISP FNCube(Side)

DEF FNSlash$(String$[30])
  FOR I=1 TO 30
  IF String$[I,I]="/" THEN String$[I,I]=";"
  NEXT I
  FNSlash$=String$[1,18]
FN END
PRINT# 1, A(1); FNSlash$(B$)
```

Description

A maximum of 30 parameters can be passed into the function. The formal parameters listed in the `DEF FN` statement must match the actual parameters listed in the calling `FN` statement in type—numeric versus string. The actual parameters are passed into the user-defined function by value; any changes made to parameters within the user-defined function are local to the function and not available to the rest of the program. However, all program variables (except those whose names are the same as formal function parameters) are available in the user-defined function.

Function definitions are local to the program or subprogram in which they are located.

If a string parameter passed into a function is longer than 18 characters, it must be dimensioned within the function `DEF FN` statement. When a string user-defined function passes a string expression back to the program, that expression can be no longer than 18 characters.

User-defined functions must not be recursive. `DEF FN` cannot be included in a multistatement line.

Single-Line Functions. `DEF FN` is a declaratory statement; it is ignored if the function is not referenced. Single-line functions must include the function assignment (= *numeric expression* or = *string expression*).

ON ERROR branching should be disabled before execution of a single-line function. Otherwise, an error could cause premature exit from the function.

Multiple-Line Functions. The `DEF FN` statement defines the beginning of the function; `FN END` defines the end. An `FN`...= statement within the function defines the value passed back to the program. Branching statements should not be used to exit the function.

The block of statements defining the function can be placed anywhere within the program, except that a function cannot be nested within another function.

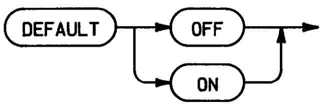
Related Keywords

`FN`

DEFAULT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The DEFAULT statement specifies how warnings are handled by the system.



D

Examples

```
DEFAULT OFF
IF Angle=0 THEN DEFAULT ON
```

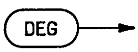
Description

With default on, warnings generate a message and, if relevant, a default value. With default off, warnings generate a message and halt execution. The power-on condition is default on.

DEG

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `DEG` statement sets degrees as the unit in which angles are measured.



Description

When `DEG` is executed, all angle parameters in statements and functions are interpreted as degrees. (There are 360 degrees in a circle.) All functions returning an angle return a value in degrees.

The angle mode of a program is global. When a subprogram is called, the current angle mode is carried into the subprogram. If a subprogram changes the angle mode and then returns to the main program, the new mode is carried back to the main program.

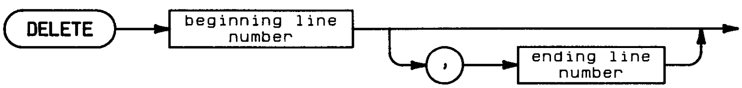
Related Keywords

`GRAD`, `RAD`

DELETE

Keyboard Executable ■
Programmable □
In an IF...THEN □

The DELETE command deletes program lines from the current program or subprogram in memory.



Item	Description	Range
beginning line number	integer constant identifying a program line	1 through 65,535
ending line number	integer constant identifying a program line	1 through 65,535

Examples

```
DELETE 30
DELETE 30,90
```

Description

Specifying only the beginning line number deletes that line. Specifying both parameters deletes all lines within that range.

When both a main program and one or more subprograms are present in memory, DELETE acts upon the program specified by the previous FINDPROG statement.

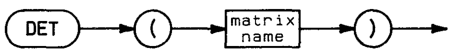
Related Keywords

FINDPROG, SCRATCH, SCRATCHSUB

DET

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `DET` function returns the determinant of the specified matrix.



Item	Description	Range
matrix name	name of a two-dimensional numeric array	any valid name

Examples

```
Denominator=DET(Matrix1)
IF DET(A)=0 THEN 300
```

Description

The specified matrix must be a square matrix. (The number of rows must equal the number of columns.)

Related Keywords

`DETL`

DETL

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The DETL function returns the determinant of the last matrix inverted in a MAT...INV statement, or the determinant of the cofficient matrix (first argument in parentheses) in the most recently executed MAT...SYS statement.

(DETL)→

D

Examples

```
A=DETL
IF DETL=0 THEN 400
```

Description

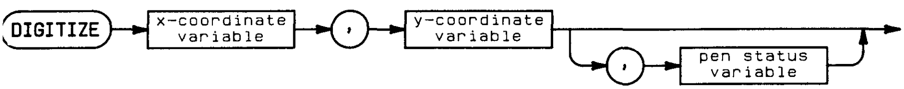
The matrix whose determinant is returned must have been previously specified in a MAT...INV statement or a MAT...SYS. The most recently executed statement is used.

Related Keywords

DET, MAT...INV, MAT...SYS

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `DIGITIZE` statement inputs the physical pen position and status from the current plotting device and assigns those values to the specified variables.



D

Item	Description	Range
x-coordinate variable	simple numeric variable or array element	any valid name
y-coordinate variable	simple numeric variable or array element	any valid name
pen status variable	simple numeric variable or array element	any valid name

Examples

```
DIGITIZE Xposition, YPosition, Penstatus
DIGITIZE x(I), y(I)
```

Description

The pen x- and y-coordinates are interpreted according to the current units. The pen status variable is assigned the value 0 if the pen is up, 1 if the pen is down.

When `DIGITIZE` is executed, program execution is suspended until the pen coordinates (and optional status) are entered from the plotting device.

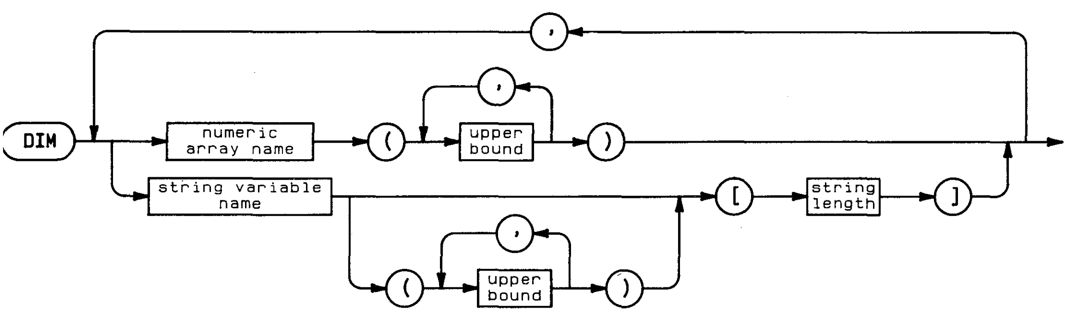
Digitizing is aborted by pausing the program.

Related Keywords

`CURSOR`, `WHERE`

Keyboard Executable	■
Programmable	■
In an IF...THEN	□

The DIM statement allocates memory for REAL numeric arrays, string variables, and string arrays.



Item	Description	Range
numeric array name	name of a numeric array	any valid name
upper bound	integer constant	1 through 65,530
string variable name	name of a simple or array string variable	any valid name
string length	integer constant	1 through 65,530

Examples

```
DIM A(300), B(2,50), C$[20]
DIM D$(25)[30], E$(3,3)[3]
```

Description

One- and two-dimensional arrays are allowed.

If an array is to be explicitly dimensioned, the dimensioning statement must be executed before any of the elements of the array are referenced. If an element is referenced before the array is explicitly dimensioned, an array is implicitly dimensioned with upper bound(s) equal to 10. If a string variable is referenced before the string length is dimensioned, the string length is implicitly dimensioned to 18.

A variable can be dimensioned only once within a program; an attempt to dimension a variable that has already been explicitly or implicitly dimensioned causes an error.

A program can contain any number of `DIM` statements. If the program contains an `OPTION BASE` statement, dimensioning statements must occur after the option base has been declared.

The dimension(s) of a variable are global, known to the program and any subprograms to which the variable is passed.

Related Keywords

`INTEGER`, `REAL`, `SHORT`

DIRECTORY

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The DIRECTORY statement displays a directory of the main program and the subprograms currently in system memory.

Description

The directory lists the subprograms in their order in system memory, along with the deallocated size, the number of lines, and the allocation status of each subprogram. An arrow (>) indicates the current (sub)program. The subprogram names listed in the directory are the names with which the subporgrams were initially created (using FINDPROG) and stored.

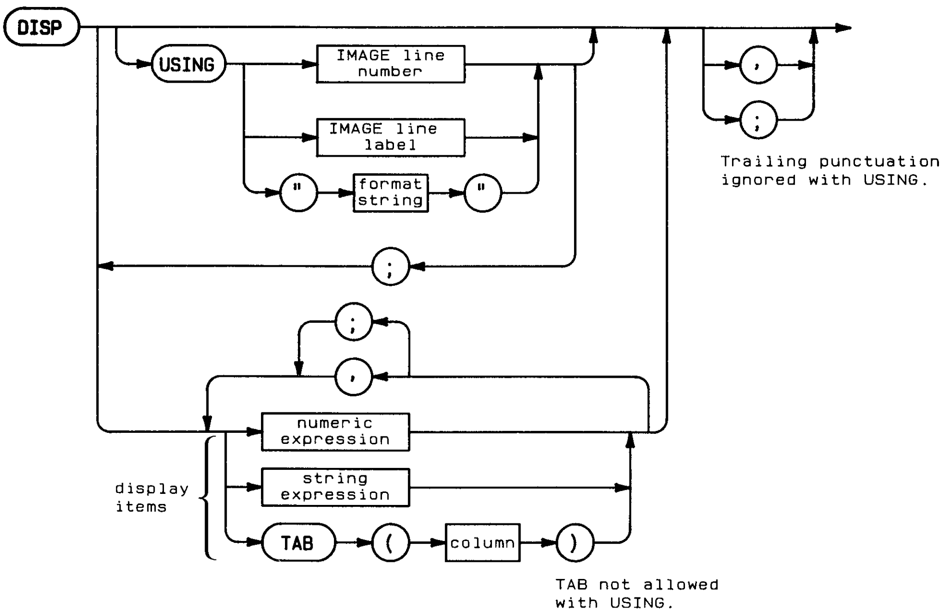
Related Keywords

FINDPROG, SCRATCHSUB

DISP

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The DISP statement outputs the display items to the current display line.



Item	Description	Range
IMAGE line number	integer constant identifying an IMAGE statement	1 through 65,535
IMAGE line label	name identifying an IMAGE statement	any valid line name
format string	string expression containing one or more field specifiers (see IMAGE statement for syntax)	—
column	numeric expression, rounded to an integer	−99,999 through 99,999; negative numbers are interpreted as 1
numeric expression	—	—
string expression	—	—

D

Examples

```
DISP Number; Letter$
DISP TAB(10);A$,"Results=";Result
DISP USING "DC3D.5D,4X,7A";A,"dollars"
DISP USING 100; A,B$,C
```

Description

The keyword USING provides for specifying the format of output. When DISP is executed without USING, a standard format is used.

Simple DISP (Without USING). Simple DISP uses standard number format (see glossary) for numeric items, and displays numeric and string items in either of two field widths:

- When display items are separated by semicolons, they are displayed in *narrow* format with a leading blank or minus sign. Strings are output with no leading or trailing blanks.
- When display items are separated by commas, they are displayed in *wide* format, left-justified in 21-column fields. Items longer than 21 characters occupy more than one field.

When the TAB function is included as a display item, the cursor moves to the designated column. Negative column numbers are treated as TAB(1). Column numbers greater than the line length are reduced MOD (line length). When TAB is used to control format, display items should be separated by semicolons; using commas causes output to be displayed in wide format.

When the list of display items is exhausted, an end-of-Line (EOL) sequence, ordinarily carriage return/line feed, is sent to the display. The EOL can be suppressed by including a comma or semicolon after the last display item.

D

Control Characters. Control characters can be included as display items by specifying their ASCII code as argument in the CHR$ function or by using the metacharacter ~ followed by the character decimal code.

Formatted Output. DISP USING uses a format string contained in the statement itself or in a referenced IMAGE statement to format the output. The format string, consisting of one or more field specifiers separated by delimiters (comma or slash), is used from left to right. Display items are paired with their corresponding field specifiers. Certain field specifiers do not use a display item (for example, X).

If the format string is exhausted before all the display items have been processed, the format string is reused from the beginning. Extra field specifiers are ignored. If a field is larger than the numeric item, the number is right-justified in the field. A warning is issued if the number is larger than the field. (A minus sign requires a digit position if M or S is not included in the field specifier.) Numbers are rounded to the number of decimal placed indicated by the field specifier. Standard number format can be chosen by using the image specifier K.

The TAB function cannot be used with DISP USING.

When the list of display items is exhausted, an end-of-line (EOL) sequence, ordinarily carriage return/line feed, is sent to the display. The EOL can be suppressed by placing the image specifier # at the beginning the format string. Unlike with simple DISP, a terminating semicolon or comma is ignored and does not suppress the EOL sequence.

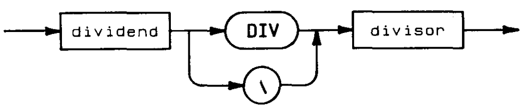
Refer to IMAGE for the syntax of the format string.

Related Keywords

IMAGE, OUTPUT, PRINT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The DIV operator returns the integer portion of the quotient resulting from a division operation. The DIV operation can also be indicated by the symbol \.



Item	Description	Range
dividend	numeric expression	—
divisor	numeric expression	≠0

Examples

```
C=A DIV B
DISP (A+B)\C
```

Description

A DIV B is equivalent to the expression IP(A/B).

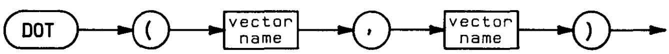
Related Keywords

MOD

DOT

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `DOT` function returns the dot product of two vectors.



Item	Description	Range
vector name	name of a one-dimensional numeric array	any valid name

Examples

```
DISP DOT(A,B)
IF DOT(C,D)=0 THEN 600
```

Description

The dot product (scalar product) of two vectors is computed by summing the products of the corresponding elements of the two vectors. The two vectors must be the same size.

Related Keywords

`MAT`

DRAW

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The DRAW statement lowers the pen and moves it to the specified x-, y-coordinate position. The pen remains down until it is raised by another statement.

Item	Description	Range
x-coordinate	numeric expression, interpreted in the current units	—
y-coordinate	numeric expression, interpreted in the current units	—

Examples

```
DRAW 10,10
DRAW XPosition,XPosition*5
```

Description

DRAW uses the current units mode (UU's or GU's) and line type. In UU's mode, lines cannot be drawn outside the plotting boundaries. In GU's mode, the plotting boundaries become equivalent to the graphics limits; therefore, lines can be drawn anywhere within the graphics limits.

In both UU's mode and GU's mode, the logical pen can be moved outside the plotting area. However, the physical pen cannot be moved beyond the plotting boundaries.

Related Keywords

IDRAW, LINE TYPE, MOVE, PLOT

DTB$

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The DTB$ (*decimal-to-binary*) function returns a string containing the base 2 representation of the decimal argument.

Item	Description	Range
numeric argument	numeric expression, truncated to an integer	—

Examples

```
A$=DTB$(45)
DISP DTB$(X(1)/X(2))
```

Related Keywords

BTD, DTH$, DTO$, HTD, OTD

DTH$

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The `DTH$` (*decimal-to-hexadecimal*) function returns a string containing the base 16 representation of the decimal argument.

Item	Description	Range
numeric argument	numeric expression, truncated to an integer	—

Examples

```
DISP DTH$(5700)
IF DTH$(I(5))="A4" THEN J=12
```

Related Keywords

BTD, DTB$, DTO$, HTD, OTD

DTO$

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `DTO$` *(decimal-to-octal)* function returns a string containing the base 8 representation of the decimal argument.

Item	Description	Range
numeric argument	numeric expression, truncated to an integer	—

Examples

```
Y$=DTO$(A(1))
DISP DTO$(512+X)
```

Related Keywords

`BTD`, `DTB$`, `DTH$`, `HTD`, `OTD`

DTR

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `DTR` (*degrees-to-radians*) function interprets the numeric argument as an angle measured in degrees, and returns the value of the angle in radians.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Radians=DTR(Degrees)
DISP DTR(90)
```

Description

The argument and value returned by `DTR` are independent of the current trigonometric mode.

Related Keywords

`RTD`

DUMP ALPHA

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `DUMP ALPHA` statement copies the contents of the alpha display to the PRINTER IS device.

Related Keywords

`DUMP GRAPHICS`

DUMP GRAPHICS

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The DUMP GRAPHICS statement copies the contents of the graphics display to the internal printer.

DUMP GRAPHICS →

D

Description

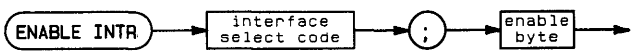
The internal printer (located at interface select code 2) must be the current PRINTER IS device. The display must be the current PLOTTER IS device.

The contents of the graphics display is copied dot-by-dot.

Related Keywords

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ENABLE INTR` statement enables the specified interface for generating an interrupt which can cause end-of-line branching.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
enable byte	numeric expression, truncated to an integer and moduloed 256	—

Examples

```
ENABLE INTR 7; 8
ENABLE INTR Isc; BTD("101")
```

Description

The binary representation of the enable byte is written into Control Register 1 of the specified interface. The significance of each bit is interface-dependent.

The end-of-line branch taken when an interrupt is generated is specified by an `ON INTR` statement.

Executing `ENABLE INTR` is equivalent to executing a `CONTROL` statement to Control Register 1.

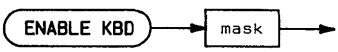
Related Keywords

`CONTROL, ON INTR, STATUS`

ENABLE KBD

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ENABLE KBD` statement enables and disables various keyboard keys during program execution and/or keyboard input (`INPUT` and `LINPUT`).



Item	Description	Range
mask	numeric expression, rounded to an integer	0 through 255

Examples

```
ENABLE KBD 16
ENABLE KBD KeyMask
```

Description

The binary equivalent of the decimal mask is used to activate (enable) and deactivate (disable) various portions of the keyboard. The keyboard is divided into four areas:

- The (RESET)* key.
- The (PAUSE)* key.
- The special function keys.
- All other keys.

* Machine-dependent key or control sequence.

Keys can be activated and deactivated separately for program execution (while the program is running) and keyboard input (while the program is halted at an INPUT or LINPUT statement). Setting a bit (1) activates the key(s); clearing a bit (0) deactivates the key(s).

activated

Bit Number	Decimal Value	Operating Mode	Key(s) Affected
7	128	program execution	RESET *
6	64	program execution	PAUSE *
5	32	program execution	special function keys
4	16	program execution	all other keys
3	8	keyboard input	RESET *
2	4	keyboard input	PAUSE *
1	2	keyboard input	special function keys ~~and all other keys~~
0	1	keyboard input	~~special function keys and~~ all other keys

* Machine-dependent key or control sequence.

reset KP
pause

7 8

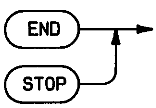
Related Keywords

INPUT, LINPUT, ON KEY#, ON KYBD

END

Keyboard Executable □
Programmable ■
In an `IF...THEN` ■

The `END` or `STOP` statement is the last statement executed by a program.



Description

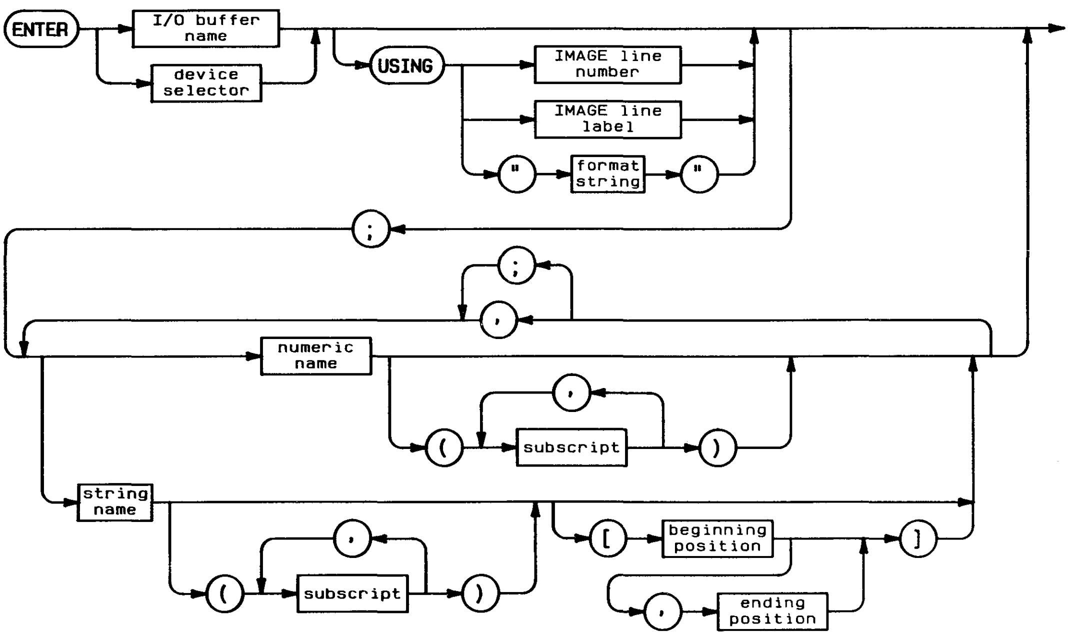
`END` and `STOP` are interchangeable. The statements are optional and can appear anywhere in the program. More than one `END` and/or `STOP` statements are allowed.

Related Keywords

`STOP`

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ENTER statement inputs bytes of data from a device or buffer and assigns the data to the specified numeric and/or string variables.



Item	Description	Range
device selector	numeric exprssion, rounded to an integer (see glossary)	—
I/O buffer name	name of a string variable declared as an I/O buffer	—
IMAGE line label	name identifying an IMAGE statement	any valid line label
IMAGE line number	integer constant identifying an IMAGE statement	1 through 65,535
format string	string expression consisting of one or more field specifiers (see page 2-126 for syntax)	—
numeric name	name of a numeric variable	any valid name
string name	name of a string variable	any valid name
subscript	numeric expression, rounded to an integer	1 through 65,530
beginning position	numeric expression, rounded to an integer	1 through 65,530
ending position	numeric expression, rounded to an integer	1 through 65,530

E

Examples

```
ENTER 701 USING Enterformat; A, B$, C
ENTER 5; Var1, Var2, Var3
```

Table of Image Specifiers and Delimiters For ENTER

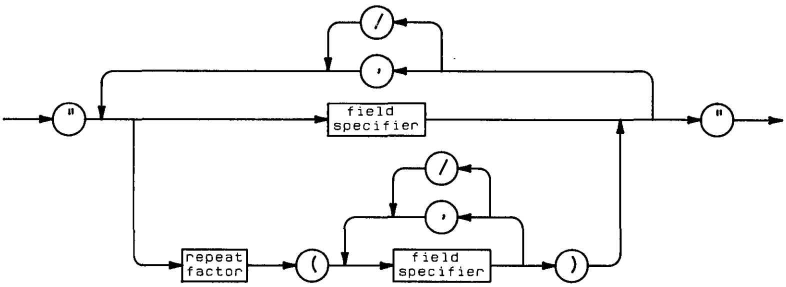
Image Specifier	Meaning
X	Directs the computer to skip one character.
D, Z, *, ., S, M	All six specifiers accept one character to be used in building a numeric variable. The characters may be the digits 0 through 9, the decimal point, and signs. The six different specifiers are provided for documentation purposes and for compatibility between OUTPUT and ENTER format strings.
K	The number or string is input in free field format (see glossary).
A	Inputs one string character.
B	Inputs one byte of binary data and enters its decimal equivalent into a numeric variable.
W	Inputs two bytes of binary data to be used in building a 16-bit, 2's complement binary word. The first byte entered is the most significant. The decimal equivalent of the resulting word is entered into a numeric variable.
C	Accepts one character for building a numeric variable, and provides for ignoring all commas while the number is being entered. (Without this specifier, a comma ends the entry of the number.)
E	Inputs an exponent consisting of the letter E, a sign, and three digits.
e	Inputs an exponent consisting of the letter E, a sign, and two digits.
/	Causes computer to skip to the beginning of a new field. The new field is indicated by a line feed.
#	When used as a statement terminator specifier, eliminates the requirement for a line feed to terminate the ENTER statement; the ENTER statement terminates as soon as the last variable in the statement has been satisfied. When used as a field terminator specifier, eliminates the line feed as a terminating condition during free-field entry; line feeds entered are placed in the string.
%	When used as a statement terminator specifier, allows EOI or line feed as terminating condition. When used as a field terminator specifier, allows EOI as an additional terminating condition.

E

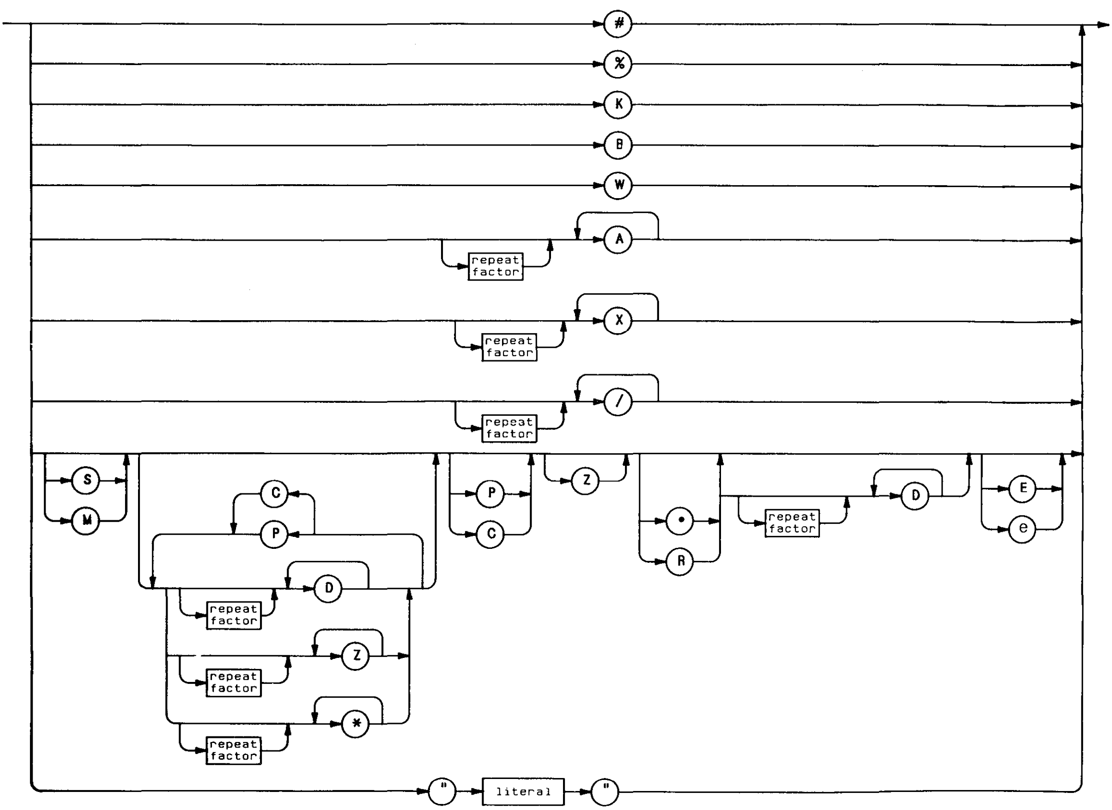
Related Keywords

CONVERT, IMAGE, IOBUFFER

format string



field specifier



EPS

The `EPS` function returns machine epsilon, the smallest positive REAL number.

Examples

```
DISP EPS
```

Related Keywords

`INF`

ERRL

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ERRL function returns the line number of the program line generating the most recent error or warning.

Examples

```
PRINT ERRL
IF ERRL=200 THEN GOSUB 700
```


Related Keywords

ERRM, ERRN, ERROM, ERRSC, ON ERROR

ERRM

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ERRM statement displays the error message generated by the most recent error.

Description

If no error has occurred since power on, reset, SCRATCH, LOAD, or GET, the system displays `Error 0 : 0`.

ERRM is useful as part of an ON ERROR recovery routine, where no error message would otherwise be displayed.

Related Keywords

ERRL, ERRN, ERROM, ERRSC, ON ERROR

ERRN

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ERRN` function returns the error number of the most recent error or warning.

Examples

```
DISP ERRN
IF ERRN=49 THEN GOSUB Assignment
```


Description

If no error has occurred, `ERRN` returns 0.

Related Keywords

`ERRL`, `ERRM`, `ERROM`, `ERRSC`, `ON ERROR`

ERROM

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ERROM` function returns a number indicating which BASIC module returned the most recent error or warning.

Examples

```
DISP ERROM
IF ERRN=113 AND ERROM=232 THEN 400
```


Description

`ERROM` is used to distinguish between two or more errors having the same error number but originating from different BASIC modules.

Related Keywords

`ERRL`, `ERRM`, `ERRN`, `ERRSC`, `ON ERROR`

ERRSC

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `ERRSC` function returns the interface select code of the interface that generated the most recent interface-dependent error.

Examples

```
IF ERRSC=7 THEN STATUS 7,1;C
DISP ERRSC
```


Description

When an interface error occurs, `ERRSC` returns the interface select code of that interface until another I/O error occurs at another interface. If no interface error has occurred, `ERRSC` returns 0.

Related Keywords

`ERRL`, `ERRM`, `ERRN`, `ERROM`, `ON ERROR`

EXOR

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `EXOR` operator returns a 1 or 0 based on the logical exclusive-OR of the operands.

Item	Description	Range
operand	numeric expression	—

Examples

```
T=A(1) EXOR A(2)
IF You EXOR Cize THEN YouHealthy
```

Description

A non-zero operand (positive or negative) is interpreted as a logical 1. An operand of zero is interpreted as a logical 0. The following table describes the results of performing an `EXOR` operation.

Exclusive OR

A	B	A EXOR B
0	0	0
0	1	1
1	0	1
1	1	0

Related Keywords

`AND`, `NOT`, `OR`

EXP

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `EXP` numeric function returns the natural (base *e*) antilogarithm by raising *e* to the power of the argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

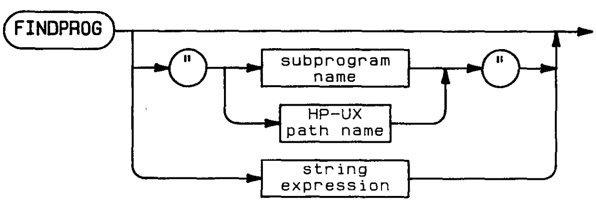
```
K=A*EXP(-E/RT)
PRINT A;EXP(A)
```

Related Keywords

`LOG`

Keyboard Executable ■
Programmable ■
In an IF...THEN □

The FINDPROG statement locates (and retrieves, if necessary) the specified subprogram in system memory or mass storage. When FINDPROG is executed from the keyboard, a system pointer is positioned at the subprogram so that it can be listed and edited.



Item	Description	Range
subprogram name	literal	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

F

Examples

```
FINDPROG
FINDPROG "SubSort"
FINDPROG FileName$&"2"
```

Description

When FINDPROG is executed without a parameter, the pointer is moved to the main program.

FINDPROG first searches computer memory for the specified subprogram. If the subprogram is not found, the current working directory or specified mass storage location is searched. The HP-UX path name must be used if the subprogram is not located in computer memory or in the current working directory. If the subprogram is found in mass storage, it is brought into system memory.

If the specified subprogram is not found in system memory or in mass storage, the pointer is moved to a new block of system memory. The system displays NEW PROGRAM, indicating that a new subprogram can now be entered from the keyboard without overwriting other programs currently in memory. The FINDPROG name must be used when the new subprogram is stored.

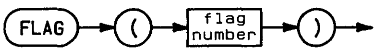
Related Keywords

CALL, DIRECTORY, STORE

FLAG

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The FLAG function returns the status of the specified flag—set (1) or clear (0).



Item	Description	Range
flag number	numeric expression, rounded to an integer	+1 through +64

Examples

```
IF FLAG(1) THEN 200
IF FLAG(A)=FLAG(B) THEN GOSUB 1000
```


Related Keywords

CFLAG, FLAG$, SFLAG

FLAG$

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The FLAG$ function returns an eight-character string whose binary representation shows the status of the 64 flags.

Examples

```
DISP FLAG$
IF FLAG$="H2a?"&CHR$(12)&"1fM" THEN GOTO 400
```

Description

The left-most (most significant) bit of the left-most character represents the status of flag 1.

When the FLAG$ string is displayed, executable control characters are interpreted. Non-executable control characters are ignored.

Related Keywords

CFLAG, FLAG

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `FLIP` statement causes the keyboard to toggle between typewriter mode operation and BASIC mode operation.*

```
(FLIP)──►
```

Description

In typewriter mode, the keyboard produces unshifted lowercase letters and shifted uppercase letters. In BASIC mode, the keyboard produces unshifted uppercase and shifted lowercase letters. Only letter keys are affected. The default condition is typewriter mode.

When `ON KYBD` branching has been enabled for letter keys, the branch is taken only when the typed character in the current keyboard mode matches a character in the `ON KYBD` string expression.

Related Keywords

`ON KYBD`

* Implementation of FLIP is machine-dependent.

