



**HEWLETT
PACKARD**

Reference Manual



Notice

Hewlett-Packard Company makes no express or implied warranty with regard to the program material offered or the merchantability or the fitness of the program material for any particular purpose. The program material is made available solely on an “as is” basis, and the entire risk as to its quality and performance is with the user. Should the program material prove defective, the user (and not Hewlett-Packard Company nor any other party) shall bear the entire cost of all necessary correction and all incidental or consequential damages. Hewlett-Packard Company shall not be liable for any incidental or consequential damages in connection with or arising out of the furnishing, use, or performance of the program material.



HP-71

Reference Manual

January 1984

00071-90010 Rev. B

Contents

How To Use This Manual	5
Introduction	5
What Is a Keyword?	5
Finding Keyword Entries	5
Format for Keyword Dictionary Entries	6
How To Read the Syntax Diagrams	8
Using Blanks in Keyword Inputs	9
Using the Glossary	9
Using the Keyword Index	9
HP-71 Keyword Dictionary	10
System Characteristics	314
Scope of Environments	314
Variables	315
Simple Numeric Variables	316
Numeric Array Variables	316
Simple String Variables	316
String Array Variables	316
Array Bounds and Referencing	316
Math Reference	317
Precedence of Operators	317
Numeric Precision	317
Range of Numbers	317
Numeric Expressions	318
System Flags	319
Keyboard and Display Control	319
Input Keystrokes	319
Editing Keystrokes	320
System Keystrokes	321
Escape Keystrokes	322
Display Control	322
CALC Mode keystrokes	322
HP-71 Character Set and Character Codes	322
Control Characters	326
HP-71 Display Escape Code Sequences	328
Reset Conditions	329

System Memory Requirements	330
Memory Usage During Evaluation of Expressions	333
Mathematical Discussion of HP-71 Statistical Arrays	334
Matched Samples	334
Summary Statistics	335
Recursive Calculation of Statistics	336
Simple Linear Regression	337
IEEE Proposal for Handling Math Exceptions	338
Introduction	338
Setting and Clearing Math Exception Flags	338
The Five Math Exception Flags	338
Extended Default Values	339
Not a Number	340
Infinity	341
Denormalized Numbers and -0	341
Classes of Numbers	342
The Unordered Comparison Operator	343
Table of Comparisons (X Compared to Y)	344
Glossary	346
Errors, Warnings, and System Messages	378
Introduction	378
Alphabetical Message Listing	378
Numerical Message Listing and Descriptions	380
Math Errors (1 through 21)	380
System Errors (22 through 27)	382
Program Errors (28 through 56)	383
File and Device Errors (57 through 65)	387
Card Reader Errors (66 through 74)	389
Syntax Errors (75 through 88)	390
Card Reader Messages (89 through 97)	392
HP-71 Exception Flag Summary	393
IVL (Invalid Operation)	393
DVZ (Division By Zero)	393
OVF (Overflow)	393
UNF (Underflow)	393
INX (Inexact Result)	393
Keyword Index and Summary	394
Subject Index	406

How To Use This Manual

Introduction

The *HP-71 Reference Manual* is a reference tool for users who are already familiar with HP-71 operation and BASIC language programming. If you are new to the HP-71, you should first turn to the *HP-71 Owner's Manual* to familiarize yourself with the computer's features.

This manual provides a source of nontutorial information concerning HP-71 keywords, character sets, memory requirements, error messages, and other topics. Included also are a glossary of HP-71 terms (page 346) and a Keyword Summary and Index (page 394—at the back of the manual).

The HP-71 Keyword Dictionary covers keyword use, syntax, parameter guidelines, operating details, and related keywords. Before you begin referring to the dictionary, read through the information on the next four pages to familiarize yourself with the dictionary's format and use.

What Is a Keyword?

The term *keyword* refers to your HP-71's BASIC statements, functions, and operators. All but nine of the keywords are programmable. These nine are termed *nonprogrammable statements*. (In some computer literature, such nonprogrammable keywords are referred to as *commands*.)

Finding Keyword Entries

The HP-71 keyword entries are placed in alphabetical order. Each entry begins at the top of a new page and can be quickly identified by the keyword name printed in large, blue characters. The keywords for some trigonometric functions have acceptable alternate spellings. Such alternates are indicated in parentheses at the top of the page, following the more common spelling. *Middle* keywords, such as `T□`, cannot be executed alone and therefore are not listed separately.

Combined Keyword Descriptions. Where two or more keyword entries begin with the same word(s) and perform similar operations, their descriptions are grouped together under one heading. For example, `ON ERROR GOSUB` and `ON ERROR GOTO` are described under the page heading **ON ERROR GOSUB / GOTO**. In these instances, common features such as identical input parameters are represented once for all keywords in the group.

Operator Descriptions. Logical operators (`AND`, `EXOR`, `NOT`, and `OR`) appear in alphabetical order in the Keyword Dictionary. The `@` character, the `&` concatenation operator, and the math operators (`+`, `-`, `*`, `/`, `%`, and `^`) appear at the end of the dictionary. Relational operators (`<`, `=`, `>`, `#`, `?`, `<=`, `>=`, and `<>`) are listed under "Precedence of Operators" on page 317, but do not appear in the Keyword Dictionary.

Format for Keyword Dictionary Entries

Each of the numbered circles shown below links a feature of the Keyword Dictionary format to a corresponding illustration in the sample dictionary entry on the facing page.

- ① **Keyword Name:** Identifies the keyword entry and includes, within parentheses, the keyword's acceptable alternate spelling, if any.
- ② **Purpose:** Describes what operations the keyword is designed to perform. If the keyword is one of the nine nonprogrammable statements, this fact is noted in this description.
- ③ **Keyword Type and Execution Options:** The filled-in squares (■) in this chart indicate the keyword type (statement, function, or operator) and whether you can execute the keyword from the keyboard, in CALC mode, and/or after THEN or ELSE in an IF ... THEN ... ELSE statement.
- ④ **Syntax Diagram:** Illustrates the required and optional syntax for HP-71 keywords. A description of how to read the syntax diagrams is provided on page 8 (following the sample of the `DISP` dictionary entry).
- ⑤ **Examples:** Illustrate some of the different ways you can use the keyword. (The examples are separate from each other and, unless otherwise indicated, should *not* be read as if they are part of the same instruction.) Examples that use quoted strings are shown with pairs of double quotes (" ... "). Except in the case of quotes within quotes (" ' ... ' "), or where a pair of double quotes enclose one single quote (" ... ' ... "), a pair of single quotes (' ... ') can be used in place of a pair of double quotes.
- ⑥ **Input Parameters Table:** Further specifies the parameters used in the syntax diagram.

Note: The various types of input parameters referred to in the syntax diagrams and "Input Parameters" tables are defined in the glossary.
- ⑦ **Comments:** Additional information about the use of the keyword.
- ⑧ **Related Keywords:** Other keywords that have either a functional similarity to the keyword being described or an influence on its results.

1 DISP

2 DISP displays numeric and string data.

3

■ Statement

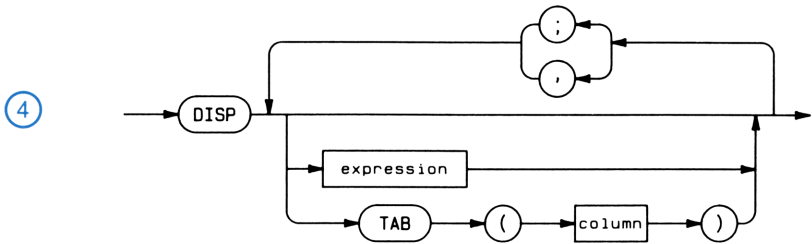
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



5 Examples

```
DISP I;TAB(I+5);"*"           DISP
DISP "Hello";A$;".","Welcome"  A$,"DATE: ",D$
```

6 Input Parameters

Item	Description	Restrictions
expression	Numeric or string expression.	None.
column	Numeric expression rounded to an integer.	Greater than zero.

7 Comments

You can omit the statement name itself in all cases except immediately after THEN or ELSE.

-
-
-

8 Related Keywords

DISP USING, ENG, FIX, PRINT, SCI, STD, STR\$, WIDTH.

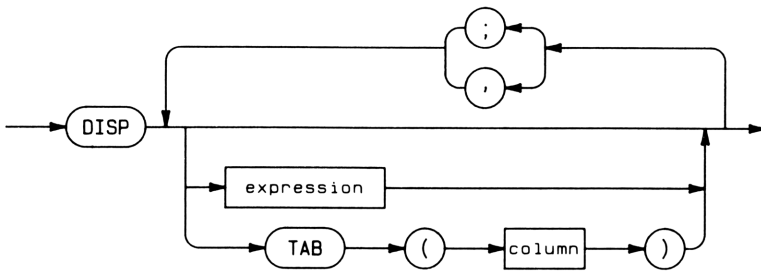
How To Read the Syntax Diagrams

The diagrams indicate acceptable keyword input syntax. (Incorrect syntax in an input prevents the computer from accepting that input, which results in an error message.)

Note: The syntax diagrams illustrate the general use of blanks in keyword inputs. For a more detailed discussion of this topic, refer to “Using Blanks in Keyword Inputs,” on the next page.

Double quotes are used in these diagrams. However, you can also use single quotes. (The opening and closing quote symbols must match.)

Example.



- Items enclosed in ovals and circles can be entered in either uppercase or lowercase letters. Blanks should not be embedded unless explicitly shown.
- Items in the boxes correspond directly to entries in the “Item” column of the “Input Parameters” table included in the keyword description.
- All valid paths through the diagrams are indicated by arrows. Looping paths indicate a parameter or sequence that can be repeated.
- An item is optional if there is a valid path around it. Options are generally shown as descending loops.
- An arrow (→) terminating the exit path from a statement indicates that you can use the `␣` character to append another statement to the illustrated statement.
- An arrow and an “X” (→X) terminating the exit path from a statement indicates that no statement can be appended after the subject statement.
- An arrow with a vertical bar (→|) used to terminate the exit path from a statement indicates that other statements can be appended, but they will not be executed following execution of the illustrated statement.

Using Blanks in Keyword Inputs

You can use blanks as separators between keywords, parameters, and punctuation unless otherwise indicated in a keyword entry. However, blanks are not needed as separators except where required by the HP-71 to prevent misinterpretation or error conditions when processing a keyword input.

The HP-71 uses blanks as delimiters for unquoted strings. Without these delimiters, using an unquoted string for an input such as a file type name or an unquoted file specifier can cause an error. For example, the HP-71 accepts file type names having up to five characters. Thus, to tell the computer to “create file ABC of type TEXT,” you would use the statement

```
CREATE TEXT ABC
```

which must have blanks as shown. Omitting the blanks results in the statement

```
CREATETEXTABC
```

which the HP-71 interprets as “create file BC of type TEXTA” (which is a nonexistent type). This second interpretation results in an error message because there is no such file type. Similarly, if you wanted your HP-71 to copy file A to file B, you would enter `COPY A TO B`. (Using the same statement without blanks—`COPYATOB`—tells the computer to copy file ATOB to main RAM.)

Ambiguity in program statements can also occur in certain uses of an unquoted file specifier that contains a device specifier. For example, to tell the computer to copy file A from a plug-in device to file B, you would use a program statement such as

```
100 COPY A:PORT TO B
```

which must have blanks, as shown. Omitting the blanks results in the statement

```
100 COPYA:PORTTOB
```

which causes the computer to interpret the string `COPYA:` as a label. Because the remainder of the statement (as interpreted) does not contain a valid statement, an error results.

Using the Glossary

The Glossary (page 346) describes the terminology used in this manual and in your *HP-71 Owner's Manual*, and includes syntax diagrams that illustrate some of the common terms used in the keyword dictionary to specify keyword inputs.

Using the Keyword Index

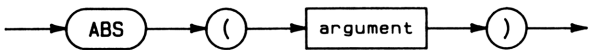
The keyword index, which appears at the end of this manual, provides you with both a reference for locating keywords by page number and a listing of keywords by functional category (such as “Input/Output” statements and “File Management” statements).

HP-71 Keyword Dictionary

ABS

ABS(x) returns the value of x with its sign set to “+.” This also applies to the following two cases:

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
E=ABS(X-X1)
PRINT "negative part ="; -ABS(X)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

ABS (x) returns the value of x with its sign set to “+.” This also applies to the following two cases:

- ABS (-0) = +0
- ABS (NaN) = NaN, with its sign field forced to +.

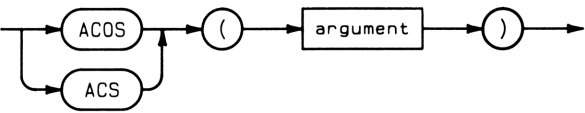
Related Keywords

SGN.

ACOS (ACS)

ACOS (*arccosine*) returns the principal value of the angle having a cosine equal to its argument. The angle returned is expressed in the current angular setting.

- ☐ Statement
- ☒ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☒ CALC Mode
- ☒ IF . . . THEN . . . ELSE



Examples

ACOS (X+.5*Y/Z)

T=T+ACOS (SQRT(1-Y^2)/R)

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	− 1 through 1.

Comments

The range of the result of ACOS is shown in the table to the right.

You can change the angular setting by executing DEGREES (when in Radians setting), or RADIANS (when in Degrees setting). Refer also to the OPTION statement.

Angular Setting	Range of ACOS
Degrees	0 through 180
Radians	0 through PI

Related Keywords

ACS, ANGLE, ASIN, ASH, ATAN, ATN, COS, DEG, DEGREES, OPTION, RAD, RADIANS, SIN, TAN.

ADD

ADD adds the coordinates of a data point to the summary statistics in the current statistical array.

■ Statement

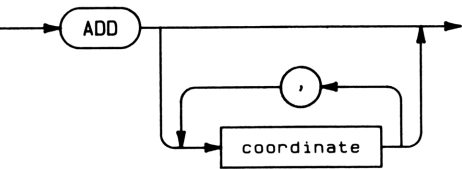
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF . . . THEN . . . ELSE



Examples

```
ADD                               ADD 1,34.87365,A,B
ADD V1,V2,V3,V4,V5
```

Input Parameters

Item	Description	Restrictions
coordinate	Numeric expression giving coordinate (value) of the data point.	None.

Comments

Numeric expressions you specify when you execute ADD represent the coordinates (values) of the data point. The number of expressions must be within the range specified by a preceding STAT statement (0 through 15). Any missing coordinates are assumed equal to zero.

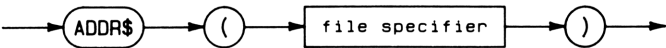
Related Keywords

CORR, CLSTAT, DROP, LR, MEAN, PREDV, SDEV, STAT, TOTAL.

ADDR\$

ADDR\$ (*address string*) returns a *string* representing the hexadecimal address of the file you specify.

- ☐ Statement
- ☒ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

```
A$=ADDR$ ("FILE1")
DISP ADDR$ (B$)
DISP ADDR$ ("FILE2:PORT")
```

Input Parameters

Item	Description	Restrictions
file specifier	String expression containing a file specifier.	File name with optional port specifier.

Comments

ADDR\$ returns the address of the beginning of the file header.

Related Keywords

DTH\$, HTD, PEEK\$, POKE.

ADJABS

ADJABS (*adjust-absolute*) performs an *absolute* adjust on the system clock.

■ Statement

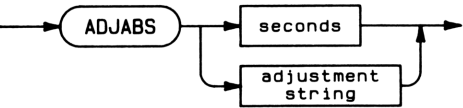
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
ADJABS 124.7           ADJABS "+00:05:45"       ADJABS A$
ADJABS -3300           ADJABS "-00:02:14"
ADJABS "00:03:35"      ADJABS N
```

Input Parameters

Item	Description	Restrictions
seconds	Numeric expression.	Seconds (s) in the range $-360000 < s < 360000$.
adjustment string	String expression in form "HH: MM: SS" or "-HH: MM: SS."	Maximum of 99 ^h 59 ^m 59 ^s .

Comments

ADJABS changes only the HP-71's clock time; the current clock speed is not affected. To adjust both the clock time and the clock speed, refer to the ADJUST statement on the next page.

Related Keywords

ADJUST, AF, EXACT, SETTIME.

ADJUST

ADJUST simultaneously changes the clock time and specifies a clock speed correction that is stored and applied the next time you execute EXACT to calibrate the clock.

■ Statement

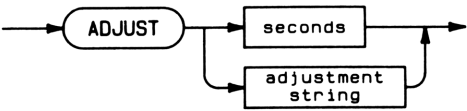
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

ADJUST 124.7

ADJUST -3300

ADJUST "00:03:35"

ADJUST "+00:05:45"

ADJUST "-00:02:14"

ADJUST A\$

ADJUST N

Input Parameters

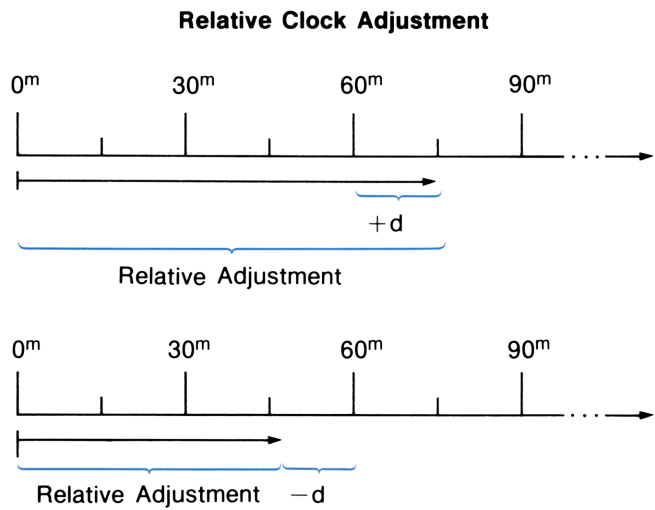
Item	Description	Restrictions
seconds	Numeric expression.	Seconds (s) in the range $-360000 < s < 360000$.
adjustment string	String expression of form "HH:MM:SS" or "-HH:MM:SS".	Maximum of 99 ^h 59 ^m 59 ^s .

Comments

ADJUST performs the following:

1. Changes the clock time by the amount you specify.
2. Stores the difference between amount you specify and the nearest multiple of 30 minutes as an error correction that will be implemented the next time you execute EXACT.

ADJUST (continued)



Executing `ADJUST` performs the relative clock adjustment and stores *d* as data for `EXACT` correction factor computation.

Because the error correction is added to any error correction already stored as a result of earlier executions of `ADJUST`, you can execute `ADJUST` as many times as you want to before executing `EXACT`. `EXACT` uses the accumulated error correction, then clears it from memory.

ADJUST Input	Time Change	Error Correction
-3300 00:03:35 -00:02:14	-55 ^m + 3 ^m 35 ^s - 2 ^m 14 ^s	+5 ^m (Measured from -60 ^m to -55 ^m .) +3 ^m 35 ^s (Measured from 00 ^m to 3 ^m 35 ^s .) -2 ^m 14 ^s (Measured from 00 ^m to -02 ^m 14 ^s .)

If you execute `ADJUST` more than once before executing `EXACT`, the corresponding error corrections are accumulated.

The resolution of the clock system is 0.0019^s (1/512th of a second). Numeric input for `ADJUST` can specify fractional seconds; string input cannot.

Related Keywords

ADJABS, AF, EXACT, SETTIME.

AF returns the current value of the clock system *adjustment factor* (expressed in seconds), and gives you the option of setting a new adjustment factor.

☐ Statement

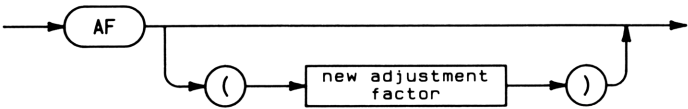
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

LET X = AF

DISP AF

LET Y = AF (1000)

Input Parameters

Item	Description	Restrictions
new adjustment factor (in seconds)	Numeric expression rounded to an integer.	Must be either 0 or within one of the following ranges: <ul style="list-style-type: none">• $10 \leq new\ af \leq 8,388,607$.• $-8,388,608 \leq new\ af \leq -10$.

Comments

If you specify a new adjustment factor with AF, that factor replaces the current adjustment factor in the HP-71's clock. However, regardless of whether you specify a new adjustment factor, AF always returns the *current* adjustment factor.

The HP-71 adjusts the clock's internal time base by applying a 1-second correction at the intervals specified by the adjustment factor. A positive AF adds the correction; a negative AF subtracts the correction. Specifying a new adjustment factor in the range of either $1 \leq new\ af \leq 9$ or $-9 \leq new\ af \leq -1$ results in an Invalid AF (error 27) message. Specifying $new\ af > 8,388,607$ or $new\ af < -8,388,608$ defaults the new adjustment factor to 0.

AF (continued)

Related Keywords

EXACT.

AND

AND returns a 1 or a 0, based on the logical AND of its operands.

☐ Statement

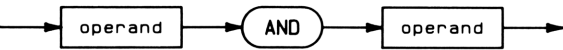
☐ Function

☒ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
IF Z = 25 AND A = C THEN GOSUB 100
Q = R AND S
```

Input Parameters

Item	Description	Restrictions
operand	Numeric expression.	Subject to operator precedence.

Comments

The operands of AND are considered to be logically false if zero and logically true if nonzero. The table to the right indicates the range of results for AND.

The precedence of AND in relation to the HP-71's other operators is described under "Precedence of Operators" on page 317.

Operand		Result
Left	Right	
False	False	0
False	True	0
True	False	0
True	True	1

Related Keywords

EXOR, NOT, OR.

ANGLE

ANGLE returns the polar angle determined by the (x,y) coordinate pair. The angle returned is expressed according to the current angular setting.

☐ Statement

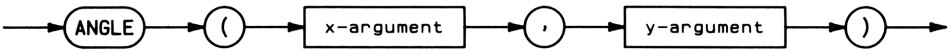
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF . . . THEN . . . ELSE



Examples

```
T = ANGLE (X,Y)
```

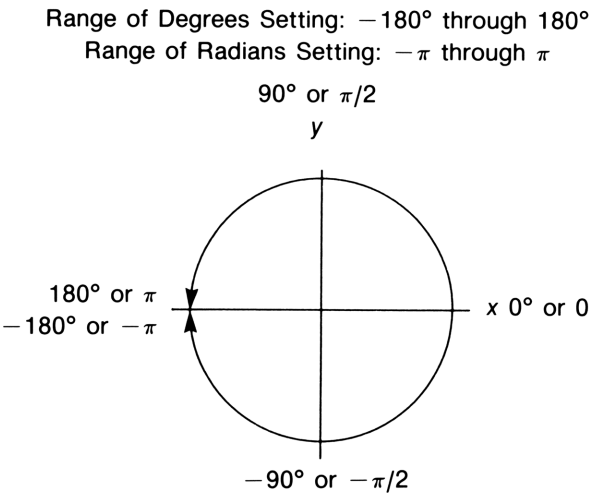
Input Parameters

Item	Description	Restrictions
x-argument	Numeric expression representing first coordinate of pair (x,y).	None.
y-argument	Numeric expression representing second coordinate of pair (x,y).	None.

ANGLE (continued)

Comments

The arguments need not both be finite. (Notice, however, that there are invalid argument *pairs*. For example, ANGLE (INF, INF) is invalid.) Certain cases distinguish the sign of a zero argument. The range of ANGLE is shown in the following illustration.



Where $s(a) = \text{sign}(a)$ —except for $s(+0) = 1$ and $s(-0) = -1$ —the following table defines ANGLE (x, y).

Normal Inputs		ANGLE (x, y)
x	y	
± 0	± 0	y
$s(x) = 1$	Any Number	ATAN(y/x)
$s(x) = -1$	$s(y) = 1$	ATAN(y/x) + 180 (in Degrees setting)
$s(x) = -1$	$s(y) = -1$	ATAN(y/x) - 180 (in Degrees setting)

Related Keywords

ACOS, ACS, ASIN, ASN, ATAN, ATN, COS, DEG, DEGREES, OPTION, RAD, RADIANS, SIN, TAN.

ASIN (ASN)

ASIN (*arcsine*) returns the principal value of the angle having a sine equal to its argument. The angle returned is expressed in the current angular setting.

☐ Statement

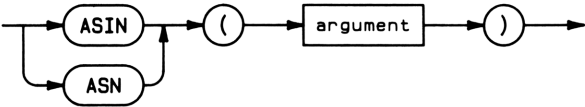
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF . . . THEN . . . ELSE



Examples

```
PRINT ASIN (X/R)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	−1 through 1.

Comments

You can change the angular setting by executing DEGREES (when in Radians setting) or RADIANS (when in Degrees setting). Refer also to the OPTION statement. The range of the result of ASIN is shown in the table to the right.

Angular Setting	Range of ASIN
Degrees	−90 through 90
Radians	−PI/2 through PI/2

Related Keywords

ACOS, ACS, ANGLE, ATAN, ATN, ASN, COS, DEG, DEGREES, OPTION, RAD, RADIANS, SIN, TAN.

ASSIGN

ASSIGN# associates a symbolic channel number with a specified file and opens that file.

- Statement

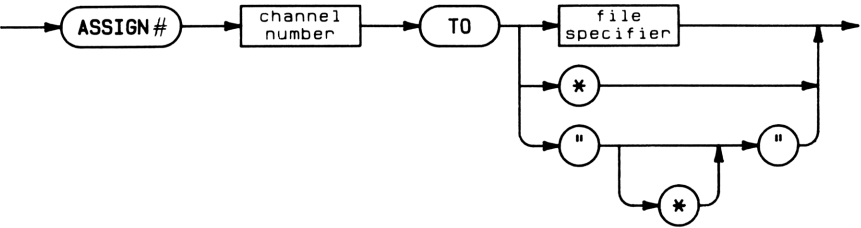
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

ASSIGN# 1 TO FILE1

ASSIGN# L TO FILE2:PORT (2)

ASSIGN# 6 TO *

ASSIGN# 6 TO ""

Input Parameters

Item	Description	Restrictions
channel number	Numeric expression rounded to an integer.	1 through 255.
file specifier	String expression or unquoted string.	File name with optional device specifier. Cannot reference CARD device.

ASSIGN # (continued)

Comments

A given channel can be assigned to only one file at a time. Thus, if you specify a channel that is already assigned to another file, that file is closed and the channel is then assigned to the new file.

An assigned channel is automatically released, and its associated file closed, by assigning the channel number to *, "!", or ". Also:

- Executing `RUN` or `END ALL` releases all channels.
- Executing `END` or `STOP`, or encountering the end of the program releases all channels in the local environment.
- Executing `END SUB` releases only those channels local to the subprogram.

If `ASSIGN#` specifies a file, but no device, and cannot find the file, the HP-71 creates a `DATA` file (of size zero, with a 256-byte record length) in main RAM and gives this file the name specified by `ASSIGN#`. However, if `ASSIGN#` specifies a file and a device, but the file cannot be found on this device, an error condition occurs.

Related Keywords

`CREATE.`

ATAN (ATN)

ATAN (*arctangent*) returns the principal value of the angle having a tangent equal to its argument. The angle returned is expressed in the current angular setting.

☐ Statement

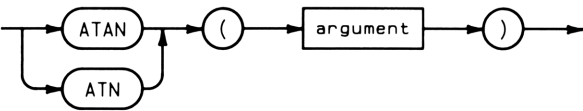
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A = ATAN (T)          PRINT ATAN (Y/X)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	−INF through INF.

Comments

You can change the angular setting by executing DEGREES (when in Radians setting) or RADIANS (when in Degrees setting). Refer also to the OPTION statement. The range of the result of ATAN is shown in the table to the right.

Angular Setting	Range of ATAN
Degrees	−90 through 90
Radians	−PI/2 through PI/2

Related Keywords

ACOS, ACS, ANGLE, ASIN, ASN, ATN, COS, DEG, DEGREES, OPTION, RAD, RADIANS, SIN, TAN.

AUTO

AUTO begins automatic line numbering for editing the current file, and is nonprogrammable.

■ Statement

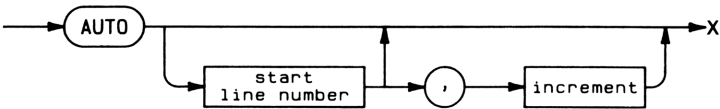
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

AUTO

AUTO 25

AUTO 100, 25

Input Parameters

Item	Description	Restrictions
start line number	Integer constant identifying a program line. Default: 10.	1 through 9999.
increment	Integer constant. Default: 10.	1 through 9999.

Comments

AUTO numbers new program lines as they are entered and stored. Automatic numbering begins with the start line number and continues with the specified or default increment. If an automatically displayed line number corresponds to an existing line in program memory, that line is displayed. If the current line number added to the increment value exceeds 9999, a *wrap-around* takes place. That is:

Next Line Number = (*Current Line Number* + *Increment*) − 9999

AUTO (continued)

Any one of the following operations terminates `AUTO` operation:

- Typing over the displayed line number with a syntactically correct line that does not begin with a line number, then pressing `END LINE`.
- Deleting the line by pressing `END LINE` when the display following the line number is blank.
- Pressing `ATTN` .

BEEP, BEEP OFF/ON

BEEP causes a tone to sound at the specified frequency and duration. BEEP OFF disables the beeper. BEEP ON enables the beeper.

■ Statement

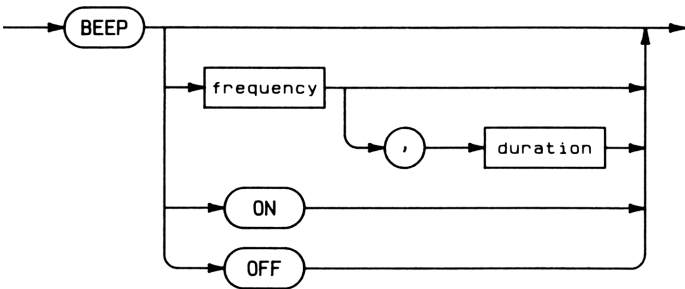
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

BEEP

BEEP F

BEEP 261,1

BEEP OFF

BEEP ON

Input Parameters

Item	Description	Restrictions
frequency	Numeric expression rounded to an integer. Default: 500 Hz.	Refer to "Comments," below.
duration	Numeric expression rounded to the nearest thousandth. Default: 0.25 seconds.	$D \leq 1048.575$ seconds.

Comments

The beeper has two volume settings; normal (default) and loud. You can switch the volume to a loud tone by setting flag -25 (the system beep volume flag). Clearing flag -25 returns the beep volume to its normal (default) level.

BEEP, BEEP OFF/ON (continued)

BEEP Statement. The frequency of the tone is subject to the resolution of the built-in tone generator. The maximum frequency is approximately 4900 Hz. A specified duration greater than the maximum indicated under “Restriction” on the preceding page defaults to the maximum.

BEEP OFF Statement. `BEEP OFF` deactivates the beeper by setting flag `-2` (the beep flag). When deactivated, the beeper does not operate for any purpose, including execution of `BEEP`.

Note: After executing `BEEP OFF`, a tone does not sound when the HP-71 detects an error condition.

`BEEP OFF` does not affect the current setting of the beep volume flag (flag `-25`).

BEEP ON Statement. `BEEP ON` activates the beeper by clearing flag `-2` (the Beep flag). `BEEP ON` does not affect the current setting of flag `-25` (the beep volume flag).

BYE

BYE turns off the HP-71.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

BYE

Comments

General Operation. Pressing after executing BYE turns on the HP-71. Executing BYE in a program causes the HP-71 to automatically resume execution of the program—at the instruction following BYE—when the computer is turned on again. However, when executing BYE from the keyboard, any statements concatenated after BYE are not executed when the computer is turned on again.

Timer Control of Program Execution. If the HP-71 activates a timer by executing ON TIMER, then subsequently executes BYE, the computer turns itself off as described above. However, when the timer expires, the computer turns on, executes the branch indicated by the timer, then continues executing the program.

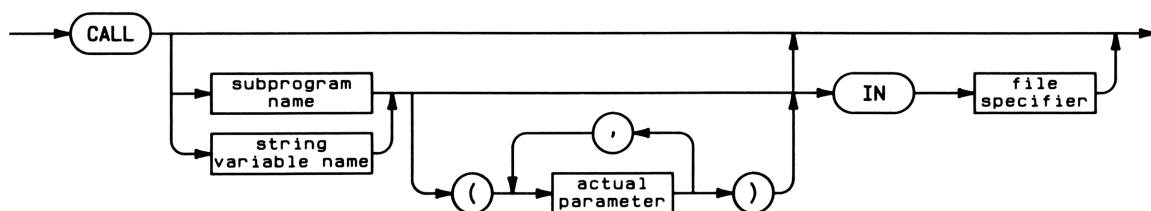
Related Keywords

OFF.

CALL

CALL transfers program execution to a specified subprogram and may also pass parameters to that subprogram.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF ... THEN ... ELSE



Examples