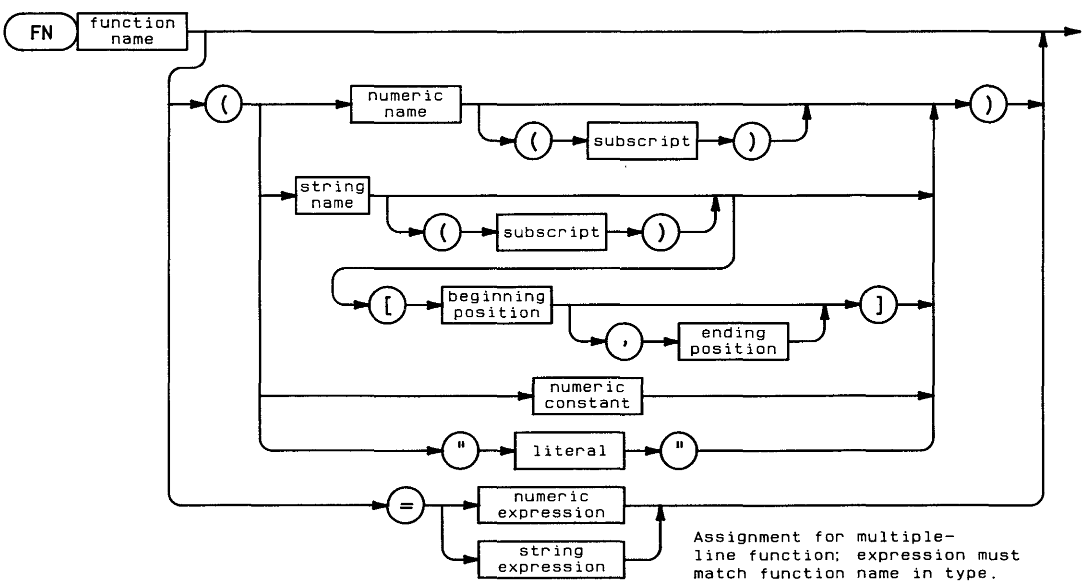
FLOOR

See **INT**

Keyboard Executable □
Programmable ■
In an IF...THEN ■

The FN keyword is a prefix used before the name of a user-defined function to identify a call to the function. Optional parameters in parentheses are passed to the function. The function returns a value used by the expression containing the function call.

FN...= is used within a multiple-line, user-defined function to assign a value to the function. FN END defines the end of multiple-line functions.



Item	Description	Range
numeric name	name of a simple numeric variable or numeric array	any valid name
subscript	numeric expression, rounded to an integer	1 through 65,530
beginning position	numeric expression, rounded to an integer	1 through 65,530
ending position	numeric expression, rounded to an integer	1 through 65,530
literal	string constant	—
numeric constant	a numeric expression that can contain digits 0 through 9, plus or minus sign, a decimal point, and exponential notation	—
string name	name of a simple string variable or string array	any valid name
numeric or string expression	(see glossary)	—

Examples

```
Y=FNInverse /A
FNMultilineFunction$=A$ & "***"
```

F

Description

When FN invokes a user-defined function, the function type (numeric versus string) must match the context of the expression invoking the function. For example, the value returned by a string function cannot be assigned to a numeric variable.

The parameters passed into a user-defined function by FN must match the DEF FN parameter list in number and type (numeric versus string). The parameters are passed by value; any changes made to the value of program variables within a user-defined function are *not* carried back to the program. Numeric and string variables, elements of numeric and string arrays, and substrings can be passed to a function.

The FN...= statement must appear somewhere within a multiple-line function to assign the function a value which is returned to the program.

Recursive user-defined functions are not allowed; a function cannot invoke itself.

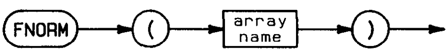
Related Keywords

DEF FN

FNORM

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `FNORM` function returns a value computed by squaring each element of the specified array, summing the squares, and then taking the square root of the sum.



Item	Description	Range
array name	name of a one- or two-dimensional numeric array	any valid name

Examples

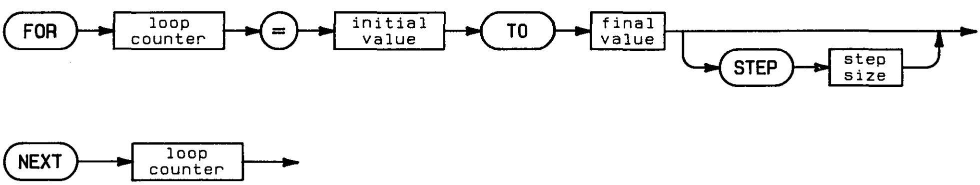
```
M=FNORM(Array3)
```


Related Keywords

`CNORM`, `RNORM`

Keyboard Executable ■
Programmable ■
In an IF...THEN □

The FOR and NEXT statements together define a program loop that is repeated until a loop counter passes a specified value.



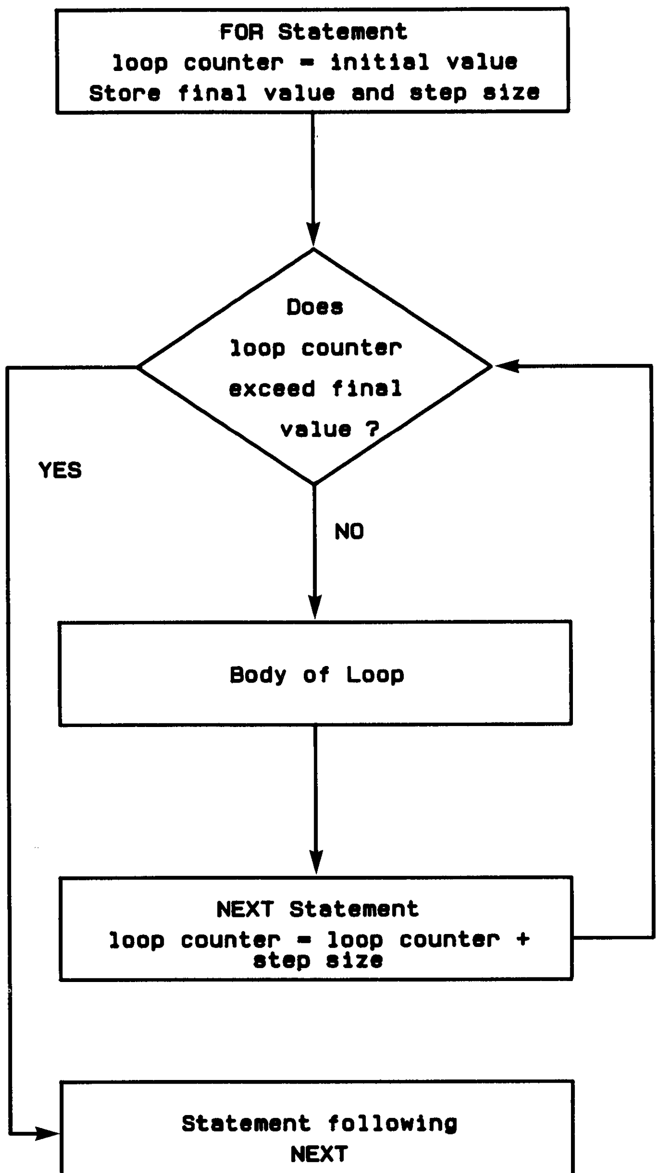
Item	Description	Range
loop counter	simple numeric variable name	cannot be an array variable
initial value	numeric expression	—
final value	numeric expression	—
step size	numeric expression (default=1)	—

F

Examples

```
100 FOR Counter=1 TO 100
110     DISP Counter
120 NEXT Counter

200 FOR I=N TO N+M STEP stepsize
220     A(I)= .592*ABS(I^3)
230     IF A(I)>X THEN 400
240     PRINT I, A(I)
250 NEXT I
```


FOR Statement
loop counter = initial value
Store final value and step size
Does
loop counter
exceed final
value ?
YES
NO
Body of Loop
NEXT Statement
loop counter = loop counter +
step size
Statement following
NEXT

Description

The `FOR` statement defines the beginning of the loop, sets the loop counter equal to the specified value, and stores the final value and step size. Each time the `NEXT` statement is executed, the loop counter is incremented (or decremented, in the case of a negative step value) by the step value and then compared to the final value. If the final value has not been passed, program execution is transferred to the statement immediately following the `FOR` statement. If the final value has been passed, program execution continues with the line immediately following the `NEXT` statement. (The loop counter is not equal to the final value when the loop has been exited.)

Because the loop counter is tested immediately after the `FOR` statement is executed (see flowchart), the loop is not executed at all if the loop counter initial value is already past the final value. For example, a loop beginning with the statement `FOR I=3 TO 5 STEP -.3` will not be executed, since 3 is already past (less than) the final value 5.

The loop can be exited by unconditional or conditional branching; the loop counter retains is current value. The loop may be re-entered in the body of the loop or at the `FOR` statement. Entering a loop at the `FOR` statement reinitializes the loop counter.

The `FOR` statement stores the loop counter, final value, and step size, and these values remain unchanged for the loop until the `FOR` statement is executed again. When the loop counter, final value, and step size are numeric expressions containing variables, the values of those variables can be changed within the loop without affecting how many times the loop is executed. However, changing the value of the loop counter within the loop can affect how many times the loop is executed. The loop counter can be used in expressions defining the initial value, final value, and step size.

Each `FOR` statement must have one, and only one, matching `NEXT` statement. When `FOR...NEXT` loops are *nested*, one loop must be contained entirely within another.

FP

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `FP` function returns the fractional part of the numeric argument. The function returns a value greater than −1 and less than +1. A negative argument returns a negative value.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Y=FP(X+Q)
IF FP(X)=0 THEN DISP "X IS AN INTEGER"
```

Related Keywords

`IP`

FRAME

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The FRAME statement draws a frame around the plotting area using the current line type and pen number.

Description

After the frame is drawn, the pen is positioned at the lower left corner of the frame and the pen is up.

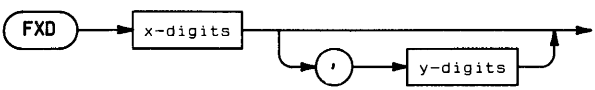
Related Keywords

CLIP, LINE TYPE, LOCATE

FXD

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `FXD` statement specifies the number of digits to the right of the decimal point in labels plotted by `LAXES` and `LGRID`.



Item	Description	Range
x-digits	numeric expression, rounded to an integer	parameters outside the range 0 through 7 are interpreted as FXD(0)
y-digits	numeric expression, rounded to an integer	parameters outside the range 0 through 7 are interpreted as FXD(0)

Examples

```
FXD 3
FXD 3,5
```

Description

FXD allows for formatting LAXES and LGRID labels with 0 through 7 digit positions to the right of the decimal point. A maximum of eight digits plus sign are allowed in the label. The x-digits parameter specifies the format for x-axis labels; y-digits specifies the format for y-axis labels. If the y-digits parameter is omitted, the x-axis and y-axis labels are formatted using the x-digits parameter.

If a label is too large or too small for the specified label format, it is plotted in exponential notation.

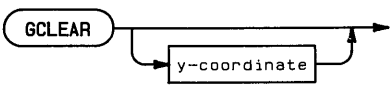
Related Keywords

LAXES, LGRID

GCLEAR

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

When the display is the current plotting device, the `GCLEAR` statement clears the graphics display to the current background color.



Item	Description	Range
y-coordinate	numeric expression	—

Examples

```
IF X=0 THEN GCLEAR
GCLEAR Y*2
```

Description

If a y-coordinate position is specified, the screen in cleared from that position to the bottom of the display. The y-coordinate is interpreted in the current scaling units. The current background color is determined by the current pen number.

GCLEAR Background Color

Pen Number	Background Color After GCLEAR
≥ 1	black
-1 or ≤ -2	white
0	uses previous pen number

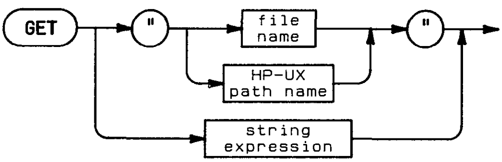
If the current plotting device is a peripheral plotter, GCLEAR sends a page eject command.

Related Keywords

PEN

Keyboard Executable ■
Programmable □
In an IF...THEN □

The GET command retrieves the specified text file from mass storage and attempts to enter the contents into memory as program lines.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
GET "aPet"
GET A$
```

...GET

Description

GET retrieves ASCII character strings from the specified UNIX ASCII text file. The file must not contain control characters. Each record is read as a separate character string. When a string consists of a valid BASIC program statement preceded by a line number, the string is entered into system memory as a program line. If a string cannot be properly interpreted as a program line due to a syntax error, it is entered into system memory as a comment line. When GET encounters a character string that is not preceded by a valid line number, it displays the line.

The retrieved lines are read into system memory without scratching the program already there. If an incoming line has the same line number as a line already in memory, the new line overwrites the original line.

When the GET operation is finished, the system displays `...end of get.`

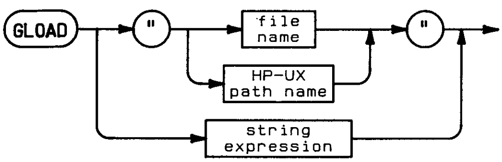
Related Keywords

LOAD

G
H

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `GLOAD` statement retrieves the specified BASIC/GRAF file and enters its contents into graphics display memory.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
GLOAD "Filename"
GLOAD "/v1/filename"
GLOAD "/Dir1/Dir2/filename"
```

Description

If the file name is used alone (rather than as part of an HP-UX path name), GLOAD uses the current working directory.

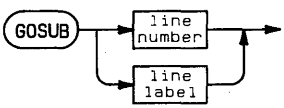
Executing GLOAD clears the previous contents of the graphics display as the contents of the BASIC/GRAF file are entered into graphics display memory. The alpha display can be viewed again programmatically by executing ALPHA, or from the keyboard by pressing any key.

Related Keywords

MASS STORAGE IS, GSTORE

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The GOSUB statement causes program execution to branch unconditionally to the subroutine located at the specified line.



Item	Description	Range
line number	integer constant identifying a program line	1 through 65,535
line label	name of a program line	any valid name

Examples

```
GOSUB 760
GOSUB marine
```

Description

The specified line must be in the same program or subprogram as the GOSUB statement. If the specified statement is declaratory (for example, DIM, REM, or DATA), the program branches to the next executable statement.

When GOSUB is executed, execution of the subroutine continues until a RETURN statement causes branching to the statement following the GOSUB statement.

Subroutines can be recursive; i.e., a subroutine can invoke itself.

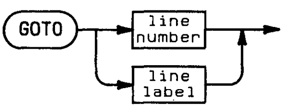
Related Keywords

GOTO, ON...GOSUB, ON...GOTO, RETURN

GOTO

Keyboard Executable ☐
Programmable ■
In an `IF...THEN` ■

The `GOTO` statement causes program execution to branch unconditionally to the specified line.



Item	Description	Range
line number	integer constant identifying a program line	1 through 65,535
line label	name of a program line	any valid name

Examples

```
200 GOTO 340
300 GOTO Increment
400 IF Happy THEN Smile
```

G
H

Description

The specified line must be within the same program or subprogram as the `GOTO` statement. If the specified statement is declaratory (for example, `DIM`, `REM`, or `DATA`), the program branches to the next executable statement.

When `GOTO` is used after `THEN` or `ELSE` in an `IF...THEN` (`...ELSE`) statement, the `GOTO` keyword can be omitted.

Related Keywords

`GOSUB`, `IF...THEN...ELSE`, `ON...GOSUB`, `ON...GOTO`

GRAD

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `GRAD` statement sets grads as the unit in which angles are measured.

Description

When `GRAD` is executed, all angle parameters in statements and functions are interpreted as grads. (There are 400 grads in a circle.) All functions returning an angle return a value in grads.

The angle mode of a program is global. When a subprogram is called, the current angle mode is carried into the subprogram. If a subprogram changes the angle mode and then returns to the main program, the new mode is carried back to the main program.

Related Keywords

`DEG`, `RAD`

GRAPHICS

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The GRAPHICS statement displays the contents of graphics CRT memory.

Description

The display must be the current PLOTTER IS device.

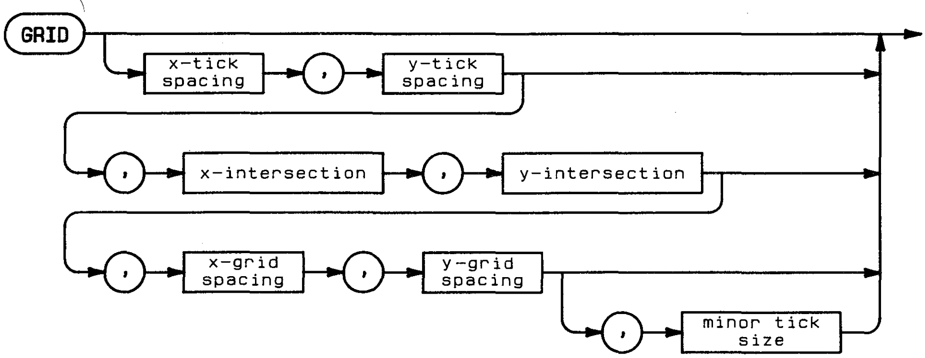
The GRAPHICS statement has no effect on the contents of alpha or graphics CRT memory. The GRAPHICS and ALPHA statements allow you to alternately view the graphics and alpha displays without affecting display memory.

Related Keywords

ALPHA

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The GRID statement draws a grid pattern onto the plotting area using the current line type and pen number.



Item	Description	Range
x-tick spacing	numeric expression, interpreted in current units (default=0, no ticks)	—
y-tick spacing	numeric expression, interpreted in current units (default=0, no ticks)	—
x-intersection	numeric expression interpreted in the current x-axis units (default=lower-left corner)	—
y-intersection	numeric expression interpreted in the current y-axis units (default=lower-left corner)	—
x-grid spacing	numeric expression, rounded to an integer, specifying the number of tick intervals between vertical grid lines (default=1)	—
y-grid spacing	numeric expression, rounded to an integer, specifying the number of tick intervals between horizontal grid lines (default=1)	—
minor tick size	length of a minor tick, in graphics units (default=2)	—

Examples

```
GRID 5,10
GRID 5,10,Xcross,Ycross
GRID t(1),t(2),30,30,2,4,3
```

Description

The grid is drawn across the entire plotting area using the current line type. Grid lines are drawn symetrically from the intersection of the two axes such that a grid line on each axis corresponds with the origin.

The x- and y- tick spacing parameters specify the distance between tick marks on each axis. Negative numbers are interpreted as positive values by taking the absolute value. The default value of 0 draws no ticks.

The x-intersection parameter specifies, in current x-axis units, the point where the x-axis intersects the y-axis. The y-intersection parameter sepcifies, in current y-axis units, the point where the y-axis intersects the x-axis.

The x- and y- grid spacing parameters specify the number of intervals between grid lines. For example, a major count of 4 means that every fourth tick is a grid line. The default value of one draws each tick as a grid line.

The minor tick size parameter specifies the length of the ticks in graphics units. The default length is 2 GU's.

If `GRID` is executed without parameters, the number of grid lines drawn depends on the current scaling.

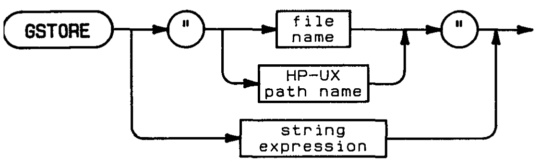
Related Keywords

`AXES`, `LGRID`, `LINE TYPE`

GSTORE

Keyboard Executable	■
Programmable	■
In an IF...THEN	■

The GSTORE statement stores the contents of graphics display memory into a BASIC/GRAF file with the specified name.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
GSTORE "Filename"
GSTORE FILE$
GSTORE "/Dir1/Dir2/filename"
```

G H

Description

If the file name is used alone (rather than as part of an HP-UX path name), the GSTORE operation uses the current working directory.

When GSTORE is executed, the system searches the specified directory for a BASIC/GRAF file with the specified name. If the file is found, the current contents of the graphics display memory is stored in that file, overwriting the previous contents. If no such file is found, then the file is created.

An error is returned if the file name exists with another file type.

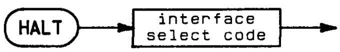
Related Keywords

GLOAD, MASS STORAGE IS

HALT

Keyboard Executable	■
Programmable	■
In an `IF...THEN`	■

The `HALT` statement terminates any currently active interrupt transfer.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10

Examples

```
HALT 7
HALT Isc
```

Description

If `HALT` is executed with an interrupt transfer active and an EOT branch enabled, the branch will be taken.

Interface-dependent action:

- HP-IB: Leaves the bus in the present state.
- Serial, BCD, and GPIO: Does not affect external lines. `RESET` or `ABORTIO` may be necessary after a halt to return the handshake lines to the proper state of the next operation.
- HP-IL: Active controller: If a data transfer is in progress, a Not Ready for Data (NRD) is sent. If the interface is not involved in the transfer, `RESUME` can be used to continue the transfer.

  Non-controller: Leaves the loop in its present state.

Related Keywords

`ABORTIO`, `ON EOT`, `RESET`

HMS

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `HMS` function converts a string in hours:minutes:seconds (HH:MM:SS) format to an integer number of seconds.

Item	Description	Range
string argument	string expression	(see **Description**)

Examples

```
DISP HMS("09:55:34")
LoopTime$=HMS(A$&":"&B$&":"&C$)
```

Description

The string expression must evaluate to a string in the form HH:MM:SS, where:

- HH (hours) consists of two digits in the range 00 through 99.
- MM (minutes) and SS (seconds) are each two digits in the range 00 through 59.

Related Keywords

`DATE`, `HMS$`, `MDY`, `MDY$`, `TIME`

HMS$

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `HMS$` function converts a specified number of seconds to hours:minutes:seconds (HH:MM:SS) format.

Item	Description	Range
numeric argument	non-negative numeric expression, rounded to an integer, interpreted as number of seconds	<360,000

Examples

```
Header$=HMS$(A)
DISP HMS$(12000)
```

Description

`HMS$` returns a string in the range 00:00:00 (`HMS$(0)`) through 99:59:59 (`HMS$(359999)`).

Related Keywords

`DATE, HMS, MYD, MYD$, TIME`

HTD

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `HTD` (*hexadecimal-to-decimal*) function interprets the string argument as the hexidecimal (base 16) representation of an integer and returns the numeric decimal equivalent.

Item	Description	Range
string argument	string expression containing the base 16 representation of an integer	characters must be 0 through 9, A through F; cannot exceed the range of integers

Examples

```
Y=HTD(J$&"B")
IF D=HTD("A") THEN 700
```


Related Keywords

`BTD, DTB$, DTH$, DTO$, OTD`

IDRAW

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `IDRAW` statement draws a line from the current pen position to a position calculated by incrementing the current position by the specified x-and y-increments.

Item	Description	Range
x-increment	numeric expression	—
y-increment	numeric expression	—

Examples

```
IDRAW 10,50
IDRAW RATIO*10, B(1)
```

Description

`IDRAW` uses the current units mode (UU's or GU's) and line type. In UU's mode, lines cannot be drawn outside the plotting boundaries. In GU's mode, the plotting boundaries become equivalent to the graphics limits; therefore, lines can be drawn anywhere within the graphics limits.

In both UU's mode and GU's mode, the logical pen can be moved outside the plotting area. However, the physical pen cannot be moved beyond the plotting boundaries.

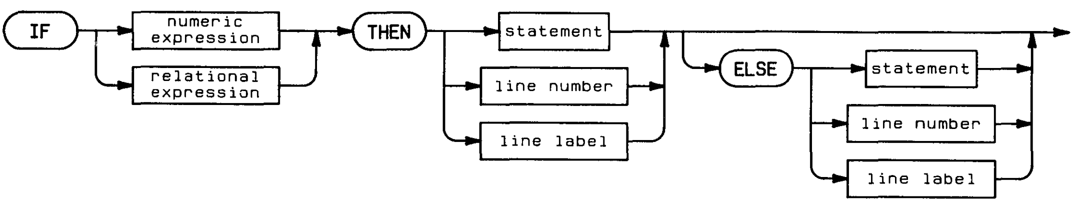
Related Keywords

`DRAW`, `IMOVE`, `IPLOT`, `LINE TYPE`, `PLOT`

IF...THEN...(ELSE)

Keyboard Executable ☐
Programmable ■
In an IF...THEN ☐

The IF...THEN...(ELSE) statement causes conditional branching to the specified program line, based on the value of a relational or numeric expression.



Item	Description	Range
relational expression	an expression comparing two numeric or string expressions using relational operators (=, <, >, <=, >=, <> or #).	—
numeric expression	evaluated as true if non-zero and false if zero	—
statement	a programmable statement allowable "In an IF...THEN"	refer to individual keyword legal usage tables
line number	integer constant identifying a program line	1 through 65,535
line label	name of a program line	any valid name

Examples

```
IF SIN(Angle) THEN DrawLine
IF Variable<5 THEN 200 ELSE PRINT Variable
```

Description

When the expression following `IF` evaluates as true (non-zero), the portion of the statement following `THEN` is executed. When the expression following `IF` is false and the statement includes `ELSE`, the portion of the statement following `ELSE` is executed. When the expression following `IF` is false and the statement does not include `ELSE`, program execution proceeds to the next line.

`THEN` and `ELSE` can be followed by:

- A line number or line label. This is interpreted as an implied `GOTO`.
- An executable statement. The statement must be one permitted "In an `IF...THEN`." If the executable statement is a `GOSUB` statement, the subroutine `RETURN` statement returns execution to the line following the `IF...THEN` statement.
- A sequence of statements concatenated with `@`.

Related Keywords

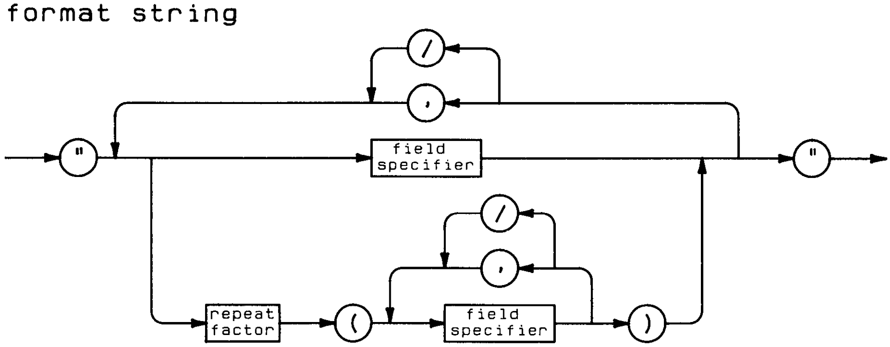
`GOSUB`, `GOTO`

IMAGE

Keyboard Executable	☐
Programmable	■
In an IF...THEN	☐

The IMAGE statement contains a format string referenced by DISP USING, PRINT USING, ENTER USING, OUTPUT USING, LABEL USING, or ENTER USING. The format string contains one or more field specifiers that describe the format of the incoming or outgoing data.

format string



Item	Description	Range
field specifier	literal consisting of one or more image specifiers (see syntax diagram, page 2-176)	—
repeat factor	integer constant	>0
format string	character string consisting of one or more field specifiers	—

Examples

```
IMAGE 2ZCDDD.2D,4X,12A,K
IMAGE #, 4(M3*.3DE,2X,2(3A))
IMAGE "Results = ",2(4D.2D,3X)
```

Description

When the format string is part of an `IMAGE` statement, it is not enclosed in quotes. A format string is enclosed in quotes when it is part of a `DISP USING`, `PRINT USING`, `LABEL USING`, `OUTPUT USING`, or `ENTER USING` statement.

The format string consists of one or more *field specifiers*, separated by delimiters. Most field specifiers designate a format for a particular item. Items are paired with their corresponding field specifiers from left to right. Certain field specifiers are not paired with an item. For example, X specifies a blank space between two items and / specifies an end-of-line sequence.

A field specifier consists of one or more image specifiers. The image specifiers within a field specifier describe the format of one PRINT, DISP, LABEL, OUTPUT, or ENTER item. Items must match their field specifiers in type. For example, a string expression must be formatted with a field specifier appropriate for string data rather than one for numeric items. Certain image specifiers can be preceded by a repeat factor. For example, 4A specifies four character spaces. Certain image specifiers are used to control the EOL sequence sent to devices.

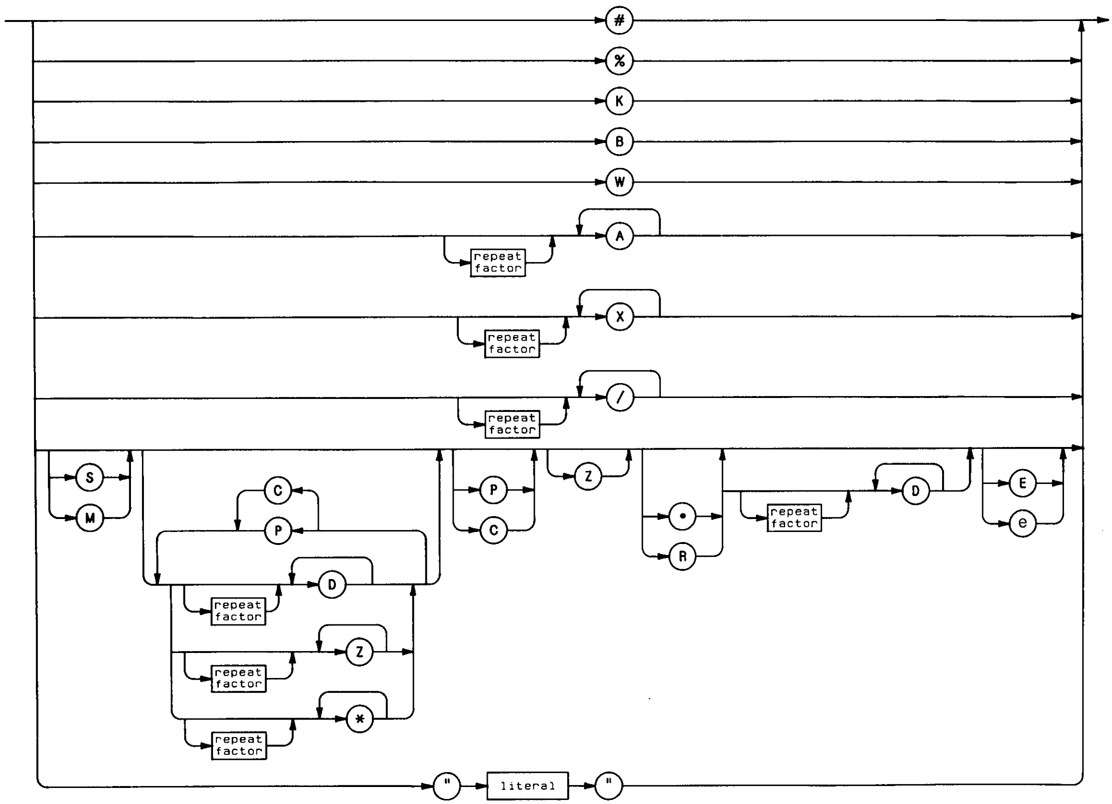
If the format string is exhausted before the entire list of items is output, the format string is reused from the beginning. Extra field specifiers are ignored.

If a field specifier is larger than a numeric item, the number is right-justified in the field. An IMAGE overflow occurs when a numeric item requires more digits spaces to the left of the decimal point than are specified. The overflow is reported as a warning (DEFAULT ON) or error (DEFAULT OFF). In the case of a warning, the default value assigned to the item may be incorrect. If a numeric item contains more decimal places than the field specifier, the number is rounded to fit the field.

If a string item is longer than the field specifier, it is truncated to fit the field. If the string item is shorter than the field specifier, the string is left-justified in the field.

`IMAGE` statements are declaratory; they are ignored if they are not referenced.

field specifier



I
K

Table of Image Specifiers and Delimiters for DISP, PRINT, OUTPUT, and LABEL*

Image Specifier	Meaning
X	Outputs a blank space.
D	Digit position to left or right of the radix symbol. If the field to the left of the radix is larger than the number, the number is right-justified with leading blanks. If no sign is specified, a minus sign occupies one digit position. If a sign image is specified, the sign is positioned to the left of the left-most digit.
Z	Digit position to left of the radix symbol. If the field to the left of the radix is larger than the number, the number is right-justified with leading zeros.
*	Digit position to left of the radix symbol. If the field to the left of the radix is larger than the number, the number is right-justified with leading asterisks.
K	Strings are in compact format with no leading or trailing blanks. Numbers are in standard number format with no leading or trailing blanks.
A	Character position for a string character; When the specified field is larger than the string, characters are left-justified.
B	Outputs a value as one 8-bit byte of data. Values outside the range 0 through 255 are reduced MOD(256). Numbers outside the range 0 through 32,767 return the character ■. Numbers are rounded to the nearest integer.†
W	Outputs a value as two, 8-bit bytes of a 16-bit word. The most significant byte is output first. Numbers outside the range -32,768 through 32,767 uses 32,767. Negative numbers are output in 16-bit 2's complement format.†
.	Radix; specifies a decimal point in that postion.
R	Radix; places a comma in that position.
literal ‡	String constant consisting of any of the following: keyboard characters, the CHR$ function, and metacharacter sequences. The literal image is output without quotation marks.

* This table applies to formatted DISP, PRINT, OUTPUT, and LABEL. See ENTER for additional information.

† When output is directed to the printer or display, the character(s) with decimal codes corresponding to the data bytes are output.

‡ Literal images cannot be used with OUTPUT USING.

I
K

Table of Image Specifiers and Delimiters for DISP, PRINT, OUTPUT, and LABEL* (Continued)

Image Specifier	Meaning
C	Digit separator; places a comma in that position. Comma is output only if digits on both sides of the separator are output.
P	Digit separator; places a period in that position. Period is output only if digits on both sides of the separator are output.
E	Exponential format; exponent consists of three digits plus sign.
e	Exponential format; exponent consists of two digits plus sign.
S	Sign; + or −.
M	Sign; blank or −.
" "	Literal; outputs characters enclosed between quotes.
/	Image specifer or delimiter; performs a carriage return/line feed.
#	Placed at beginning of format string to suppress output of an end-of-line sequence.

Related Keywords

DISP, ENTER, PRINT, OUTPUT

IMOVE

Keyboard Executable	■
Programmable	■
In an `IF...THEN`	■

The `IMOVE` statement lifts the pen and moves it from the current position to a position calculated by incrementing the current pen position by the specified x- and y-displacements.

Item	Description	Range
x-increment	numeric expression, interpreted in the current units	—
y-increment	numeric expression, interpreted in the current units	—

Examples

```
IMOVE 5,10
IMOVE A-10,B
```

Description

`IMOVE` uses the current units mode (UU's or GU's). The physical pen cannot move beyond the plotting boundaries (equivalent to the graphics limits in GU's mode). However, the logical pen can be moved beyond the plotting boundaries or graphics limits.

Related Keywords

`DRAW`, `IDRAW`, `IPLOT`, `LINE TYPE`, `MOVE`, `PLOT`

INF

The `INF` function returns machine infinity, the largest positive REAL number.

Examples

```
DISP INF
```

Related Keywords

`EPS`

Keyboard Executable	■
Programmable	□
In an IF...THEN	□

The INIT command initializes the BASIC program currently in memory.

(INIT)——►

Description

Initializing a program:

- Erases variable assignments made from the keyboard.
- Allocates memory to all program variables and assigns them values of 0 and the null string.
- Checks the program for prerun errors; for example, referencing a nonexistent line, duplicate user-defined functions, and dimensioning the same variable more than once.
- Sets the lowest numbered line as the first line to be executed when the program is run.
- Cancels any enabled event-initiated branching.
- Clears program flags.

Refer to the table of Reset Conditions for additional information.

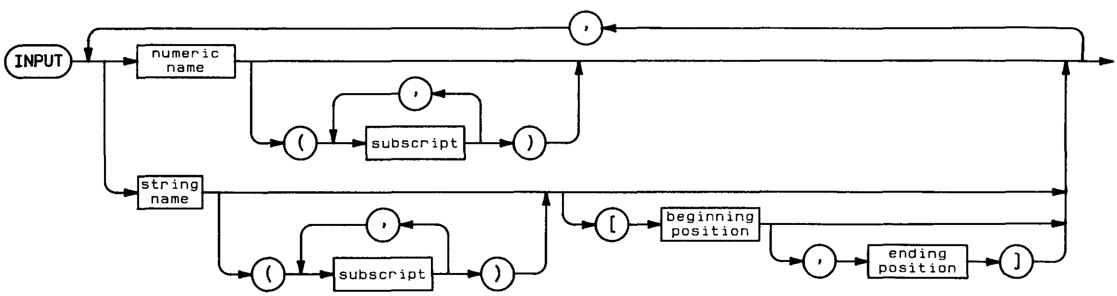
Related Keywords

CONT, PAUSE, RUN

INPUT

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The INPUT statement is used to assign values entered from the keyboard to program variables.



Item	Description	Range
numeric name	name of a simple numeric variable or numeric array	any valid name
string name	name of a simple string variable or string array	any valid name
subscript	numeric expression, rounded to an integer	1 through 65,530
beginning position	numeric expression, rounded to an integer	1 through 65,530
ending position	numeric expression, rounded to an integer	1 through 65,530

I
K

Examples

```
30 INPUT Variable1,Variable2$,Array1(2,3)
50 INPUT Array2$(3),Array2$(4)[3,5],Array2$(6)[3]
```

Description

Executing `INPUT` causes program execution to halt until a value has been entered from the keyboard for each input item. Items are separated by commas. The entire list of items must be entered at once. An error is returned if the number of items entered does not equal the number of items listed in the input statement.

Individual items must match the specified `INPUT` variable(s) in type (numeric versus string). The input statement can include simple numeric and string variables, numeric and string array elements, and substrings. Entries from the keyboard can include numbers, numeric expressions containing numbers and operators, and character strings. If quotation marks appear anywhere in the input string, they are regarded as part of the string. The null string can be assigned to an INPUT string variable only when the `INPUT` statement contains only that item.

When `INPUT` is executed, a question mark is displayed on the current alpha display line. A `DISP` (`USING`) statement, executed just before the `INPUT` statement, can be used to display a more informative prompt. The question mark appears on a separate line from the `DISP` (`USING`) prompt unless that statement suppresses the end-of-line sequence. If the EOL sequence is suppressed, the question mark is displayed on the same line as the prompt, immediately after the last character. The `DISP` EOL sequence is suppressed by terminating the statement with a semicolon. The `DISP USING` EOL sequence is suppressed by including the # image specifier in the format string.

Live keyboard operations are not allowed while the program is halted at `INPUT`. If a program is paused from the keyboard at an `INPUT` statement, executing `CONT` resumes program execution at the line following the `INPUT` statement; the INPUT variables do not receive assignments.

ON KEY#, ON KYBD, ON TIMER#, and ON ERROR branching are temporarily disabled during execution of an `INPUT` statement.

I
K

Related Keywords

`LINPUT`

INT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The INT function returns the greatest integer less than or equal to the numeric argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
DISP INT(35.77*X)
IF X/2=INT(X/2) THEN PRINT "Variable X is Even"
```

Description

The functions INT and FLOOR perform identical operations. INT differs from IP for negative arguments. For example, IP(-5.6) returns -5, whereas INT(-5.6) returns −6.

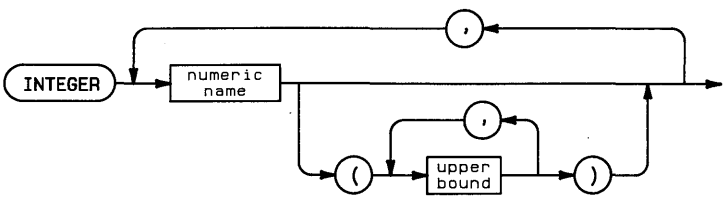
The FLOOR function is identical to INT.

Related Keywords

FLOOR, FP, IP

Keyboard Executable ■
Programmable ■
In an IF...THEN □

The INTEGER statement declares and reserves memory for integer variables.



Item	Description	Range
numeric name	name of a simple numeric variable or numeric array	any valid name
upper bound	integer constant	1 through 65,535

Examples

```
INTEGER IntegerVariable,IntegerArray1(10),IntegerArray2(5,3)
```

I
K

Description

All numeric variables are REAL unless declared SHORT or INTEGER.

When the numeric variable name is used with one or two upper bound(s) enclosed in parentheses, the variable is dimensioned to be a one- or two-dimensional array. The default lower bound of the array is 0. The `OPTION BASE` statement is used to set the lower bound equal to 1.

When a REAL number is assigned to an INTEGER variable, the number is rounded. Overflow occurs if the value of the number is outside the range of integers.

When variables are passed to a subprogram by address, the precision declarations accompany the variable into the subprogram.

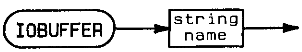
Related Keywords

`DIM, SHORT, REAL`

IOBUFFER

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `IOBUFFER` statement declares a string variable as an I/O buffer.



Item	Description	Range
string name	name of a simple string variable	any valid name

Examples

```
IOBUFFER OneDollar$
```

Description

The previously dimensioned length of the string is the size of the buffer. When the buffer is declared, four pointers are established for controlling buffer activity. In addition, four status registers and two control registers provide for monitoring the buffer pointers:

I/O Buffer Status Registers

Register	Default Value	Function
SR0	1	Buffer empty pointer
SR1	0	Buffer fill pointer
SR2	0	Active-in select code
SR3	0	Active-out select code

I/O Buffer Control Registers

Register	Default Value	Function
CR0	1	Buffer empty pointer
CR1	0	Buffer fill pointer

- The *buffer empty pointer* has an initial value of 1. Its value changes when data bytes are removed from the buffer:

  1. A byte of data is accessed by an ENTER or TRANSFER statement.
  2. The buffer empty pointer is incremented by 1.

  The value of the buffer empty pointer is stored in control/status register 0. The value of the pointer is restored to 1 when the buffer is empty.

- The *buffer fill pointer* has an initial value of 0. Its value changes as bytes of data are placed in the buffer:

  1. The buffer fill pointer is incremented by 1.
  2. A byte of data is placed in the buffer.

  The value of the buffer fill pointer is stored in control/status register 1. The value of the pointer is restored to 0 when the buffer is empty.

- The *active-in select code* equals 0 when there is no active input TRANSFER operation for the buffer. When an input TRANSFER for the buffer is active, the active-in select code equals the interface select code of the source interface. The value of the active-in select code is stored in control/status register 2.

- The *active-out select code* equals 0 when there is no active output TRANSFER operation for the buffer. When an output TRANSFER for the buffer is active, the active-out select code equals the interface select code of the destination interface. The value of the active-out select code is stored in control/status register 3.

A buffer is empty when the buffer empty pointer equals the buffer fill pointer plus one. A buffer is full when the buffer fill pointer equals the dimensioned length of the string variable. When a buffer becomes empty, the buffer fill pointer is reset to 0 and the buffer empty pointer is reset to 1. The active-in select code and active-out select code are not affected by the buffer becoming empty. When a buffer becomes empty, data in the buffer is not lost. The data can be accessed again by changing the value of the buffer fill pointer.

If a conversion table is to be used for a buffer, the `CONVERT` statement must be executed *after* the buffer has been declared with an `IOBUFFER` statement.

Related Keywords

`CONTROL`, `CONVERT`, `ENTER`, `OUTPUT`, `STATUS`, `TRANSFER`

IP

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The IP function returns the integer part of the numeric argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

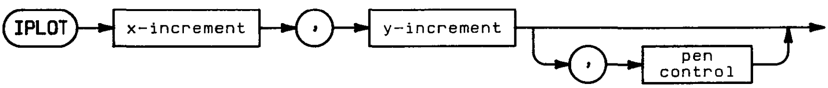
```
PRINT IP(number)
Counter=IP(X+9.6)
```

Related Keywords

FLOOR, FP, INT

Keyboard Executable	■
Programmable	■
In an IF...THEN	■

The IPLOT statement moves the pen from the current pen position to a position calculated by incrementing the current pen position by the specified x- and y-displacements. The optional pen control parameter specifies the up/down status of the pen.



Item	Description	Range
x-increment	numeric expression, interpreted in the current units	—
y-increment	numeric expression, interpreted in the current units	—
pen control	numeric expression, rounded to an integer (default=+1; pen lowered after move)	—

Examples

```
IPLOT X,Y,P
IPLOT 5,10
```

I
K

Description

IPLOT uses the current units (GU's or UU's) and line type. In UU's mode, lines cannot be drawn outside the plotting boundaries. In GU's mode, the plotting boundaries are equivalent to the graphics limits; therefore, lines can be drawn anywhere within the graphics limits.

In both UU's mode and GU's mode, IPLOT can position the logical pen outside the plotting area. However, IPLOT cannot position the physical pen outside the plotting boundaries. If none of the line is inside the current plotting area, the physical pen is not moved, but the logical pen position is updated.

The optional pen control parameter specifies the up and down position of the pen as follows:

Pen Control

Pen Control Parameter	Pen Action
positive, even	pen moved and then lifted
positive, odd	pen moved and then lowered
negative, even	pen lifted and then moved
negative, odd	pen lowered and then moved

If no pen control parameter is specified, the up/down status of the pen before IPLOT is executed determines whether the pen is up or down as it moves. If the pen is up, it is lowered when it reaches its new position.

Related Keywords

LINE TYPE, PLOT, RPLOT

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `KEY LABEL` statement displays the key labels assigned to the user-defined (special function) keys during program execution.

Examples

```
IF KCode=150 THEN KEY LABEL
```

Description

When it is executed in a program, `KEY LABEL` displays the key labels assigned by `ON KEY#` statements in the program.

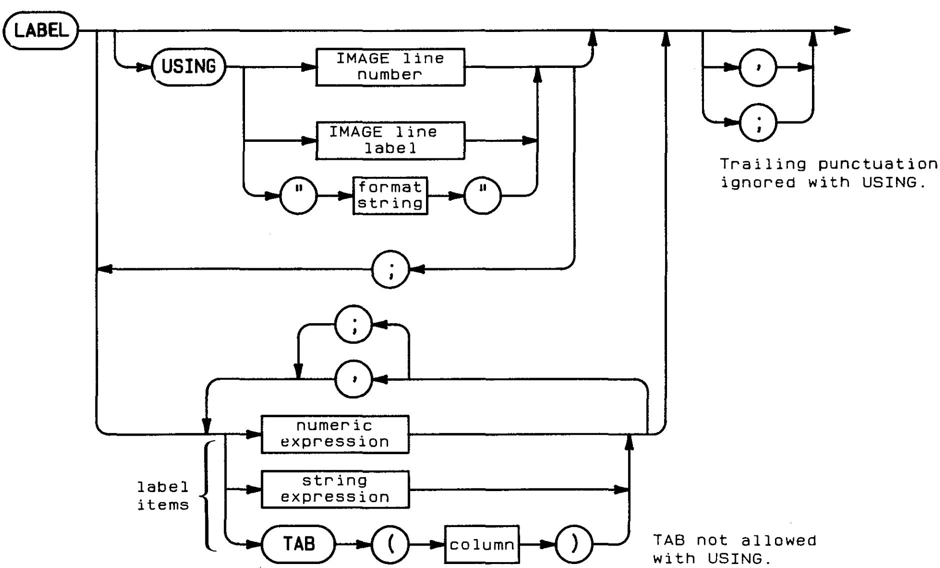
Executing `KEY LABEL` from the keyboard displays the key labels for the typing aids assigned to the user-defined keys. The typing aid assignments are changed by executing `ON KEY#` from the keyboard.

Related Keywords

`OFF KEY#`, `ON KEY#`

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LABEL statement plots alphanumeric labels on the plotting device at the current pen postion.



Item	Description	Range
IMAGE line number	integer constant identifying an IMAGE statement	1 through 65,535
IMAGE line label	name identifying an IMAGE statement	any valid line name
format string	string expression containing one or more field specifiers (see IMAGE statement for syntax)	—
column	numeric expression, rounded to an integer	-99,999 through 99,999; negative numbers are interpreted as 1
numeric expression	—	—
string expression	—	—

Examples

```
LABEL "Velocity (m/s)"
LABEL A$
LABEL USING "5Z.2D"; Earnings
LABEL USING Format; Ylabel$ & "(millions)"
```

Description

Labels can be positioned anywhere within the graphics limits. They are drawn using the current pen and line type 1. (The current line type remains in effect for lines and axes.)

LABEL Without Using. Simple LABEL standard number format (see glossary) for numeric items, and displays numeric and string items in either of two field widths:

- When label items are sparated by semicolons, they are drawn in *narrow* format with a leading blank or minus sign. Strings are output with no leading or trailing blanks.
- When label items are separated by commas, they are drawn in *wide* format, left-justified in 21-column fields. Items longer than 21 characters occupy more than one field.

When the TAB function is included as a label item, the cursor moves to the designated column. Negative column numbers are treated as TAB(1). Column numbers greater than the line length are reduced MOD (line length). When TAB is used to control format, label items should be separated by semicolons; using commas causes output to be displayed in wide format.

After all the label items have been drawn, an end-of-line sequence is sent to the logical pen, moving the pen to a position underneath the first character of the label. The EOL sequence can be suppressed by including a comma or semicolon after the last label item.

LABEL Appearance and Position. The following statements control the appearance of labels:

- The `CSIZE` statement determines the height, aspect ratio, and slant of the label characters.
- The `LORG` statement determines the position of the label with respect to the pen position at the time the `LABEL` statement is executed.
- The `LDIR` statement determines the angle at which the label is drawn.

Formatted Labels. The `LABEL USING` statement uses a format string contained in the statement itself or in an accompanying `IMAGE` statement to format the output. The format string, consisting of one or more field specifiers separated by delimiters (`,` or `/`), is used from left to right. Label items are paired with their corresponding field specifiers. Certain field specifiers do not use a label item (for example, `X`). If the format string is exhausted before all the display items have been processed, the format string is reused from the beginning. Extra field specifiers are ignored. If a field is larger than the numeric item, the number is right-justified in the field. A warning is issued if the number is larger than the field. (A minus sign requires a digit position if `M` or `S` is not included in the field specifier.) Numbers are rounded to the number of decimal places indicated by the field specifier.

Refer to `IMAGE` for the syntax of the format string.

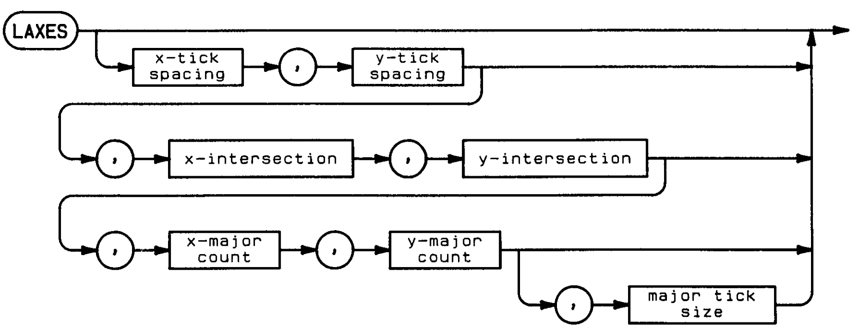
Related Keywords

`CSIZE, IMAGE, LDIR, LORG`

LAXES

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

LAXES (*label axes*) statement draws a pair of axes and labels them with the current scale units at each major tick mark.



Item	Description	Range
x-tick spacing	numeric expression, interpreted in the current units ~~(default=0, no ticks)~~ 10 ticks per axis	—
y-tick spacing	numeric expression, interpreted in the current units ~~(default=0, no ticks)~~	—
x-intersection	numeric expression, interpreted in the current units (default=lower-left corner)	—
y-intersection	numeric expression, interpreted in the current units (default=lower-left corner)	—
x-major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks on the x-axis (default=1)	—
y-major count	numeric expression, rounded to an integer, specifying the number of tick intervals between major tick marks on the y-axis (default=1)	—
major tick size	length of a major tick, in graphics units	—

L

Examples

```
LAXES (Xmax-Xmin)/10,2
LAXES 1,2,Xsect,Ysect
LAXES 1,2,40,20,3,1
```

Description

The axes are drawn across the entire plotting area using the current line type. Tick marks are drawn symmetrically from the intersection of the two axes such that a major tick mark on each axis corresponds with the origin. Labels are drawn using line type 1. They are placed outside the plotting boundaries below the x-axis and to the left of the y-axis.

The x and y tick-spacing parameters specify the distance between tick marks on each axis. When the tick-spacing parameter is positive, the labels are drawn perpendicular to the axis. When the tick-spacing parameter is negative, the labels are drawn parallel to the axis.

The x-intersection parameter specifies, in current x-axis units, the point where the x-axis intersects the y-axis. The y-intersection parameter specifies, in current y-axis units, the point where the y-axis intersects the x-axis.

The x- and y-major count parameters specify the number of intervals between major ticks. For example, a major count of 4 means that every fourth tick is major tick. The default value of 1 draws each tick as a major tick.

The major tick size parameter specifies the length of the major ticks in graphics units. The default length is 2 GU's. Minor ticks are always ½ the size of major ticks.

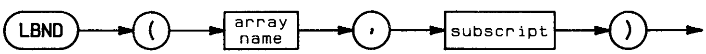
Related Keywords

AXES, GRID, LGRID, LINE TYPE

LBND

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LBND function returns the lower bound of the specified array.



Item	Description	Range
array name	name of a one- or two-dimensional numeric array	any valid name
subscript	numeric expression, rounded to an integer	1 through 2

Examples

```
DISP LBND(array,1)
MAT S=B(LBND(B,1):5,3)
```

Description

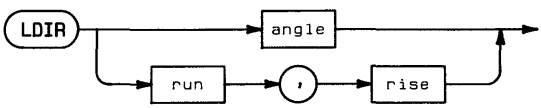
LBND always returns the current option base. The second parameter (subscript) is ignored. (The parameter is used with the corresponding UBND function to specify which upper bound is to be returned in the case of two-dimensional arrays.)

Related Keywords

UBND

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LDIR (*label direction*) statement specifies the angle at which labels are drawn.



Item	Description	Range
angle	numeric expression, interpreted according to the current trigonometric mode	—
run	numeric expression, interpreted in the current scale units	—
rise	numeric expression, interpreted in the current scale units	—

Examples

```
LDIR 60
LDIR A(I),A(I)*1.3
```

Description

The specified angle is interpreted according to the current trigonometric mode (DEG, RAD, or GRAD). This angle measures the counterclockwise rotation between the horizontal axis and the label direction.

The run and rise parameters determine the direction of a vector drawn in the new label direction.

Related Keywords

DEG, GRAD, LABEL, PDIR, RAD

LEN

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `LEN` (*length*) numeric function returns the number of characters in the string argument.

Item	Description	Range
string argument	string expression	—

Examples

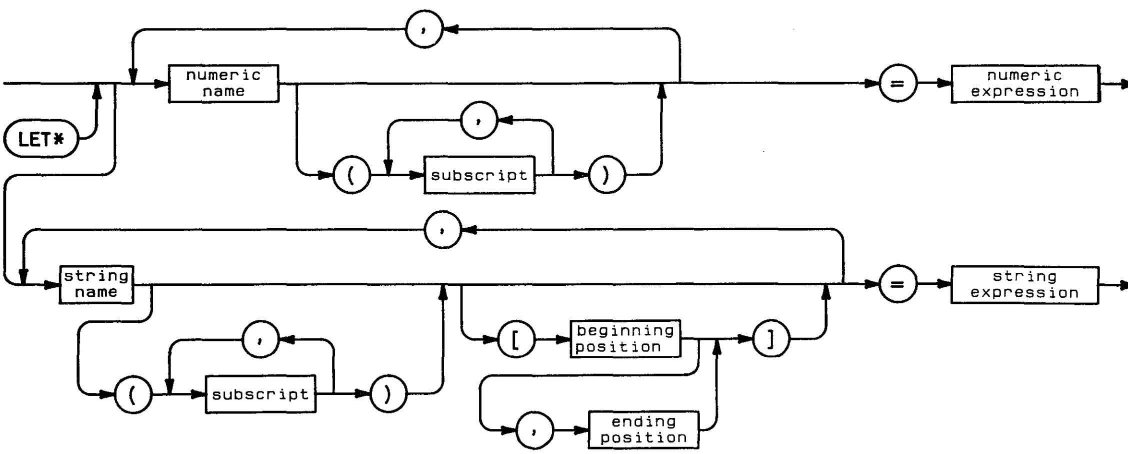
```
Y=LEN(A$)
IF LEN(String$)<=10 THEN String$=String$&" "
```

Description

The value returned is the current number of characters in the string, regardless of its dimensioned length. The length of the null string is 0.

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LET statement assigns values to variables. The keyword is optional in program statements. Assignments from the keyboard must include the keyword.



* Keyword is optional when statement is executed in a program.

Item	Description	Range
numeric name	name of a simple numeric variable or numeric array	any valid name
string name	name of a simple string variable or string array	any valid name
subscript	numeric expression, rounded to an integer	1 through 65,530
beginning position	numeric expression, rounded to an integer	1 through 65,530
ending position	numeric expression, rounded to an integer	1 through 65,530

L

Examples

```
LET Variable=5*X
Svariable$="ABC"&H$
LET A(2,4), B(2,5)=7
String1$(3)[2,5]="fghi"
```

Description

`LET` assigns the numeric or string value on the right side of the equation to one or more variables on the left side. Any variables used on the right side must previously have been assigned.

A REAL expression is rounded when assigned to an INTEGER or SHORT variable. The REAL expression must evaluate to a number within the INTEGER or SHORT range.

When a string expression is assigned to a string variable, the expression must evaluate to a sequence of characters less than or equal to the dimensioned size of the string variable. When a string expression is assigned to a substring, excess characters are truncated. For example, `A$[1,2]="abcde"` assigns the characters `ab` to the first two characters of variable A$.

The following rules apply to string assignments:

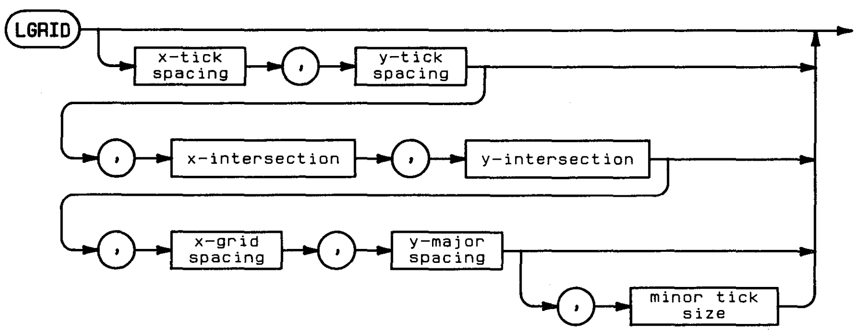
- The expression on the right must evaluate to a string less than or equal to the dimensioned size of the variable.
- When an expression is assigned to a substring, excess characters are truncated. For example, `A$[n,n+1]="abcde"` assigns `ab` to positions n and n+1 of A$.
- When a substring reference contains only the beginning position, characters are entered into the string starting at that position. For example, `A$[n]="qrs"` assigns `qrs` to character positions n, n+1, and n+2.
- `A$[n,n]="abc"` assigns `a` to position n.
- A substring in which the ending position is one less than the beginning position specifies the null string. For example, `A$=B$[4,3]` is equivalent to `A$=""`.
- Substring expressions `A$[n+2,n]`, `A$[n+3,n]`, etc., return an error.

L

LGRID

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LGRID statement draws a grid within the current plotting area and labels each grid line with the current scale units.



Item	Description	Range
x-tick spacing	numeric expression, interpreted in current units (default=0, no ticks)	—
y-tick spacing	numeric expression, interpreted in current units (default=0, no ticks)	—
x-intersection	numeric expression interpreted in the current x-axis units (default=lower-left corner)	—
y-intersection	numeric expression interpreted in the current y-axis units (default=lower-left corner)	—
x-grid spacing	numeric expression, rounded to an integer, specifying the number of tick intervals between vertical grid lines (default=1)	—
y-grid spacing	numeric expression, rounded to an integer, specifying the number of tick intervals between horizontal grid lines (default=1)	—
minor tick size	length of a minor tick, in graphics units (default=2)	—

L

Examples

```
LGRID 5,10
LGRID Xspace,Xspace*2,Xsect,Ysect
LGRID 5,10,30,30,2,4,3
```

Description

The grid is drawn across the entire plotting area using the current line type. Grid lines are drawn symmetrically from the intersection of the two axes such that a grid line on each axis corresponds with the origin. Each grid line is labeled with the current scale units. Labels are drawn outside the plotting boundaries below the x-axis and to the left of the y-axis using line type 1.

The x- and y-tick spacing parameters specify the distance between tick marks on each axis. When the tick-spacing parameter is positive, the labels are drawn perpendicular to the axis. When the tick-spacing parameter is negative, the labels are drawn parallel to the axis.

The x-intersection parameter specifies, in current x-axis units, the point where the x-axis intersects the y-axis. The y-intersection parameter sepcifies, in current y-axis units, the point where the y-axis intersects the x-axis.

The x- and y-grid spacing parameters specify the number of intervals between grid lines. For example, a major count of 4 means that every fourth tick is a grid line. The default value of one draws each tick as a grid line.

The minor tick size parameter specifies the length of the ticks in graphics units. The default length is 2 GU's.

If `LGRID` is executed without parameters, two labeled axes are drawn.

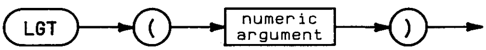
Related Keywords

`AXES, GRID, LAXES, LINE TYPE`

LGT

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `LGT` function returns the base 10 logarithm of the argument.



Item	Description	Range
numeric argument	numeric expression	>0

Examples

```
A(2)=A(1)*LGT(T)
IF LGT(X)=2 THEN DISP X
```

Related Keywords

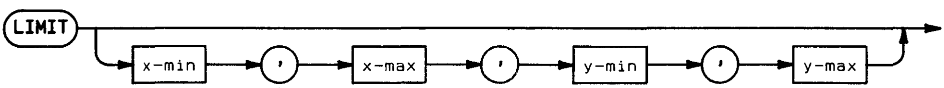
`LOG`

L

LIMIT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LIMIT statement specifies the graphics limits of the plotting device and activates the graphics default conditions (see glossary). The graphics limits must be within the physical limits of the plotting device.



Item	Description	Range
x-min	numeric expression, interpreted as millimeters	—
x-max	numeric expression, interpreted as millimeters	—
y-min	numeric expression, interpreted as millimeters	—
y-max	numeric expression, interpreted as millimeters	—

Examples

```
LIMIT 0,125,0,75
LIMIT A(1),A(2),A(3),A(4)
```

Description

The LIMIT parameters specify the coordinates, in millimeters, of the lower-left and upper-right corners of the plotting area. The origin is the lower-left corner of the physical limits. The parameters must specify coordinates within the physical limits of the plotting device. When `LIMIT` is executed, the physical and logical pens are moved to the lower left corner of the graphics limits—(0,0) in GU's.

Executing `LIMIT` overrides any previously set graphics limits; the new limits remain in effect until a new `LIMIT` statement is executed, or until the default graphics limits are activated (see glossary) by reset or by executing a `PLOTTER IS` statement.

When `LIMIT` is executed without parameters, program execution halts until coordinates are entered from the plotting device.

The order of LIMIT parameters can be changed to produce reflected graphics output:

Reflecting Plots

LIMIT Statement Parameters	Effect
x-max, x-min, y-min, y-max	reflects output across y-axis
x-min, x-max, y-max, y-min	reflects output across x-axis
x-max, x-min, y-max, y-min	reflects output across origin

Labels are reflected by using negative CSIZE parameters (refer to **CSIZE**).

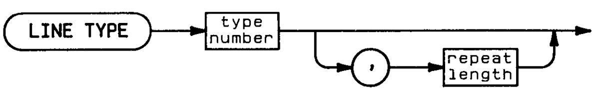
Related Keywords

`CSIZE, LOCATE, PLOTTER IS, RATIO, SCALE, SETGU, SETUU, SHOW`

LINE TYPE

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LINE TYPE statement selects the line type for drawing lines, axes, frames, and grids on the graphics display. For peripheral plotters and some displays, LINE TYPE also provides for selecting the repeat length of the line pattern.



Item	Description	Range
type number	numeric expression, rounded to an integer (default=1)	—
repeat length	numeric expression, rounded to an integer, interpreted as GU's (default=4 GU's for peripheral plotters)	—

Examples

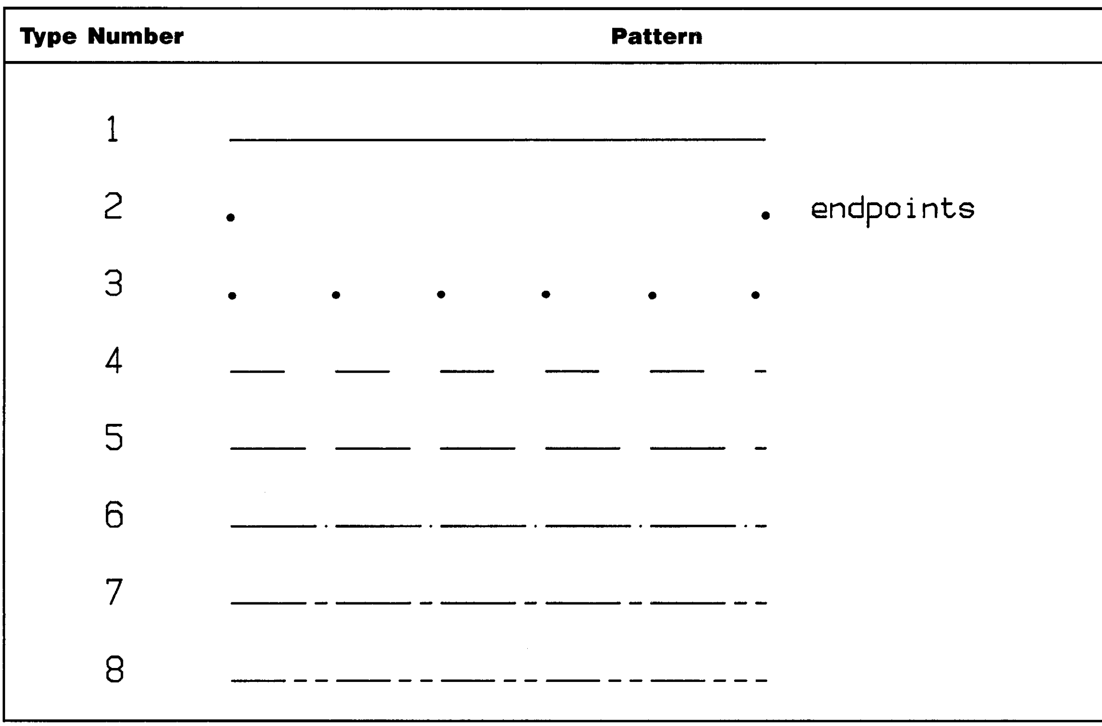
```
LINE TYPE 5
LINE TYPE A,10
```

Description

Line types 1 through 8 are available on the graphics display. Type numbers outside this range default to line type 1.

The repeat length is always expressed in GU's, regardless of the current units. The default value of the display repeat length is machine-dependent. The repeat length parameter may be ignored by some display devices.

Display Line Types



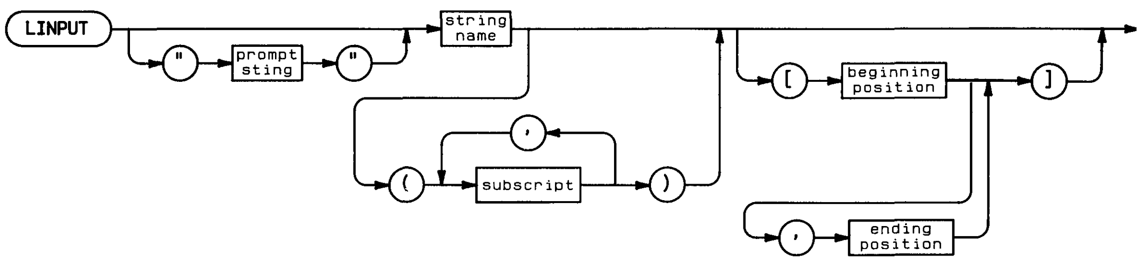
Related Keywords

PEN

LINPUT

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `LINPUT` statement accepts alphanumeric input from the keyboard, interprets the input as a character string, and assigns the character string to the specified string variable.



Item	Description	Range
prompt string	literal composed of characters from the keyboard	—
string name	name of a simple string variable or string array element	any valid name
subscript	numeric expression, rounded to an integer	1 through 65,530; maximum of two allowed
beginning position	numeric expression, rounded to an integer	1 through 65,530
ending position	numeric expression, rounded to an integer	1 through 65,530

Examples

```
LINPUT "Enter your data", A$
LINPUT Variable$
```

L

Description

When `LINPUT` is executed, a prompt appears on the current line of the alphanumeric display and remains there until the LINPUT item is satisfied. If no prompt is specified, the default prompt `?` is used. Using a null string for the prompt string suppresses the default prompt.

The `LINPUT` statement allows commas, quotation marks, and leading and trailing blanks in the character string assigned to the string variable. Unlike the `INPUT` statement, multiple inputs and variable assignments are not allowed.

Pressing (RETURN) terminates data input. If no characters are entered, the null string is assigned to the string variable.

Event-initiated branching (`ON KEY#`, `ON ERROR`, `ON KYBD`, `ON EOT`, `ON INTR`, `ON TIMEOUT`, `ON TIMER#`) is disabled while `LINPUT` is being executed.

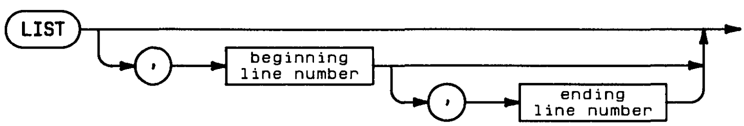
Related Keywords

`INPUT`

LIST

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LIST statement lists the current program or subprogram in system memory on the alpha display.



Item	Description	Range
beginning line number	integer constant	1 through 65,535
ending line number	integer constant	1 through 65,535

Examples

```
LIST
LIST 40,40
```

L

Description

The beginning line number and ending line number specify the portion of the program to be listed. If no ending line number is specified, listing begins at the beginning line number and fills the display.

When both parameters are omitted, the listing fills the screen. Listing begins at the first line of the program except in the following cases:

- When program execution has been halted by a program error, by execution of `PAUSE`, or by pausing the program from the keyboard, listing begins at the line at which execution halted.
- Executing `LIST` repeatedly displays successive segments of the program.

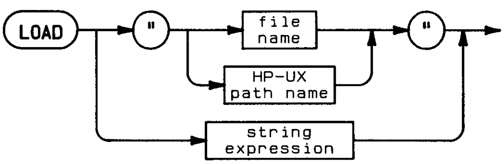
Related Keywords

`PLIST`

LOAD

Keyboard Executable ■
Programmable □
In an `IF...THEN` □

The `LOAD` command retrieves the specified BASIC/PROG file and loads the program into system memory.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
LOAD "Filename"
LOAD "/vol1/filename"
LOAD "/Directory1/Directory2/filename"
```

Description

If the file name is used alone (rather than as part of an HP-UX path name), the LOAD operation uses the current working directory.