```
CALL
CALL SETUP1
CALL C1$(A,B$,X+Y,"YES")
CALL CONTROL(A1(4),P(),Q(),D#[3,10]) IN IOSUBS
CALL "EXECUTE"((L), 3, N) IN GENERAL
```

CALL (continued)

Input Parameters

Item	Description	Restrictions
subprogram name	Quoted or unquoted string.	One to eight characters. The first character must be a letter; any remaining characters can be letters and/or digits in any combination.
string variable name	Refer to Glossary.	Must evaluate to subprogram name.
actual parameter	Constant, expression, variable name, or channel expression (numeric expression preceded by #).	Expressions cannot reference user-defined functions. Rounded value of a channel expression must be in the range of 1 through 255.
file specifier	String expression or unquoted string.	File name, with optional device specifier. Expressions cannot reference user-defined functions.

Comments

A subprogram may call itself. User-defined functions may not appear anywhere in the parameter list.

Passing Parameters.

- The parameters must be of the same type (numeric, string, or channel number) as the corresponding parameters in the subprogram's SUB statement.
- Numeric expressions are passed by value to the corresponding numeric variable.
- A numeric variable may be passed by value by enclosing it in parentheses.
- An array passed by reference may have () or (,) following the variable name indicating the number of dimensions (1 or 2) of the passed array. However, the () or (,) is necessary *only* in the SUB parameter list.
- String expressions or substrings are passed by value (up to a maximum length of 32 characters).
- String variables can also be passed by reference.

CALL (continued)

A channel number in the actual parameter list is a # followed by an integer constant or a numeric expression. However, the corresponding formal parameter must be a # followed by an integer constant. A channel number opened by a subprogram (and not passed as a parameter) is local to the subprogram. If a subprogram has no parameter list, then the subprogram's channel numbers are the same as those for the calling program, but all variables are local to the subprogram.

Note: Because of this feature, if the subprogram takes no parameters due to an omission in `CALL` or `SUB`, you may unexpectedly discover channel numbers intended for local use are being used by the calling program.

Subprograms. You can enter more than one subprogram in a program file. A subprogram must start with `SUB` and end with `END SUB`. Any program lines between two subprograms will not be executed in normal program flow. You can store subprograms in the same file as a main program, as long as the the main program precedes any subprograms. It is recommended that you use a different name for each of your subprograms. Otherwise, there is a possibility that one of two or more commonly-named subprograms will always be called, regardless of your intent, while any other subprograms having that name will not be found by the computer.

The HP-71 searches for a subprogram as follows:

1. The current program file.
2. Any other program files in HP-71 memory, in the order in which they appear in the system catalog.
3. Program files in any plug-in memory modules, in port number order.
4. If the subprogram is not found, and `CALL` passes no parameters, then the subprogram name is searched for as a program *file* name. (If a file name is specified by the `IN` keyword, as shown in the syntax diagram on page 31, only that file will be searched.)

Executing `CALL` creates a local environment* for the subprogram. (That is, a new set of variables is created which does not disturb the old variables unless they were passed as parameters). When execution returns from the subprogram to the calling program, the HP-71 reactivates the calling program's environment.

* For further information concerning local and global environments, refer to "Scope of Environments" on page 314, and to your *HP-71 Owner's Manual*.

CALL (continued)

Executing CALL From the Keyboard. The current file can be executed in a local environment by executing `CALL` `ENDLINE` from the keyboard.

Note: If you execute `CALL` from the keyboard and

1. You concatenate other statements after `CALL`, and
2. `CALL` invokes a subprogram that is subsequently suspended, then resumed,

then, when the HP-71 returns from that subprogram it does not execute any of the remaining concatenated statements.

Related Keywords

`END SUB`, `SUB`.

CAT returns a catalog of file information.

■ Statement

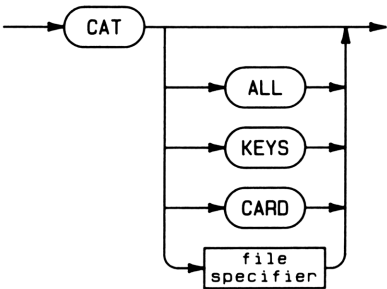
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

CAT TESTPR3

CAT ABC:PORT

CAT:MAIN

CAT:PORT(1)

CAT KEYS

CAT

CAT A\$

CAT ALL

CAT CARD

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default: File name without device specifier catalogs named file. Device specifier without file name catalogs all files on that device. Executing CAT without specifying either file name or device specifier catalogs the current file.	A device specifier or a file name with optional device specifier.

CAT (continued)

Comments

The catalog display provides the following file information:

- Name.
- Protection, coded as one of the following: `P` (private), `S` (secure), `E` (private and secure), or blank (no protection).
- Type.
- Length (in bytes).
- Creation date, time, and port number (if any). The date is displayed in the *MM/DD/YY* format, and the time is displayed in the *HH:MM* format.

When you use `CAT` to list the information on more than one file in a memory device, the files appear in the order in which they were created, beginning with the oldest file. The `▲` and `▼` keys scroll from one file to the next. Pressing `g▲` positions the HP-71 to the first (oldest) file in the catalog. Pressing `g▼` positions the HP-71 to the last (newest) file in the catalog.

Specifying Devices. `CAT:device id` returns information on all files residing on the given device. `CAT:MAIN` returns information on all files residing in MAIN memory, while `CAT:PORT(1)` returns information on all files residing on port #1.

Cataloging Devices. If one or more memory devices containing files are plugged into your HP-71:

- Executing `CAT ALL` returns information on the files in main RAM and in any plug-in memory modules and independent RAMs.
- Executing `CAT:PORT` returns information only on the files in any plug-in memory modules and independent RAMs.

Pressing `f▼` during a `CAT ALL` or `CAT:PORT` operation causes the HP-71 to move from the current memory device catalog to the next plug-in memory module.

Assigning a Displayed BASIC File to “Current” Status. Pressing the `EDIT` typing aid (`fEDIT`) while in any multiple-file catalog causes the currently displayed file to become the current file. An `Invalid File Type` condition occurs if the file you select is not of type BASIC.

Cataloging a Magnetic Card Containing an HP-71 File. Executing `CAT CARD` displays `CAT: Align then ENDLN` to prompt you to read a magnetic card into the HP-71. After you pull the card through the card reader, the HP-71 displays the file name, track number, and catalog information for the card in the same way as described in the preceding text for other file media.

CAT (continued)

Note: Because of the overhead information needed for card storage, the indicated file size may be larger than the actual file size. In BASIC files, this discrepancy will be, at the most, one byte. However, for TEXT files—which are always “padded” to a 256-byte boundary—the discrepancy will be in the range of 0 to 255 bytes.

Cataloging a Magnetic Card Containing an HP-75 File. The HP-71 catalogs HP-75 magnetic cards in the same way as for HP-71 cards, except that the file type is displayed as `Ø` for any file that is not a “LIF1” type.

Cataloging Key Assignments. Executing `CAT KEYS` displays catalog information for the `keys` file, which is the system file of current key assignments.

Note: If you create a file using keywords from a plug-in module or device, then remove the module or device and subsequently execute `CAT`, one or both of the following may occur:

- Instead of displaying file type, the HP-71’s five-digit internal code for the file type may be displayed.
- File size may be represented by a value that is 1 to 125 bytes greater than the actual file size.

Reexecuting the `CAT` instruction with the subject module or device plugged in eliminates both discrepancies.

Executing `DELAY` with a line rate of 0.5 and a character rate of infinity (`DELAY .5, INF`) prevents the catalog header from scrolling. Refer to the **DELAY** keyword entry.

Related Keywords

`CAT$.`

CAT\$

CAT\$ returns catalog information for the specified file.

☐ Statement

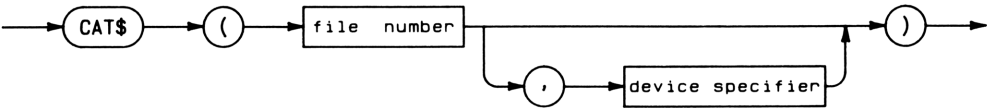
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
DISP CAT$(1)                                A$=CAT$(C1,";PORT(2)")
A$ = CAT$(I)C1,8]
```

Input Parameters

Item	Description	Restrictions
file number	Numeric expression (rounded to an integer) that corresponds to the file's position in the HP-71 or plug-in device.	None.
device specifier	String expression. Default: ;MAIN.	First character must be a colon.

Comments

For a specified file, CAT\$ returns in string form the same catalog information as that returned by CAT. For files in main RAM, the length of the string returned is 38 characters; for files on plug-in memory devices and independent RAMs, the string length is 43 characters. The last character is a blank. The character positions in the string are assigned as follows:

File Name	Security Code	File Type	File Size In Bytes	Date Created	Time Created	Port Number If In a Port
1–8	10	12–16	18–22	24–31	33–37	39–42

CAT\$ (continued)

Note: If the file type is unrecognized, the five-digit signed integer file type number occupies characters 11 through 16.

A positive file number refers to a file's position on the memory device. For example, `CAT$(2)` returns information on the second file in main RAM. If the specified file number is less than or equal to zero, and no second parameter is given, then `CAT$` returns information on the current file.

Specifying a file number greater than the last file in the specified device returns the null string. The null string is also returned if the file number is less than or equal to zero and a second parameter is specified.

If a device specifier string includes a file name, an `Invalid Filespec` (error 58) condition occurs.

Related Keywords

`CAT.`

CEIL

CEIL (*ceiling*) returns the smallest integer greater than or equal to a specified argument.

☐ Statement

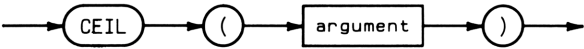
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

A1=CEIL(X1)

PRINT "Ceiling = ";CEIL(Y)

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

CEIL returns a value of type REAL. If the value of the numeric expression is an integer, that value is returned. For example:

- CEIL(1.5) returns 2.
- CEIL(-1) returns -1.
- CEIL(-1.5) returns -1.

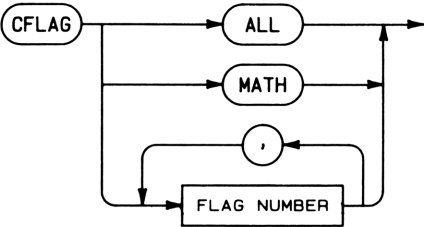
Related Keywords

FLOOR, FP, INT, IP.

CFLAG

CFLAG (*clear flag*) clears user and/or system flags specified by keyword or by a flag number list.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF . . . THEN . . . ELSE



Examples

CFLAG ALL

CFLAG MATH

CFLAG 1,2,INX

Input Parameters

Item	Description	Restrictions
flag number	Numeric expression rounded to an integer.	−32 through 63.

Comments

CFLAG ALL clears all user flags. CFLAG MATH clears the math exception flags. CFLAG with a flag number list clears the system and user flags specified by the values (rounded to integers) of the numeric expressions in the list. System flags numbered less than −32 cannot be cleared by CFLAG.

Related Keywords

OVZ, FLAG, INX, IVL, OVZ, SFLAG, UNF.

CHAIN

CHAIN purges the current file, copies the specified file into main RAM, and begins executing that file.

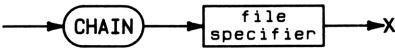
- ☒ Statement

☐ Function

☐ Operator
- ☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
CHAIN SEGMENTS          CHAIN TEST2:CARD          CHAIN P$
CHAIN DEMO:PORT(2)
```

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string.	File name with optional device specifier.

Comments

Chaining allows programs of unlimited size to be run by breaking up such programs into segments. A CHAIN statement in the first segment directs the system to copy and run a second segment. A CHAIN statement in the second segment directs the system to copy and run a third segment, and so on. When execution of a chained file begins, that file becomes the current file.

Note: When CHAIN calls the next file, the calling file is purged from memory before execution of the chained file.

CHAIN releases local environments and clears all program control information associated with a prior suspended program.

Related Keywords

COPY, RUN.

CHARSET

CHARSET (*character set*) defines the alternate character set in the ASCII code range of 128 through 255.

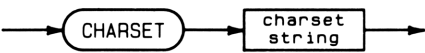
- Statement

□ Function

□ Operator
- Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
CHARSET CHR$(64)&CHR$(128)&CHR$(126)&CHR$(1)&CHR$(2)&CHR$(0)
CHARSET CHARSET$&CHR$(1)&CHR$(128)&CHR$(1)&CHR$(128)&CHR$(1)
CHARSET C$
CHARSET " "
```

Input Parameters

Item	Description	Restrictions
charset string	String expression.	None.

Comments

The character set string is interpreted as a series of six-byte groups. Each successive group defines a successive character from an alternate character set. The first group defines character 128; the second group defines character 129, and so on, up to character 255. To add a new character to an existing character set, use the CHARSET\$ function in the same way as shown in the second example, above (and described under the next keyword entry).

HP-71 Character Sets

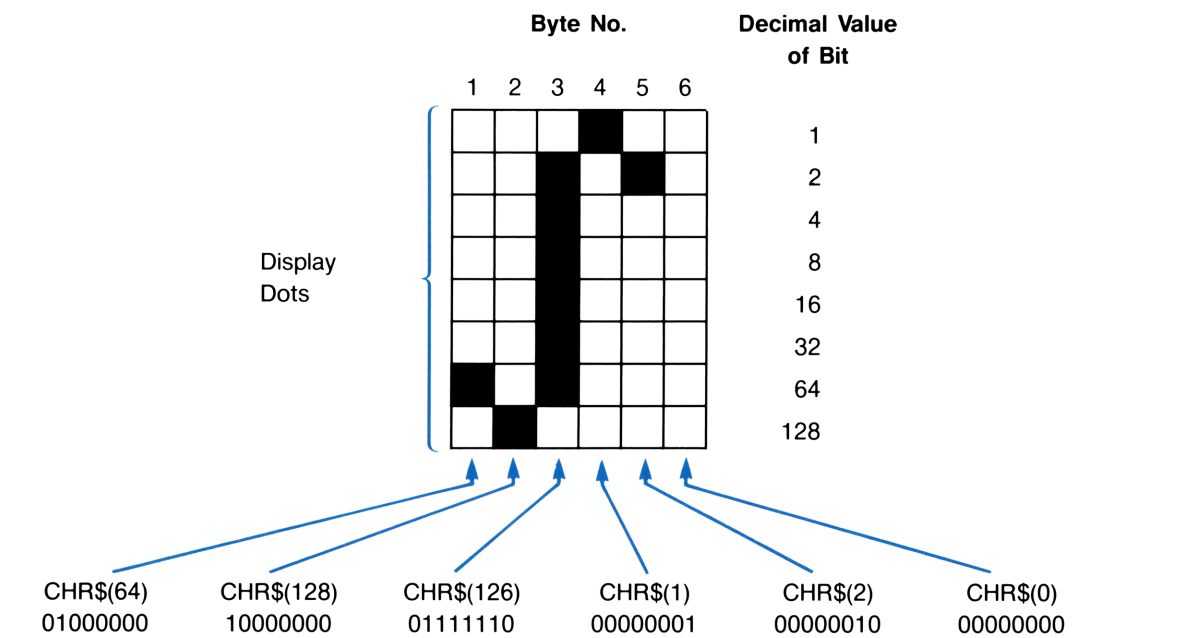
Standard	{	0
HP-71	{	1
Character	{	.
Set	{	.
	}	127
Alternate	{	128
HP-71	{	.
Character	{	.
Set	{	.
	}	255

CHARSET (continued)

Once an alternate character set is established, you can access any character in the set using CHR#. For example, DISP CHR\$(132) displays the fifth character in the current alternate character set. If you specify less than 128 alternate characters and then use CHR# to select a character number outside of the defined range, the HP-71 returns the corresponding character from the standard HP-71 character set. Thus, if (1) you have an alternate set consisting of 11 characters (which would be numbered from 128 through 138), and (2) you specify the nonexistent 12th character by executing CHR\$(139), then the HP-71 returns the 12th character in the standard HP-71 character set. That is, the computer returns the same character that would be returned by executing CHR\$(11).

All predefined characters in the HP-71 are six columns wide—although the sixth column is left blank to allow for space between characters. Each six-byte group in the character set string corresponds to the dots of six consecutive columns in the HP-71 display. The first byte in a group specifies the leftmost column in the displayed version of the character. The sixth specifies the rightmost column. The lowest-valued bit of each byte corresponds to the topmost dot in a column, and the highest-valued bit corresponds to the lowest dot in the column. (Refer to the next illustration.)

The first of the statements listed under “Examples” on the preceding page defines an alternate character set having only one character (which corresponds to character number 128). This character resembles the symbol used in integration.



CHARSET (continued)

If you specify more than 128 characters (768 bytes) in an alternate character set, the HP-71 uses only the first 128. If the last group in the string does not contain six bytes, then the remaining bytes of that character are assumed to be zero.

An alternate character set uses three and one-half bytes of user memory, plus six bytes for each character in the set. `CHARSET " "` deactivates the alternate set and restores the memory to the user's available space.

Certain plug-in ROMs may activate alternate character sets without using this statement. In this case the character set resides in ROM and requires only seven bytes of user memory (RAM).

Related Keywords

`CHARSET$, CHR$.`

CHARSET\$

CHARSET\$ (*character set*) returns a string representing the current alternate character set.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A$ = CHARSET$                                GDISP CHARSET$CI*6,I*6+5J
```

Comments

The length of the returned string is determined by the number of alternate characters currently defined. Each character you define adds 6 bytes to the string; up to 128 characters can be defined. The maximum string length is 6×128 bytes.

CHARSET\$ returns the string of bytes specified in the last CHARSET statement. (Refer to the **CHARSET** keyword entry for the significance of the bytes.)

Related Keywords

CHARSET, CHR\$.

CHR\$

CHR\$ (*string conversion*) converts a numeric value into an ASCII character.

☐ Statement

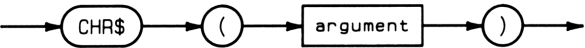
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

A\$[X,X] = CHR\$(47+X)

E\$ = CHR\$(27)

Input Parameters

Item	Description	Restrictions
argument	Numeric expression rounded to an integer.	0 through 255. (CHR\$ performs modulo 256 on numeric arguments.)

Comments

CHR\$ returns the one-character string having the specified ASCII value. For a table of ASCII characters and their equivalent decimal values, refer to pages 322 through 326.

Related Keywords

CHARSET, DISP, GDISP, NUM.

CLAIM PORT

CLAIM PORT returns an independent RAM module to main RAM status, and is nonprogrammable.

■ Statement

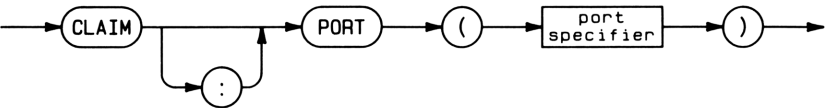
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
CLAIM PORT (0)           CLAIM PORT (A)
```

Input Parameters

Item	Description	Restrictions
port specifier	Numeric expression truncated to two digits after the decimal point; interpreted as <i>p.dd</i> , where: <i>p</i> = port number <i>dd</i> = device number.	$0 \leq p \leq 5$. $0 \leq dd \leq 15$.

Comments

Port 0 contains the internal chain of devices and the HP-IL port. Ports 1 through 4 are the four ports in the front of the HP-71. Port 5 is the card reader port.

Device number refers to the position of a plug-in device in a chain of such devices. In the HP-71, the internal RAM (Port 0) device numbers are 0.00, 0.01, 0.02, and 0.03 (where each device number represents 4K of RAM).

Because CLAIM changes the system configuration, all file pointers are reset, the workfile becomes the current file, and all FOR ... NEXT loops are terminated.

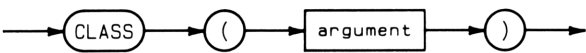
Related Keywords

FREE, SHOW.

CLASS

CLASS returns a value indicating the numeric class of the argument.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
IF CLASS(X)=-2 THEN TINYNEG
IF ABS(CLASS(X)) <= 3 THEN FINITE
ON ABS(CLASS(X)) GOTO ZERO,DENORM,NORMAL,INFINITE,QNAN,SNAN
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

The primary use for CLASS is program control. CLASS also provides a mechanism to discriminate between +0 and -0.

CLASS (continued)

CLASS returns a nonzero integer in the range of -6 through 6 , where:

- The sign of the result agrees with that of the argument.
- The result separates the machine representable numbers into 12 classes, as follows:

Class of Argument (x)	CLASS(x)	CLASS($-x$)
Zero ($+0$ or -0)	1	-1
Denormalized ($\text{MINREAL} \leq \text{ABS}(X) < \text{EPS}$)	2	-2
Normalized ($\text{EPS} \leq \text{ABS}(X) < +\text{INF}$)	3	-3
Infinity ($+\text{INF}$, or $-\text{INF}$)	4	-4
Quiet NaN	5	-5
Signalling NaN	6	-6

Related Keywords

SGN.

CLSTAT

CLSTAT (*clear statistical array*) clears (sets to zero) all elements in the currently specified statistical array.

■ Statement

□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
CLSTAT
```

```
IF I=1 THEN CLSTAT
```

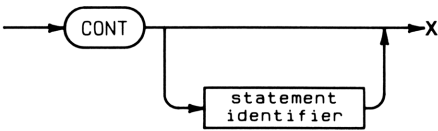
Related Keywords

```
ADD, CORR, DROP, LR, MEAN, PREDV, SDEV, STAT, TOTAL.
```

CONT

CONT (*continue execution*) continues execution of a suspended program, and is nonprogrammable.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
CONT                                CONT SECTION4                        CONT L$
CONT 250                            CONT "TEST5"
```

Input Parameters

Item	Description	Restrictions
statement identifier	Line number or label of a program statement.	Any valid line number or label reference.

Comments

CONT Operation. A running program will be suspended by a PAUSE statement, an error condition, pressing [ATTN], or single-stepping ([SST]).

Executing CONT without specifying a line number or label reference (or pressing [F] [CONT]) resumes execution at the suspend statement. Specifying a line number or label reference with CONT resumes program execution at that line number or statement. If the HP-71 does not find the specified line number, but does find a higher-numbered line, execution is continued from this line number.

If an error occurs during program execution, executing CONT continues the program with the statement causing the error. This feature enables you to debug a program that has been halted by an error, then resume execution. For example, errors related to variables can be corrected and the program continued from the statement in error.

CONT (continued)

Note: Editing a program clears the suspended state of a program. Thus, attempting to continue a program after editing that program causes execution to begin at the start of the program.

Continuing a program that is not suspended is equivalent to running the program.

Suspended Programs. When a program is suspended for any reason, the **SUSP** annunciator is displayed. While a program is suspended, memory is allocated to the suspended program for local variables, subprograms, and other program control information. Modifying a program or executing `END`, `END ALL`, or `STOP` clears the **SUSP** annunciator and releases the memory used for program control. `CONT` clears the **SUSP** annunciator, but does not release memory.

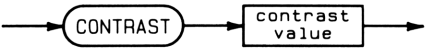
Related Keywords

`END`, `PAUSE`, `STOP`. See also the descriptions of the `ATTN`, `SST`, and `CONT` keys in your *HP-71 Owner's Manual*.

CONTRAST

CONTRAST adjusts the display contrast (viewing angle).

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
CONTRAST 10
```

Input Parameters

Item	Description	Restrictions
contrast value	Numeric expression rounded to an integer.	0 through 15.

Comments

CONTRAST controls the parameter specifying the angle at which the display is most easily viewed.

The contrast value may be varied between 0 and 15, where 0 is the shallowest angle and 15 is the steepest. This corresponds to the lowest and highest contrast, respectively.

The default display contrast value is 9.

COPY

COPY creates a new (destination) file and copies information from an existing (source) file to the new file.

■ Statement

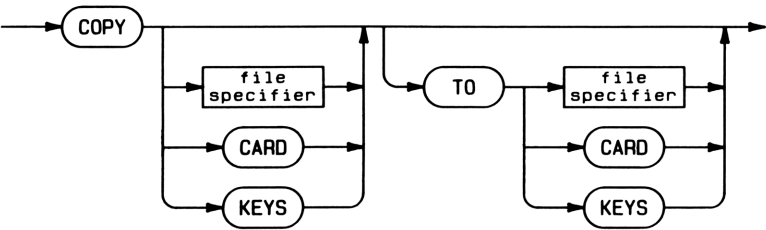
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
COPY TO DEM01
COPY FILE1 TO FILE2
COPY TO CARD
COPY A$ TO CARD
COPY "TEST" TO TEST5:PCRD
```

```
COPY GAMES:PORT
COPY CARD
COPY CARD TO GRAPH
COPY K1:PORT(2) TO KEYS
COPY KEYS TO KEYASN:PORT(1)
```

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Defaults: Refer to comments, below.	Any valid file specifier.

Comments

The source and destination files for a COPY operation can exist in main RAM, in an independent RAM, on a magnetic card, or on an external file storage device (except that when copying a file from a card, the destination file must be in main RAM). When copying into main RAM or to independent RAM, the destination file is a new file. The source file is never altered.

COPY (continued)

Defaults.

- When the current file is the source file, it is unnecessary to specify a source file name.
 - It is unnecessary to specify either the source or destination file names when the following three conditions apply:
 1. The current file is the source file.
 2. The current file is in a device that is plugged into a port.
 3. You want the destination file to be created in main RAM with the same name as the source file.
- Note:** The destination file specifier can be omitted *only* when the source file is in an external device or on a magnetic card. (The HP-71 treats a device plugged into a port as an external device.) If the source file is a special system file (such as `workfile` or `keys`), the HP-71 converts the destination file name to uppercase.
- When the destination *device* is not specified, it is assumed to be main RAM.

If a file having the same name as the destination file already exists in the destination device, a `File Exists` (error 59) condition results when `COPY` attempts to create the new (destination) file.

Card Reader Operation. You can use the reserved word `CARD` or the device specifiers `:CARD` or `:PCRD` to reference the card reader. The `:PCRD` specifier following the destination file name creates a “private” card file.

When copying a file to a card, the source file specifier must refer to a file that is currently in either the HP-71 or a memory device plugged into the computer. If the source file specifier is omitted, the current file is copied to the card. If the destination file specifier is omitted, the source file name is used as the destination file name on the card. For example, `COPY AFILE TO CARD` creates a card file named `AFILE`. `COPY AFILE TO BFILE:PCRD` creates a card file named `BFILE` from the source file (named `AFILE`).

If the source file is a special system file (such as `workfile` or `keys`), and the destination file-name is omitted, (as in `COPY KEYS TO CARD`), the HP-71 converts the destination file name to uppercase.

When copying a file to a card, each side of each card must be passed through the card reader twice. The second pass verifies that the data was recorded correctly. If the data was not recorded correctly, the HP-71 prompts you to rewrite the card.

When copying a card to memory, if the destination file name is omitted, the file name on the card becomes the destination file name.

COPY (continued)

Specifying `:PCRD` as the source device is equivalent to specifying `:CARD`. It has no effect on the privacy of the destination file.

Attempting to copy a card that was written on the HP-75 and contains a non-LIF1 file type causes an `Unknown Card` (error 69) condition. This condition can also be caused by an HP-71 file of an unknown type. (An unknown HP-71 file type results if the computer requires a certain LEX file or plug-in module to recognize the file type, and that LEX file or module is not present.)

Key Assignments. You can use `COPY KEYS` to store the current key assignments in a file. When copying from a key assignment file to `keys` (which is the current system key assignment file):

- If `keys` does not already exist, simply execute `COPY file specifier TO KEYS`.
- If `keys` already exists, either purge it (by executing `PURGE KEYS`) or rename it (by executing `RENAME KEYS TO file specifier`), then execute `COPY file specifier TO KEYS`. (Attempting to copy a key assignment file to `keys` when `keys` already exists results in a `File Exists` — error 59 — condition.)

Related Keywords

`PRIVATE`.

CORR

`CORR` (*correlation*) returns the sample correlation between a pair of variables in the current statistical array.

☐ Statement

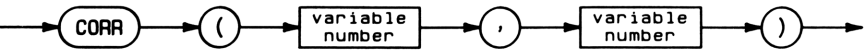
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A1=CORR(X1,Y1)
PRINT "Correlation = ";CORR(1,2)
```

Input Parameters

Item	Description	Restrictions
variable number	Numeric expression rounded to an integer.	Must be a value in the range of 1 through the current <code>STAT</code> array dimension.

Comments

`CORR` returns a `REAL` value.

Related Keywords

`ADD`, `CLSTAT`, `DROP`, `LR`, `MEAN`, `PREDV`, `SDEV`, `STAT`, `TOTAL`.

COS

COS (*cosine*) returns the cosine of its argument.

☐ Statement

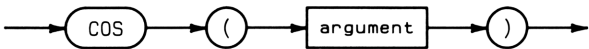
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
X=R*COS(T)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	Must be a finite number.

Comments

COS assumes that the argument is expressed in the current angular setting. The HP-71 first reduces the argument by 360° (or 2π if in radians setting). If in radians setting, this reduction takes place with a 31-digit representation of π for increased accuracy. Also:

- $\text{COS}(90 + n * 180) = [(-1)^n] * 0$; $n = 0,1,2,3, \dots$ (Degrees setting).
- $\text{COS}(-x) = \text{COS}(x)$.

Related Keywords

ANGLE, ACOS, ACS, ASIN, ASH, ATAN, ATN, DEG, DEGREES, OPTION, RAD, RADIANS, SIN, TAN.

CREATE

CREATE creates a data file in main RAM or in an independent RAM.

■ Statement

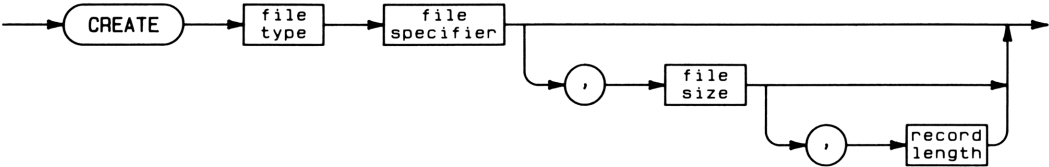
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF... THEN... ELSE



Examples

```
CREATE TEXT MEM0245
CREATE DATA A$,20,50
CREATE TEXT "FILE6:PORT(1)",2000
CREATE SDATA FILE41C,200
```

Input Parameters

Item	Description	Restrictions
file type	Unquoted string indicating the type of file to create.	TEXT, LIF1, DATA, or SDATA file type.
file specifier	String expression or unquoted string.	File name with optional device specifier.
file size	} Numeric expression rounded to an integer. Default: Refer to "Comments."	DATA and SDATA files: 1 through 65,535; TEXT files: 1 through 1,048,575.
record length		

Comments

The file type tells the HP-71 which format to use when writing information into the file.

Executing CREATE does not open the created file. To open a newly created data file for writing, use the ASSIGN# statement.

CREATE (continued)

A file placed in memory as a result of executing `CREATE` will increase in size if you use `PRINT#` to sequentially write to a new record immediately following the current last record of the file.

TEXT (LIF1) Files. A TEXT file contains variable length records of ASCII character data. The name “LIF1,” by which this file type is referenced in the HP-75, can be used in place of the name “TEXT.” However, the HP-71 still regards as a TEXT file any file carrying a “LIF1” designation. This file type is the same as that of the HP Logical Interchange Format (LIF) file type “1” which serves as an interchange file format among most Hewlett-Packard computers. The file size is specified in bytes. Both the minimum file length and the default file size are zero. Because a LIF1 file contains records of varying length, you cannot specify the file’s record length.

DATA files. A DATA file type has fixed length records which can contain both numeric and string values. The file size is specified in number of records, with a default size of zero. The record length is specified in number of bytes. If not specified, or specified as 0, the record length defaults to 256 bytes.

SDATA files. The SDATA file type is the same file format as the DA (data) file produced by the HP-41 calculator. Thus, SDATA files are register-oriented. Each record is 8 bytes in length. The HP-71 can write only numeric values to this type of file. Notice, however, that the HP-41 can store both numeric and alpha data in this file, which can be read by the HP-71 `READ#` statement. (Refer to **READ#** and **PRINT#** for further information.) The file size is specified by the number of records in the file, with a default size of zero.

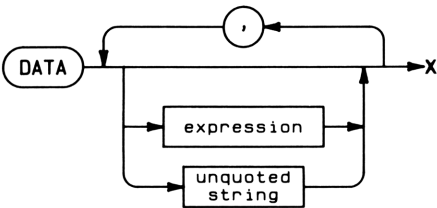
Related Keywords

`ASSIGN#`, `PRINT#`, `READ#`, `RESTORE#`.

DATA

DATA statements contain data that can be read by READ statements.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☐ Keyboard Execution
- ☐ CALC Mode
- ☐ IF ... THEN ... ELSE



Examples

```
DATA 123,456,"Hello"  
DATA PI*2,FGH"QQQ"  
DATA "This is a string" & " expression"
```

Input Parameters

Item	Description	Restrictions
expression	Refer to glossary.	None.
unquoted string	Refer to "Punctuation and Space in Data Items," on the next page.	None.

Comments

DATA statements provide the means for ordering string and/or numeric data items for assignment to corresponding variables listed in READ statements. Each time a running program encounters a variable in a READ statement, it assigns to that variable the next consecutive data item from a DATA statement. If a READ statement variable is encountered after all of the data items in a DATA statement have been assigned, the first data item in the next consecutively-numbered DATA statement is assigned to that variable. A program or subprogram can contain any number of DATA statements at any locations.

DATA (continued)

Data Statements in Subprograms. When a subprogram is called, the HP-71 remembers the location of the next item to be read in the calling program in anticipation of returning from the subprogram. Within the subprogram, the first item read is the first item in the lowest numbered `DATA` statement within the subprogram. When program execution returns to the calling program, the `READ` operations resume where they were suspended when the subprogram was called.

Constants and Variables. A numeric or string constant must be read into a variable that can store the constant. In the HP-71 a numeric constant (or expression) in a `DATA` statement can be read into either a numeric variable or a string variable. However, a string constant can be read only into a string variable. That is:

- Numeric values in a `DATA` statement can correspond to either numeric or string variables in a `READ` statement used to access that `DATA` statement. A numeric value read into a string variable is interpreted as an unquoted string.
- String values in a `DATA` statement can correspond only to string variables in a `READ` statement used to access that `DATA` statement. Attempting to read a string value into a numeric variable causes an immediate error condition.

If a string can be interpreted as a valid string expression (such as `DATA CHR$(X)`) followed by a comma or the end of the line, then it will be evaluated as such when read. Otherwise, the computer treats it as an unquoted string.

Punctuation and Spaces in Data Items. A data item can be any one of the following:

- Numeric expression.
- String expression.
- Unquoted string.

To place an unmatched quote mark in a string expression, you must enclose the mark within a pair of the opposite type quote marks. For example:

```
DATA "Class of '67", 43
DATA 'Elapsed Time: 53''
```

Also, the computer ignores any leading or trailing blanks in an unquoted string expression. Thus, to include such blanks in a string expression, they must be within quotes (that is, in a *quoted* string).

Since `DATA` statements can contain `!` and `@` symbols, `DATA` cannot be followed by a comment or another statement.

DATA (continued)

Related Keywords

READ, RESTORE.

DATE

DATE returns the current clock date.

- ☐ Statement
- ☒ Function
- ☐ Operator

- ☒ Keyboard Execution
- ☒ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

```
A=DATE          DISP DATE
```

Comments

DATE returns the current clock date as an integer in a year/day (YYDDD) format, where YY is the year and DDD is the day number of the year.

Related Keywords

DATE\$, SETDATE.

DATE\$

DATE\$ (*date string*) returns the current calendar date as an eight-character string in a year/month/day (YY/MM/DD) format.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A$ = DATE$           DISP DATE$
```

Related Keywords

DATE, SETDATE.

DEF FN

DEF FN (*define function*) indicates the beginning of a user-defined function definition.