`LOAD` scratches any BASIC programs, subprograms, and variable assignments in memory.

`LOAD` cannot be used to load subprograms. `FINDPROG` retrieves subprograms and makes them available for editing.

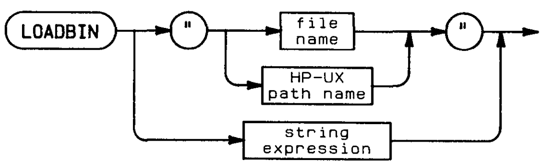
Related Keywords

`FINDPROG`, `MASS STORAGE IS`, `STORE`

LOADBIN

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LOADBIN statement retrieves the specified binary file, enters it into BASIC memory, and makes all the binary program entry points available to CALLBIN.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
LOADBIN "Gdraw"
LOADBIN "mylogon/gdraw"
LOADBIN FILE$
```

Description

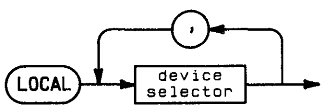
LOADBIN loads the binary program without scratching BASIC memory. Program entry points are not checked; linking must be done outside the BASIC environment.

Related Keywords

CALLBIN, SCRATCHBIN

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LOCAL statement returns one or more an instruments to local control after they have been placed under remote control by the REMOTE statement.



Item	Description	Range
device selector	numeric expression, rounded to an integer (see glossary)	—

Examples

```
LOCAL 703,706
LOCAL 100*Isc+Address
```

Description

If two or more device selectors are specified, they must be valid combinations of an interface select code and primary address; the devices must be at the same interface select code.

Interface-dependent action:

- HP-IB:

  If the computer is System Controller and the device selector is an interface select code, Remote Enable (REN) is set false.

  If the computer is Active Controller and the device selector contains a primary address, the interface addresses the specified device(s) and sends Go To Local (GTL), leaving ATN true. (Use RESUME to set ATN false.)

  If the device is in remote with local lockout set, the device must receive GTL or have REN set false to be returned to local control.

- Serial, BCD, GPIO: Error.
- HP-IL:

  The computer must be active controller.

  If the device selector in an interface select code, Not Remote Enable (NRE) is sent.

  If the device selector includes a primary address, Unlisten (UNL), Listen Address n (LADn), and Go To Local (GTL) are sent.

  If the device is in remote with local lockout set, the device must receive Go To Local (GTL) or Not Remote Enable (NRE) before it returns to local control.

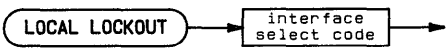
Related Keywords

LOCAL LOCKOUT, REMOTE

LOCAL LOCKOUT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LOCAL LOCKOUT statement sends the Local Lockout message (LLO), which prevents an operator from placing the specified device(s) under manual (local) control.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10

Examples

```
LOCAL LOCKOUT Isc
LOCAL LOCKOUT 7
```

Description

The computer must be active controller. The LLO message is received by all devices on the interface. If a device is in the LOCAL state when LLO is sent, the message does not take affect until the device receives a Remote message and becomes addressed to listen.

Interface-dependent action:

- HP-IB: ATN is left true. (If necessary, use RESUME to set ATN false.) Local Lockout remains in effect until the Remote Enable (REN) line is set false.
- HP-IL: Local Lockout remains in effect until the Not Remote Enable (NRE) command is sent.
- Serial, BCD, and GPIO: Error.

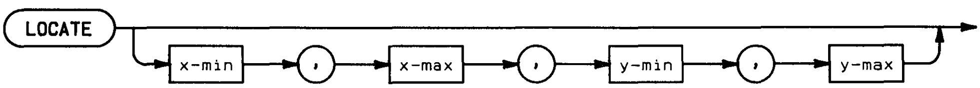
Related Keywords

LOCAL, REMOTE

LOCATE

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LOCATE statement specifies plotting boundaries in graphics units (GU's).



Item	Description	Range
x-min	numeric expression, interpreted as GU's	—
x-max	numeric expression, interpreted as GU's	—
y-min	numeric expression, interpreted as GU's	—
y-max	numeric expression, interpreted as GU's	—

Examples

```
LOCATE 20,60,50,100
LOCATE 20,20+X,50,50+Y
```

Description

The LOCATE parameters define the plotting boundaries in GU's. These boundaries replace any previously defined plotting boundaries. When the system is in UU's mode, no lines can be drawn beyond the plotting boundaries. However, labels can be drawn outside the plotting area and within the graphics limits.

When LOCATE is executed prior to SCALE, MSCALE, or SHOW, the user units are mapped onto the LOCATE-defined plotting area. If a CLIP statement is executed after LOCATE, the CLIP boundaries replace the LOCATE boundaries.

The LOCATE plotting boundaries are canceled when `LIMIT`, `PLOTTER IS`, or `UNCLIP` are executed. The `SETGU` statement deactivates the plotting boundaries; they are restored by executing `SETUU`.

When `LOCATE` is executed without parameters, program execution halts until plotting boundaries are entered from the plotting device.

The LOCATE parameters can be exchanged to reflect the plot (see `LIMIT`).

Related Keywords

`CLIP`, `LIMIT`, `LOCATE`, `PLOTTER IS`, `SETGU`, `SETUU`, `UNCLIP`

LOG

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The LOG numeric function returns the natural (base e) logarithm of the argument.

Item	Description	Range
numeric argument	numeric expression	>0

Examples

```
T=1/K*LOG(N1/N2)
IF LOG(A)<=2 THEN 900
```

Related Keywords

EXP, LGT

LORG

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `LORG` (*label origin*) statement specifies the position of labels relative to the current pen position.

Item	Description	Range
label position	numeric expression, rounded to an integer (default=1)	1 through 9

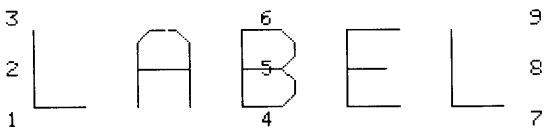
Examples

```
LORG 5
LORG X
```

Description

Label positions outside the range 1 through 9 are interpreted as LORG 1.

The following illustration shows the relationship between the label and the logical pen position. The numbers show the logical pen position before the label is drawn using the various label position numbers.



Related Keywords

LABEL, LDIR

LWC$

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `LWC$` function returns a string formed by replacing all uppercase letters in the argument with lowercase letters.

Item	Description	Range
string argument	string expression	—

Examples

```
DISP LWC$("QWERTY")
IF LWC$(A$)="y" THEN GOSUB Positive
```

Description

The `LWC$` function affects only the letters A through Z (characters with ASCII code 65 through 90).

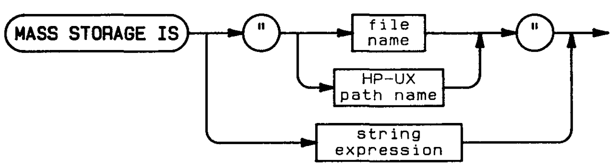
Related Keywords

`UPC$`

MASS STORAGE IS

Keyboard Executable	■
Programmable	■
In an IF...THEN	■

The MASS STORAGE IS statement designates the specified directory file as the current working directory.



Item	Description	Range
file name	literal; name of a directory file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name of a directory file (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name of a directory file	—

Examples

```
MASS STORAGE IS "textfiles"
MASS STORAGE IS "/vol1/dir2/dir3"
```

Description

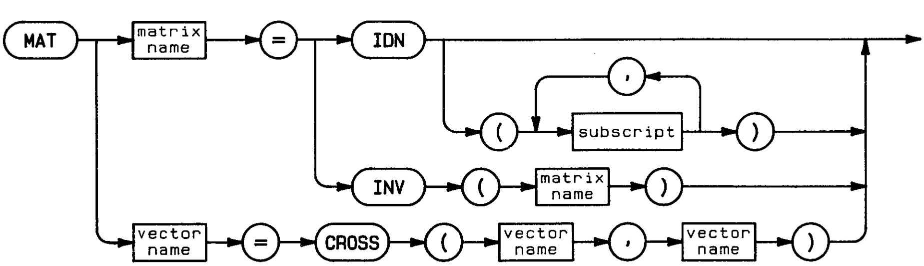
The specified file must be a directory file. Once a directory file has been designated the current working directory, files in that directory can be accessed by file name alone.

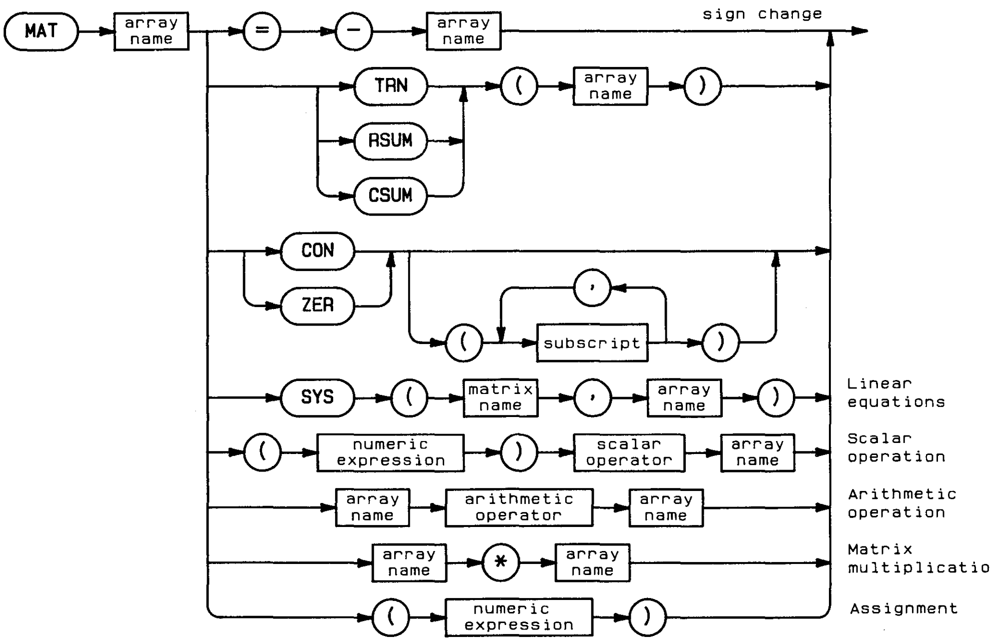
MAT

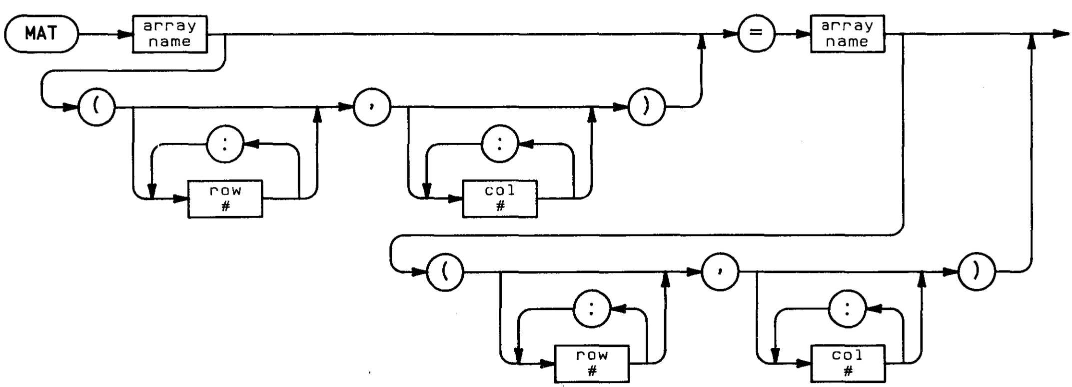
Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The MAT statement performs a number of operations on arrays. The statement can be constructed to perform arithmetic and scalar operations, matrix multiplication, and to initialize arrays to constant values. Through the use of secondary keywords, the statement performs a variety of special vector and matrix operations.

Item	Description	Range
array name	name of a one- or two-dimensional numeric array	any valid name
vector name	name of a one-dimensional array	any valid name
matrix name	name of a two-dimensional array	any valid name
scalar operator	operator used in scalar arithmetic with an array	+ − * /
arithmetic operator	operator used in arithmetic operations involving two arrays	+ − . /
numeric expression	(see glossary)	—
row #	numeric expression, rounded to an integer	valid row number
col #	numeric exression, rounded to an integer	valid column number




MAT
array name
=
−
array name
sign change
TRN
RSUM
CSUM
(
array name
)
CON
ZER
,
(
subscript
)
SYS
(
matrix name
,
array name
)
Linear equations
(
numeric expression
)
scalar operator
array name
Scalar operation
array name
arithmetic operator
array name
Arithmetic operation
array name
*
array name
Matrix multiplication
(
numeric expression
)
Assignment


MAT
array name
=
array name
(
:
row #
,
:
col #
)
(
:
row #
,
:
col #
)

Examples

`MAT NegArray=-PosArray`	Sign change.
`MAT A=(B)`	Arithmetic assignment.
`MAT A=CSUM(Array)`	Sum of columns.
`MAT A=SYS(Matrix,Array)`	System of linear equations.
`MAT A=INV(M)*C`	Inverse; matrix multiplication.
`MAT Q=(2*X)/Array`	Scalar operation.
`MAT Q=Array1.Array2`	Arithmetic operation.
`MAT B=ZER(3,3)`	Initializing (and redimensioning, if necessary) an array.
`MAT A=Array`	Copying an entire array.
`MAT A(3:5,2:7)=Array(1:3,1:6)`	Copying a portion of an array.
`MAT Vector=CROSS(Vector1,Vector2)`	Cross product.

Description

The `MAT` statement allows you to:

- Change the sign of every element in an array.
- Calculate the Inverse (`INV`), and Transpose (`TRN`) of a matrix.
- Produce an identity matrix (`IDN`).
- Calculate the cross product (vector product) of two, 3-element vectors (`CROSS`).
- Calculate the sum of the rows (`RSUM`) and the sum of the columns (`CSUM`) of an array.
- Solve a system of *n* linear equations with *n* unknowns (`SYS`).
- Assign the value 1 (`CON`) or zero (`ZER`) to all the elements of an array.
- Add, subtract, multiply, and divide a numeric expression and an array (scalar operation).
- Add, subtract, multiply, and divide the elements of two arrays (arithmetic operation).
- Perform matrix multiplication between two arrays.
- Copy all or a portion of an array into all or a portion of another array.

Identity (IDN). The secondary keyword `IDN` produces an identity matrix by assigning the value 1 to all diagonal elements (elements for which the row subscript equals the column subscript). If the matrix is not a square matrix before execution of the `MAT = IDN` statement, the matrix must be redimensioned within the statement by specifying redimension subscripts.

Inverse (INV). The secondary keyword `INV` calculates the inverse of a square matrix. (A matrix multiplied by its inverse produces an identity matrix.) When the determinant of a matrix equals 0, the inverse cannot be calculated.

If the result matrix is not the same size and shape as the operand matrix, the system attempts to redimension it. An error is returned if the result array is not large enough to be properly redimensioned.

Transpose (TRN). The secondary keyword `TRN` produces the transpose of a array by exchanging the rows and columns of the operand array. The transpose of an n-by-m array is an m-by-n array; each element is defined by interchanging the subscripts.

The result array must be dimensioned to be at least as large as the current size of the operand array. If necessary, the system redimensions the result array to the proper shape.

Cross Product (CROSS). The secondary keyword `CROSS` calculates the cross product (vector product) of two, 3-element vectors. The two operand arrays and the result array must be vectors.

Summing Rows and Columns (RSUM and CSUM). The secondary keyword `RSUM` computes the sum of each row of the operand array and assigns those values to the elements of a one-column vector. If the result array is a vector, it is redimensioned, if necessary, to have as many elements as the number of rows in the operand array. If the result array is a matrix, it is first redimensioned to have one column and as many rows as the operand array.

The secondary keyword `CSUM` computes the sum of each column of the operand array and assigns those values to the elements of a one-row vector. As with `RSUM`, the result array is redimensioned, if necessary, to a vector of the proper size.

Solving the Matrix Equation AX=B. The secondary keyword `SYS` solves the matrix equation AX=B for the unknown array X. This statement is most often used when solving a system of n linear equations in n unknowns:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \text{ and } \mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

A is the coefficient matrix, B is the constant array, and X is the result array containing the solution to the system of equations. When B and X are matrices, `SYS` simultaneously solves two different systems of n equations in n unknowns.

Assigning Values 1 and 0 To Elements. The secondary keyword `CON` assigns the value 1 to all elements of the result array. Optional parameters redimension the array to the specified size.

The secondary keyword `ZER` assigns the value 0 to all elements of the result array. As with `CON`, the optional parameters redimension the array to the specified size.

Scalar Operations. A scalar operation statement performs an arithmetic operation between a numeric expression and each element of the operand array. Array elements can be added to (+), subtracted from (−), multiplied by (*), and divided into (/) a specified numeric value.

A scalar operation can also be used to change the sign of every element in an array. For example,

```
MAT B = -A
```

assigns values to the elements of array B by changing the sign of every element in array A.

Arithmetic Operations Between Arrays. An arithmetic operation statement performs addition (+), subtraction (−), multiplication (.), or division (/) between corresponding elements of two arrays.

Matrix Multiplication. If A and B are the two operand arrays and C is the result array, matrix multiplication is defined by the equation:

$$C_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

where n is the number of elements in a column in array A.

Matrix multiplication follows these general rules:

- The result array has the same number of rows as the first operand array and the same number of columns as the second operand array.
- Matrix multiplication is legal only if the column size of the first operand array equals the row size of the second operand array.
- The system allows multiplication of a row vector and a column vector. However, two row vectors or two column vectors are not allowed.

Arithmetic Assignment. An arithmetic assignment evaluates the numeric expression enclosed in parentheses and assigns that value to every element of the specified array.

Copying Arrays. An array copy statement copies all or a portion of an operand array to all or a portion of a result array.

The following rules apply to copying an entire array to another entire array:

- If both arrays are matrices, the result array is first redimensioned to have the same number of rows and columns as the operand matrix.
- If the result array is a vector, the operand array must be a vector, a one-column matrix, or a one-row matrix. The result vector is first redimensioned to have the same number of elements as the operand array.
- If the result array is a matrix and the operand array is a vector, the result matrix is first redimensioned to have one column and as many rows as the number of elements in the operand vector.

The following rules apply to copying values from and/or into a portion of an array (subarray):

- If all elements of the operand array are to be copied, do not specify row or column numbers after the operand array name. If all elements of the result array are to be assigned values, do not specify the row or column numbers after the result array name. The values of array elements are transferred in order from left to right along each row, and from top row to bottom row.
- If no row or column numbers are specified after the result array, the result array is redimensioned before the values are assigned. If row or column numbers are specified after the result array, values are assigned to the specified elements, but no redimensioning occurs.
- If an array is a vector, specify only the row number.
- If an entire row is to be copied or assigned values, the column numbers may be omitted; however, a comma must be placed after the row number. If an entire column is to be copied or assigned values, the row numbers may be omitted; however, a comma must be placed before the column number. For example, `MAT B(,4)=MAT A` copies all the elements in vector A into column 4 of array B.

- If only one row or column is to be copied, specify the row or column number. If more than one row or column are copied, specify the beginning and ending row or column number, separated by a colon. For example, `MAT B(3,1:4)=MAT A(2:5,2)` copies elements from column 2, rows 2 through 5 of array A into row three, columns 1 through 4 of array B.
- If the operand and result arrays are both matrices, the number of rows (and columns) specified after the result array must equal the number of rows (and columns) copied from the operand array.

A column from an operand *matrix* cannot be copied into a row of a result array using one statement. Conversely, a row from an operand matrix cannot be copied into a column of the result array using one statement. In both cases, values must first be copied to an intermediate vector.

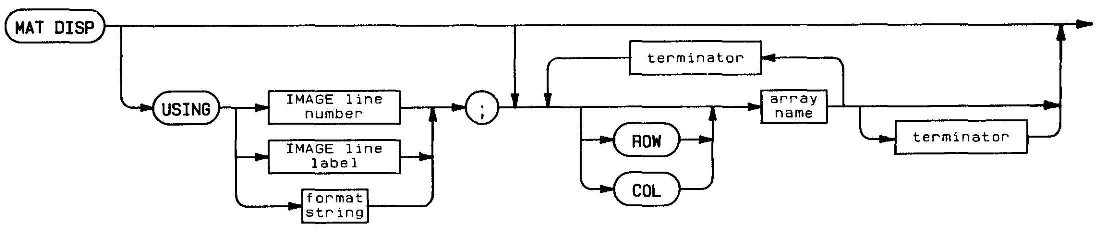
Related Keywords

`REDIM`

MAT DISP

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The MAT DISP statement displays the specified array(s).



Item	Description	Range
array name	name of a numeric array	any valid variable name
IMAGE line number	integer constant identifying an IMAGE statement	1 through 65,535
IMAGE line label	name identifying an IMAGE statement	any valid line name
format string	string expression consisting of one or more field specifiers (see IMAGE statement for syntax)	—
terminator	comma, semicolon, or slash	—

Examples

```
MAT DISP A
MAT DISP ROW A$; COL B/
MAT DISP USING 200; COL Array1
```

Description

MAT DISP provides two forms of output: simple (without USING) and formatted (with USING).

The optional keywords ROW and COL specify the arrangement of the displayed array elements. Specifying ROW causes elements to be displayed by rows. Each row begins on a new line, and the elements in each row are displayed in order from the first column to the last column. Specifying COL causes elements to be displayed by columns. Each column begins on a new line, and the elements of a column are displayed in order from the top row to the bottom. The default arrangement is ROW. More than one line may be required to display a row or column.

Simple MAT DISP (without USING). A terminator is placed after the array name to specify the horizontal spacing between elements. A final terminator after the last array name in the statement specifies spacing for that array. Unlike the DISP statement, the end-of-line sequence is not suppressed.

MAT DISP Terminators

Terminator	Spacing Between Elements
;	Close spacing; elements are separated by two spaces. A minus sign occupies one space.
,	Wide spacing; elements are left-justified in 21-column fields.
/	One element per line.

...MAT DISP

Formatted MAT DISP (with USING). MAT DISP USING uses a format string contained in the statement itself, or in a referenced IMAGE statement, to define the format of the output. The format string, consisting of one or more field specifiers separated by delimiters, is used from left to right. Elements are paired with their corresponding field specifiers. If the format string is exhausted before all the display items have been processed, the format string is re-used from the beginning. Extra field specifiers are ignored. If a field is larger than a number, the number is right-justified in the field. A warning is issued if an element is larger than the field. Numbers are rounded to the number of decimal places indicated by the field specifier.

The comma, semicolon, and slash terminators can be used interchangeably. Spacing is controlled entirely by the format string. A final terminator does not suppress the end-of-line sequence.

Refer to **IMAGE** for the syntax of the format string.

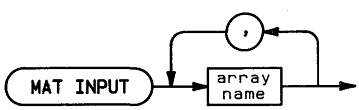
Related Keywords

IMAGE, MAT PRINT

MAT INPUT

Keyboard Executable	□
Programmable	■
In an `IF...THEN`	■

The `MAT INPUT` statement inputs values into the specified array(s).



Item	Description	Range
array name	name of a numeric array	any valid variable name

Examples

```
MAT INPUT B
MAT INPUT NumericArray, StringArray
```

Description

When `MAT INPUT` is executed, the program prompts for elements of the first specified array by displaying the variable name of the first element—for example, `Array(0,0)`. One or more values, separated by commas, can be entered at a time. Values are assigned to array elements from left to right along a row, from top row to bottom. When one or more values have been entered, `MAT INPUT` prompts for the next element to be assigned. Input into the array continues until all the elements have been assigned values. If an array becomes full in the middle of an input line, the remaining elements on the line are ignored.

...MAT INPUT

If a second array is specified, input into it starts at the next input line after the first array is full.

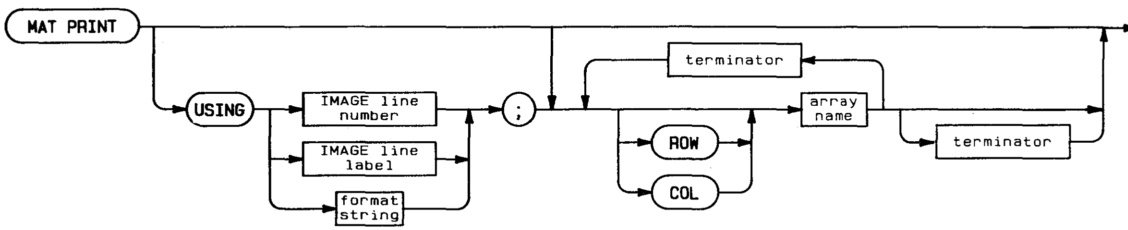
Input continues until all the specified arrays are full.

Related Keywords

`MAT READ`

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `MAT PRINT` statement outputs the specified array(s) to the PRINTER IS device.



Item	Description	Range
array name	name of a numeric array	any valid variable name
`IMAGE` line number	integer constant identifying an `IMAGE` statement	1 through 65,535
`IMAGE` line label	name identifying an `IMAGE` statement	any valid line name
format string	string expression consisting of one or more field specifiers (see `IMAGE` statement for syntax)	—
terminator	comma, semicolon, or slash	—

Examples

```
MAT PRINT A
MAT PRINT ROW A; COL B/
MAT PRINT USING 200; COL Array1
```

...MAT PRINT

Description

MAT PRINT provides two forms of output: simple (without USING) and formatted (with USING).

The optional keywords ROW and COL specify the arrangement of the printed array elements. Specifying ROW causes elements to be printed by rows. Each row begins on a new line, and the elements in each row are printed in order from the first column to the last column. Specifying COL causes elements to be printed by columns. Each column begins on a new line, and the elements of a column are printed in order from the top row to the bottom. The default arrangement is ROW. More than one line may be required to print a row or column.

Simple MAT PRINT (without USING). A terminator is placed after the array name to specify the horizontal spacing between elements. A final terminator after the last array name in the MAT PRINT list specifies spacing for that array. Unlike the PRINT statement, the end-of-line sequence is not suppressed.

MAT PRINT Terminators

Terminator	Spacing Between Elements
;	Close spacing; elements are separated by two spaces. A minus sign occupies one space.
,	Wide spacing; elements are left-justified in 21-column fields.
/	One element per line.

Formatted MAT PRINT (with USING). MAT PRINT USING uses a format string contained in the statement itself, or in a referenced IMAGE statement, to define the format of the output. The format string, consisting of one or more field specifiers separated by delimiters, is used from left to right. Elements are paired with their corresponding field specifiers. If the format string is exhausted before all the print items have been processed, the format string is reused from the beginning. Extra field specifiers are ignored. If a field is larger than a number, the number is right-justified in the field. A warning is issued if an element is larger than the field. Numbers are rounded to the number of decimal places indicated by the field specifier.

The comma, semicolon, and slash terminators can be used interchangeably. Spacing is controlled entirely by the format string. A final terminator has no effect on the output.

Refer to **IMAGE** for the syntax of the format string.

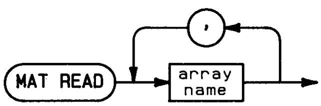
Related Keywords

`DISP, IMAGE, MAT DISP`

MAT READ

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The MAT READ statement reads values from DATA statements and assigns them to array elements.



Item	Description	Range
array name	name of a numeric array	any valid name

Examples

```
MAT READ NumericArray
MAT READ A, B
```

Description

The values are read from DATA statements and assigned to array elements from left to right along a row, from top row to bottom. Arrays are filled in the order in which they are listed. If there are not enough data elements to satisfy MAT READ, the program returns an error and program execution halts.

Related Keywords

MAT INPUT

MAX

The MAX function compares two numeric arguments and returns the larger of the two values.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Y=MAX(10,X)
Counter=IP(MAX(I,J))
```

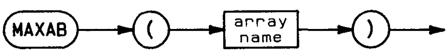
Related Keywords

MIN

MAXAB

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The MAXAB function computes the absolute value of each element in the specified array and returns the largest value.



Item	Description	Range
array name	name of a one- or two-dimensional numeric array	any valid name

Examples

```
DISP MAXAB(Array1)
IF MAXAB(Array1)=1 THEN 500
```

Related Keywords

AMAX, AMAXCOL, AMAXROW, MAXABCOL, MAXABROW

MAXABCOL

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `MAXABCOL` function returns the column number of the element whose absolute value was returned by the most recently executed `MAXAB` function.

Examples

```
YSubsript=MAXABCOL
A(3,MAXABCOL)=12
```

Description

If two or more elements in different columns have the largest absolute value, the lowest column number is returned.