■ Statement

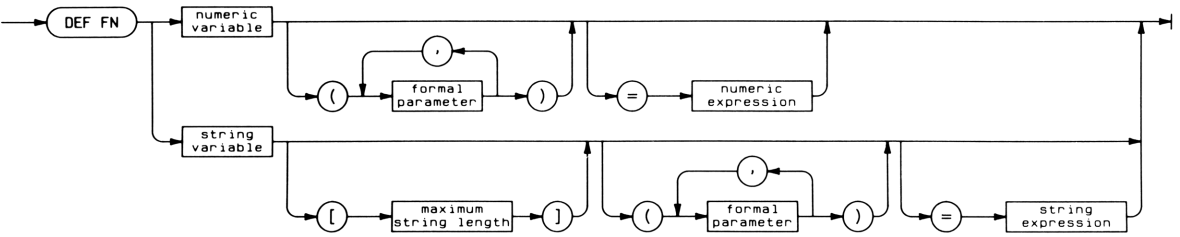
□ Function

□ Operator

□ Keyboard Execution

□ CALC Mode

□ IF ... THEN ... ELSE



Examples

```
DEF FNF(X,Y,Z)=(X*3+Y*4)/Z
DEF FNA(I,J,K,B$)
DEF FNF$(A,B,C$)=C$[A,B]
```

```
DEF FNL$[C128](T$,R$,X)
DEF FNO
```

Input Parameters

Item	Description	Restrictions
numeric variable	Letter, or letter followed by a digit.	None.
formal parameter	Numeric or string variable name.	None.
numeric expression	Refer to the Glossary.	None.
string variable	Letter followed by \$ or letter and digit followed by \$.	None.
maximum string length	Numeric constant. Default: 32.	0 to 65535.
string expression	Refer to the Glossary.	None.

DEF FN (continued)

Comments

A user-defined function computes a single value, and is used by specifying the function's name and actual parameter list in an expression; for example: `3*FNF(X)+1`. (Refer to the **FN** keyword entry on page 116.) A function definition can appear anywhere in a main program or subprogram, and can be composed of either a single statement or multiple statements. A `DEF FN` statement with no `=` symbol is the beginning of a multiple-statement function definition that must have a corresponding `END DEF` statement.

A user-defined function can have from 0 to 14 parameters. The formal parameters must match the actual parameters in both number and type. All parameters passed to a function are passed by value. Parameters appearing in the function definition list are *local* to the function and are inaccessible from the main program. However, all global variables having names that differ from the function's formal parameters can be accessed from within the function.

A string function returns a string value. The string length of a formal string parameter is automatically increased, if necessary, to match the length of the actual string value passed at execution. However, if the length of the formal string parameter becomes greater than the length of the actual string, an error results. For example, in the following program segment, the actual string value (`A$`) is three characters, but the corresponding formal string parameter (`T$`) contains five characters. Thus, line 110 would cause an error because the formal string length cannot be increased beyond that specified by the actual string length.

```
10 A$="ABC"
20 DISP FNL$(A$)
.
.
.
100 DEF FNL$(T$)
110 T$="12345"
120 END DEF
```

A function can call itself (which is termed *recursion*).

Related Keywords

`END DEF.`

DEF KEY

DEF KEY (*define key*) assigns a character string to the specified key.

■ Statement

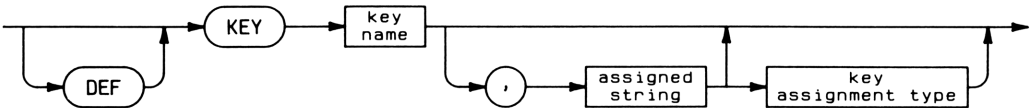
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
DEF KEY "#5","GOSUB 134"
KEY "C",A$;
DEF KEY E$, "RUN 200";
KEY "#64",'RUN "SUB1"'
```

```
DEF KEY "g1","CALL SUB1(I)"
DEF KEY "FQ","CALL PMT"
DEF KEY "A"
```

Input Parameters

Item	Description	Restrictions
key name	String expression indicating key to be defined. (Refer to glossary.)	Less than five characters.
assigned string	String expression.	None.
key assignment type	; or : symbol, or blank. (Refer to "Assigning Keys," on the next page.)	None.

Comments

Key assignments are useful for repetitive character string entry, data entry, partial command entry, direct execution, and generally customizing the keyboard to your needs.

DEF KEY (continued)

Assigning Keys. The following table describes the three types of key assignments:

Assignment Type	Punctuation Used To Define Key Assignment Type	Result of Pressing the Assigned Key
Typing Aid	Semicolon (;).	Displays the assigned string as though you manually typed it in.
Direct Execution	Colon (:).	The assigned string is executed without altering the display.
Immediate Execution	None (default).	The HP-71 displays the assigned string, then performs the same as if you typed in the assigned string and pressed <code>END LINE</code> .

Key assignments are only active when the user keyboard is active. If the first character of a multiple-character key name string is #, the string is interpreted as a key number. If the first character of a two-character string is upper- or lowercase “F” or “G,” the string is interpreted as the `[f]`- or `[g]`- shift of the key specified by the second character. Any excess or unnecessary characters in the key name string cause an error condition.

The numbers to use when entering a DEF KEY assignment are shown in the table to the right.

Note: The `[f]`- and `[g]`- shift keys cannot be redefined.

Key Set	Key Numbering
Primary Key Functions	1 through 56 (<code>[Q]</code> , <code>[W]</code> , <code>[E]</code> , ... <code>[+]</code>)
<code>[f]</code> - Shift Functions	57 through 112 (<code>[IF]</code> , <code>[THEN]</code> ... <code>[g]</code> - <code>[CONT]</code>)
<code>[g]</code> - Shift Functions	113 through 168 (<code>[q]</code> , <code>[w]</code> ... <code>[?]</code>)

The HP-71 stores any current key assignments in a special system file named `keys`. Each time you execute DEF KEY, the computer updates the `keys` file.

Deassigning Keys. DEF KEY also deassigns keys by returning them to their default assignments. To deassign a key, enter DEF KEY and the key name *only*, then press `END LINE` .

DEF KEY (continued)

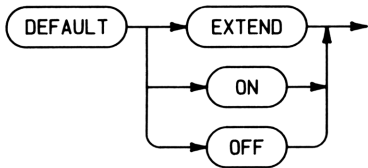
Related Keywords

CAT KEYS, COPY KEYS, FETCH KEY, LIST KEYS, PURGE KEYS, RENAME KEYS, USER, VIEW .

DEFAULT EXTEND/ON/OFF

DEFAULT sets the math exception traps to specific values.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
DEFAULT OFF                                DEFAULT EXTEND
DEFAULT ON                                IF A<0 THEN DEFAULT EXTEND
```

Comments

The trap actions may be set individually for each of the five exception flags by using the TRAP function described on page 293. However, there are some groups of actions that are common and can easily be set by the DEFAULT statements:

- DEFAULT OFF : Sets the traps for the UNF, OVF, DVZ, and IVL flags to zero. Sets the trap for the INX flag to 1. For the INX flag (flag -4) a trap value of 1 or 2 produces the same results. Refer to the description of the TRAP function in your *HP-71 Owner's Handbook*.
- DEFAULT ON : Sets the traps for the INX, UNF, OVF, DVZ, and IVL flag to 1.
- DEFAULT EXTEND: Sets the traps for the INX, UNF, OVF, and DVZ errors to 2. Sets the trap for the IVL flag to 1.

Related Keywords

TRAP.

DEG

DEG (*radians to degrees conversion*) converts arguments expressed in radians to degrees.

☐ Statement

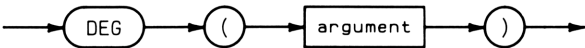
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
DEG(X+.1*Y)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

The conversion constant used by DEG is accurate to 15 digits, which often produces more accurate results than a conversion which does not use this function.

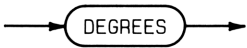
Related Keywords

```
DEGREES, RAD, RADIANS.
```

DEGREES

DEGREES sets the unit of measure for expressing angles to degrees. It is a short form of the OPTION ANGLE DEGREES statement.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
DEGREES                                IF A=0 THEN DEGREES
```

Comments

After you execute DEGREES, executing any function that returns an angle will return the angle in degrees units. Operations using parameters to represent angles interpret the angles in degrees.

The Degrees setting is the HP-71 default angular setting. Both the Degrees setting and its counterpart, the Radians setting, are global.

Executing DEGREES clears both the RAD annunciator and system flag -10.

Related Keywords

OPTION, RADIANS.

DELAY

DELAY sets the rate at which lines and characters within a line will scroll in the display.

■ Statement

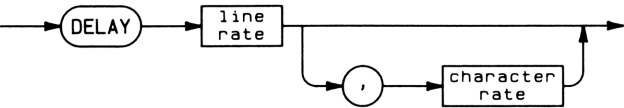
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF . . . THEN . . . ELSE



Examples

```
DELAY 1           DELAY 0,INF
```

Input Parameters

Item	Description	Restrictions
line rate	Numeric expression rounded to the nearest 1/32 second. Values greater than or equal to 8 are considered infinite.	Zero to infinity.
character rate	Numeric expression rounded to the nearest 1/32 second. Values greater than or equal to 8 are considered infinite. Default: 0.125 second.	Zero to infinity.

Comments

The *character rate* is the number of seconds to delay between characters in the display. A zero character rate causes the display to be immediately advanced to show the last part of the line. An infinite character rate causes the first part of the line to remain in the display indefinitely.

The *line rate* is the number of seconds to hold each display line (after character movement halts) before displaying the next line. An infinite line rate causes the line to remain in the display until you press a key. A line rate of zero implies that lines are not held in the display at all.

The HP-71 interprets as infinite any line and character rates that equal or exceed eight seconds. Rates of less than 0 are interpreted as 0.

DELAY (continued)

Related Keywords

DISP, LIST.

DELETE

DELETE deletes one or more program lines from the current file, and is nonprogrammable.

- Statement

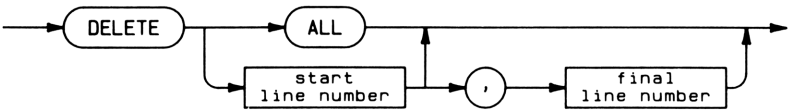
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

DELETE 10

DELETE ALL

DELETE 100,200

Input Parameters

Item	Description	Restrictions
start line number	Integer constant identifying a program line.	1 through 9999.
final line number	Integer constant identifying a program line. Default: Start line number.	Start line number through 9999.

Comments

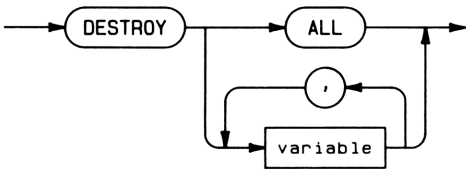
To delete a single line, specify only that line number. (You can also delete a single line without using the DELETE statement by entering the line number and pressing **END LINE**). To delete a block of program lines, specify the start and final line numbers in the DELETE statement.

DELETE ALL deletes all lines in the current file. (The file header remains.)

DESTROY

DESTROY deletes variables and arrays from memory.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF ... THEN ... ELSE



Examples

DESTROY X

DESTROY ALL

DESTROY A,R\$,M

Input Parameters

Item	Description	Restrictions
variable	Numeric or string variable name.	None.

Comments

Destroys variables and arrays, reclaiming the memory they consumed. Memory-reclaiming deletion of a specified array or variable occurs when DESTROY is executed. If the computer is executing a subprogram or user-defined function, only the variables created by that entity are deleted. DESTROY ALL deletes all variables in the current entity (program, subprogram, or user-defined function). Executing DESTROY ALL within a user-defined function causes program execution to immediately exit from that function in the same way that it does when executing END DEF.

Related Keywords

DIM, INTEGER, REAL, SHORT.

DIM

DIM (*dimension*) allocates memory for string or REAL variables and arrays.

■ Statement

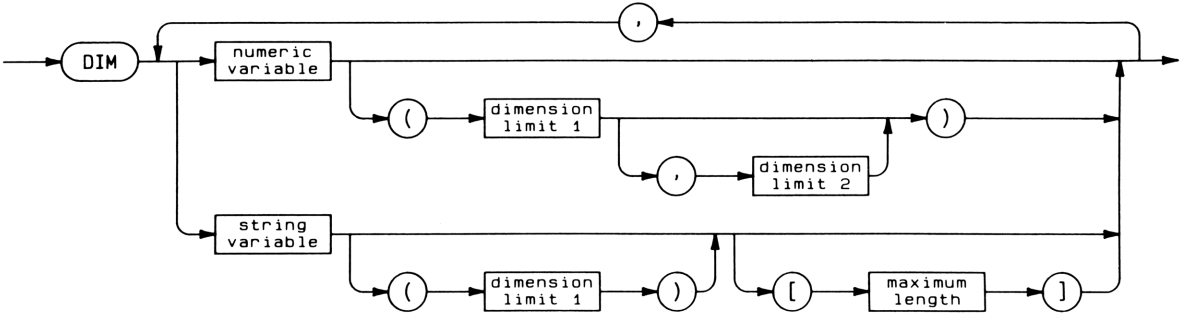
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

DIM Z

DIM W9(7,3)

DIM Z2, W(4,4)

DIM G\$[50]

DIM A\$(12)

DIM C\$(25)[30]

Input Parameters

Item	Description	Restrictions
numeric variable	Letter followed by optional digit.	None.
string variable	Letter followed by \$, or letter and digit followed by \$.	None.
dimension limit 1	Numeric expression rounded to an integer.	Option Base setting to 65535.
dimension limit 2		
maximum length	Numeric expression rounded to an integer. Default: 32.	1 to 65535.

DIM (continued)

Comments

DIM creates REAL variables and arrays, and string variables and arrays. Creation occurs upon execution of DIM. The dimension limits are evaluated at creation time. The lowest-numbered subscript in any dimension is 0 or 1, depending on the OPTION BASE setting when the array is created. OPTION BASE has no effect on maximum string length or on substring indices; the first character always occurs at position 1. Numeric elements are initialized to zero and string elements are initialized to null (no characters).

If DIM specifies a simple numeric variable that already exists, the variable is reinitialized to zero. Other variables are redimensioned, but not reinitialized (unless the data type—or, if a string variable, the maximum string length—is changed). If DIM expands an array, it also initializes all newly-created elements in the array. Notice that redimensioning does not necessarily preserve an element’s position within an array, but does preserve the sequence of elements within an array. (Refer to “Declaring Arrays (DIM, REAL, SHORT, INTEGER)” in section 3 of your HP-71 Owner’s Manual.)

The following tables indicate the conditions that apply to REAL numeric variables and arrays, and to string variables.

REAL Numeric Variables

Initial Value	0
Numeric Precision	12 Decimal Digits
Exponent Range	± 499
Maximum No. of Array Dimensions	2
Maximum Dimension Limit	65535
Memory Usage in Bytes	
• Simple Variable	9.5
• Array	8*(Dim1 - Base + 1)*(Dim2 - Base + 1) + 9.5

DIM (continued)

String Variables

Initial Value	Null
Default Maximum Length	32 Characters
Possible Maximum Length	65535 Characters
Maximum No. of Array Dimensions	1
Maximum Dimension Limit	65535
Memory Usage in Bytes	
• Simple Variable Memory Usage	Maximum Length + 11.5
• Array	$(dim - base + 1) * (Maximum\ Length + 2) + 9.5$

Related Keywords

DESTROY, INTEGER, OPTION BASE, REAL, SHORT.

DISP

DISP (*display*) displays numeric and string data.

■ Statement

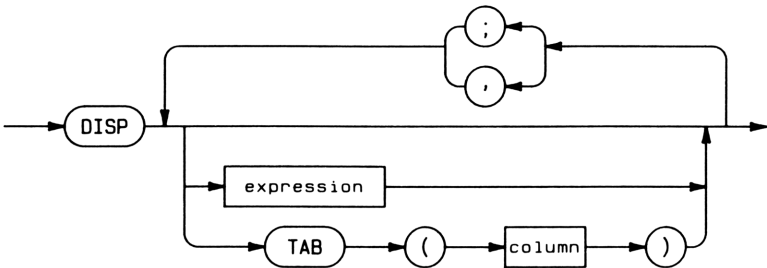
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
DISP I;TAB(I+5);"@"
DISP "Hello";A$;",";"Welcome"
```

```
DISP
A$,"DATE: ",D$
```

Input Parameters

Item	Description	Restrictions
expression	Numeric or string expression.	None.
column	Numeric expression rounded to an integer.	Greater than zero.

Comments

You can omit the statement name itself in all cases except immediately after THEN or ELSE.

The numeric format depends on the currently selected display format. (Refer to the STR\$ keyword entry for details on various formats.) A leading blank is added before positive numbers and a trailing blank is appended to the end of all numbers. These blanks are associated with the displayed number itself, and not with the punctuation on either side of it. No blanks are added to either side of string items.

DISP (continued)

After `DISP` is executed, the display remains unchanged until explicitly changed by another statement, keyboard entry, or program error.

Punctuating DISP Statements. Items to be displayed must be separated by semicolons or commas, which themselves affect the display format. When the HP-71 encounters a semicolon after an item, it displays the next item immediately adjacent to the first item (although numbers always have one or two blanks associated with them, as described above). However, if the remaining line width is not sufficient for the entire item, it will be displayed on the next line.

Commas between items cause the HP-71 to format the items in display “zones.” A display zone is 21 characters wide. (The last zone in a display is less than 21 characters if it exceeds the remaining display width.) The normal display width of 96 characters includes five zones: four 21-character zones and one 12-character zone. When the HP-71 encounters a comma between items in the display list, it skips to the beginning of the next zone (or to the next line if the remaining line width is not sufficient for the entire item). To cause a blank zone of 21 characters, enter two adjacent commas in the display list.