Related Keywords

`AMAXCOL`, `MAXAB`, `MAXABROW`

MAXABROW

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The MAXABROW function returns the row number of the element whose absolute value was returned by the most recently executed MAXAB function.

Examples

```
XSubscript=MAXABROW
MAT C=Array2(1:MAXABROW,4)
```

Description

If two or more elements in different rows have the largest absolute value, the lowest row number is returned.

Related Keywords

AMAXCOL, MAXAB, MAXABCOL

MDY

Keyboard Executable	■
Programmable	■
In an IF...THEN	■

The MDY function converts a string expression in the form MM/DD/YYYY to the equivalent Julian Day number.

Item	Description	Range
string argument	string expression in the form “MM/DD/YYYY”	“10/15/1582” through “11/25/4046”

Examples

```
DISP MDY("04/20/1984")-MDY("10/03/1983")
IF MDY(Day$)<2446160 THEN 2000
```

Description

The allowable parameters correspond to Julian Day numbers 2,299,161 through 3,199,160.

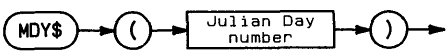
Related Keywords

DATE, DATE$, MDY$

MDY$

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The MDY$ function interprets a numeric expression as the Julian Day number and converts it to a string expression in the form MM/DD/YYYY.



Item	Description	Range
Julian Day number	numeric expression, rounded to an integer, interpreted as the Julian Day number	2,299,161 through 3,199,160

Examples

```
DISP MDY$(3000000)
Day$=MDY$(X)
```

Description

The allowable parameters correspond to October 15, 1582 through November 25, 4046.

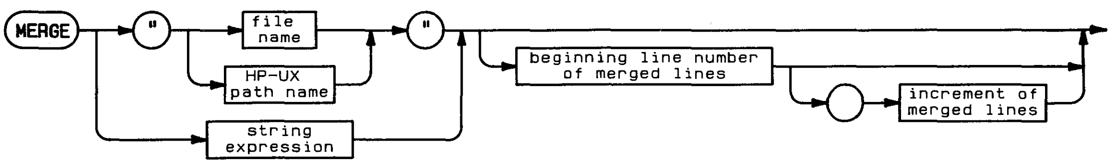
Related Keywords

DATE, DATE$, MDY

MERGE

Keyboard Executable ■
Programmable □
In an IF...THEN □

The MERGE command merges a program or subprogram retrieved from mass storage with the current program or subprogram in system memory.



Item	Description	Range
file name	literal; name of a BASIC/PROG or BASIC/SUBP file	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—
beginning line number of merged lines	integer constant identifying a program line (default=last line number of current program+10)	1 through 65,535
increment of merged lines	integer constant (default=10)	1 through 65,535

Examples

```
MERGE "Traffic"
MERGE "ER"200,5
```

...MERGE

Description

If the file name is used alone (rather than as part of an HP-UX path name), the MERGE operation uses the current working directory. The current working directory is selected by the `MASS STORAGE IS` statement.

`MERGE` retrieves the specified BASIC/PROG or BASIC/SUBP file from mass storage, renumbers the retrieved program lines, and adds them to the current (sub)program in system memory. The merged program is renumbered according to the beginning line number of merged lines and the increment of merged lines specified in the `MERGE` command. If the optional parameters are omitted, the beginning line number of merged lines is obtained by incrementing the last line number in system memory by 10.

When programs are merged using the optional parameters, any merged lines renumbered to the same line numbers as lines currently in memory overwrite those lines.

The message `...end of merge` is displayed at the conclusion of the merge operation.

Related Keywords

`FINDPROG`, `REN`

MIN

The `MIN` function compares two numeric arguments and returns the smaller of the two values.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Y=MIN(10,X)
Counter=IP(MIN(I,J))
```

Related Keywords

`MAX`

MOD

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The MOD operator returns the remainder resulting from a division operation.

Item	Description	Range
dividend	numeric expression	—
divisor	numeric expression	—

Examples

```
C=8 MOD 3
IF Hours MOD Trip<3 THEN 300
```

Description

The MOD operation is defined by the equation:

$$A \text{ MOD } B = A - B * \text{INT}(A/B)$$

where INT(A/B) is the greatest integer less than or equal to A/B. By definition, A MOD 0 is A.

Related Keywords

DIV

MOVE

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The MOVE statement lifts the pen and moves it to the specified x-, y-coordinate position. The pen remains up until it is lowered by another statement.

Item	Description	Range
x-coordinate	numeric expression, interpreted in the current units	—
y-coordinate	numeric expression, interpreted in the current units	—

Examples

```
MOVE 10,10
MOVE XPosition,XPosition*5
```

Description

MOVE uses the current units mode (UU's or GU's). The physical pen cannot move beyond the plotting boundaries (equivalent to the graphics limits in GU's mode). However, the logical pen can be moved beyond the plotting boundaries or graphics limits.

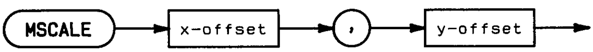
Related Keywords

IMOVE, PLOT

MSCALE

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `MSCALE` statement specifies millimeter user units scaling of the plotting area and the location of the origin.



Item	Description	Range
x-offset	numeric expression, interpreted as millimeters	—
y-offset	numeric expression, interpreted as millimeters	—

Examples

```
MSCALE 10,5
MSCALE A*10,A
```

Description

The `MSCALE` parameters specify, in millimeters, the offset of the origin from the lower-left corner of the plotting area. `MSCALE` scales the current plotting area, which is a function of the units mode (GU's or UU's) and the previously executed statements.

- In GU's mode, `MSCALE` scales the entire graphics area previously specified by `PLOTTER IS` or `LIMIT`).
- In UU's mode, `MSCALE` scales the plotting area previously specified by `LOCATE`. If `LOCATE` has not been executed, the entire graphics area is scaled.

After executing `MSCALE`, the system in set to UU's mode.

Related Keywords

`LIMIT`, `LOCATE`, `PLOTTER IS`, `SCALE`, `SHOW`

See **FOR...NEXT**.

NORMAL

Keyboard Executable ■
N **Programmable** ■
In an IF...THEN ■

The NORMAL statement cancels *print-all* mode and program tracing (TRACE, TRACE VAR, and TRACE ALL) operations.

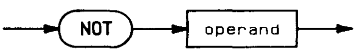
Related Keywords

AUTO, PRINTALL, TRACE

NOT

Keyboard Executable	■
Programmable	■
In an IF...THEN	■

The NOT operator returns 1 if its operand equals 0. Otherwise, 0 is returned.



Description

A non-zero expression (positive or negative) is interpreted as a logical 1; a zero is interpreted as a logical 0. The following table describes the results of performing a NOT operation.

Logical NOT

A	NOT A
0	1
non-zero	0

Related Keywords

AND, EXOR, OR

NPAR

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The NPAR function returns the number of parameters passed to a subprogram by a CALL statement.

Examples

```
ON NPAR GOTO 200,300,400
IF NPAR=2 THEN SUBEXIT
```

Related Keywords

CALL, SUB

NUM

Keyboard Executable	■
Programmable	■
In an `IF...THEN`	■

The `NUM` numeric function returns the decimal value of the first character in the string argument.

Item	Description	Range
string argument	string expression	—

Examples

```
X=NUM(String$[A,A])
IF NUM(A$)=32 THEN Skip
```

Description

The value returned is in the range 0 through 255. When the argument is the null string, `NUM` returns 0.

Related Keywords

`CHR$`

OFF CURSOR

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `OFF CURSOR`* statement removes the cursor from the alpha display. The cursor position remains unchanged.

O

```
(OFF CURSOR)──►
```

Related Keywords

`ON CURSOR`

* Implementation of OFF CURSOR is machine-dependent.

OFF EOT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The OFF EOT statement disables end-of-line branching for termination of a TRANSFER operation on the specified interface.

Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10

Examples

```
OFF EOT 7
OFF EOT Isc
```

Description

When a transfer terminates after OFF EOT has been executed, the system retains a pending end-of-line branch. The branch is taken immediately when ON EOT is executed for that interface. Only one (the most recent) end-of-line branch can be pending.

Related Keywords

ON EOT, TRANSFER

OFF ERROR

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `OFF ERROR` statement cancels event-initiated branching previously enabled by a `ON ERROR` statement. Further errors halt program execution.

Related Keywords

`ON ERROR`

OFF INTR

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The OFF INTR statement cancels end-of-line branching for interface interrupts previously established by ON INTR.

Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10

Examples

```
OFF INTR 7
OFF INTR Isc
```

Description

If the interface is enabled for interrupts (by ENABLE INTR), OFF INTR prevents branching when an interrupt occurs. However, the interrupt is retained by the system; the branch will be taken immediately if ON INTR is executed. If more than one interrupt occurs, only the last one is retained.

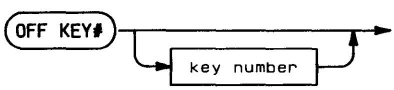
Related Keywords

CONTROL, ENABLE INTR, ON INTR

OFF KEY#

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The OFF KEY# statement cancels end-of-line branching previously enabled by an ON KEY# statement.



Item	Description	Range
key number	numeric expression, rounded to an integer	must correspond to a special function key

Examples

```
OFF KEY# 1
OFF KEY# N
```

Description

If the key number is omitted, all current run-time ON KEY# assignments are canceled.

Related Keywords

ON KEY#

OFF KYBD

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `OFF KYBD` statement cancels end-of-line branching previously enabled by an `ON KYBD` statement.

Item	Description	Range
string expression	characters and/or escape sequences representing the keys for which branching is disabled.	—

Examples

```
OFF KYBD "1234567890"
OFF KYBD A$ & "#*"
```

Description

When the optional parameter is omitted, `OFF KYBD` cancels branching for all previously enabled keys.

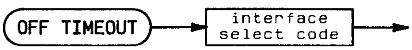
Related Keywords

`OFF KEY#`, `ON KYBD`

OFF TIMEOUT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The OFF TIMEOUT statement cancels end-of-line branching for timeouts on the specified interface.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10

Examples

```
OFF TIMEOUT 7
OFF TIMEOUT Isc
```

Description

When a timeout (specified by SET TIMEOUT) occurs after OFF TIMEOUT has been executed, the system retains a pending end-of-line branch. The branch is taken immediately when ON TIMEOUT is executed for that interface.

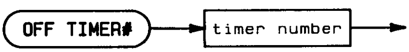
Related Keywords

ON TIMEOUT

OFF TIMER#

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The `OFF TIMER#` statement cancels end-of-line branching for the specified timer.



Item	Description	Range
timer number	numeric expression, rounded to an integer	1 through 3

Examples

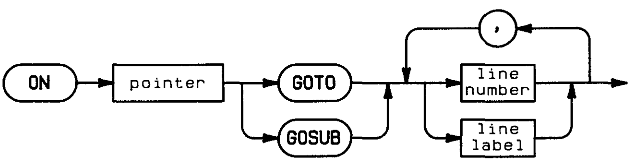
```
OFF TIMER# 3
OFF TIMER# TimerNumber
```

Related Keywords

`ON TIMER#`

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The ON...GOTO/GOSUB statements transfer program execution to one of the specified program lines based on the value of a pointer.



Item	Description	Range
pointer	numeric expression, rounded to an integer	(see **Description**)
line number	integer constant identifying a program line	1 through 65,535
line label	name of a program line	any valid name

Examples

```
250   ON P(1) GOTO 200, 400, 640
740   ON .5*Pointer1 GOSUB Subroutine1, Subroutine2
612   IF Y THEN ON Y GOTO 330, Odd, 700
```

Description

When the pointer evaluates to 1, execution is transferred to the first line number or line label. When the pointer evaluates to 2, execution is transferred to the second line number/label, and so on. An error is returned if the pointer evaluates to a number less than 1 or greater than the number of line numbers/labels. In practice, the maximum value of the pointer equals the number of line numbers/labels that can be typed into a program line.

If the `GOSUB` keyword is used, execution is tranferred to the specified subroutine. When the subroutine `RETURN` statement is executed, execution branches to the statement immediately following `ON...GOSUB`.

Related Keywords

`GOSUB`, `GOTO`, `RETURN`

ON CURSOR

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ON CURSOR* statement displays the cursor after it has been previously turned off by the OFF CURSOR statement.

Related Keywords

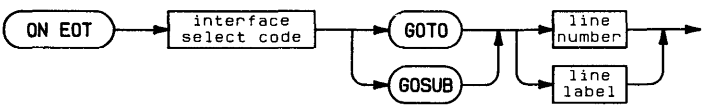
OFF CURSOR

* Implementation of ON CURSOR is machine-dependent.

ON EOT

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The ON EOT statement defines and enables end-of-line branching when the last byte of data is transferred by a TRANSFER statement.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
line number	integer constant	1 through 65,535
line label	name of a program line	any valid name

Examples

```
ON EOT 7 GOTO 1000
ON EOT Isc GOSUB TransferDone
```

Description

When ON EOT is executed, any pending end-of-line branch for a previous, unserviced transfer termination at the specified interface is taken immediately.

ON EOT overrides any previous ON EOT for that interface.

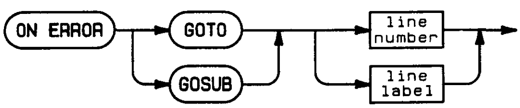
Related Keywords

OFF EOT, TRANSFER

ON ERROR

Keyboard Executable □
Programmable ■
In an IF...THEN ■

The ON ERROR statement defines and enables an event-initiated branch to be taken when a run-time error occurs.



Item	Description	Range
line number	integer constant identifying a program line	1 through 65,535
line label	name of a program line	any valid name

Examples

```
ON ERROR GOSUB 100
ON ERROR GOTO Recovery
```

...ON ERROR

Description

ON ERROR branching occurs immediately when a run-time error is detected, and has higher priority than any other event-initiated routine. When an `ON ERROR...GOSUB` statement is used, the recovery routine `RETURN` statement returns execution to the program line following the one that generated the error. If an error occurs in the middle of a multistatement line, the rest of the line is not executed.

The ON ERROR declaration remains active during the recovery routine unless it is disabled by executing `OFF ERROR`. In general, `OFF ERROR` should be executed at the beginning of the recovery routine to prevent an infinite loop between the line containing the error and the beginning of the recovery routine.

ON ERROR declarations take precedence over all other program interrupts. (Refer to the Branch Precedence Table on page 4-13.)

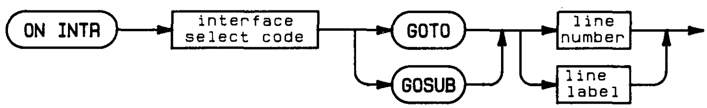
ON ERROR declarations are local to the program or subprogram in which they are executed.

Related Keywords

`OFF ERROR`

Keyboard Executable	☐
Programmable	■
In an IF...THEN	■

The ON INTR statement defines an end-of-line branch to be taken when an interface generates an interrupt.



Item	Description	Range
interface select code	numeric expresion, rounded to an integer	3 through 10
line number	an integer constant specifying a valid line number	1 through 65,535
line label	name identifying a program line	any valid line name

Examples

```
ON INTR 7 GOTO 300
ON INTR 3 GOSUB Service
```

Description

ON INTR establishes the end-of-line branch to be taken when an interface interrupt occurs. Interrupt causes are specified by setting the appropriate bit(s) in Control Register 1 using the ENABLE INTR or CONTROL statement.

When an interrupt occurs and is serviced, end-of-line branching is disabled for the interface until enabled using ENABLE INTR or CONTROL. It is not necessary to execute ON INTR again.

If a pending end-of-line branch from a previous, unserviced interrupt exists when ON INTR is executed, the branch is taken immediately. Only one interrupt per select code is retained by the system.

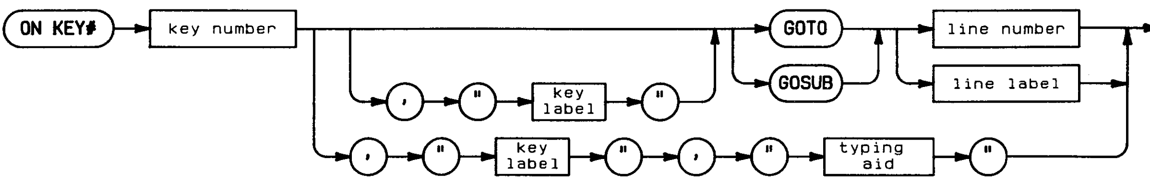
Executing a second ON INTR statement for the same interface overrides the previous ON INTR. If two interfaces interrupt during the same line, the order in which the interrupts are serviced is determined by their precedence. (Refer to the Branch Precedence Table, page 4-13, for additional information.)

Related Keywords

CONTROL, ENABLE INTR, OFF INTR

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ON KEY# statement defines the functions of the user-defined (special function) keys.



Item	Description	Range
key number	numeric expression, rounded to an integer	must correspond to the number of one of the special function key
key label	literal or string expression evaluating to displayable characters	1 through 8 characters
line number	integer constant identifying a program line	1 through 65,535
line label	name of a program line	any valid name
typing aid	literal composed of displayable characters and/or control characters	1 through 32 characters

Examples

```
ON KEY# 3, "BREAK" GOSUB Break
ON KEY# 4, "Path","dir1/dir2/dir3"
```

...ON KEY#

Description

The syntax and function of ON KEY# has two forms:

- When ON KEY# is executed within a program, it defines and enables a branch to be taken when the specified user-defined key is pressed. The optional key label parameter provides for displaying a key label when KEY LABEL is executed in the program.
- When ON KEY# is executed from the keyboard, it defines a typing aid for a sequence of characters. Typing aids are in effect whenever a program is not running. The optional key label parameter provides for displaying a key label.

  If the the typing aid string consists of a keyboard executable statement or command followed by a carriage return, pressing the key executes the statement or command immediately.

When ON KEY# is executed within a program, end-of-line branching is enabled for the specified key. If the ON KEY#...GOSUB statement is used, the subroutine RETURN statement causes branching to the statement following the one being executed when the key was pressed.

ON KEY# end-of-line branching is disabled by executing OFF KEY#.

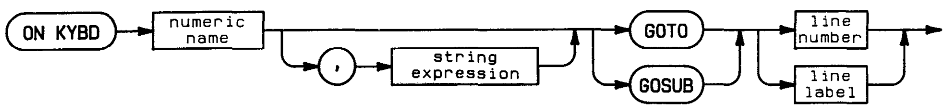
Refer to the Branch Precedence Table on page 4-13 for additional information.

Related Keywords

ENABLE KBD, KEY LABEL, OFF KEY#

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ON KYBD statement defines and enables an event-initiated branch to be taken when the specified key(s) is(are) pressed during program execution.



Item	Description	Range
numeric name	name of a simple numeric variable or numeric array element.	—
string expression	characters and/or escape sequences representing the keys for which branching is enabled	—
line number	integer constant identifying a program line	1 through 65,535
line label	name of a program line	any valid name

Examples

```
ON KYBD Keys,"1234567890" GOSUB Numberkeys
ON KYBD A1,CHR$(27)&"w" GOTO 130
```

...ON KYBD

Description

Executing ON KEYBD enables end-of-line branching to the specified program line when any of the keys listed in the string expression are pressed. Alphanumeric keys are identified in the string expression by their displayable character (for example, a and A for the unshifted and shifted (A) key) or by their numeric key code (for example, CHR$(65) for (A)). Keys without displayable characters (*special keys*, such as tab, cursor control, and special function keys) must be identified by their escape sequences.* For example, the following statement enables ON KYBD branching for (A) and (◄), assuming ESC D for the (◄) key:

```
ON KYBD Keyvar,"A"&CHR$(27)&"D" GOTO 100
```

When a keystroke triggers an interrupt that causes branching to the specified program line, the key code of the key pressed is assigned to the numeric variable. That variable assignment remains in effect until the variable is reassigned by an assignment statement or by pressing another key specified in the ON KYBD statement. For example, pressing (A) assigns the value 65 to variable Keyvar.

The most recent ON KYBD declaration overrides any previous ON KYBD statement. Keys enabled in the previous statement remain active; however, branching will occur to the most recently specified program line, and the variable assignment will be made to the most recently specified variable.

When the optional string expression is omitted, branching remains in effect for all previously specified keys.

When ON KYBD branching is enabled for any of the special function keys, it overrides ON KEY# branching previously specified for those keys. ON KYBD declarations are local to the program or subprogram in which they are enabled.

One or more enabled keys can be disabled by executing OFF KYBD.

Related Keywords

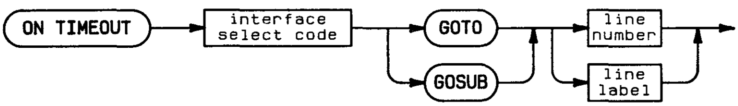
OFF KYBD, ON KEY#

* Escape sequences are machine dependent.

ON TIMEOUT

Keyboard Executable	☐
Programmable	■
In an IF...THEN	■

The ON TIMEOUT statement enables end-of-line branching when an interface timeout occurs on the specified interface.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
line number	integer constant	1 through 65,535
line label	name of a program line	any valid name

Examples

```
ON TIMEOUT 7 GOTO 300
ON TIMEOUT Isc GOSUB Recover
```

...ON TIMEOUT

Description

The amount of time the system will wait for completion of a handshake is set by SET TIMEOUT. If ON TIMEOUT is executed after the SET TIMEOUT limit has been exceeded for that interface, the end-of-line branch is taken immediately.

A timeout will not occur when a peripheral device stops handshaking in the middle of a transfer operation (TRANSFER INTR or TRANSFER FHS). However, a transfer can timeout if the interface or device cannot be addressed to start the transfer.

ON TIMEOUT overrides any previous ON TIMEOUT for the specified interface.

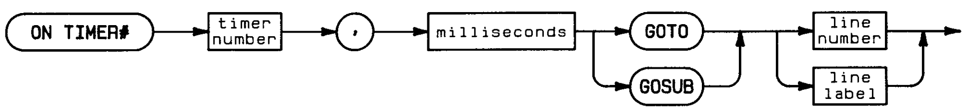
Related Keywords

OFF TIMEOUT, SET TIMEOUT

ON TIMER#

Keyboard Executable ☐
Programmable ■
In an `IF...THEN` ■

The `ON TIMER#` statement defines an end-of-line branch to be taken when the specified time interval has elapsed.



Item	Description	Range
timer number	numeric expression, rounded to an integer	1 through 3
milliseconds	numeric expression	⩾1
line number	integer constant identifying a program line	1 through 65,535
line label	name of a program line	any valid name

Examples

```
ON TIMER# 2, 5000 GOTO Service
ON TIMER# TNumber,TLimit GOSUB 1000
```

...ON TIMER#

Description

When ON TIMER# is executed, the specified timer is set to zero and activated. When the milliseconds interval has elapsed, the branch is taken at the end of the current program line. After the branch has been taken, the timer is reset to zero and immediately reactivated. If the ON TIMER#...GOSUB statement is used, the subroutine RETURN statement causes branching to the statement following the one being executed when the key was pressed.

ON TIMER# branching remains in effect until an OFF TIMER# statement is executed for that timer, or until the program chains another program into memory. Timers continue to come due when the program is paused or delayed (by a WAIT statement), but the branch is not immediately taken. Pending branches are taken when the program is continued or when the WAIT interval has elapsed.

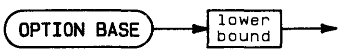
Related Keywords

OFF TIMER#

OPTION BASE

Keyboard Executable	☐
Programmable	■
In an IF...THEN	☐

The OPTION BASE statement specifies the lower bound of all arrays in a program.



Item	Description	Range
lower bound	integer constant (default=0)	0 or 1

Examples

```
OPTION BASE 1
```

Description

An OPTION BASE statement can occur only once in a program, and must precede any explicit variable declarations. The option base is the lower bound of all numeric and string arrays in the program. (Upper bounds are declared in the dimensioning statements—REAL, SHORT, INTEGER, and DIM.)

The option base declaration is global; the option base is passed to any subprograms called by the program. An error may result if a subprogram attempts to specify another option base.

When a program chains another program, the option base of the two programs must agree.

Related Keywords

DIM, INTEGER, REAL, SHORT

OR

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `OR` operator returns a 1 or 0 based on the logical inclusive-OR of the operands.

Item	Description	Range
operand	numeric expression	—

Examples

```
IF A OR B THEN C
Decision=Yes OR No
```

Description

A non-zero operand (positive or negative) is interpreted as a logical 1; an operand of zero is interpreted as a logical 0. The following table describes the result of performing a logical OR.

Inclusive OR

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Related Keywords

`AND, EXOR, NOT`

OTD

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The OTD (*octal-to-decimal*) function interprets the string argument as the octal (base 8) representation of an integer and returns the numeric decimal equivalent.

Item	Description	Range
string argument	string expression containing the base 8 representation of an integer	characters must be 0 through 7; cannot exceed the range of integers

Examples

```
U=OTD("3567")
IF I=OTD(H$) THEN 45
```

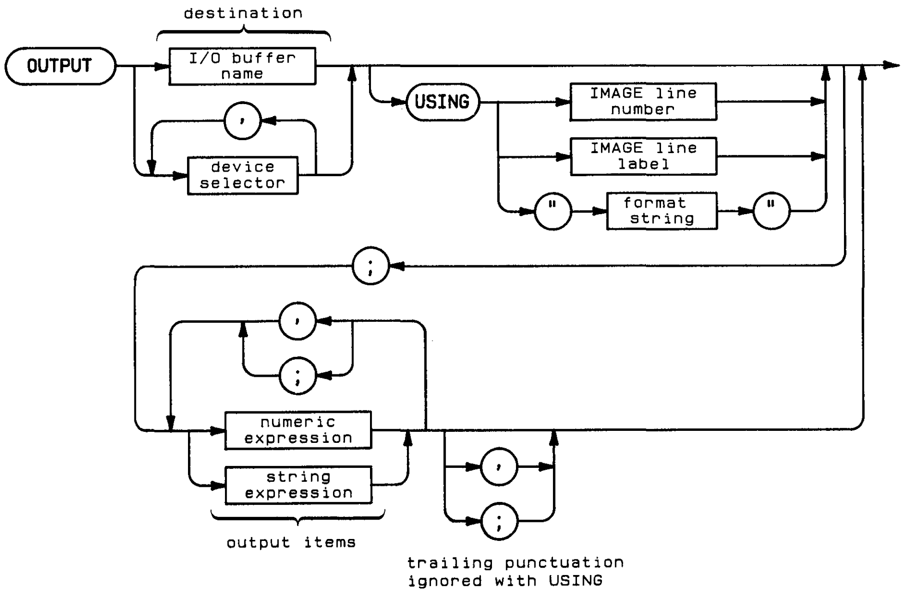
Related Keywords

BTD, DTB$, DTH$, DTO$, HTD

OUTPUT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The OUTPUT statement outputs items to the specified destination.



Item	Description	Range
device selector	numeric expression, rounded to an integer (see glossary)	—
I/O buffer name	name of a string variable declared as an I/O buffer	—
IMAGE line label	name identifying an IMAGE statement	any valid line label
IMAGE line number	integer constant identifying an IMAGE statement	1 through 65,535
format string	string expression consisting of one or more field specifiers (see IMAGE for syntax)	—
numeric expression	—	—
string expression	—	—

Examples

```
OUTPUT 701,702;"abcde", D1; Q$
OUTPUT Buffer$ USING 300;A,A$,B,"Hello"
```

Description

Bytes of numeric or string data are output to the specified device(s) or I/O buffer. If a CONVERT operation is enabled for that device or buffer, the conversion is performed immediately before the byte is output.

Simple OUTPUT (without USING). The simple OUTPUT statement (without USING) outputs items using two different field widths:

- When items are separated by semicolons, they are output in *narrow* format. Numbers are output in standard number format with a leading blank or minus sign and a trailing space. Strings are output with no leading or trailing blanks.
- When items are separated by commas, they are output in *free field* format, left-justified in 21-column fields. Numbers are output in standard number format with a leading space or minus sign. Trailing spaces are output to fill the unused portion of the field. Strings have no leading spaces; trailing spaces are added to fill the field.

Automatic End-of-Line Sequence. When the output list is exhausted, an end-of-Line (EOL) sequence, ordinarily carriage return/line feed, is sent. The EOL can be suppressed by placing the image specifier # at the beginning of the format string in the `OUTPUT USING` or `IMAGE` statement. The EOL sequence is also suppressed by placing a comma or semicolon at the end of the output list in a simple `OUTPUT` statement.

O **Formatted Output.** The `OUTPUT USING` statement uses a format string contained in the statement itself, or in an accompanying `IMAGE` statement, to format the output. The format string, consisting of one or more field specifiers separated by delimiters (, or /), is used from left to right. Output items are paired with their corresponding field specifiers. A field specifier consists of one or more image specifiers. Certain field specifiers do not use a display item (for example, X).

If the format string is exhausted before all the output items have been processed, the format string is reused from the beginning. Extra field specifiers are ignored. If a field is larger than the numeric item, the number is right-justified in the field.

A warning is issued if the number is larger than the field, and the number output may be incorrect. (A minus sign requires a digit position if M or S is not included in the field specifier.) Numbers are rounded to the number of decimal places indicated by the field specifier.

A trailing comma or semicolon after the last output item is ignored; trailing punctuation does not suppress the EOL sequence.

Refer to `IMAGE` for the syntax of the format string.

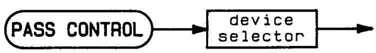
Related Keywords

`CONVERT, DISP, IMAGE, IOBUFFER, PRINT, TRANSFER`

PASS CONTROL

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `PASS CONTROL` statement passes Active Controller responsibility to the specified device.



Item	Description	Range
device selector	numeric expression, rounded to an integer (see glossary)	—

Examples

```
PASS CONTROL 710
PASS CONTROL 8
```

Description

Interface-dependent action:

- HP-IB:

  If the device selector is an interface select code, the specified device must be addressed to talk before PASS CONTROL is executed. Executing PASS CONTROL causes the interface to send the Take Control (TCT) message; ATN is set false.

  If the device selector contains a primary address, the interface sends the specified device's talk address, followed by the Take Control (TCT) message, and sets ATN false.

P

- HP-IL:

  If the device selector is an interface select code, the recieving device must be addressed to talk before PASS CONTROL is executed. Executing PASS CONTROL causes the interface to send the Take Control (TCT) message.

  If the device selector contains a primary address, the interface sends the specified device's talk address, followed by the Take Control (TCT) message.

- Serial, BCD, and GPIO: Error.

Related Keywords

ABORTIO, ENABLE INTR, ON INTR, REQUEST, RESET

PAUSE

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The PAUSE statement pauses program execution.

Description

When PAUSE is executed, program execution is suspended at the end of the current line. To resume execution, execute CONT.

If a halted program is edited, it must be initialized before execution can continue. To continue an edited program, use RUN, or INIT followed by CONT.

When a program is paused from the keyboard during execution of a multistatement line, the line is completed before the program halts. If the line includes a branching statement, execution halts at the end of the statement to which the program branched.

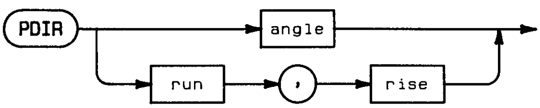
Related Keywords

CONT, INIT, RUN

PDIR

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The PDIR (*plot direction*) statement specifies a rotation of coordinates which is applied to incremental plotting (IPLOT, IMOVE, and IDRAW) and relative plotting (RPLOT).



P

Item	Description	Range
angle	numeric expression, interpreted according to the current trigonometric mode	—
run	numeric expression, interpreted according to the current scale units	—
rise	numeric expression, interpreted according to the current scale units	—

Examples

```
PDIR ACS(P(I))
PDIR 30,30
```

Description

The angle measures the conterclockwise rotation between the horizontal axis and the new x-axis. The run and rise parameters determine a vector drawn in the direction of the new x-axis.

Axes and labels are not affected by `PDIR`.

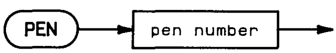
Related Keywords

`DEG`, `GRAD`, `RAD`

PEN

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The PEN statement selects a pen on the current plotting device.



Item	Description	Range
pen number	numeric expression, rounded to an integer	device dependent

Examples

```
PEN -1
PEN A
```

Description

On a periperal plotter, no checking is done to verify that the specified pen number exists. Pen 0 returns the current pen to the stall. Negative pen numbers are interpreted as pen 0.

When the display is the plotting device, pen numbers are interpreted as follows:

Graphics Display Pen Operation

Pen Number	Effect
≥ 1	Plots white dots.
0	Pen is deactivated and does not plot.
-1 or < -2	Plots black dots.
-2	Performs an exclusive OR, plotting white dots over black dots and black dots over white dots.

Related Keywords

GCLEAR

PENUP

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

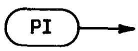
The `PENUP` statement lifts the pen on the current plotting device.

Description

After `PENUP` is executed, no drawing takes places until the pen is dropped manually or by executing a statement that drops the pen:

PLOT	IPLOT	RPLOT
DRAW	IDRAW	LABEL
XAXIS	YAXIS	AXES
GRID	LGRID	LAXES

The `PI` function returns the value of π with full machine precision.



Examples

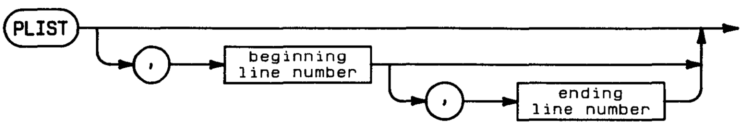
```
C=2*PI*R
IF A<2*PI THEN GOSUB 500
```

PLIST

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `PLIST` statement lists the current program or subprogram in system memory on the PRINTER IS device.

P



Item	Description	Range
beginning line number	integer constant	1 through 65,535
ending line number	integer constant	1 through 65,535

Examples