The effect of commas and semicolons can be used to added advantage when these symbols are the last ones in the display list. For example, ending the list with a semicolon causes subsequent characters to be displayed adjacent to the last output. Ending the list with a comma causes subsequent characters to be displayed on the same line, but in the next display zone. Ending the list with neither causes subsequent characters to be displayed on the next line by sending a carriage return/line feed to terminate the line.

The `DISP` display list is equivalent to the ANSI minimal BASIC print list.

Use of TAB. `TAB` positions `DISP` (and `PRINT`) output at the column you specify. If the current column position is beyond the specified `TAB`, the computer first moves to the next line, then positions itself to the specified column. If the column position value exceeds the current line width, the computer reduces the position value by a multiple of the line width (in a manner similar to the `RED` function), then moves to the reduced column position.

Related Keywords

`DISP USING`, `ENG`, `FIX`, `PRINT`, `SCI`, `STD`, `STR$`, `WIDTH`.

DISP USING

DISP USING (*display using*) displays the display list items in a user-specified image format.

■ Statement

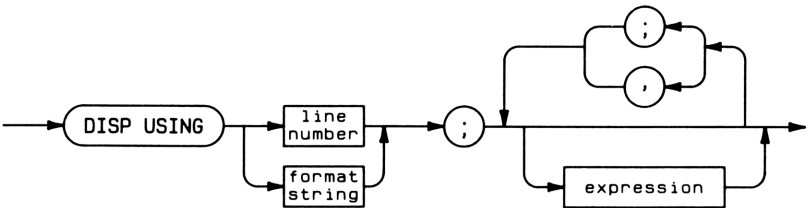
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
DISP USING 200; X, 345.99
DISP USING S$; A$, 1.25+X
DISP USING "#,K,2<XX,3D.D>"; "Output=",Y,2.34E2
DISP USING "2X,'hello',X,8A,'!'; N$
DISP USING '8A,5X,"$"3D.DD'; "Profit=", P
```

Input Parameters

Item	Description	Restrictions
line number	Integer constant identifying a program line.	1 through 9999.
format string	String expression.	Refer to the IMAGE keyword entry.
expression	numeric or string expression.	None.

DISP USING (continued)

Comments

`DISP` may be omitted in all cases except after `THEN` or `ELSE`. The display items in the display list must be separated by commas or semicolons. Notice that, contrary to the use of commas and semicolons with the `DISP` statement, such punctuation has no effect on the spacing between displayed or printed items.

`DISP USING` requires a format string to format the output items. If there are no display items, there may or may not be any output to the display, depending on the items entered in the format string.

If `DISP USING` references a line number, `IMAGE` must be the first statement in that line. When executed from the keyboard, the computer searches for an `IMAGE` statement at the referenced line number in the current file.

If `DISP USING` contains a string expression for the image, that expression must evaluate to a valid format string, as described in the **IMAGE** keyword entry.

Example.

Program segment:

```
10 S$= '2X,' ' & "Today's" & ' ',10A,"$"2D.DD'
20 DISP USING S$; " Special",2.95
```

Program output:

```
Today's Special    $2.95
```

Related Keywords

`DISP`, `IMAGE`, `PRINT USING`, `WIDTH`.

DISP\$

DISP\$ (*display string*) returns a string containing all readable characters in the display.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A$=DISP$                                IF DISP$="" THEN 100 ELSE 200
```

Comments

DISP\$ returns a string of up to 96 characters containing all DISP\$ readable characters in the display. (A “readable” character is any character displayed while the cursor is on, as it is during keyboard entry.) DISP\$ allows a number (VAL(DISP\$)) or string keyed into the display to be used directly by a user-defined key or a subsequently-run program. If there are no readable characters in the display, the null string is returned. In general, characters displayed by a DISP statement are *not* readable.

DIV

`DIV` (*integer quotient division*) returns the integer portion of the quotient of the dividend and the divisor. That is:

$$x \text{ DIV } y = \text{IP}(x/y).$$

☐ Statement

☐ Function

☒ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
N=X DIV 10
```

```
PRINT "Hours =";M DIV 60
```

Input Parameters

Item	Description	Restrictions
dividend divisor	} Numeric expression.	Subject to operator precedence. Division by zero and division of Inf by Inf are not allowed.

Comments

The backslash character (ASCII 92) is an alternative form of the `DIV` operator.

Related Keywords

`/` (division operator), `RMD`.

DROP

DROP removes the coordinates of a data point from the data set represented by the summary statistics in the current statistical array.

■ Statement

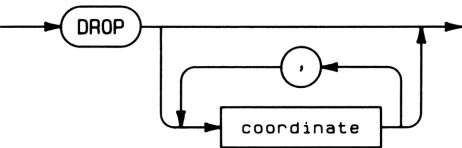
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF . . . THEN . . . ELSE



Examples

DROP

DROP 2,12.3,A,C

Input Parameters

Item	Description	Restrictions
coordinate	Numeric expression giving coordinate (variable value) of data point.	None.

Comments

The number of coordinates must be within the range specified by a preceding STAT statement. Any missing coordinates are assumed to equal zero.

Related Keywords

ADD, CLSTAT, CORR, LR, MEAN, PREDV, SDEV, STAT, TOTAL.

DTH\$

DTH\$ (*decimal-to-hexadecimal string*) converts a decimal number to a five-digit string representing its hexadecimal value, with leading zeroes included.

☐ Statement

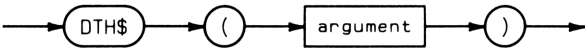
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

A\$=DTH\$(HTD("2F8D7")+64)

DISP DTH\$(16^5-1)

Input Parameters

Item	Description	Restrictions
argument	Numeric expression rounded to an integer.	0 through 1,048,575.

Comments

A typical usage is converting a decimal number to a hexadecimal address in conjunction with the ADDR\$, HTD, and PEEK\$ functions, and the POKE statement.

Related Keywords

ADDR\$, HTD, PEEK\$, POKE.

DVZ

DVZ (*divide-by-zero*) returns the divide-by-zero flag number (−7).

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
IF FLAG<DVZ> THEN STOP           A=TRAP<DVZ,0>
```

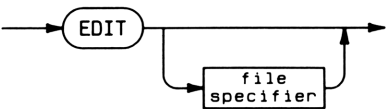
Related Keywords

CFLAG, DEFAULT, FLAG, INX, IVL, OVF, SFLAG, TRAP, UNF.

EDIT

EDIT enables you to enter a new BASIC program or to edit a BASIC program already in memory, and is nonprogrammable.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

EDIT FILE6

EDIT

EDIT A:PORT

EDIT "STD1"

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default: System workfile.	File name with optional device specifier.

Comments

The BASIC file you specify with EDIT becomes the current file. If the file does not exist, EDIT creates it. If you do not specify a file, the workfile becomes the current file. When you execute EDIT, the HP-71 displays the specified file's catalog information.

You can perform LIST, RUN, MERGE, COPY, RENAME, and other file operations on the current file without specifying its file name. Also, the file's program lines may be viewed and edited using the scroll keys.

EDIT *file name*:MAIN initiates a search in main RAM *only* for the specified file. If the file is not found, it is created in main RAM.

EDIT (continued)

`EDIT file name:PORT(n)` initiates a search of port *n* *only* for the specified file. If the file is not found and if the device plugged into that port is an independent RAM, the file is created in that device.

`EDIT file name:PORT` (no particular port specified) initiates a search of all plug-in memory devices for the specified file. If the file is not found, `EDIT` creates the file on the first independent RAM having enough room.

If you do not specify any device, and `EDIT` does not find the file, the HP-71 creates the file in main RAM. As is always the case when no device is specified, the HP-71 searches main RAM first, then searches any plug-in memory devices.

`EDIT` clears any program control information. This includes collapsing all (internally-maintained) execution stacks and releasing any local environments. (In this regard, `EDIT` performs the equivalent of `END ALL`.) Consequently, executing `EDIT` while a program is suspended (**SUSP** annunciator displayed) removes the program from the suspended state and clears the **SUSP** annunciator. Also, because `EDIT` clears the aforementioned stacks, it cannot be used in a `FOR...NEXT` loop.

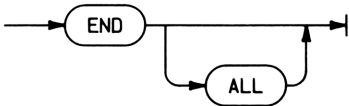
Related Keywords

`CAT`.

END

END terminates a subprogram, user-defined function, or program.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF ... THEN ... ELSE



Examples

```
END                                END ALL
```

Comments

END in a Program. END is the last statement executed in a program. (Program execution can also be terminated with STOP.) When a program has been suspended (refer to the **PAUSE** keyword entry), you can clear the suspended state by executing END from the keyboard.

END in a Subprogram. Executing END in a subprogram is equivalent to executing END SUB. Refer to “END SUB Comments” on the next page.

END in a User-Defined Function. Executing END in a user-defined function is equivalent to executing END DEF. Refer to “END DEF Comments” on the next page.

The END ALL Statement. END ALL releases all levels of local variables and memory associated with a program, any subprogram(s) it called, and any user-defined functions in either the program or the subprogram(s) it calls.

Related Keywords

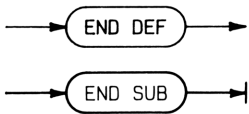
```
END DEF, END SUB, STOP.
```

END DEF/SUB

END DEF (*end function definition*) causes a normal return from a multiple line user-defined function call.

END SUB (*end subprogram*) causes a normal return from a subprogram invoked by a CALL statement.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input type="checkbox"/> IF ... THEN ... ELSE



Examples

```
END DEF          END SUB
```

Comments

END DEF Comments. END DEF indicates the end of a user-defined function, and must be the last statement in a multiple statement user-defined function definition. END DEF terminates function execution and returns control to the expression containing the FN call. The value of the function is set by the most recent execution of FN *variable name* = *expression*.

When a user-defined function is invoked, it not only uses memory to create its local variables, but it also requires memory to retain information about the global environment. *If program execution halts during execution of a user-defined function, the local variables and memory used by that function are not automatically released.* However, you can release this memory by executing END DEF from the keyboard. This terminates the user-defined function and also affects control of execution in one of the following two ways:

- If the user-defined function was invoked by a program, then suspended, executing END DEF from the keyboard suspends program execution at the statement following the statement that called the function.
- If the user-defined function was invoked from the keyboard, then suspended, executing END DEF from the keyboard returns control to the keyboard and does not continue the statement that invoked the function.

When used in a user-defined function, END and DESTROY ALL operate in the same way as END DEF.

END DEF/SUB (continued)

END SUB Comments. Encountering `END SUB` *during subprogram execution* terminates the subprogram, releases the local variables and memory associated with the terminated subprogram, and returns program execution to the statement following the `CALL` that invoked the subprogram. If the subprogram was invoked by a `CALL` statement from the keyboard, control returns to the keyboard. (In this case, if there are one or more keywords concatenated after `CALL`, the HP-71 executes these keywords before returning control to the keyboard.)

When a subprogram is *suspended*, executing `END SUB` from the keyboard terminates the subprogram, releases any local variables and memory associated with the subprogram, and affects control of execution in one of the following two ways:

- If the subprogram was called from another program, execution returns to that program and is suspended at the statement following the `CALL`.
- If the subprogram was called from the keyboard, control returns to the keyboard. This means that if you concatenated one or more keywords after `CALL`, those keywords are *not* executed.

You can use `END` in place of `END SUB` to end a subprogram. Also, `END SUB` operates in the same way as `END` if no local environment exists.

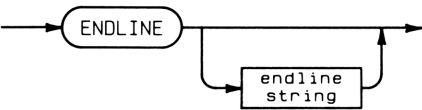
Related Keywords

`DEF FN, SUB.`

ENDLINE

ENDLINE specifies the end-of-line sequence used in PRINT and PLIST statements.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
ENDLINE
ENDLINE ""
ENDLINE CHR$(13)&CHR$(10)&CHR$(10)
```

Input Parameters

Item	Description	Restrictions
endline string	String expression. Default: CR/LF (carriage return/line feed).	Up to three characters.

Comments

The specified string is appended to the output of each PRINT statement whenever either of the following two conditions exist:

- The statement is *not* terminated by a comma or semicolon (, or ;).
- An end-of-line sequence is sent to the printer.

ENDLINE without a string expression restores the normal CR/LF end-of-line.

Related Keywords

PLIST, PRINT, PRINT USING.

ENG

ENG (*engineering format*) sets the engineering display format (ENG mode) and the number of significant digits to be displayed (or printed).

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

ENG 0

IF X > 9999 THEN ENG 11

Input Parameters

Item	Description	Restrictions
number of digits	Numeric expression rounded to an integer. Default: A value less than 0 defaults to 0. A value greater than 11 defaults to 11.	0 through 11.

ENG (continued)

Comments

Display format statements control the format setting for displaying numbers. The display setting remains in effect until you execute another `ENG`, `SCI`, `FIX`, or `STD` statement.

In `ENG` format the displayed value appears as

(Sign) Mantissa E (Sign) Exponent,

where, for normalized numbers,

$1 \leq \text{mantissa} < 1000$,

and the exponent is a multiple of 3. In `ENG` format, the HP-71 displays one significant digit more than the rounded integer value you specify in the `ENG` statement. For example, `ENG 3` produces the following four-digit outputs:

- 1.234E0
- 12.34E0
- 123.4E0

If a displayed value has an exponent of -499 , it is displayed in `SCI` format to the number of digits specified in the `ENG` statement. Denormalized numbers have a mantissa of less than 1.

Related Keywords

`FIX`, `SCI`, `STD`, `STR$`.

EPS

EPS (*epsilon*) returns the smallest positive, normalized number that the HP-71 can represent (1.0 E−499).

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF . . . THEN . . . ELSE



Examples

```
IF X<EPS THEN PRINT X
```

Comments

The value of EPS is the “underflow threshold.”

Related Keywords

```
INF, MAXREAL, MINREAL.
```

ERRL

ERRL (*error line*) returns the line number where the most recent program error or warning occurred.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
DISP "Error on line");ERRL      IF ERRL=200 THEN GOTO
                                ERREXIT
```

Comments

The current ERRL value is affected only by errors and warnings occuring during program execution. An error occurring in a subprogram sets ERRL equal to the appropriate line number in that subprogram, and not the line number of the main (calling) program.

If no error or warning has occured in a running program since the last memory reset, executing ERRL returns 0. If the last error occurred during execution of a nonBASIC program, ERRL also returns 0.

Related Keywords

ERRM\$, ERRN.

ERRM\$

ERRM\$ (*error message string*) returns the message text of the most recent error or warning.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF . . . THEN . . . ELSE



Examples

```
DISP ERRM$           E$=ERRM$
```

Comments

ERRM\$ returns the message text identified by the number of the most recent error (ERRN). If no error or warning has occurred since the last memory reset, ERRN is zero and ERRM\$ returns the null string.

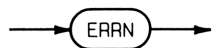
Related Keywords

ERRL, ERRN.

ERRN

ERRN (*error number*) returns the number of the most recent error or warning.

- | | |
|--|--|
| <input type="checkbox"/> Statement | <input checked="" type="checkbox"/> Keyboard Execution |
| <input checked="" type="checkbox"/> Function | <input checked="" type="checkbox"/> CALC Mode |
| <input type="checkbox"/> Operator | <input checked="" type="checkbox"/> IF ... THEN ... ELSE |



Examples

```
DISP "Error number"; ERRN
IF ERRN=255071 THEN DISP "External Error"
```

Comments

Both syntax and run-time errors update the number returned by **ERRN**. If no error or warning has occurred since you initialized your HP-71, **ERRN** returns 0.

Related Keywords

ERRL, ERRM\$.

EXACT

`EXACT` calibrates the system clock and tells the HP-71 that the time currently stored is the correct time.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

`EXACT`

Comments

`EXACT` sets the Exact flag (−46), and directs the HP-71 to compute an internal adjustment factor that is used to continuously update a fast- or slow-running clock. The recommended use of `EXACT` is as follows:

1. After either powering up your HP-71 or performing a level three reset, execute `SETTIME` to set the clock. Then immediately execute `EXACT` to set the Exact flag and begin the first adjustment period.
2. Following an interval of several days, weeks, or months, use `ADJUST` and/or `SETTIME` one or more times to correct the clock. The HP-71 accumulates all corrections except multiples of a half hour (refer to the `ADJUST` and `SETTIME` statements). Ensure that the clock time is correct by comparing it to a reliable time information source. Then execute `EXACT`.

Note: A loss of precision occurs if the corrections made between executions of `EXACT` do not correspond exactly to the true time. However, the longer the interval between executions of `EXACT`, the smaller this precision error will be in proportion to any error resulting from a slow or fast clock. Thus, correcting the clock after several weeks gives better results than correcting the clock after only one day.

After executing `EXACT`, if further corrections are necessary, repeat the process at step 2.

EXACT (continued)

The interval between two `EXACT` commands is termed an *adjustment period*. Executing `EXACT` affects the adjustment period as follows:

- Recomputes the adjustment factor based on:
 - The current adjustment factor.
 - The time corrections accumulated during the current adjustment period.
 - The length of the adjustment period.
- Begins a new adjustment period.

If the absolute value of the computed adjustment factor (af_c) is in the range $0 < af_c < 10$, an `Invalid AF (error 27)` condition occurs.

To access the adjustment factor, refer to the **AF** keyword entry.