```
PLIST 100
PLIST 100,200
```

Description

The beginning line number and ending line number specify the portion of the program to be listed. If no ending line number is specified, listing begins at the beginning line number and continues for the entire (sub)program. When both parameters are omitted, the entire program is listed.

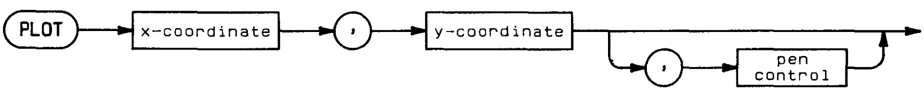
Related Keywords

`LIST`

PLOT

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `PLOT` statement moves the pen from the current pen position to the specified x- and y-coordinate position. The optional pen control parameter specifies the up/down status of the pen.



Item	Description	Range
x-coordinate	numeric expression, interpreted in the current units	—
y-coordinate	numeric expression, interpreted in the current units	—
pen control	numeric expression, rounded to an integer (default= +1; pen lowered after move)	—

Examples

```
PLOT X,Y,P
PLOT 5,10
```

Description

`PLOT` uses the current units (GU's or UU's) and line type. In UU's mode, lines cannot be drawn outside the plotting boundaries. In GU's mode, the plotting boundaries are equivalent to the graphics limits; therefore, lines can be drawn anywhere within the graphics limits.

In both UU's mode and GU's mode, `PLOT` can position the logical pen outside the plotting area. However, `PLOT` cannot position the physical pen outside the plotting boundaries. If none of the line is inside the current plotting area, the physical pen is not moved, but the logical pen position is updated.

The optional pen control parameter specifies the up and down position of the pen as follows:

Pen Control

Pen Control Parameter	Pen Action
positive, even	pen moved and then lifted
positive, odd	pen moved and then lowered
negative, even	pen lifted and then moved
negative, odd	pen lowered and then moved

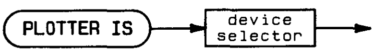
If no pen control parameter is specified, the up/down status of the pen before `PLOT` is executed determines whether the pen is up or down as it moves. If the pen is up, it is lowered when it reaches its new position.

Related Keywords

`IPLOT`, `LINE TYPE`, `RPLOT`

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The PLOTTER IS statement specifies the device to which graphics output is sent.



Item	Description	Range
device selector	numeric expression, rounded to the nearest integer (default=1)	see glossary

Examples

```
PLOTTER IS 705
PLOTTER IS I
```

Description

In addition to selecting the plotting device, the PLOTTER IS statement:

- Reads the graphics limits of the plotting device.
- Activates the graphics default conditions (see **graphics default conditions** in the glossary).

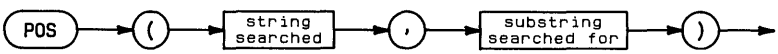
Related Keywords

ASSIGN, LIMIT

POS

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `POS` numeric function returns the position of the first character of a substring within another string.



P

Item	Description	Range
string searched	string expression	—
substring searched for	string expression	—

Examples

```
Index=POS(A$,"1")
DISP String$[POS(String$,"L"),30]
```

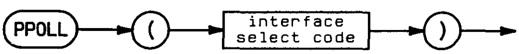
Description

If the substring searched for is the null string or is not contained within the string searched, `POS` returns 0. If the substring searched for occurs in more than one place, only the first occurence is returned.

PPOLL

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The PPOLL numeric function returns the results of a parallel poll operation.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10

Examples

```
ParPol=PPOLL(7)
IF PPOLL(Isc)=8 THEN GOSUB 300
```

Description

Interface-dependent action:

- HP-IB and HP-IL: The computer must be active controller. The value returned is a byte representing eight status-bit messages of devices on the interface bus. Each device capable of responding asserts one bit of the response byte.
- Serial, BCD, and GPIO: Error.

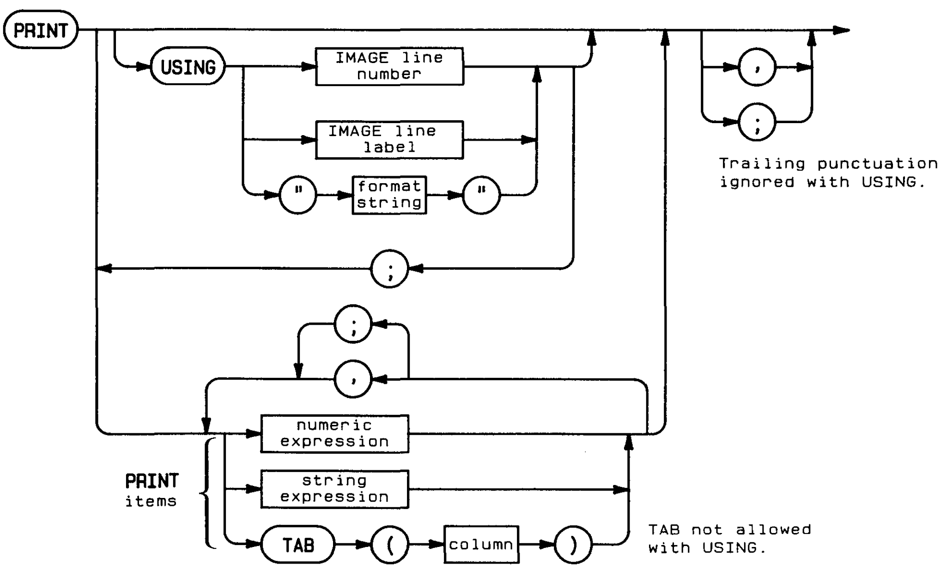
Related Keywords

SPOLL

PRINT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The PRINT statement outputs the print items to the current PRINTER IS device.



P

Item	Description	Range
IMAGE line number	integer constant identifying an IMAGE statement	1 through 65,535
IMAGE line label	name identifying an IMAGE statement	any valid line name
format string	string expression containing one or more field specifiers (see IMAGE statement for syntax)	—
column	numeric expression, rounded to an integer	negative numbers are interpreted as 1
numeric expression	—	—
string expression	—	—

Examples

```
PRINT Number; Letter$
PRINT TAB(10);A$,"Results=";Result
PRINT USING "DC3D.5D,4X,7A";A,"dollars"
PRINT USING 100; A,B$,C
```

Description

The keyword USING provides for specifying the format of output. When PRINT is executed without USING, a standard format is used.

Simple PRINT (Without USING). Simple PRINT uses standard number format (see glossary) for numeric items, and displays numeric and string items in either of two field widths:

- When display items are separated by semicolons, they are displayed in *narrow* format with a leading blank or minus sign. Strings are output with no leading or trailing blanks.
- When display items are separated by commas, they are displayed in *wide* format, left-justified in 21-column fields. Items longer than 21 characters occupy more than one field.

When the TAB function is included as a print item, the column parameter positions the next character on the print line. Negative column numbers are treated as TAB(1). Column numbers greater than the line length are reduced MOD (line length). When TAB is used to control format, display items should be separated by semicolons; using commas causes output to be displayed in wide format.

When the list of print items is exhausted, an end-of-line (EOL) sequence, ordinarily carriage return/line feed, is sent to the printer. The EOL can be suppressed by including a comma or semicolon after the last print item.

P

Control Characters and Alternate Character Sets. Control characters are included as print items by specifying their ASCII code as argument in the CHR$ function or by using the metacharacter ~ followed by the character decimal code.

Formatted Output. The PRINT USING statement uses a format string contained in the statement itself or in a referenced IMAGE statement to format the output. The format string, consisting of one or more field specifiers separated by delimiters (comma or slash), is used from left to right. Print items are paired with their corresponding field specifiers. Certain field specifiers do not use a print item (for example, X).

If the format string is exhausted before all the print items have been processed, the format string is reused from the beginning. Extra field specifiers are ignored. If a field is larger than the numeric item, the number is right-justified in the field. A warning is issued if the number is larger than the field. (A minus sign requires a digit position if M or S is not included in the field specifier.) Numbers are rounded to the number of decimal placed indicated by the field specifier. Standard number format can be chosen by using the image specifier K.

TAB cannot be used with PRINT USING.

When the list of print items is exhausted, an end-of-line (EOL) sequence, ordinarily carriage return/line feed, is sent to the display. The EOL can be suppressed by placing the image specifier # at the beginning the format string in the PRINT USING or IMAGE statement. Unlike with simple PRINT, a terminating semicolon or comma is ignored and does not suppress the EOL sequence.

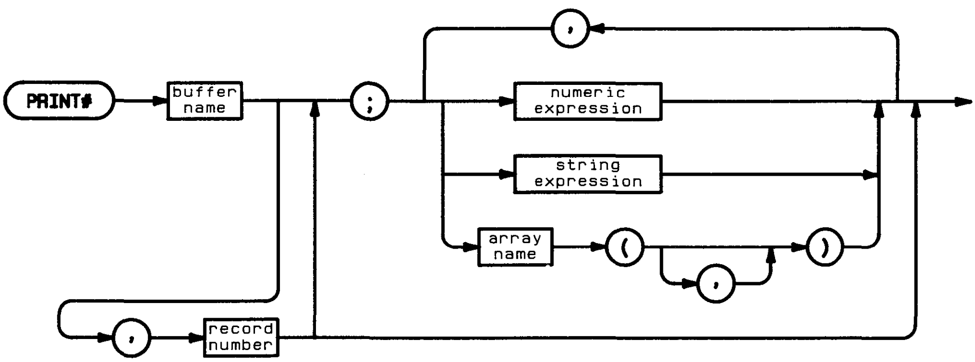
Refer to IMAGE for the syntax of the format string.

Related Keywords

DISP, IMAGE, OUTPUT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The PRINT# statement outputs data to an open BASIC/DATA file.



Item	Description	Range
buffer number	numeric expression, rounded to an integer	1 through 10
record number	numeric expression, rounded to an integer	—
numeric expression	(see glossary)	—
string expression	(see glossary)	—
array name	name of a numeric or string array	any valid name

Examples

```
PRINT# 1; Variable
PRINT# BufferNumber, record; A(4)*7, B$[7,12]
```

Description

The buffer number must have been previously assigned to the file with an ASSIGN# statement. The ASSIGN# statement places the file pointer at the beginning of the file.

Serial Access. When the record number is omitted, data is written serially. In serial access, data is written to the file sequentially; items are placed in the next logical record when the current record becomes full.

As each PRINT# item is written into the file, the file pointer advances beyond that data. When the entire list of PRINT# items has been written, the file pointer remains positioned after the last data item read and an end-of-file marker is positioned there. A subsequent PRINT# statement continues writing data from that position.

Serial printing continues until all the data is printed, or until the medium is full. The data file is automatically expanded, if necessary, to accommodate all the PRINT# items. Serial printing also halts when the file is closed, or when a random access READ# or PRINT# is executed.

Random Access. When the record number is included, data is written using random access. The record number must not exceed the total number of records in the file.

When the PRINT# statement is executed, the file pointer is moved to the beginning of the specified logical record. As an item of data is written into the record, the file pointer advances to the next position in the record and an end-of-record marker is placed in that position. A random PRINT# operation cannot extend across logical record boundaries. An error is returned if the file pointer moves beyond the end of the record.

Executing a random access PRINT# without a list of data causes the file pointer to move to the beginning of the specified logical record.

Related Keywords

ASSIGN#, READ#

PRINT ALL

Keyboard Executable	■
Programmable	■
In an IF...THEN	■

The PRINT ALL statement produces a printed copy of alphanumeric information as it is displayed on the alpha display.

Description

PRINT ALL directs a copy of all displayed alphanumeric output to the PRINTER IS device. This includes output from DISP, DISP USING, and LIST, keyboard input, and error messages generated from the keyboard or from a running program.

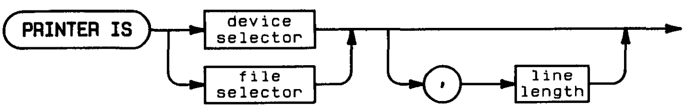
Related Keywords

CRT IS, PRINTER IS

PRINTER IS

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The PRINTER IS statement selects the destination for PRINT and PLIST output.



P

Item	Description	Range
device selector	numeric expression, rounded to an integer	see glossary
file selector	numeric expression, rounded to an integer	11 through 20
line length	numeric expression, rounded to an integer (default=80)	1 through 220

Examples

```
PRINTER IS 701
PRINTER IS P1
```

Description

Output from `PRINT` (`USING`) and `PLIST` is sent to the PRINTER IS device or to the specified file. The alpha display is the default printing device at power-on.

The line length specifies the maximum number of characters sent to the PRINTER IS device before an end-of-line (EOL) sequence is automatically sent. The EOL character(s) are not counted as part of the line length. When a PRINT USING format string specifies output that exceeds the PRINTER IS line length, the line is broken at the line length and the format is continued at the beginning of the next line.

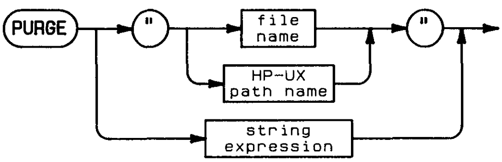
Related Keywords

`ASSIGN, IMAGE, PLIST, PRINT`

PURGE

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The PURGE statement deletes the entry for the specified file from its directory.



P

Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
PURGE "myfile"
PURGE "/vol1/dir1/dir2/myfile"
```

Description

If the file name is used alone rather than as part of an HP-UX path name, the file must be located in the current working directory.

A purged file can no longer be accessed. The space previously occupied by the file becomes available for creation of other files.

Related Keywords

ASSIGN#

RAD

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The RAD statement sets radians as the unit in which angles are measured.

Description

When RAD is executed, all angle parameters in statements and functions are interpreted as radians. (There are 2π radians in a circle.) All functions returning an angle return a value in radians.

The angle mode of a program is global. When a subprogram is called, the current angle mode is carried into the subprogram. If a subprogram changes the angle mode and then returns to the main program, the new mode is carried back to the main program.

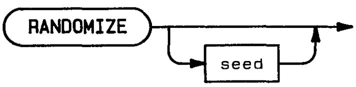
Related Keywords

DEG, GRAD

RANDOMIZE

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The RANDOMIZE statement specifies a new seed for the RND function.



Item	Description	Range
seed	numeric expression, rounded to an integer	range of integers

R

Examples

```
RANDOMIZE
RANDOMIZE Seed
```

Description

The seed determines the sequence of pseudorandom numbers generated. Using the same seed causes RND to generate the same series of numbers.

The seed is global, and is passed between the main program and any subprogram(s).

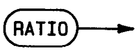
Related Keywords

RND

RATIO

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `RATIO` function returns the ratio of the dimensions of the graphics limits—horizontal dimension divided by vertical dimension.



Examples

```
R=RATIO
LOCATE 5,RATIO*20,10,50
```


Description

The graphics limits from which `RATIO` is computed are set by executing `PLOTTER IS` or `LIMIT`.

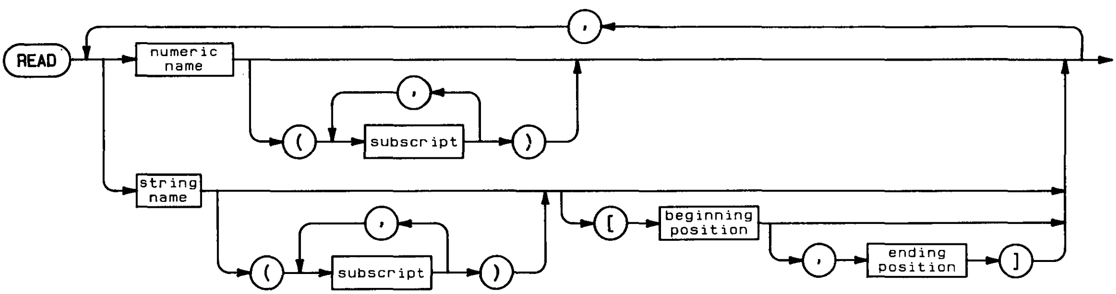
Related Keywords

`LIMIT`, `PLOTTER IS`

READ

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The READ statement reads numeric and/or string constants from one or more DATA statements and assigns those values to program variables.



Item	Description	Range
numeric name	name of a simple numeric variable or numeric array	any valid name
string name	name of a simple string variable or string array	any valid name
subscript	numeric expression, rounded to an integer	1 through 65,530
beginning position	numeric expression, rounded to an integer	1 through 65,530
ending position	numeric expression, rounded to an integer	1 through 65,530

Examples

```
READ Variable1,Variable2$
READ A(1,2),B,C$,D$[3,5],E$(4)[7]
```

Description

READ uses a data pointer to indicate the data item to be read. When program execution begins, the data pointer is positioned at the left-most item in the lowest-numbered DATA statement. When the data list in a particular DATA statement is exhausted, the pointer moves to the next-higher numbered DATA statement. Attempting to read past the last data item in the program generates an error.

The order in which DATA statements are used can be changed using the RESTORE statement.

Each subprogram has its own data pointer, and can use only its own DATA statements. When a subprogram is called, its first READ statement uses the first DATA statement in that subprogram. When execution returns to a calling program, the calling program resumes use of its own data pointer starting from the pointer's last position.

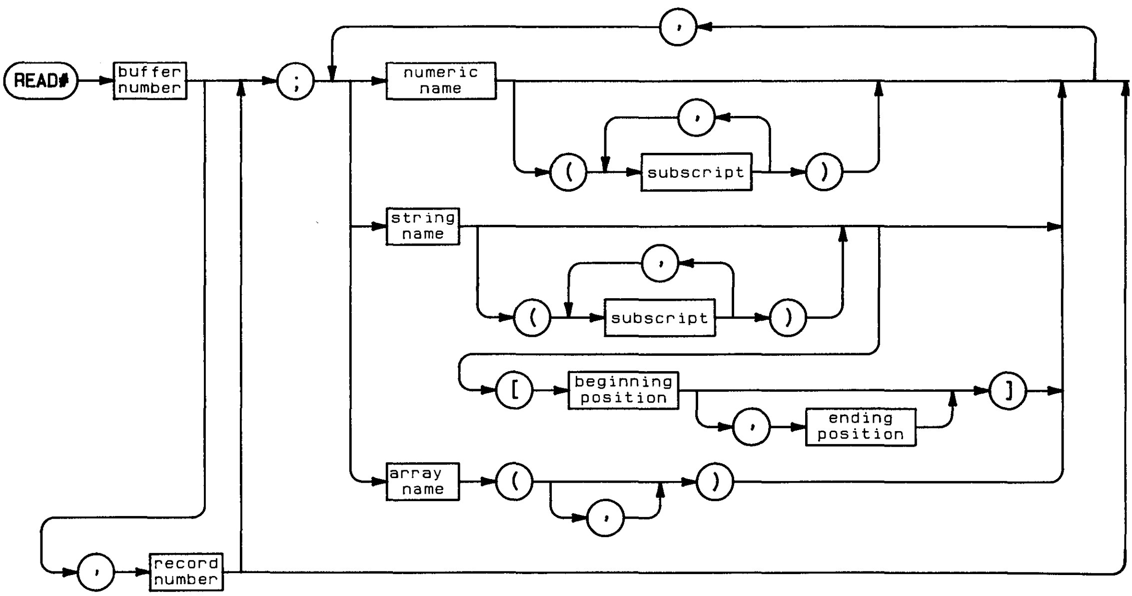
Related Keywords

DATA, RESTORE

READ#

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `READ#` statement retrieves data from an open BASIC/DATA file and assigns the data to the specified variable(s).



Item	Description	Range
buffer number	numeric expression, rounded to an integer	1 through 10
record number	numeric expression, rounded to an integer	none
numeric name	name of a simple numeric variable or numeric array element	any valid name
string name	name of a simple string variable or string array element	any valid name
array name	name of a numeric or string array	any valid name
subscript	numeric expression, rounded to an integer	1 through 65,530
beginning position	numeric expression, rounded to an integer	1 through 65,530
ending position	numeric expression, rounded to an integer	1 through 65,530

R

Examples

```
READ# 1;Variable
READ# BufferNumber,record;A(4),B$[7,12]
```

Description

The buffer number must have been previously assigned to the file with an `ASSIGN#` statement. The `ASSIGN#` statement places the file pointer at the beginning of the file.

Data read from the file must match the READ# variables in type (numeric versus string). Numeric data need not agree in precision. The data is converted to the precision of the READ# variable.

Serial Access. When the record number is omitted, data is read serially. As an item of data is read from the file into a READ# variable, the file pointer advances to the next item. When the entire list of READ# variables has been satisfied, the file pointer remains positioned after the last data item read. A subsequent `READ#` statement continues reading data from that position. Serial access continues until the file is closed, all the data has been read, or a random access `READ#` or `PRINT#` statement is executed.

Random Access. When the record number is included, data is read using random access. The record number must not exceed the total number of records in the file.

When the random `READ#` statement is executed, the file pointer is moved to the beginning of the specified logical record. As an item of data is read from the record into a READ# variable, the file pointer advances to the next item in the record. A random READ# operation cannot extend across logical record boundaries. An error is returned if the file pointer encounters the end of the logical record before all the READ# variables have been satisfied.

Executing a random access `READ#` without a list of variables moves the file pointer to the beginning of the specified logical record.

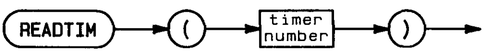
Related Keywords

`ASSIGN#`, `PRINT#`

READTIM

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The READTIM function returns the integer number of seconds elapsed on the specified system timer after the timer is set by an ON TIMER# statement in a program.



Item	Description	Range
timer number	a numeric expression, rounded to an integer	0 through 3

Examples

```
IF READTIM(1)<5 THEN GOSUB SendData
DISP READTIM(A)
```

Description

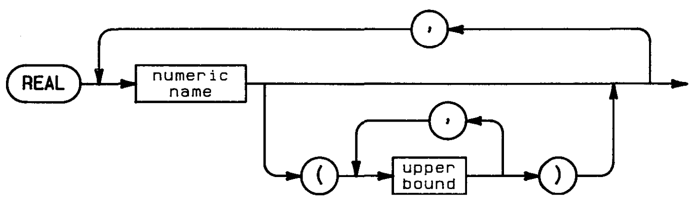
If the timer has not been set or has been disabled by OFF TIMER#, READTIM returns 0. Timer #0 is the system clock; READTIM(0) returns the value of the clock seconds counter.

Related Keywords

OFF TIMER#, ON TIMER#, SETTIME

Keyboard Executable ■
Programmable ■
In an IF...THEN □

The REAL statement declares and reserves memory for full precision floating point numeric variables.



Item	Description	Range
numeric name	name of a simple numeric variable or numeric array	any valid name
upper bound	integer constant	1 through 65,530

Examples

```
REAL Variable,Array1(10),Array2(5,3)
```

Description

All numeric variables are REAL unless declared SHORT or INTEGER.

When the numeric variable name is used with one or two upper bound(s) enclosed in parentheses, the variable is dimensioned to be a one- or two-dimensional array. The default lower bound of the array is 0. The `OPTION BASE` statement is used to set the lower bound equal to 1.

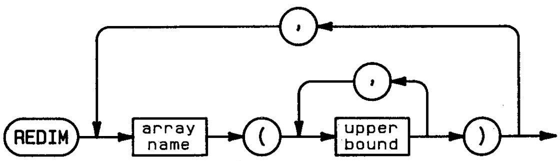
When variables are passed to a subprogram by address, precision declarations accompany the variable into the subprogram.

Related Keywords

`DIM, INTEGER, SHORT`

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `REDIM` statement changes the subscript range of a previously dimensioned array.



Item	Description	Range
array name	name of a numeric array	any valid name
lower bound	numeric expression, rounded to an integer (default=option base value)	1 through 65,530
upper bound	numeric expression, rounded to an integer	1 through 65,630

R

Examples

```
REDIM A(3)
REDIM FirstArray(4,5),SecondArray(5)
```

Description

Redimensioning an array reassigns elements to different positions in the array. Elements are stored in order from left to right along each row, from the top row to the bottom.

The following rules apply to redimensioning arrays:

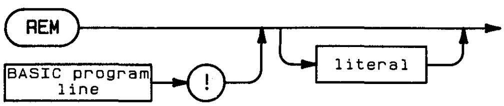
- The number of dimensions of the array must not change.
- The total number of elements in the new working size cannot exceed the number originally dimensioned.

If `REDIM` specifies an array that has not yet been explicitly dimensioned, the array is first dimensioned to a two-dimensional array with upper bounds equal to 10, and then immediately redimensioned.

REM

Keyboard Executable	☐
Programmable	■
In an `IF...THEN`	☐

The `REM` statement allows comments in a program.



Item	Description	Range
literal	string constant composed of characters from the keyboard	characters with ASCII codes 0 through 31 not allowed
BASIC program line	a proper BASIC statement or multistatement line	—

Examples

```
10 REM  Written 12/5/83
20 !
30 DISP "Insert disc in drive" ! User must insert disc #4
```

Description

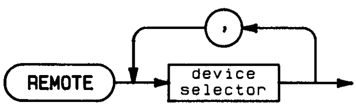
The comment delimiter, `!`, can be used anywhere after the line number; all characters following the delimiter are considered part of the comment.

If a `REM` statement is included in a multistatement line, it must be the last statement in the line.

REMOTE

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The REMOTE statement places the specified device(s) into remote control.



Item	Description	Range
device selector	numeric expression, rounded to an integer (see glossary)	—

R

Examples

```
REMOTE 710
REMOTE A1,A2,A3
```

Description

If two or more device selectors are listed, they must include primary addresses, and the devices must be on the same interface. If the device selector is an interface select code, the remote state is enabled for all devices on the bus having remote/local capabilities.

Interface-dependent action:

- HP-IB: The computer must be system controller. The bus is placed into remote operation.

  If the device selector is an interface select code, the interface sets Remote Enable (REN) true. Devices do not go into remote state until they are addressed to listen.

  If the device selector contains a primary address, the interface sets REN true, sends Unlisten (UNL), and then sends the listen address of the specified device(s). `REMOTE` leaves ATN true.
- HP-IL: The computer must be active controller.

  If the device selector is an interface select code, Remote Enable (REN) is sent.

  If the device selector(s) include a primary address, Remote Enable (REN), Unlisten (UNL), and Listen Address (LAD) are sent.
- BCD: Sets a partial field separator.
- Serial and GPIO: Error.

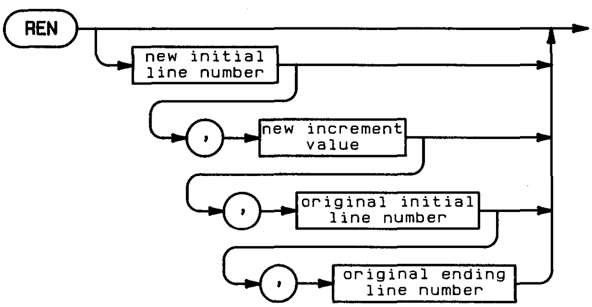
Related Keywords

`LOCAL`, `LOCAL LOCKOUT`, `RESUME`

REN

Keyboard Executable ■
Programmable □
In an IF...THEN □

The REN command renumbers all or portions of the current program or subprogram.



R

Item	Description	Range
new initial line number	integer constant (default=10)	1 through 65,535
new increment value	integer constant (default=10)	1 through 65,535
original initial line number	integer constant (default=1)	1 through 65,535
original ending line number	integer constant (default=65,535)	1 through 65,535

Examples

```
REN 500,2,1,60000
REN 10,1
```

Description

The program lines to be renumbered are delimited by the original initial line number and the original ending line number. Both original line numbers must exist in the program. The first line in the delimited segment is assigned the new initial line number. Successive lines are renumbered according to the specified new increment value. An error occurs if renumbering causes the new ending line number to exceed 65,535, or if either original line number does not exist.

When `REN` changes a line number, all references to that line number within the (sub)program (for example, `GOTO` *line number*) are automatically updated.

`REN` cannot be used to change the order of program lines. An error occurs if renumbering causes newly renumbered program lines to overlap previous or following lines. In the case of an error, renumbering halts and line numbers are returned to their original values.

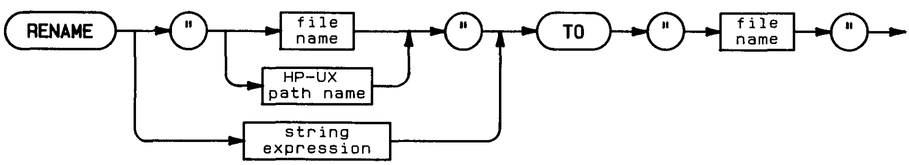
Related Keywords

`SCAN`, `XREF L`

RENAME

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `RENAME` statement changes the name of the specified file in its directory.



R

Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed, BASIC/SUBP files not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	BASIC/SUBP files not allowed
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
RENAME "name1" TO "name2"
RENAME "/disc1/oldname" TO "newname"
```

Description

RENAME removes the old name from the directory and replaces it with the new name. The parameter following TO must be a simple file name.

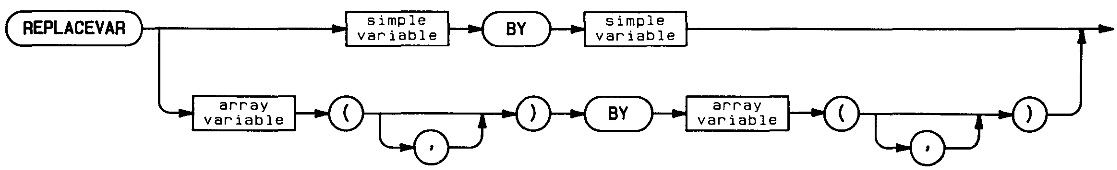
If the old file name is used alone rather than as part of the HP-UX path name, the file must be located in the current working directory.

R

REPLACEVAR

Keyboard Executable ■
Programmable □
In an IF...THEN □

The REPLACEVAR command replaces all occurrences of the specified variable name in a program or subprogram with another variable name.



R

Item	Description	Range
simple variable name	simple numeric or string variable name	any valid name
array variable name	numeric or string array name	any valid name

Examples

```
REPLACEVAR A BY B
REPLACEVAR string$() BY twine$()
```

Description

The new variable name must match the replaced variable name in type—simple numeric, simple string, numeric array, or string array. A one-dimensional array variable is indicated by parentheses following the variable name. For two-dimensional arrays, a comma must be included within the parentheses.

The messages `Replacing...` and `...end of replace` indicate the beginning and end of the replacement operation.

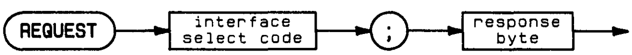
Related Keywords

`SCAN, XREF L, XREF V`

REQUEST

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `REQUEST` statement is used by the non-active controller to send a response byte to the active controller.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
response byte	numeric expression, truncated to an integer and moduloed 256	—

Examples

```
REQUEST 7;64
REQUEST Isc;64+X
```

Description

Interface-dependent action:

- HP-IB and HP-IL: The computer must be non-controller. Executing `REQUEST` sets up a serial poll response byte, which is sent to the active controller in response to a serial poll operation. If bit 6 (decimal value 64) of the response byte is set, the computer sends Service Request (SRQ) to the active controller in response to the incoming serial poll. The active controller's serial poll clears SRQ.

- Serial: A BREAK, defined by the response byte, is sent. The transmit line is held in a space condition (0-state) for the number of character times specified in the response byte, followed by a mark condition (1-state) for five character times.
- BCD and GPIO: Error.

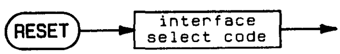
Related Keywords

`PASS CONTROL`, `SPOLL`

RESET

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `RESET` statement performs a hardware reset of the interface, returning it to its power-on state.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10

Examples

`RESET 7, RESER Isc`

Description

When `RESET` is executed, the interface performs a self-test, and the control registers are set to their default values.

If EOT branching is enabled, resetting the interface during an active transfer causes the branch to be taken.

Interface-dependent action:

- HP-IB: If the computer is system controller, HP-IB sends Interface Clear (IFC), then Remote Enable (REN).
- Serial: Modem control lines are turned off.
- BCD: Data lines are set to high-impedance state, handshake lines are set false, and I/O lines are set to input state.
- GPIO: Ports A and B are set to high-impedance state, ports C and D are set to off state, CTL lines are set false, and OUTA and OUTB are set to indicate output.
- HP-IL: If the interface is system controller, Interface Clear (IFC), Auto Address Unconfigure (AAU), and Auto Address 1 (AAD1) are sent, followed by Not Remote Enable (NRE) and Remote Enable (REN).

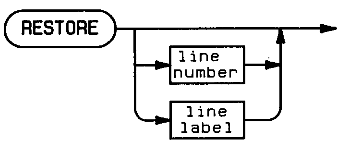
Related Keywords

ABORTIO, HALT, ON EOT

RESTORE

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The RESTORE statement specifies which DATA statement will be accessed by the next READ operation.



R

Item	Description	Range
line number	integer constant identifying a program line (default=first DATA statement in a program or subprogram)	1 through 65,535
line label	name of a program line	any valid name

Examples

```
100 RESTORE
200 RESTORE 130
```

Description

The specified statement must be a DATA statement located in the same program or subprogram. When that data statement has been used, the data pointer moves to the next-higher numbered DATA statement.

Related Keywords

DATA, READ

RESUME

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The RESUME statement re-enables I/O operations after they have been disabled by execution of HALT or SEND.

Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10

Examples

```
RESUME 7
RESUME Isc
```

Description

Interface-dependent action:

- HP-IB: The computer must be active controller. The ATN line is set false.
- Serial: The transmitter is enabled (refer to the interface documentation for additional information).
- HP-IL: The computer must be active controller. The Send Data (SDA) message is sent if a transfer is not already in progress.
- BCD and GPIO: Error.

Related Keywords

CONTROL, HALT, SEND

RETURN

Keyboard Executable □
Programmable ■
In an IF...THEN ■

The RETURN statement is used within a subroutine to cause branching to the statement following the invoking GOSUB.

RETURN →

Description

When an invoking GOSUB (or ON...GOSUB) is embedded in a multistatement line, RETURN returns program execution to the statement following the GOSUB on that line. A GOSUB interrupt (for example, ON KEY# 5 GOSUB 100) returns execution to the line following the line on which the interrupt occurred.

Related Keywords

GOSUB, ON...GOSUB

REV$

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `REV$` function returns a string formed by reversing the sequence of characters in the specified string.

Item	Description	Range
string argument	string expression	—

Examples

```
Backwards$=REV$("ABCDE")
DISP REV$(String$[2,7])
```

RMD

The `RMD` function divides the first numeric argument by the second numeric argument and returns the remainder from the division.

Item	Description	Range
dividend	numeric expression	—
divisor	numeric expression	—

Examples

```
ANGLE=RMD(A,360)
IF RMD(X,Y)=0 THEN 300
```

Description

For non-zero values of Y, RMD(X,Y) returns a value according to the equation:

$$RMD(X,Y) = X\text{-}Y * IP(X/Y)$$

When y=0, RMD(X,Y)=X. `RMD` and the `MOD` operator return the same result when X and Y have the same sign.

Related Keywords

`MOD`

RND

The RND function returns a pseudorandom number greater than or equal to 0 and less than 1.

Examples

```
IF RND>.5 THEN DISP "HEADS"
```

Description

The sequence of random numbers returned depends on the seed. BASIC uses a default seed whenever the system is reset. The RANDOMIZE statement is used to change the seed.

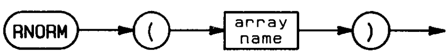
Related Keywords

RANDOMIZE

RNORM

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The RNORM function returns the *row norm* of an array. The row norm is computed by summing the absolute values of the elements in each row of the array and selecting the largest value.



Item	Description	Range
array name	name of a one- or two-dimensional array	any valid name

R

Examples

```
SUM=RNORM(Array1)
IF RNORM(A)üRNORM(C) THEN B=RNORM(A)
```

Related Keywords

CNORM, FNORM, RNORMROW

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `RNORMROW` function returns the row number of the row having the largest sum of absolute values, using the array specified in the most recently executed `RNORM` function.

RNORMROW →

Examples

```
A(RNORMROW,3) = 2.5E4
DISP RNORMROW
```


Description

Row numbering starts with zero for option base 0.

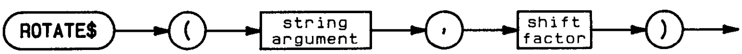
Related Keywords

`ABSUM`, `RNORM`

ROTATE$

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The ROTATE$ function shifts the characters in a string by the specified number of positions, rotating characters "bumped" off one end of the string to the other end.



Item	Description	Range
string argument	string expression	—
shift factor	numeric expression, rounded to an integer	—

Examples

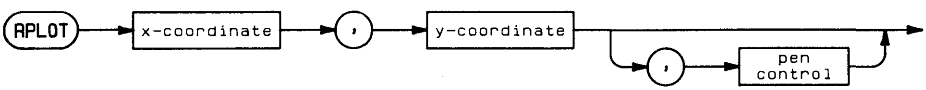
```
DISP ROTATE$("ABCDEFG",2)
IF ROTATE$(Line1$,1)="x" THEN Y=2
```

Description

The sign of the shift factor determines which way characters are rotated. A positive shift factor causes characters to be right-shifted, with characters at the end of the string rotated to the beginning. A negative shift factor causes characters to be left-shifted, with characters at the beginning of the string rotated to the end.

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The RPLOT statement moves the pen from the current pen position to the specified x- and y-coordinate position, using a local coordinate origin. The optional pen control parameter specifies the up/down status of the pen.



Item	Description	Range
x-coordinate	numeric expression, interpreted in the current units	—
y-coordinate	numeric expression, interpreted in the current units	—
pen control	numeric expression, rounded to an integer (default=1; pen lowered after move)	—

Examples

```
RPLOT X,Y,P
RPLOT 5,10
```

Description

The x- and y-coordinates are interpreted as increments to a local origin. RPLOT does not affect the local origin.

The local origin is the current logical pen position at the completion of any of the following statements:

```
AXES   DRAW   FRAME   GRID   IDRAW   IMOVE   IPLOT   LABEL   MOVE   PLOT
```

RPLOT uses the current units (GU's or UU's) and line type. In UU's mode, lines cannot be drawn outside the plotting boundaries. In GU's mode, the plotting boundaries are equivalent of the graphics limits; therefore, lines can be drawn anywhere within the graphics limits.

In both UU's mode and GU's mode, RPLOT can position the logical pen outside the plotting area. However, RPLOT cannot position the physical pen outside the plotting boundaries.

The optional pen control parameter specifies the up and down position of the pen as follows:

Pen Control

Pen Control Parameter	Pen Action
positive, even	pen moved and then lifted
positive, odd	pen moved and then lowered
negative, even	pen lifted and then moved
negative, odd	pen lowered and then moved

If no pen control parameter is specified, the up/down status of the pen before RPLOT is executed determines whether the pen is up or down as it moves. If the pen is up, it is lowered when it reaches its new position.

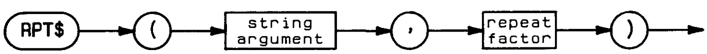
Related Keywords

IPLOT, LINE TYPE, PLOT

RPT$

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The RPT$ function returns a string consisting of the string argument repeated the specified number of times.



Item	Description	Range
string argument	string expression	—
repeat factor	numeric expression, rounded to an integer	—

Examples

```
DISP RPT$(String$,5)
Q$=RPT$("12345  ",N)
```

Description

A repeat factor less than +1 returns a null string. A repeat factor that produces a result string greater than 65,530 characters causes an error.

RTD

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The RTD (*radians-to-degrees*) function interprets the numeric argument as an angle measured in radians, and returns the value of the angle in degrees.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Degrees= RTD(Radians)
DISP RTD(PI*B)
```

Description

The argument and value returned by RTD are independent of the current trigonometric mode.

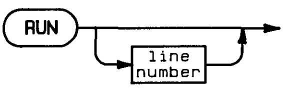
Related Keywords

DTR

RUN

Keyboard Executable	■
Programmable	□
In an `IF...THEN`	□

The `RUN` command starts program execution from the beginning or from the specified line.



Item	Description	Range
line number	integer constant (default=first program line)	1 through 65,535

Examples

```
RUN
RUN 4500
```

Description

If a line number is specified, it must be a valid line number in the main program. If the main program does not contain the specified line, execution starts at the next higher number line. An error results if no higher numbered line exists.

Execution of `RUN` occurs in two steps—prerun initialization and program execution. During prerun initialization:

- Memory is allocated to all program variables, and the variables are set to 0 and the null string.
- Any variable assignments previously made from the keyboard are scratched.
- The program is checked for prerun errors; for example, referencing a non-existent statement, duplicate user-defined functions, dimensioning the same variable more than once.

If an error is detected, prerun halts and an error message is returned.

When prerun initialization is completed, program execution begins. If the specified line number does not exist, execution begins with the next higher numbered line.

Refer to the table of Reset Conditions on pages 4-8 and 4-9 for additional information.

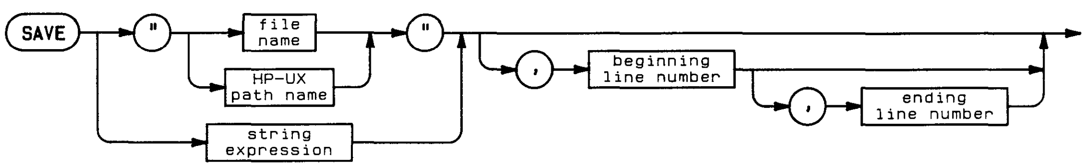
Related Keywords

`CONT, INIT, PAUSE`

SAVE

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SAVE` statement converts program lines currently in memory to ASCII character strings and copies the strings to the specified text file.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—
beginning line number	integer constant (default=first program line)	1 through 65,535
ending line number	integer contant (default=last program line)	1 through 65,535

Examples

```
SAVE "porpoises"
SAVE "OurShip",50,200
```

Description

If the specified file of the proper type already exists, the saved lines are copied to that file, erasing and overwriting the current contents. If the file does not exist, it is created in the specified directory. The current working directory is used if the file name is used without an HP-UX path name.

The beginning line number and ending line number specify the portion of the program to be saved. If the ending line number is omitted, lines from the beginning line number to the end of the program are saved. If both parameters are omitted, the entire program is saved.

The text files created and accessed by `SAVE` are non-BASIC files.

Related Keywords

`GET`, `STORE`

SCALE

Keyboard Executable	■
Programmable	■
In an `IF...THEN`	■

The `SCALE` statement specifies a user units scale of the plotting area.

Item	Description	Range
x-min	numeric expression	—
x-max	numeric expression	—
y-min	numeric expression	—
y-max	numeric expression	—

Examples

```
SCALE 0,100,0,100
SCALE G,G+300,G-50,2G
```

Description

`SCALE` scales the current plotting area, which is a function of the units mode (GU's or UU's) and the previously executed statements.

- In GU's mode, `SCALE` scales the entire graphics area previously specified by `PLOTTER IS` or `LIMIT`.
- In UU's mode, `SCALE` scales the plotting area previously specified by `LOCATE`. If `LOCATE` has not been executed, the entire graphics area is scaled.

The `SCALE` statement must be executed *after* the plotting area (graphics limits or LOCATE-defined area) has been established. Regardless of the current units mode, executing `SCALE` leaves the system in UU's mode.

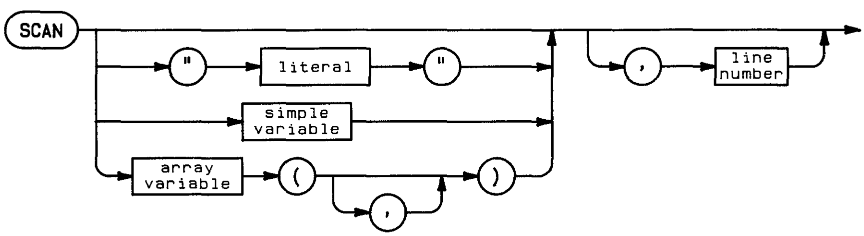
SCALE parameters can be exchanged to reflect the plot (see **LIMIT**).

Related Keywords

`LIMIT, LOCATE, MSCALE, SHOW, PLOTTER IS, SETGU, SETUU`

Keyboard Executable ■
Programmable □
In an IF...THEN □

The SCAN command searches the current program or subprogram and displays all lines containing the specified variable name or character string. The messages `Scanning...` and `...end of scan` indicate the beginning and end of the scan operation.



Item	Description	Range
literal	string constant composed of characters from the keyboard	—
simple variable	name of simple numeric or string variable	—
array variable	name of numeric or string array	—
line number	integer constant identifying a program line (default=first program line)	1 through 65,535

S

Examples

```
SCAN A()
SCAN "CALL", 2000
```

Related Keywords

REPLACEVAR, XREF L, XREF V

SCRATCH

Keyboard Executable ■
Programmable □
In an `IF...THEN` □

The `SCRATCH` command erases portions of computer memory, including the current BASIC program, subprogram(s), and variable assignments.

Description

Executing `SCRATCH`:

- Erases the current BASIC program.
- Erases any subprograms in memory.
- Erases all variable assignments made from the keyboard or within programs, including common variables.
- Cancels all I/O buffer and mass storage buffer assignments.

S

Binary programs are not affected.

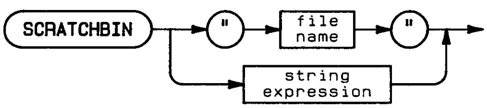
Refer to the table of Reset Conditions on pages 4-8 and 4-9 for further information.

Related Keywords

`INIT`, `SCRATCHSUB`

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SCRATCHBIN` statement erases the specified binary program from BASIC memory and reclaims the memory used by the binary.



Item	Description	Range
file name	literal; name of the binary program	14 characters maximum; slash and leading colon not allowed
string expression	expression evaluating to a file name	—

Examples

```
SCRATCHBIN "thisbinary"
SCRATCHBIN A$
```

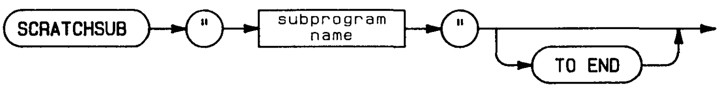
Related Keywords

`CALLBIN`

SCRATCHSUB

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The SCRATCHSUB statement scratches the specified subprogram(s) from system memory.



Item	Description	Range
subprogram name	name of the subprogram to be scratched	14 characters maximum; slash and leading colon not allowed

Examples

```
SCRATCHSUB "SubSort"
SCRATCHSUB "DeleteData" TO END
```

Description

SCRATCHSUB deletes the specified subprogram(s) without affecting the main program or other subprograms. When SCRATCHSUB is executed without the optional TO END keywords, only the specified subprogram is scratched.When SCRATCHSUB is executed from the keyboard with the optional TO END keywords, the specified subprogram and all subprograms located after it in the directory listing are scratched.

SCRATCHSUB can be executed within the main program or within subprograms. However, a subprogram cannot scratch itself ar any subprogram from which it was directly or indirectly called.

Related Keywords

DIRECTORY, SCRATCH

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SEC` function returns the secant of the angle argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
C=SEC(Angle)
IF SEC(Angle)=T THEN 400
```

Description

The angle argument is interpreted according to the current trigonometric mode—RAD (radians), DEG (degrees), or GRAD (grads). The default mode is RAD.

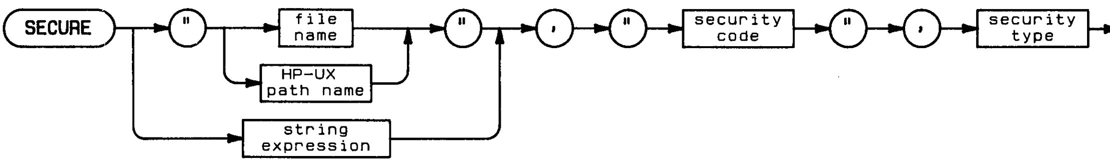
Related Keywords

`DEG`, `GRAD`, `RAD`

SECURE

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The SECURE statement secures BASIC files against being listed, copied, or overwritten.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—
security code	string expression; only the first two characters are used	⩾2 characters
security type	numeric expression, rounded to an integer and moduloed 4	security type 3 is ignored

Examples

```
SECURE "myfile", "nl",0
SECURE "/vol1/dir1/dir2/dir3/myfile",Sc$,2
```

Description

If the file name is used alone (rather than as part of an HP-UX path name), the file must be in the current working directory.

The security code is associated with the file for security types 0 and 1 only. The first two characters are used; any others are ignored.

Non-BASIC files cannot be secured within BASIC.

File Security

Security Type	File Type	Protection
0	BASIC/PROG BASIC/SUBP	Prevents LIST, PLIST, and editing.
1	BASIC/PROG BASIC/SUBP	Prevents LIST, PLIST, editing, and file-to-file COPY. The file is ignored during directory-to-directory COPY.
2	BASIC/PROG BASIC/SUBP BASIC/DATA BASIC/GRAF	Prevents the file from being overwritten by STORE, GSTORE, or PRINT#.

A file can be secured with types 0, 1, and 2 security at the same time. However, a file cannot be secured twice with the same security type.

Files can be secured against cataloging by using a period as the first character of the file name. The file will not be listed in a directory catalog. However, the file itself can be cataloged (i.e., CAT "filename" or CAT "HP-UX path name").

Regardless of the security status of a file, it can always be purged.

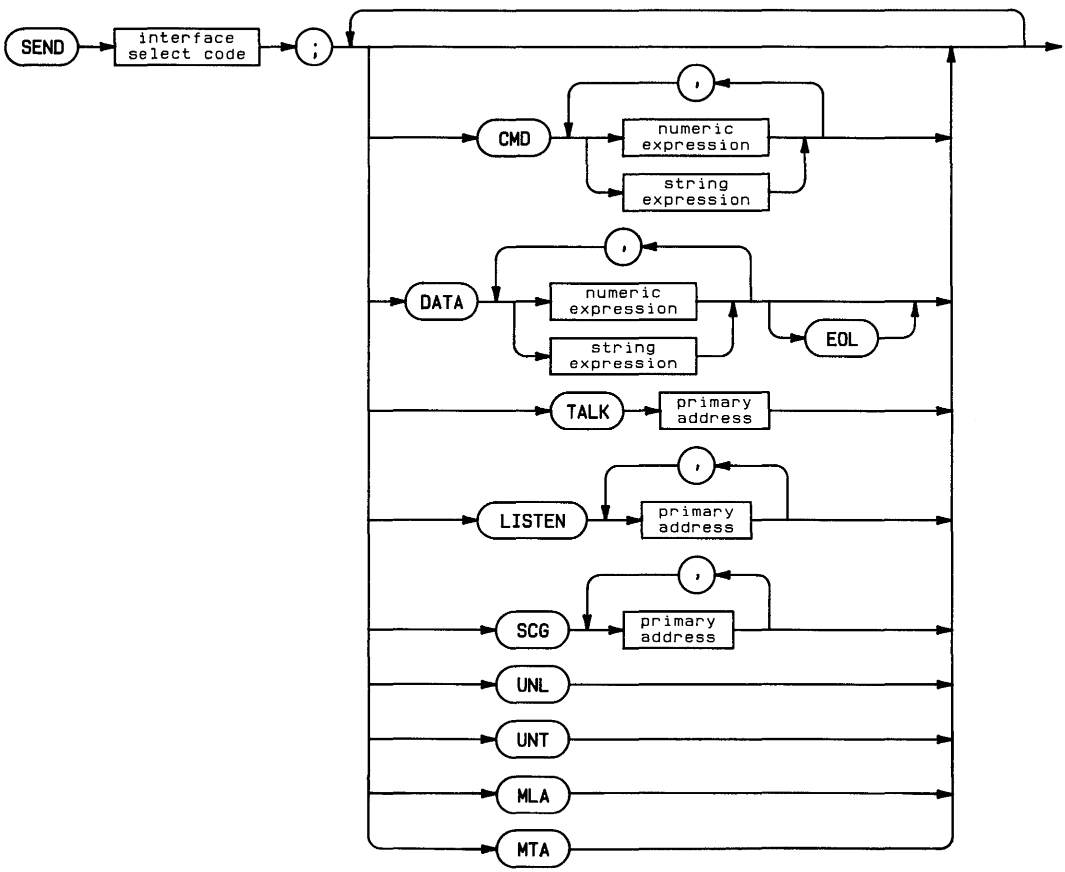
Related Keywords

PURGE, UNSECURE

SEND

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SEND` statement sends the specified command(s) or data to one or more devices.



S

Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
primary address	numeric expression, rounded to an integer	0 through 31
secondary address	numeric expression, rounded to an integer	0 through 31

Examples

```
SEND 7; CMD A$ DATA "Hello"
SEND 7; MTA UNL LISTEN 6, 14 CMD P,R SCG 6
```

Description

The secondary keywords that can be used and the action taken are interface-dependent.

HP-IB. The computer must be active controller when commands are sent. The ATN line is set true while commands are sent; the ATN line is set false while data is sent.

S

- `CMD` (commands)—sends a list of 8-bit expressions with ATN true. Primary commands have a bit pattern in the form *X00CCCCC*, where *X*=don't care and *C*=bits of the command (decimal value 0 through 31).
- `DATA` (data)—sends list of numeric or string expressions with ATN false. Any 8-bit pattern may be sent. If `EOL` is specified, the interface end-of-line sequence is sent following the data.
- `TALK`—sends a device Talk Address (TAD), decimal value 0 through 31.
- `LISTEN`—sends a device Listen Address (LAD), decimal value 0 through 31.
- `SCG` (secondary command group)—sends a secondary address to a device.
- `UNL`—sends the Unlisten command (decimal value 63). ATN is true.
- `UNT`—sends the Untalk command (decimal value 95). ATN is true.
- `MLA` (My Listen Address)—sends the listen address of the interface.
- `MTA` (My Talk Address)—sends the talk address of the interface.

Serial. The only form that can be sent is `DATA`, which sends the list of numeric and/or string expressions. If `EOL` is specified, the interface end-of-line sequence is sent.

BCD. UNL, UNT, MLA, and MTA are ignored. SCG returns an error. The following forms can be used:

- CMD (command), LISTEN, and TALK—primary addresses 0 through 6 set a partial field specifier.
- DATA (data)—the lower 4 bits of the data bytes are sent; control characters, spaces, and commas are ignored. If EOL is specified, data format checking is enabled.

GPIO. UNL, UNT, MLA, and MTA are ignored. SCG returns an error. The following forms can be used:

- CMD (command)—primary addresses 0 through 15 select port configuration. The Device Clear command pulses RESA and RESB. Selected Device Clear pulses RESA or RESB according to the most recent primary address.
- DATA—sends the list of numeric and/or string expressions. Data is sent as 8-bit bytes. If EOL is specified, the interface end-of-line sequence is sent.
- LISTEN, TALK—primary addresses 0 through 15 select the port configuration.

HP-IL. The following forms can be used:

- CMD (command)—sends the list of 8-bit expressions as command frames.
- DATA—sends the list of 8-bit expressions as data frames. If EOL is specified, the interface end-of-line sequence is sent following the data.
- TALK—sends a device Talk Address, decimal value 0 through 31.
- LISTEN—sends the device Listen Address(es), decimal value 0 through 31.
- SCG (secondary command group)—sends a secondary address frame, decimal value 0 through 31.
- UNL—sends an Auto Address Sequence and the Unlisten command frame.
- UNT—sends an Untalk command frame.
- MLA (My Listen Address)—addresses the interface to listen.
- MTA (My Talk Address)—sends the Talk Address of the interface.

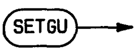
Related Keywords

OUTPUT

SETGU

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The SETGU statement sets the computer to *graphics units (GU's) mode*. In GU's mode, the plotting boundaries are equal to the graphics limits, and the plotting area is scaled in graphics units.



Examples

```
SETGU
IF X#0 THEN SETGU
```

Description

A graphics unit (GU) is defined as 1/100 of the shortest axis on the plotting device.

At power-on, reset, and when LIMIT or PLOTTER IS is executed, the computer is set to *user units mode*, with user units (UU's) set equal to graphics units. SCALE, MSCALE, or SHOW establish new user units. Executing SETGU permits plotting in GU's. After executing SETGU, plotting can be restored to previously established user units by executing SETUU.

Executing SETGU sets the plotting boundary to the graphics limits established by LIMIT or PLOTTER IS. In GU's mode, plotting boundaries set by LOCATE or CLIP are not active.

Related Keywords

LIMIT, LOCATE, MSCALE, PLOTTER IS, SCALE, SETUU, SHOW

SET I/O

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SET I/O` statement writes a byte of data to the specified interface register.

Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
register number	numeric expression, rounded to an integer	0 through 23
data byte	numeric expression, truncated to an integer and moduloed 256	—

Examples

```
SET I/O 7,16,3
SET I/O Isc,RegNum,BTD("10000011")
```

Description

The binary equivalent of the data byte is used to set and clear bits of the specified control register. `SET I/O` performs the same operation as the `CONTROL` statement, except that `SET I/O` can write to only one register at a time.

Related Keywords

`CONTROL`

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SET TIMEOUT` statement sets the maximum amount of time the system will wait for the specified interface to complete a handshake during an I/O operation.

Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
milliseconds	numeric expression, rounded to an integer	>0

Examples

```
SET TIMEOUT 7; 10000
SET TIMEOUT S1; T
```


Description

If an ON TIMEOUT end-of-line branch has been enabled, the branch is taken when the SET TIMEOUT limit is exceeded. If no ON TIMEOUT branching is in effect when the SET TIMEOUT time limit is exceeded, the system retains a pending end-of-line branch; when an `ON TIMEOUT` statement is executed, the branch is immediately taken.

I/O operations for which timeouts can occur include any OUTPUT, ENTER, TRANSFER, PRINT, and plotting operations that access an interface.

A timeout will not occur when a peripheral device stops handshaking in the middle of a transfer operation (TRANSFER INTR or TRANSFER FHS). However, a transfer can timeout if the interface or device cannot be addressed to start the transfer.

Related Keywords

OFF TIMEOUT, ON TIMEOUT

S

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SETUU` statement sets the computer to *user units (UU's) mode*. In UU's mode, user units are the current unit scaling of the plotting area.

Examples

```
SETUU
IF Y$="Y" THEN SETUU
```

Description

When `SETUU` is executed, plotting boundaries set by `LOCATE` or `CLIP` which were previously canceled by `SETGU` are reactivated. If that plotting area was previously scaled by `SCALE`, `SHOW`, or `MSCALE`, those user units are reactivated.

Executing `SCALE`, `SHOW`, or `MSCALE` also places the system in UU's mode.

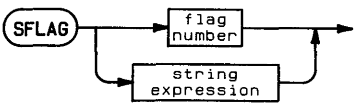
Related Keywords

`CLIP`, `LIMIT`, `LOCATE`, `PLOTTER IS`, `SETGU`

SFLAG

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The SFLAG statement sets and clears one or more flags.



Item	Description	Range
flag number	numeric expression, rounded to an integer	+1 through +64
string expression	eight characters, each interpreted as eight data bits	(use CHR$ or the metacharacter ~ for non-keyboard characters)

S

Examples

```
IF X=5 THEN SFLAG 6
SFLAG I
SFLAG "abcdefgh"
```

Description

When the SFLAG parameter is a numeric expression, it is interpreted as a flag number, and the specified flag is set. When the SFLAG parameter is a string expression, each of the eight characters are interpreted as one byte. The 8-bit binary value of each character sets (1) and clears (0) eight flags. The first character represents flags 1 through 8, the second character, flags 9 through 16, etc. If the string expression contains more than eight characters, it is truncated after the eighth character. If the string expression contains fewer than eight characters, CHR$(0) characters are appended to fill the string, and those flags are cleared.

Related Keywords

CFLAG, FLAG, FLAG$

SGN

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SGN` function returns 1 if the numeric argument is positive, −1 if the argument is negative, and 0 if the argument is 0.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
IF SGN(Y)=1 THEN GOSUB 400
Root=SGN(X)*SQR(ABS((X))
```

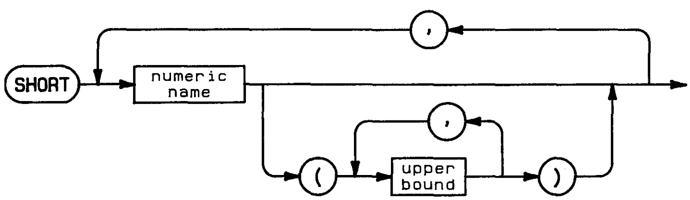

Related Keywords

`ABS`

SHORT

Keyboard Executable ■
Programmable ■
In an `IF...THEN` □

The `SHORT` statement declares and reserves memory for short precision numeric variables.



Item	Description	Range
numeric name	name of a simple numeric variable or numeric array	any valid name
upper bound	integer constant	1 through 65,535

Examples

```
SHORT ShortVariable,ShortArray1(10),ShortArray2(5,3)
```

Description

All numeric variables are REAL unless declared SHORT or INTEGER.

When the numeric variable name is used with one or two upper bound(s) enclosed in parentheses, the variable is dimensioned to be a one- or two-dimensional array. The default lower bound of the array is 0. The `OPTION BASE` statement is used to set the lower bound equal to 1.

When a SHORT simple variable or array element is printed to a data file, the value is stored in the file with REAL precision. If an entire SHORT array is printed to a data file with one statement (for example, `PRINT# 1;ShortArray()`), the elements are printed to the file with SHORT precision.

When a REAL number is assigned to a SHORT variable, the number is rounded. Overflow occurs if the number is outside the range of SHORT numbers.

When variables are passed to a subprogram by address, precision declarations accompany the variable into the subprogram.

Related Keywords

`DIM`, `INTEGER`, `REAL`

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SHOW` statement specifies a user units scale of the plotting area such that one unit of *x* equals one unit of *y* (equal unit scaling). Thus, the plotting area is scaled with unit squares.

Item	Description	Range
x-min	numeric expression	—
x-max	numeric expression	—
y-min	numeric expression	—
y-max	numeric expresison	—

Examples

```
SHOW -2,2,-4,4
SHOW A,2*B,0,3
```

Description

`SHOW` scales the current plotting area, which is a function of the units mode (GU's or UU's) and the previously executed statements.

- In GU's mode, `SHOW` scales the entire graphics area previously specified by `PLOTTER IS` or `LIMIT`).
- In UU's mode, `SHOW` scales the plotting area previously specified by `LOCATE`. If `LOCATE` has not been executed, the entire graphics area is scaled.

The user units are established such that the specified area is as large as possible and is centered within the plotting area. After executing `SHOW`, the system is set to UU's mode.

The order of the parameters can be changed to produce reflected output (see **LIMIT**).