Note: The internal clock uses the *Exact* flag. The *Inexact* flag is a math flag and has no relationship to the *Exact* flag.

Related Keywords

`ADJUST`, `AF`, `SETTIME`.

EXOR

EXOR (*exclusive or*) generates the logical Exclusive Or of its operands.

☐ Statement

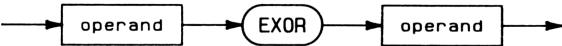
☐ Function

☒ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
IF Z0 EXOR Y0 THEN GOSUB 100      Q=R EXOR S
```

Input Parameters

Item	Description	Restrictions
operand	Numeric expression.	Subject to operator precedence.

Comments

The operands of EXOR are considered to be logically false if zero and logically true if nonzero. The possible results of this operation are summarized in the table to the right.

EXOR has the same precedence as OR, which is the lowest of all operators.

Operand		Result
Left	Right	
False	False	0
False	True	1
True	False	1
True	True	0

The precedence of EXOR in relation to the HP-71’s other operators is described under “Precedence of Operators” on page 317.

Related Keywords

AND, NOT, OR.

EXP

EXP (*natural antilogarithm*) returns the number $e = 2.718281828...$ raised to the power given by the argument.

☐ Statement

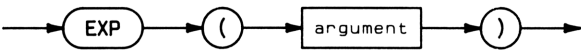
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

Y=EXP(-X*X/2)

PRINT "e to the z =",EXP(Z)

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Related Keywords

EXPM1, LOG, LOGP1.

EXPM1

EXPM1 (*natural antilogarithm minus 1*) returns the value of $e^x - 1$.

☐ Statement

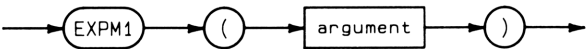
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

X=EXPM1(1.2345E-10)

PRINT "e^x-1 =" ;EXPM1(X)

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

This function allows for an accurate evaluation of the quantity $e^x - 1$, which is useful for values of x that are close to zero.

Related Keywords

EXP, LOG, LOGP1.

EXPONENT

EXPONENT returns the exponent of its normalized argument.

☐ Statement

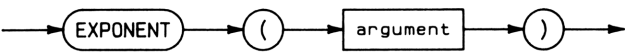
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
E=MIN(EXPONENT(X(I)),M)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

EXPONENT enables you to perform extended range computations that might otherwise produce an underflow or overflow. For a finite nonzero argument, the result is an integer in the range -510 through 499. EXPONENT(0) or EXPONENT(-0) returns -INF and sets the DVZ (division-by-zero) exception flag.

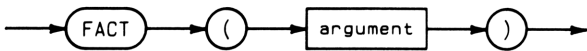
Related Keywords

```
LOG10.
```

FACT

FACT (*factorial*) returns the factorial of a nonnegative integer argument.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A = FACT(253)
PRINT "Number of permutations = ";FACT(N),
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	Nonnegative integer less than or equal to 253, or equal to Inf.

Comments

FACT returns a value of type REAL.

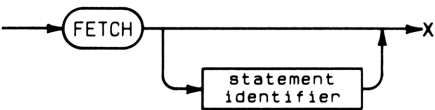
The factorial of a positive integer is the product of all positive integers less than or equal to that integer. The factorial of 0 is defined as 1. The factorial function overflows for finite arguments greater than 253. However,

```
FACT(Inf) = Inf.
```

FETCH

FETCH displays any line in the current program file for editing, and is nonprogrammable.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
FETCH 55           FETCH COMPUTE           FETCH
FETCH "SECTION2"
```

Input Parameters

Item	Description	Restrictions
statement identifier	Line number or label of a program statement.	Any valid line number or label reference.



Comments

Fetching a line sets the file pointer to that line and allows you to edit the line.

If a line number or label reference is not specified, the current line is displayed. If the line number is not found, the HP-71 displays that line number followed by the cursor. This allows you to create a new program line. In a suspended program (**SUSP** annunciator displayed), executing **FETCH** without an argument displays the line containing the next statement to be executed.

If a specified statement label is not found an error occurs.

Related Keywords

FETCH KEY, LIST. See also the descriptions of the  and  keys in the *HP-71 Owner's Manual*.

FETCH KEY

FETCH KEY displays a specified key assignment for editing, and is nonprogrammable.

☒ Statement

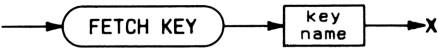
☐ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

FETCH KEY "A" FETCH KEY"#123" FETCH KEY "fA"

Input Parameters

Item	Description	Restrictions
key name	String expression.	Less than five characters.

Comments

FETCH KEY retrieves the character string assigned to the specified key. The string is displayed in one of the following formats:

- Direct execution: DEF KEY *key name* , *assigned string* ;
- Typing aid: DEF KEY *key name* , *assigned string* ;
- Immediate execution: DEF KEY *key name* , *assigned string*

If no string has been explicitly assigned to the key, the computer displays DEF KEY *key name*. (For further information concerning these formats, refer to the DEF KEY keyword entry.)

The VIEW key displays the string assigned to the next key you press. The string remains in the display as long as you hold the key down. When you release the key, the display returns to its original state. If the key is unassigned, the HP-71 displays UNASSIGNED.

Related Keywords

DEF KEY, LIST KEYS, VIEW .

FIX

`FIX` (*fixed format*) sets both the fixed display format and the number of fractional digits to be displayed (or printed).

■ Statement

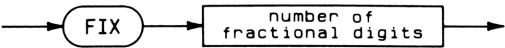
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
FIX 0                                IF X<1 THEN FIX 11
```

Input Parameters

Item	Description	Restrictions
number of fractional digits	Numeric expression rounded to an integer. Default: If the value is less than zero, <code>FIX</code> uses zero; if the value is greater than eleven, <code>FIX</code> uses 11.	0 through 11.

Comments

Display format statements control the format for displaying numbers. The display setting remains in effect until you execute another `FIX`, `ENG`, `SCI`, or `STD` statement.

In `FIX` display setting the displayed or printed value appears as:

(Sign) Mantissa.

The mantissa appears rounded to *d* places to the right of the decimal, where *d* is the specified number of digits. While the fixed display format is active, the HP-71 automatically displays a value in `SCI` format, rounded to *d* places past the decimal, in either of the following two cases:

- If the number of digits to be displayed exceeds 12.
- If a nonzero value rounded to *d* places past the decimal point would be displayed in fixed display format as zero.

Related Keywords

ENG, SCI, STD, STR\$.

FLAG

FLAG returns the current value (0 or 1) of the specified flag, and optionally sets or clears the flag.

☐ Statement

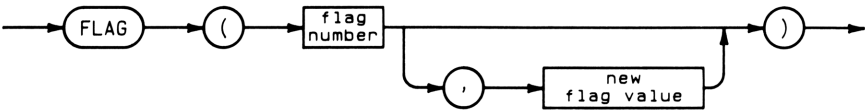
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A=FLAG(I,1)
A=FLAG(I,0)
J=FLAG(OVF,0)
```

```
J=FLAG(OVF,J)
IF FLAG(IVL) THEN GOTO IVLOP
```

Input Parameters

Item	Description	Restrictions
flag number	Numeric expression rounded to an integer.	If you do not enter a new flag value, the range is −64 through 63. If you enter a new flag value, the range is −32 through 63.
new flag value	Numeric expression.	
		None.

Comments

If you do not specify a new flag value, the flag value remains unchanged. Otherwise, the HP-71 sets the specified flag to 0 or 1, according to whether the new flag value is zero or nonzero.

You can use FLAG to set, clear, test, save, and restore any user or system flags numbered greater than −33.

Related Keywords

CFLAG, DVZ, INX, IVL, OVF, SFLAG, UNF.

FLOOR

FLOOR returns the greatest integer less than or equal to the argument.

☐ Statement

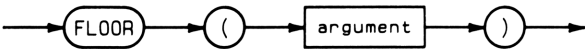
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A1=FLOOR(X1)
IF X/2 = FLOOR(X/2) THEN GOTO EVEN
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

If the value of the numeric expression is an integer, that value is returned. If the value of the expression is not an integer, FLOOR returns the greatest integer value less than or equal to the expression. For example:

- FLOOR (1.5) returns 1.
- FLOOR(-1) returns -1.
- FLOOR(-1.5) returns -2.

FLOOR and INT are identical functions.

Related Keywords

CEIL, FP, INT, IP.

FN

FN (*function*) transfers program execution to the specified user-defined function and may pass parameters to that function.

☐ Statement

☒ Function

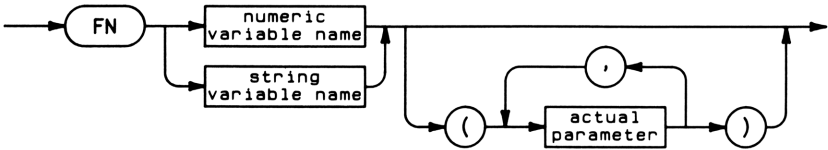
☐ Operator

☒ Keyboard Execution

☒ CALC Mode*

☐ IF . . . THEN . . . ELSE

*Multiple-line user-defined functions cannot be used in CALC mode.



Examples

X = Y + FNF(A,B*15)

PRINT A\$; FND\$(X\$,Y\$)

FNB = X + Y

FNH\$ = K\$C10,20]

Input Parameters

Item	Description	Restrictions
variable name	Numeric or string variable name.	None.
actual parameter	Numeric or string expression.	None.

Comments

When invoked in an expression, a function is evaluated and its value is returned to that expression. In a user-defined function definition, FN is used in the left-hand side of an assignment statement to assign the value to be returned from the function to the expression that invoked the function. A user-defined function can be used in either a running program or a statement executed from the keyboard. (Use the DEF FN statement to create user-defined functions.)

The actual parameters must be of the same type (numeric or string) as the corresponding parameters (formal parameters) in the DEF FN statement. All actual parameters are evaluated, then passed as values.

FN (continued)

The referenced function *must* be in the current program scope. If you execute the user-defined function from the keyboard, the computer searches only the main program or subprogram in the current file, depending upon which environment is currently active. If the currently active environment program contains more than one user-defined function with the same name, **FN** uses the one on the lowest-numbered line.

For further information concerning environments, refer to “Scope of Environments” on page 314 and to your *HP-71 Owner’s Manual*.

Note: It is possible for a subprogram to contain one or more user-defined functions having the same name as a function in the main program. In this case, if you (1) suspend program execution while in the subprogram and (2) execute the user-defined function from the keyboard, **FN** uses the function on the lowest-numbered line *in the subprogram*.

Related Keywords

DEF FN, END DEF.

FOR...NEXT

FOR is used with NEXT to define a loop that is repeated until the loop counter exceeds the specified value.

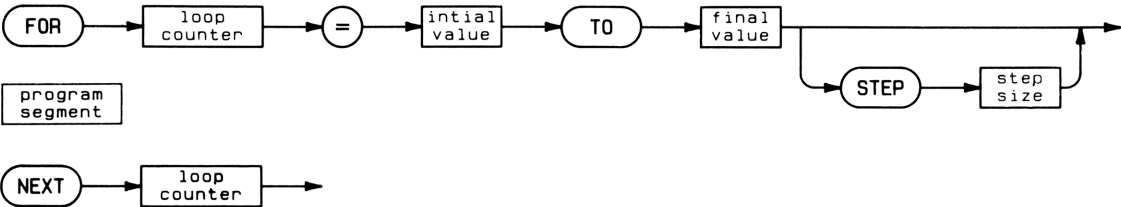
- Statement

☐ Function

☐ Operator
- Keyboard Execution

☐ CALC Mode

☐ IF ... THEN ... ELSE



Examples

```
FOR K = 1 TO 16 @ DISP 16^K @ NEXT K
FOR I = 1 TO 10 @ DISP CAT$(I) @ NEXT I
FOR A1 = 50 TO -100 STEP -.01
```

Input Parameters

Item	Description	Restrictions
loop counter	Simple numeric variable.	None.
initial value	Numeric expression.	None.
final value	Numeric expression.	None.
step size	Numeric expression. Default: 1.	None.
program segment	Any number of contiguous program lines.	None.

FOR...NEXT (continued)

Comments

General Operation. The loop counter is set to its initial value when program execution encounters the `FOR` statement. Each time execution encounters the corresponding `NEXT` statement, the step size is added to the loop counter, and the resulting new loop counter value is tested against the final value. If the new loop counter value does not exceed the final value in the direction indicated by the step size, the computer executes the loop again, beginning with the statement immediately following the `FOR` statement. If the new loop counter value exceeds the final value in the direction indicated by the step size, the computer exits from the loop and continues program execution with the statement following the `NEXT` statement. Refer to the loop test that is described under “Operating Details,” below. Notice that the step size can be positive, negative, or zero (which results in an infinite loop). Whenever you do not specify a step size, the HP-71 uses a default step size value of 1.

Operating Details. The HP-71 uses the following loop test to determine whether to execute the program segment between the `FOR` and `NEXT` statements:

If $(\text{loop counter} - \text{final value}) * \text{SGN}(\text{step size}) \leq 0$ then execute the program segment; else transfer execution to the statement following the corresponding `NEXT` statement.

When the loop is initialized, the loop counter is assigned its *initial value*, and the preceding loop test is then performed to determine whether the `FOR ... NEXT` program segment should be executed or bypassed.

The initial, final, and step size values for a loop are determined (initialized) when program execution enters the loop. If you use a variable or expression for any of these values, you can change that variable or expression value *after* the computer initializes the loop without affecting how many times the loop is repeated. However, changing the value of the loop counter after the computer initializes the loop affects the number of times that the loop will be repeated.

The loop counter variable is allowed in expressions that determine the initial, final, or step size values. However, the value of the loop counter variable is established before the HP-71 computes the final and step size values.

If you use `GOTO` to exit from a loop before the exit condition is satisfied, the loop counter contains the value it had at that time.

FOR-NEXT loops can be nested.

When the loop control values involve NaN or Inf there are exceptions to the loop test. For example, if the initial value, final value, or step size has the value of NaN upon loop entry, then an `Unordered` (error 20) condition occurs. If the IVL trap value is “2,” then the loop counter is set to NaN and the program segment between `FOR` and `NEXT` is executed at least once.

FOR...NEXT (continued)

Related Keywords

NEXT.

FP (*fractional part*) returns the fractional part of a numeric value.

☐ Statement

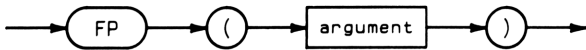
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A1 = FP(X1)
PRINT "Fractional part = ";FP(Y)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

The returned fraction has the same sign as the argument. For example:

- FP(1.5) returns .5.
- FP(-1) returns -0.
- FP(-1.5) returns -.5.

Related Keywords

CEIL, FLOOR, INT, IP.

FREE PORT

FREE PORT switches the RAM in a port from main RAM to independent RAM status. FREE PORT is nonprogrammable.

■ Statement

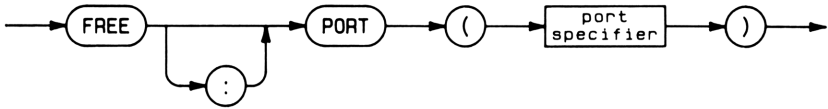
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

FREE PORT(1.01)

FREE PORT (A)

Input Parameters

Item	Description	Restrictions
port specifier	Numeric expression truncated to two digits after the decimal point. Interpreted as <i>p.dd</i> , where: <i>p</i> = port number. <i>dd</i> = device number.	$0 \leq p \leq 5$. $0 \leq dd \leq 15$.

Comments

Operation. FREE PORT separates a RAM from system memory and prepares the RAM to receive and maintain files independently of the computer's main RAM. If there is not enough memory available to remove the RAM, the computer displays an `Insufficient Memory` (error 24) message. Following execution of FREE PORT, you can remove the RAM from the HP-71 without disrupting the computer's memory configuration. (Removing a RAM from the computer without first separating it from system memory by using FREE PORT can cause a `Memory Lost` condition).

Note: Executing FREE PORT changes the system configuration. That is, when you execute FREE PORT, all file pointers are reset, the workfile becomes the current file, and all FOR ... NEXT loops are terminated.

FREE PORT (continued)

Port Information. Port 0 contains the HP-IL port and four internal memory modules. Ports 1 through 4 are the four ports in the front of the machine, numbered from left to right. Port 5 is the card reader slot.

The device number (*dd*) is the position of a plug-in device in a device chain.

Related Keywords

CLAIM, SHOW.

GDISP

GDISP (*graphic display*) sets the dot pattern in the display according to a specified string.