Related Keywords

`LIMIT`, `LOCATE`, `PLOTTER IS`, `SCALE`

SIN

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SIN` function returns the sine of the angle argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
SineX = SIN(X)
If SIN(Theta)=1 THEN DISP "Theta equals 90 degrees"
```


Description

The angle argument is interpreted according to the current trigonometric mode—RAD (radians), DEG (degrees), or GRAD (grads). The default mode is RAD.

Related Keywords

`ASN`, `DEG`, `GRAD`, `RAD`

SINGLESTEP

Keyboard Executable ■
Programmable □
In an `IF...THEN` □

The `SINGLESTEP` command executes the current program line and then halts execution.

Description

The program must be initialized (by having previously executed `INIT` or `RUN`). A paused, unaltered program need not be reinitialized. However, if a paused program is edited, it must be initialized before singlestepping.

Related Keywords

`CONT`, `INIT`

S

SPOLL

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SPOLL` function returns an integer representing the status byte of the specified device.

Item	Description	Range
device selector	numeric expression, rounded to an integer (see glossary)	—

Examples

```
DeviceStatus=SPOLL(712)
IF SPOLL(D4)<64 THEN GOSUB 800
```


Description

The computer must be active controller in order to perform a serial poll.

Interface-dependent actions:

- HP-IB:

  If the device selector is an interface select code, the interface sends Serial Poll Enable (SPE), sets ATN false, receives the status byte, sends Serial Poll Disable (SPD), and sends Untalk (UNT)

  If the device selector contains a primary address, the interface sends Unlisten (UNL), My Listen Address (MLA), the device Talk Address (TAD), Serial Poll Enable (SPE), and then sets ATN false. The interface receives the status byte and then sends Serial Poll Disable (SPD) and Untalk (UNT).

- HP-IL: SPOLL returns the first byte received in response to a serial poll of a device.

  If the device selector is an interface select code, the interface sends Send Status (SST) and then waits to receive a data byte followed by End of Transmission (EOT). The interface then sends Untalk (UNT).

  If the device selector includes a primary address, the interface sends Unlisten (UNL), My Talk Address (MTA), the device talk address, and Send Status (SST). The interface then waits to receive the data byte followed by End of Transmission (EOT). The interface then sends Untalk (UNT).

Related Keywords

PPOLL

SQR

The SQR function returns the square root of the numeric argument. Negative arguments return an error.

Item	Description	Range
numeric argument	numeric expression	⩾0

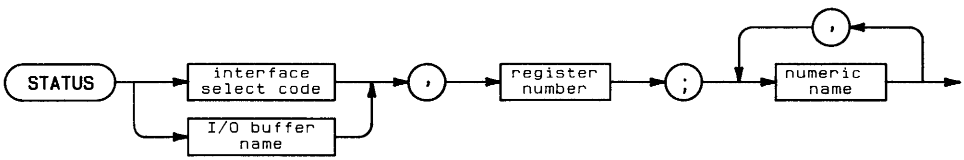
Examples

```
DISP SQR(X)
C=SQR(A^2+B^2)
```

STATUS

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The STATUS statement returns the contents of one or more interface or I/O buffer status registers.



Item	Description	Range
interface select code	numeric expression, rounded to an integer	3 through 10
I/O buffer name	name of a string variable declared as an I/O buffer	—
register number	numeric expression, rounded to an integer	0 through 15
numeric name	name of numeric variable	any valid name

Examples

```
STATUS 7,0;Register0
STATUS 7,3;Register3,Register4,Register5
```

Description

The register number must be valid for the specified interface.

When more than one numeric variable is listed, consecutive status registers are read starting at the specified register number. If the number of variables listed exceeds the number of existing registers, an error is returned; there is no wraparound to the first register.

Related Keywords

ASSERT, CONTROL, ENABLE INTR, IOBUFFER

STOP

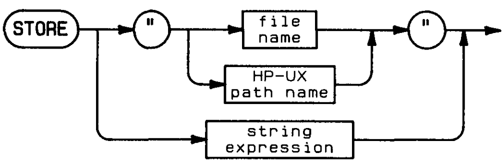
See **END**.

S

STORE

Keyboard Executable ■
Programmable □
In an IF...THEN □

The STORE command stores the current BASIC program or subprogram into a disc file of the specified name.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—

Examples

```
STORE "filename"
STORE "/disc1/filename"
STORE "/Directory1/Directory2/filename"
```

Description

If the file name is used alone (rather than as part of an HP-UX path name), the STORE operation uses the current working directory.

When `STORE` is executed, the system searches the specified directory for a BASIC/PROG file with the indicated name. If the file is found, the current (sub)program is stored in that file, overwriting the previous contents. If no such file is found, the file is created in that directory. An error is returned if the file name already exists in the directory with another file type.

When a new subprogram is stored, the file name must be the same as the FINDPROG name.

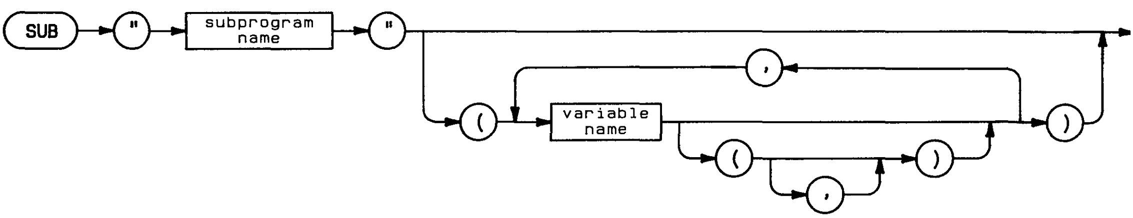
Related Keywords

`LOAD`, `MASS STORAGE IS`

SUB

Keyboard Executable ☐
Programmable ■
In an IF...THEN ☐

The SUB statement is the first statement in a subprogram. It defines the beginning of the subprogram and lists the formal parameters passed into the subprogram.



Item	Description	Range
subprogram name	literal	14 characters maximum; slash and leading colon not allowed
variable name	name of a numeric or string variable (see glossary)	any valid name

S

Examples

```
SUB "Count"
SUB "SubPlot"(Xmin,Xmax,Yvar(),Zvar(,)S$)
```

Description

All subprograms must begin with a SUB statement. The statement cannot be part of a multistatement line. A subprogram can contain only one SUB statement.

The optional variable names enclosed in parentheses list the formal parameters passed from the calling (sub)program to the subprogram. The parameters become associated, from left to right, with the pass parameters listed in the CALL statement. The variable type (simple numeric, simple string, numeric array, string array) must agree with the parameters listed in the CALL statement. Arrays are designated by a pair of parentheses after the array name; an optional comma documents 2-dimensional arrays. Variables in the main program not explicitly passed to the subprogram or held in COMmon with the subprogram are unknown to the subprogram.

The pass parameter list does not include precision declarations (REAL, SHORT, and INTEGER), nor does it specify the dimensions of simple string variables and numeric and string arrays. The precision and dimensions of variables passed by address accompany them as they are passed. When a string expression is passed by value, the formal parameter to which it is passed is dimensioned to the current length of the string.

The SUB statement can list more parameters than the calling subprogram's CALL statement. Extra parameters are set to 0 and the null string. NPAR returns the number of parameters actually passed.

Common variables can be passed into subprograms by including them in a COM statement. Unlike the parameter list of the SUB statement, the COM statement must contain both precision declarations and array size declarations.

When a subrogram is stored, it is entered into the directory as a type BASIC/SUBP file.

Related Keywords

CALL, FINDPROG, SCRATCHSUB, SUBEND, SUBEXIT

SUBEND

Keyboard Executable ☐
Programmable ■
In an IF...THEN ■

The SUBEND statement returns program execution to the calling program or subprogram.

Description

When SUBEND is executed, program execution resumes at the statement in the calling program that immediately follows the CALL statement. Comments following SUBEND are part of the subprogram.

SUBEND and SUBEXIT are interchangeable.

Related Keywords

CALL, SUB, SUBEXIT

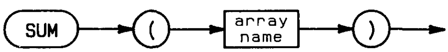
SUBEXIT

See **SUBEND**

SUM

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `SUM` function returns the sum of all the elements in the specified array.



Item	Description	Range
array name	name of a one- or two-dimensional array	any valid name

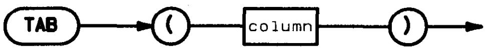
Examples

```
DISP SUM(Vector1)
Y=SUM(A)
```


Related Keywords

`ABSUM`

The TAB function is used with simple DISP and PRINT (without USING) to specify the column in which the next output item is placed. (See DISP and PRINT).



TAN

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `TAN` function returns the tangent of the angle argument.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
Tangent=TAN(Theta)
Vertical=Horizontal*TAN(x)
```

Description

The angle argument is interpreted according to the current trigonometric mode—RAD (radians), DEG (degrees), or GRAD (grads). The default mode is RAD.

T

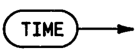
Related Keywords

`ATN`, `DEG`, `GRAD`, `RAD`

TIME

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `TIME` function returns the current value of the system clock seconds counter.



Description

The seconds counter usually represents the number of seconds elapsed since midnight. The largest value returned is 86,399. When the counter reaches this value, it is returned to 0 and the date is incremented.

Related Keywords

`DATE`, `DATE$`, `TIME$`

TIME$

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `TIME$` function returns the system clock reading in HH:MM:SS notation.

Examples

```
DISP TIME$
IF TIME$=B$ THEN 200
```

Description

The string returned is in 24-hour notation in the range 00:00:00 through 23:59:59.

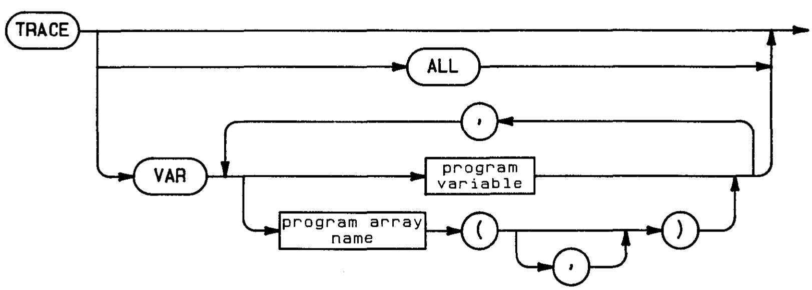
Related Keywords

`DATE`, `DATE$`, `TIME`

T

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The TRACE statement traces program variable assignments and/or the order in which program lines are executed.



Item	Description	Range
program variable	name of a simple or array program variable	any valid name
program array name	name of a program array	any valid name

Examples

```
TRACE
TRACE ALL
TRACE VAR Var1, Var2$, Array1(6), Wholearray$()
```

Description

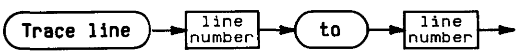
Three tracing options are available: TRACE, TRACE VAR, and TRACE ALL. Tracing results are output to the display.

...TRACE

When tracing statements are executed within a program or subprogram, tracing is local, and halts when execution is transferred to another subprogram or back to the main program. When `TRACE` or `TRACE ALL` is executed from the keyboard, it applies to the main program only. When `TRACE VAR` is executed from the keyboard, it applies to the current program or subprogram.

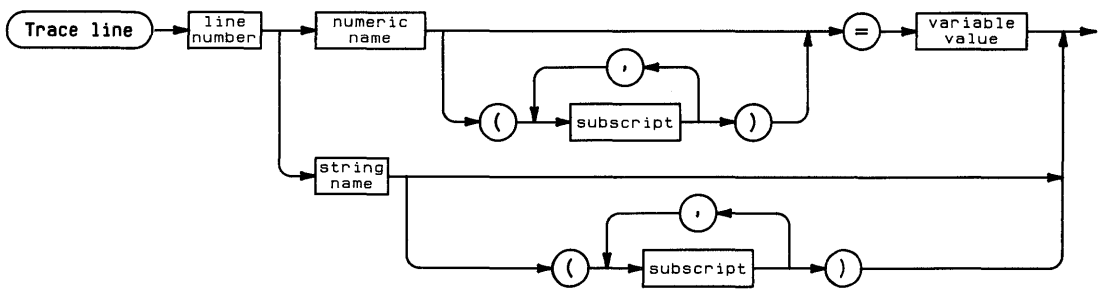
Tracing operations are canceled by executing `NORMAL`.

TRACE. `TRACE` traces the order in which program lines are executed. Nothing is output when execution proceeds sequentially from statement to next-higher numbered statement. When branching occurs, `TRACE` outputs branching information in the form:



TRACE VAR. `TRACE VAR` traces assignments to the specified program variables during program execution. Variables to be traced must be allocated before `TRACE VAR` is executed. If `TRACE VAR` is executed from the keyboard before the program is run, the program must first be initialized by executing `INIT`.

TRACE VAR outputs changes in variable assignments of program variables to in the form:



When a numeric variable receives a new assignment, the variable name and new value are output. When a string variable is assigned a new value, TRACE VAR outputs the name of the string variable without printing its new value. When a statement operates on an entire numeric array, the new value of the first element only is output.

TRACE ALL. TRACE ALL traces program execution and variable assignments from line to line, regardless of whether or not branching occurs. Changes in the values assigned to variables are reported in the same format as TRACE VAR output.

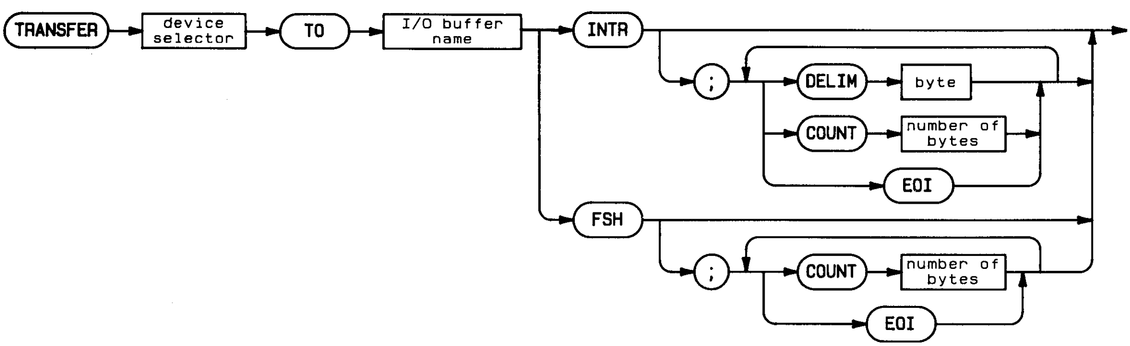
Related Keywords

NORMAL

TRANSFER (in)

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The TRANSFER (in) statement transfers bytes of data from the specified device to the specified I/O buffer.



Item	Description	Range
device selector	numeric expression, rounded to an integer (see glossary)	—
I/O buffer name	name of a string variable declared as an I/O buffer	any valid name
byte	numeric expression, rounded to an integer	0 through 255
number of bytes	numeric expression, rounded to an integer	0 through 65,530

T

Examples

```
TRANSFER 3 TO Buffer1$ FHS
TRANSFER A TO Buffer2$ INTR; DELIM 10 COUNT 300
```

Description

Characters are placed into the buffer according to the position of the *buffer fill pointer*. The transfer terminates when the buffer is full or when any of the terminating conditions specified by the following keywords is met:

- `COUNT` specifies the maximum number of characters to be transferred.
- `DELIM` specifies a terminating character. The parameter following `DELIM` is the decimal value of the last character to be placed in the I/O buffer. `DELIM` cannot be used with `FHS` transfers.
- `EOI` (End-or-identify) provides for terminating the transfer when an interface-dependent END signal is detected.

The interface may also have a programmable terminating condition.

If an ON EOT branch is enabled, the branch is taken when the transfer terminates.

Interrupt Transfer. If the `INTR` (*interrupt*) option is used, program execution continues and the interface is automatically enabled to interrupt the computer each time it is ready for a new data byte. The transfer continues to completion. If program execution stops before the transfer is complete, a warning is issued and the transfer continues to completion. The transfer must be complete before attempting to edit the program.

Fast Handshake Transfer. If the `FHS` (fast handshake) option is used, the interface and computer are dedicated to the transfer until it is completed. No interrupts or keystrokes are detected until the transfer terminates.

The procedure for clearing a system lock-up during a fast handshake transfer is machine dependent.

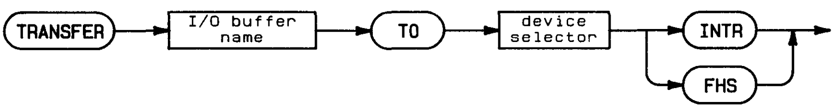
Related Keywords

`ABORTIO, CONTROL, ENTER, HALT, IOBUFFER, ON EOT, RESET, STATUS`

TRANSFER (out)

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `TRANSFER` (out) statement transfers bytes of data from the specified I/O buffer to the specified device.



Item	Description	Range
I/O buffer name	name of a string variable declared an I/O buffer	any valid name
device selector	numeric expression, rounded to an integer (see glossary)	—

Examples

```
TRANSFER Buffer1$ TO 701 INTR
TRANSFER MONEY$ TO Checking FHS
```


Description

Data is taken from the buffer at the position specified by the *buffer empty pointer*. The interface end-of-line sequence is sent following the last byte of data sent.

The transfer halts when the buffer is empty. If an ON EOT branch is enabled, the branch is taken when the transfer terminates.

Interrupt Transfer. If the `INTR` (interrupt) option is used, program execution continues and the interface is automatically enabled to interrupt the computer each time it is ready for a new data byte. The transfer continues to completion. If program execution stops before the transfer is complete, a warning is issued and the transfer continues to completion. The transfer must be complete before attempting to edit the program.

Fast Handshake Transfer. If the `FHS` (fast handshake) option is used, the system is dedicated to the transfer until it is completed. No interrupts or keystrokes are detected until the transfer terminates.

The procedure for clearing a system lock-up during a fast handshake transfer is machine dependent.

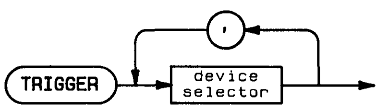
Related Keywords

`ABORTIO`, `CONTROL`, `HALT`, `IOBUFFER`, `ON EOT`, `OUTPUT`, `RESET`, `STATUS`

TRIGGER

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `TRIGGER` statement sends a Group Execute Trigger message to the specified device(s).



Item	Description	Range
device selector	numeric expression, rounded to an integer (see glossary)	—

Examples

```
TRIGGER 7
TRIGGER D1, D2
```

Description

The computer must be active controller in order to execute `TRIGGER`. If more than one device selector is specified, the device selectors must include a primary address and the devices must be located at the same interface select code.

If the device selector is an interface select code, the interface sends GET to devices addressed to listen.

If the device selector(s) contain a primary address, the interface sends UNL, LAD of the specified device(s), and GET.

Related Keywords

`RESUME`, `SEND`

TRIM$

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `TRIM$` function returns a string stripped of all leading and trailing spaces (ASCII decimal code 32). Embedded blanks are unaffected.

Item	Description	Range
string argument	string expression	—

Examples

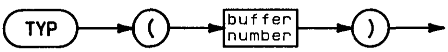
```
DISP TRIM$(Title$)
Sortdata$=TRIM$(LastName$)
```

T

TYP

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The TYP function returns the data type of the next item in a BASIC/DATA file.



Item	Description	Range
buffer number	numeric expression, rounded to an integer	1 through 10

Examples

```
IF TYP(3)=1 THEN READ# 3;Number
DISP TYP(Buffer)
```

Description

The file must be opened.

TYP returns an integer in the range 1 through 4, 8 through 10, according to the position of the file pointer and the contents of the data file. The number returned indicates the nature of the item following the current pointer location.

Values Returned by TYP

TYP Value	Data Type
1	Numeric
2	Complete string
3	End-of file
4	End-of-record
5 through 7	Not used
8	Beginning of string; the string extends into the following record
9	Middle of string; the string extends into the previous and following records
10	End of string; the string is continued from the previous record

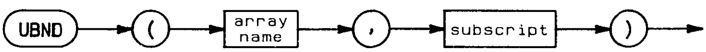
Related Keywords

READ#

UBND

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The UBND function returns the dimension (upper bound) of the first or second subscript of the specified array.



Item	Description	Range
array name	name of a one-or two-dimensional numeric array	any valid name
subscript	numeric expression, rounded to an integer	1 through 2

Examples

```
FOR I=1 TO UBND(A,2)
LET Y(UBND(Y,1),UBND(Y,2))=3
```

Related Keywords

LBND

U
V

UNCLIP

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The UNCLIP statement cancels plotting boundaries set by CLIP or LOCATE, and sets the plotting boundaries equal to the graphics limits.

Examples

```
UNCLIP
IF A$="Y" THEN UNCLIP
```

Description

Both SETGU and UNCLIP set the plotting boundaries equal to the graphics limits. The differences between the two statements are:

- UNCLIP does not switch the current plotting units to GU's. The computer remains in the current units mode.
- UNCLIP completely cancels the CLIP or LOCATE plotting boundaries. SETGU changes the current plotting area but does not cancel the plotting boundaries set by CLIP or LOCATE; they can be restored by executing SETUU.

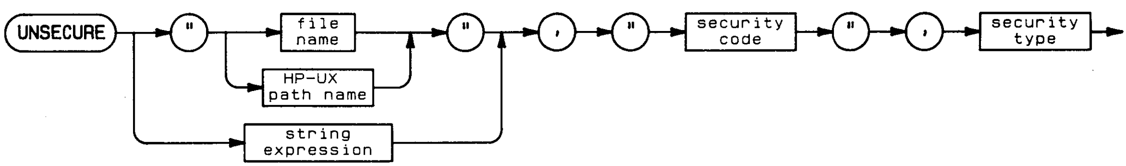
Related Keywords

CLIP, LOCATE, SETGU, SETUU

UNSECURE

Keyboard Executable	■
Programmable	□
In an `IF...THEN`	□

The `UNSECURE` command cancels security previously specified for BASIC files.



Item	Description	Range
file name	literal; name of a file in the current working directory	14 characters maximum; slash and leading colon not allowed
HP-UX path name	literal; an absolute or relative path name (see glossary)	—
string expression	expression evaluating to a file name or HP-UX path name	—
security code	string expression	first two characters are used
security type	numeric expression, truncated to an integer and moduloed 4	—

Examples

```
UNSECURE "myfile","nl",0
UNSECURE "/vol1/dir1/dir2/myfile",Code$,2
```

Description

The security type must match the security type specified for the file when it was secured. For types 0 and 1 security, the first two characters of the security code must match the code that became associated with the file when it was secured, except that lowercase and uppercase letters are interchangeable. The security code is ignored for type 2 security. `UNSECURE` has no effect for security type 3.

The following rules apply to unsecuring files:

- Files secured with type 0 can be unsecured with type 0 or 1.
- Files secured with type 1 can be unsecured for LIST, PLIST, and editing by unsecuring for type 0. COPY security remains.
- Files secured with types 0 and 1 can be unsecured for type 1. Type 0 security is automatically removed.
- When unsecuring a file for LIST, the security must be removed before the file is loaded.

Related Keywords

`SECURE`

UPC$

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `UPC$` string function returns a string in which all the lowercase letters in the argument are converted to uppercase.

Item	Description	Range
string argument	string expression	—

Examples

```
IF UPC$(String$)="YES" THEN 200
DISP UPC$(String$)&"..."
```

Related Keywords

`LWC$`

VAL

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `VAL` function converts a string expression containing digits into a numeric value.

Item	Description	Range
string argument	string expression	—

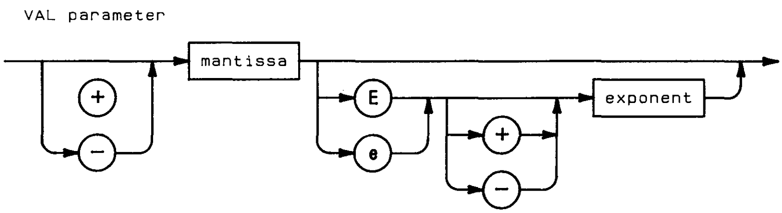
Examples

```
DISP VAL(A$)
Z=X(1)+VAL(Baseline$(X))
```

Description

The string can contain leading blanks and tab characters. The mantissa begins with the first non-blank/tab character, which must be a plus or minus sign, decimal point, or digit. Additional characters can be digits or a decimal point; there can be only one decimal point per number.

If exponential notation is used, the exponent following E or e consists of an optional sign folowed by two or three digits.



The argument must contain at least one digit. Embedded blanks and non-digit characters not used to build an exponent terminate the number.

Related Keywords

VAL$

VAL$

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `VAL$` function evaluates the numeric argument and returns the string representation of the argument in standard number format.

Item	Description	Range
numeric argument	numeric expression	—

Examples

```
C$=VAL$(D)&"00"
PRINT# 1;VAL$(Xcoordinate)
```

Description

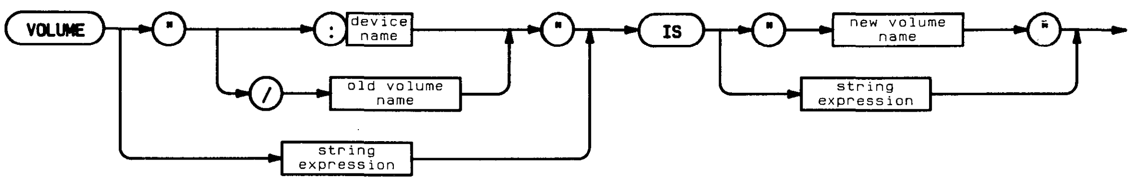
The string returned has no leading or trailing blanks. Decimal numbers have a leading zero preceding the radix.

Related Keywords

`VAL`

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The VOLUME...IS* statement changes the disc volume name and remounts the disc under this new name of the top-level directory.



Item	Description	Range
device name†	literal; name assigned to a mass storage unit by the operating system	—
old volume name	current volume name of the disc	6 characters maximum; colon, slash, and quotation marks not allowed
new volume name	volume name to be assigned to the disc	6 characters maximum; colon, slash, and quotation marks not allowed

Examples

```
VOLUME ":D301" IS "/label"
VOLUME "/topdir1" IS "topdir2"
```

* Implemented only for single-user systems with removable file systems.

† Machine-dependent parameter.

...VOLUME...IS

Description

A volume name is created on the disc when the disc is formatted. When the disc is mounted, the volume name becomes the name of the top-level directory. VOLUME...IS can change names of top-level directories only.

Related Keywords

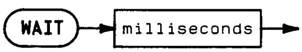
MASS STORAGE IS

U
V

WAIT

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The WAIT statement causes a delay in program execution until the specified number of milliseconds has elapsed.



Item	Description	Range
milliseconds	numeric expression	⩾1

Examples

```
WAIT N*250
IF X=7 THEN WAIT 5000
```

Description

The WAIT statement can be interrupted by pausing the program. When the program is continued, execution continues at the next statement.

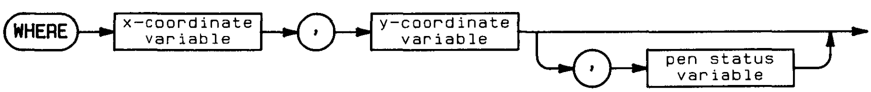
Related Keywords

PAUSE

WHERE

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The WHERE statement assigns the last known location and status of the plotting device's logical pen to the specified numeric variables.



Item	Description	Range
x-coordinate variable	name of a numeric variable	any valid name
y-coordinate variable	name of a numeric variable	any valid name
pen status variable	name of a numeric variable	any valid name

Examples

```
WHERE Xposition, YPosition, Penstatus
WHERE x(I), y(I)
```

W
X
Y

Description

The pen x- and y- coordinates are interpreted according to the current units. The pen status variable is assigned the value 0 if the pen is up, 1 if the pen is down.

The location and status of the logical pen is determined by the most recently executed statement affecting the pen. This includes all plotting statements and all statements and conditions which activate the graphics default conditions (see glossary). When the graphics default conditions are activated, the logical pen is lifted and moved to the origin (0,0).

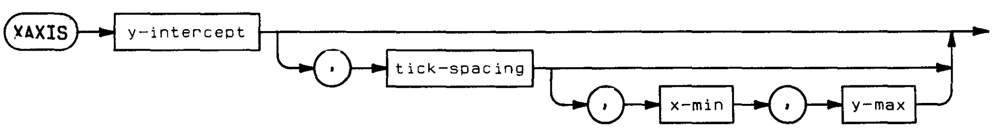
Related Keywords

CURSOR, DIGITIZE

XAXIS

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `XAXIS` statement draws a horizontal axis, with optional tick marks, at the specified y-intercept.



Item	Description	Range
y-intercept	numeric expression, interpreted according to the current units (default=0)	—
tick-spacing	numeric expression, interpreted according to the current units (default=0; no ticks)	—
x-min	numeric expression, interpreted according to the current units (default=the entire plotting area)	—
x-max	numeric expression, interpreted according to the current units (default=the entire plotting area)	—

Examples

```
XAXIS 3
XAXIS (Ymax-Ymin)/2,1
XAXIS Y(1),2,-12,12
```

W
X
Y

Description

The axis and optional tick marks are drawn using the current line type, and are clipped at the plotting boundaries. The y-intercept may lie outside the plotting area; only the portion of the axis within the plotting area is shown. The x-min and x-max parameters provide for drawing an axis across a portion of the plotting area. Parameters outside the plotting area are ignored. The default axis length is the entire plotting area.

Tick marks are 2 GU's long. The sign of the tick spacing parameter determines where ticks are placed. If the tick-spacing parameter is positive, ticks are left-justified on the x-axis. If the tick spacing parameter is negative, ticks are right-justified.

Related Keywords

AXES, LAXES, YAXIS

XREF L

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The XREF L statement displays a line cross-reference table of program line numbers, line labels, and user-defined functions in the current (sub)program.

Description

XREF L generates an entry in the line cross-reference table whenever a line number or line label is referenced. Table entries are in the form:

referenced line number [*line label*] _______ occurs on *referencing line(s)*

For example, the program lines:

```
30 IF X#5 THEN loop
500 loop: FOR I=1 to 5
```

generate the entry:

```
500 loop:__________ occurs on  30
```

The system displays ...end of xrefl when the entire table has been displayed.

Related Keywords

LIST, SCAN, XREFV

Keyboard Executable ■
Programmable ■
In an IF...THEN ■

The XREF V statement displays a cross-reference table of all the variables and user-defined functions in the current sub(program).

Description

The XREF V table contains the following information about each program variable:

- Variable—the name of the variable or user-defined function.
- Dim1—the upper bound of the first subscript in an array variable.
- Dim2—the upper bound of the second subscript in an array variable.
- Maxl—the maximum length of a string variable.
- Type—REAL, SHORT, INTEGER, or string.
- References—lines referencing the variable or user-defined function, including function definitions (DEF FN statements), function value assignments (FN...=), and function calls (FN).

The system displays ...end of xrefv when the entire table has been generated.

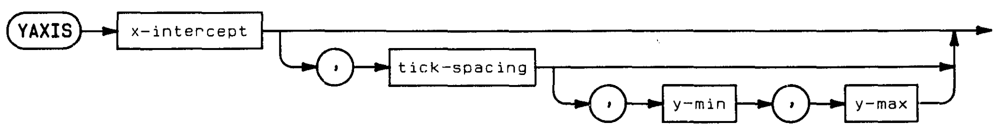
Related Keywords

LIST, SCAN, XREFL

YAXIS

Keyboard Executable ■
Programmable ■
In an `IF...THEN` ■

The `YAXIS` statement draws a horizontal axis, with optional tick marks, at the specified x-intercept.



Item	Description	Range
x-intercept	numeric expression, interpreted according to the current units (default=0)	—
tick-spacing	numeric expression, interpreted according to the current units (default=0; no ticks)	—
y-min	numeric expression, interpreted according to the current units (default=the entire plotting area)	—
y-max	numeric expression, interpreted according to the current units (default=the entire plotting area)	—

Examples

```
YAXIS 3
YAXIS X(I)/3,1
YAXIS 3,1,2,2
```

Description

The axis and optional tick marks are drawn using the current line type, and are clipped at the plotting boundaries. The x-intercept can lie outside the plotting area; only the portion of the axis within the plotting area is shown. The y-min and y-max parameters provide for drawing an axis across a portion of the plotting area. Parameters outside the plotting boundaries are ignored. The default axis length is the entire plotting area.

Tick marks are 2 GU's long. The sign of the tick spacing parameter determines where ticks are placed. If the tick spacing parameter is positive, ticks are bottom-justified on the y-axis. If the tick spacing parameter is negative, ticks are top-justified.

Related Keywords

AXES, LAXES, XAXIS

A
B
C
D
E
F
G
H
I
K
L

M
N
O
P
R
S
T
U
V
W
X
Y

82860-90002
Printed in U.S.A.

Reader Comment Card

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications.

Manual Title and/or Part number______________________________

Please circle one number for each of the statements below.

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
■ The manual is well organized.	1	2	3	4	5
■ I can find the information I want.	1	2	3	4	5
■ The information in the manual is accurate.	1	2	3	4	5
■ I can easily understand the instructions and procedures.	1	2	3	4	5
■ The manual is clearly written.	1	2	3	4	5
■ The manual contains enough examples.	1	2	3	4	5
■ The examples are appropriate and helpful.	1	2	3	4	5
■ The layout and format are attractive and useful.	1	2	3	4	5
■ The illustrations are clear and helpful.	1	2	3	4	5
■ I like the use of color in the manual.	1	2	3	4	5

Please feel free to write additional comments, particularly if you disagree with a statement above. Use additional pages if you wish—the more detailed your comments, the more useful they are to us.

Comments:______________________________

Name:______________________________

Title:______________________________

Company:______________________________

Address:______________________________

City/State/Zip:______________________________

Phone:______________________________

HEWLETT PACKARD

Printed in U.S.A. 10/84
5957-8142

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 CORVALLIS, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
ATTN: Documentation Department
Portable Computer Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

Fold here

Tape Please do not staple Tape

Reader Comment Card

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications.

Manual Title and/or Part number____________________

Please circle one number for each of the statements below.

	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
■ The manual is well organized.	1	2	3	4	5
■ I can find the information I want.	1	2	3	4	5
■ The information in the manual is accurate.	1	2	3	4	5
■ I can easily understand the instructions and procedures.	1	2	3	4	5
■ The manual is clearly written.	1	2	3	4	5
■ The manual contains enough examples.	1	2	3	4	5
■ The examples are appropriate and helpful.	1	2	3	4	5
■ The layout and format are attractive and useful.	1	2	3	4	5
■ The illustrations are clear and helpful.	1	2	3	4	5
■ I like the use of color in the manual.	1	2	3	4	5

Please feel free to write additional comments, particularly if you disagree with a statement above. Use additional pages if you wish—the more detailed your comments, the more useful they are to us.

Comments:____________________

Name:____________________

Title:____________________

Company:____________________

Address:____________________

City/State/Zip:____________________

Phone:____________________

HEWLETT PACKARD

Printed in U.S.A. 10/84
5957-8142

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 CORVALLIS, OR

POSTAGE WILL BE PAID BY ADDRESSEE

Hewlett-Packard Company
ATTN: Documentation Department
Portable Computer Division
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

Fold here

Tape Please do not staple Tape

Software Customer Support

Much of the information in this booklet is taken from the *Support Guide* that came with your HP computer. Refer to that booklet for more information on customer support.

License Agreement and Warranty

Before opening the package containing the disc(s) or software module(s), be sure to read the software's licensing agreement in the booklet marked *License Agreement and Limited Warranty*. That booklet also contains the warranty information for this software product.

Customer Assistance—Getting Help

To locate a local dealer or HP Sales and Service Office, refer to the "Directory" printed in the back of the *Support Guide* that came with your compter. If you don't have the *Support Guide*, you can call one of the following numbers for this information: In the U.S., call 800/FOR-HPPC. In Europe or the U.K., contact Hewlett-Packard, S.A. in Geneva (022/83 81 11). If you are outside the U.S. and Europe, contact Hewlett-Packard Intercontinental in California (415/857-1501).

For telephone assistance in operating your computer, call one of the following numbers.

- In the U.S. call the North American Response Center—1-800/858-8867.
- In Canada call 1-800/267-6115.
- In other countries call your HP Sales and Service Office.

Software Updates and Replacements

Updated and upgraded software and manuals are available to current owners as improvements and corrections occur. Contact your dealer or an HP Sales and Service Office for this service.

For disc-based software a new disc is provided at a nominal charge in exchange for your original master disc. For ROM-based software a new ROM cartridge is provided.

Creating copies of your master discs and your work discs minimizes the risk of damage to your original (master) software discs. Should a master software disc be destroyed, however, replacement discs are available for software distributed by Hewlett-Packard.

HEWLETT PACKARD

Edition 1	11/84	5957-8136
Edition 2	12/84	5957-8155
Edition 3	4/85	5957-8178

Printed in U.S.A. 4/85
5957-8178