- Statement

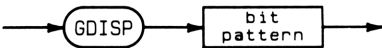
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
GDISP A#           GDISP GDISP#[C2]
```

Input Parameters

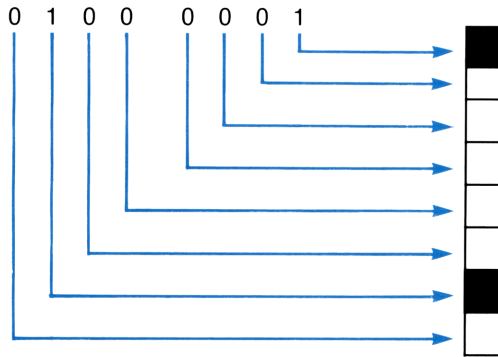
Item	Description	Restrictions
bit pattern	String expression truncated to 132 characters or extended with nulls to 132 characters. (The display is composed of 132 columns containing 8 dots per column.)	None.

Comments

GDISP sets the HP-71’s liquid crystal display bit pattern (but does not affect the display buffer contents).

The LCD window consists of a field of 132 dot columns. Each column contains eight dots. The HP-71 displays information by turning on various dots. Any character in a GDISP string has a character code that can be represented as an eight-bit binary number. (Refer to “HP-71 Character Set and Character Codes” on page 322.) Each character in a GDISP string corresponds to one column of dots in the display; each bit in the character’s binary representation corresponds to one dot in that column. The least significant bit of a binary character code defines the top dot of that character’s display column; the most significant bit defines the bottom dot of the display column. For example, GDISP uses the bit pattern for the character “A” (binary code: 0100 0001) to affect a display dot column as shown on the facing page.

GDISP (continued)



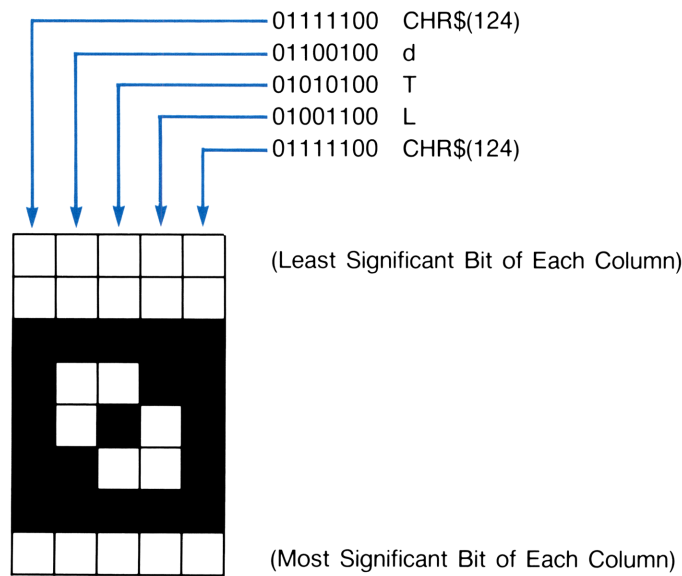
A display of this column would show the first and seventh dots.

Notice that the 1's in the binary code correspond to the displayed dots and the 0's correspond to the undisplayed dots.

A `GDISP` string character can be either a character you type in from the keyboard or a character specified by `CHR#`.

The `GDISP` string is a one-dimensional array of bytes used to determine the dots to turn on for a graphic pattern you want to place in the display. The character code for the first character defines the first dot column in the display, the second character defines the second column, and so forth. The least significant bit of each character defines the top dot of that display column and the most significant bit defines the bottom dot of the column. For example, the following illustration shows the result of executing `GDISP CHR$(124) & "dTL" & CHR$(124):`

GDISP (continued)



GDISP redefines all dot columns in the display, regardless of the WINDOW setting. However, by using GDISP and WINDOW prior to executing DISP, you can “lock” a bit pattern in a subsection of the display. (That is, you can protect a GDISP dot pattern in a designated portion of the display from being cleared by subsequent executions of DISP.

Where a GDISP-created symbol is not protected by a WINDOW setting, the symbol remains in the display until either some operation sends a character to the display or you press a key while the computer is waiting for an input.

Related Keywords

DISP, GDISP\$, WINDOW.

GDISP\$

GDISP\$ (*graphic display string*) returns a 132-character string reflecting the bit pattern in the LCD display:

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A$=GDISP$          GDISP GDISP$[2]
```

Comments

The first character of a GDISP\$ string corresponds to the first dot column of the display, and the 132nd character of this string corresponds to the last dot column of the display. For any GDISP character, the bits in that character’s binary code represent the dots in the corresponding display column. (Refer to “Comments” in the **GDISP** keyword entry.)

Executing GDISP\$ displays the current GDISP\$ string (leading and imbedded nulls are ignored). Executing GDISP GDISP\$ displays the dot pattern specified by the current GDISP\$ string. In the following program example, GDISP\$ is used in a loop to obtain the character string from the current display for use in the next display.

```
10 REM PISTON PROGRAM
20 GDISP CHR$(255) & CHR$(255)

30 FOR I = 1 TO 60
40 GDISP CHR$(24) & GDISP$
```

Displays a solid vertical bar that is two columns wide.

Begins countup loop.

Appends column corresponding to CHR\$(24) to current display, which, on the first pass through the loop, is composed only of characters resulting from execution of line 20; on subsequent passes through the loop, is composed of all characters previously specified in both line 20 and in the loop. Effects are to build on CHR\$(255) image and scroll resulting image to the right.

GDISP\$ (continued)

```
50 NEXT I
60 FOR I = 60 TO 1 STEP -1
70 GDISP GDISP$[2]
```

```
80 NEXT I
90 GOTO 20
```

Terminates countup loop.

Begins countdown loop.

Shortens by one column the displayed image created in the first loop, and scrolls image one column to the left.

Ends countdown loop.

Repeats program.

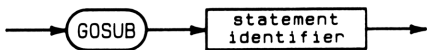
Related Keywords

GDISP.

GOSUB

GOSUB (*go to subroutine*) transfers program execution to the subroutine beginning at the specified statement. (The specified statement must be in the same program or subprogram as the **GOSUB**.) The **RETURN** associated with a **GOSUB** statement returns the program exeuction to the statement immediately following that **GOSUB**.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```

GOSUB 100
GOSUB COMPUTE6
GOSUB "PLOT10"
GOSUB G$
  
```

Input Parameters

Item	Description	Restrictions
statement identifier	Line number or label of a program statement.	Any valid line number or label reference.

Comments

If you execute **GOSUB** from the keyboard with another keyword concatenated after **GOSUB**, the computer

1. Executes the referenced program segment.
2. Executes the statement concatenated after **GOSUB** (only if the called program segment ends with **RETURN**).
3. Returns the computer to keyboard control.

GOSUB (continued)

However, if GOSUB execution is suspended and then resumed, upon encountering a RETURN the computer halts without returning to the statement concatenated after GOSUB.

All variables in a program are accessible to any subroutine within that program. Also, changing the value of a variable during execution of a subroutine within that program changes the value of the same variable in the main program.

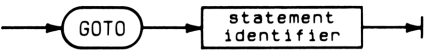
Related Keywords

GOTO, RETURN.

GOTO

When executed in a program, `GOTO` (*go to statement*) transfers program execution to the specified statement. When executed from the keyboard, `GOTO` positions the HP-71 to the specified statement.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
GOTO 100          GOTO PRINTVAR          GOTO "GRAPHX"
GOTO C$
```

Input Parameters

Item	Description	Restrictions
statement identifier	Line number or label of a program statement.	Any valid line number or label reference.

Comments

The specified line number or label must be within the current program scope.

`GOTO` is frequently used as follows:

- In a running program to perform unconditional branching.
- From the keyboard to:
 - Commence execution from a particular statement using `[SST]` or `[CONT]` (when the program is suspended).
 - Position the computer to a particular line in order to edit that line. (To view and/or edit a line within the current file, but outside of the current program scope, use `FETCH`).

(Executing `GOTO` from the keyboard displays the **SUSP** annunciator and sets the suspend statement to either the first statement in the specified line or the first statement following the specified label. Also, the line containing the suspend statement becomes the current line.)

GOTO (continued)

Related Keywords

CONT, FETCH, GOSUB. Refer also to the description of the SST key in your *HP-71 Owner's Manual*.

HTD (*hexadecimal string to decimal*) converts a string argument representing a hexadecimal number to a decimal number.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
DISP HTD("2F8D7")           A= HTD("99F1")+258
```

Input Parameters

Item	Description	Restrictions
hexadecimal value	String expression containing hexadecimal digits.	Up to five uppercase or lowercase digits.

Comments

A typical usage of HTD is to convert a hexadecimal address in conjunction with DTH\$, ADDR\$, PEEK\$, or POKE.

HTD is a numeric function that accepts a string expression as input. The string expression must represent a hexadecimal integer value in the range 0 through FFFFF. Any string expression having more than five characters or containing a character that is not a hexadecimal digit causes an error condition. A null string also causes an error condition.

Leading zeroes in the string expression will be accepted, although they are not necessary.

Related Keywords

```
ADDR$, DTH$, PEEK$, POKE.
```

IF... THEN... ELSE

IF... THEN... ELSE provides conditional execution.

■ Statement

□ Function

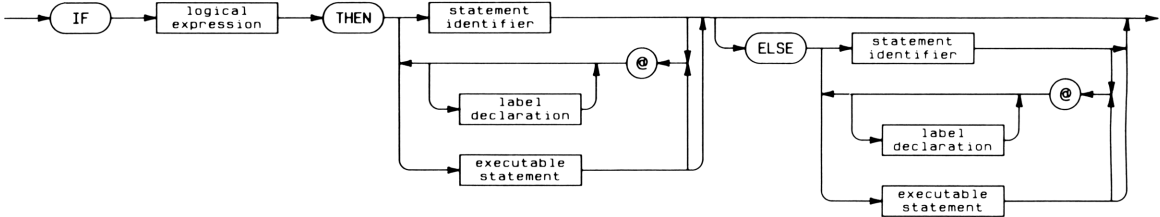
□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE*

*Following ELSE (only). Refer to "Further Restrictions" on the next page.



Examples

```
A$=CAT$(I) @ IF LEN(CAT$(I)) THEN DISP A$(1,8)
IF A=B THEN LABEL5 ELSE GOSUB 100 @ IF A>B THEN STOP
IF B THEN DISP B @ STOP ELSE 250
```

Input Parameters

Item	Description	Restrictions
logical expression	Numeric expression evaluated as true if non-zero and false if zero.	None.
statement identifier	Line number or label of a program statement.	Any valid line number or label reference.
executable statement	Executable BASIC statement.	Any statement legal in an IF construct. (Refer to "Further Restrictions," on the next page.)
label declaration	Quoted or unquoted string followed by a !.	Any valid label.

IF...THEN...ELSE (continued)

Comments

General Operation. If the logical expression evaluates to a nonzero value, it is considered true, and program control is transferred to the statement immediately following `THEN`. Execution of statements following `THEN` continues until the computer encounters either `ELSE` or the end of the line (unless a `GOTO` unconditionally branches execution to another statement). If the logical expression evaluates to zero, it is considered false, and execution is either transferred to the statement immediately following `ELSE` or, if `ELSE` is not used, to the next program line.

Note: A logical expression can be constructed with numeric or string expressions separated by relational operators, as well as with a numeric expression.

A line number or label reference immediately following `THEN` or `ELSE` is considered to be an “implied `GOTO`,” and execution is transferred to the specified statement.

Further Restrictions. Most BASIC statements are legal following `THEN` or `ELSE`. However, `FOR`, `NEXT`, `DATA`, `DEF FN`, and, generally, any nonexecutable statements are illegal after `THEN` or `ELSE`. Because `IF...THEN` constructs cannot be nested, `IF...THEN` cannot follow `THEN`, but can follow `ELSE`.

An implied `DISP` is not legal immediately following `THEN` or `ELSE`, but is otherwise legal in an `IF` construct.

IMAGE

IMAGE is used in conjunction with DISP USING and PRINT USING to control the format of displayed and printed output.

■ Statement

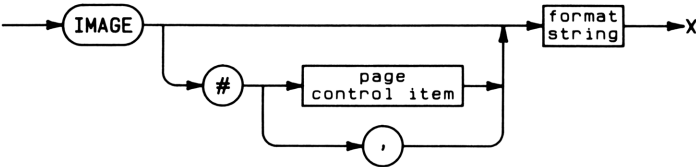
□ Function

□ Operator

□ Keyboard Execution

□ CALC Mode

□ IF ... THEN ... ELSE



Examples

```
IMAGE M3D.2D,3X,5A
IMAGE "Third Quarter",3(5X,'$'M3ZC3Z.2D)
IMAGE #,K
IMAGE 2(K),XA3","A/
```

Input Parameters

Item	Description	Restrictions
format string	Refer to "Using IMAGE Symbols to Control Output" on the next page.	None.
page control item	Refer to "Page control symbols" on page 138.	1 through 9999.

Comments

IMAGE is the last keyword recognized in a program line. Since the @ symbol is a valid IMAGE symbol, there can be no statements concatenated after IMAGE. Also, an IMAGE statement cannot contain a trailing remark, as inclusion of ! generates an error.

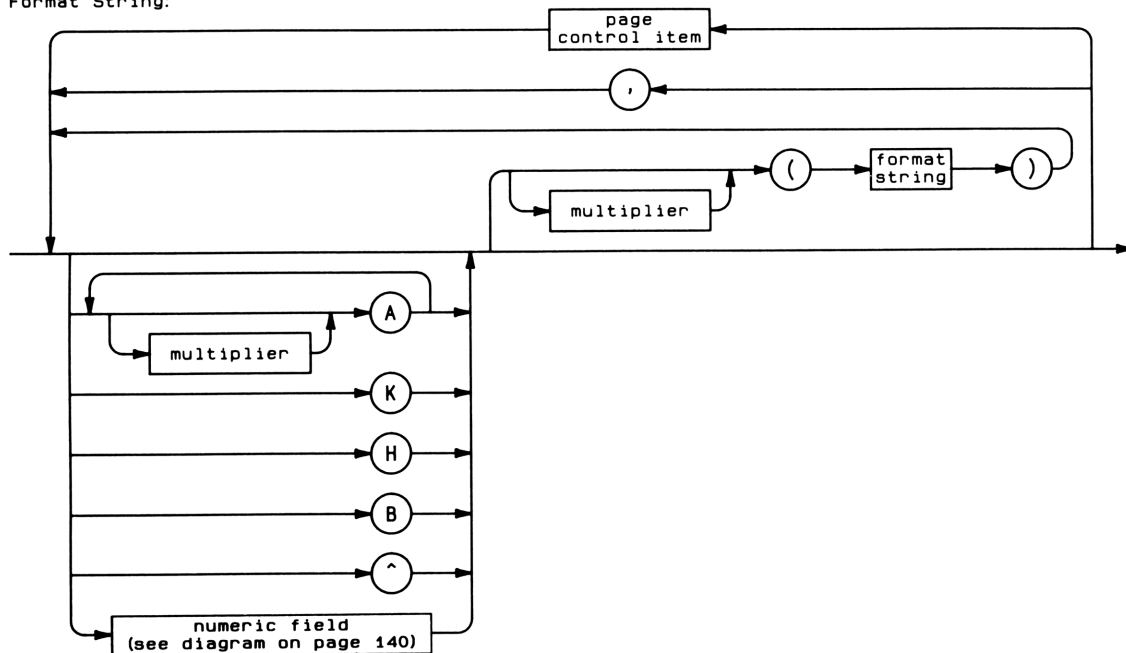
IMAGE (continued)

To be executed properly by `DISP USING` or `PRINT USING`, `IMAGE` must be the first statement in a program line. Although you can enter a program line having `IMAGE` as the last statement in a multiple-statement line, program execution does not access an `IMAGE` statement in this position, and causes an error.

You can place a program line containing `IMAGE` anywhere in a program, as location with respect to `DISP USING` or `PRINT USING` is not significant. During program execution, `IMAGE` statements are ignored in the same way as `REM` statements.

Using IMAGE Symbols to Control Output.

Format String:



The use of field specifiers in `DISP USING` or `PRINT USING` statements is shown on the next page. Also, in the following discussion, multipliers are represented by n .

IMAGE (continued)

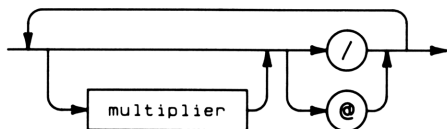
Several `IMAGE` symbols can be used with a multiplier, as shown in the syntax diagrams. (A multiplier is a numeric constant in the range 1 through 9999.)

- Carriage control symbol: Although this symbol is the first encountered in the format string, it is not acted upon until all other output has been processed.

`#` Suppresses the automatic output of the end-of-line sequence at the end of the output list.

- Page control symbols:

Page control item:



Note: Editing symbols may precede or follow any `IMAGE` character (except `#`).

`n /` `DISP USING` sends a carriage-return, line-feed to the display. `PRINT USING` sends an end-of-line sequence to the printer device.

`n @` Sends a form-feed to the output device.

- Grouping symbols:

`n ()` Parentheses are used to delimit field specifiers and to group several fields for replicated output. There is no limit, except for the amount of memory available, to the number of levels of nested parentheses.

IMAGE (continued)

- Special output symbols:

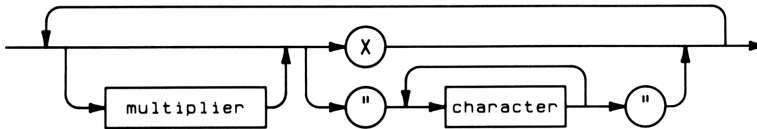
- | | |
|---|--|
| K | Compact field. Displays/prints a number or string in current display format, with no leading or trailing blanks. |
| H | Same as K, except the European radix (,) is substituted for a decimal point in numeric output. |
| B | Displays/prints a single character. The number to be output is rounded to an integer and the least significant eight bits of the number are sent. That is, a modulus (256) is performed. The B symbol is equivalent to CHR#. |
| ^ | Supression field. Causes the computer to evaluate the corresponding variable without displaying or printing the result. For example, the statement |

```
DISP USING "K,K,^,K";1,2,3,4
```

displays

```
124.
```

- Editing Symbols:



Note: Editing symbols may precede or follow any IMAGE character (except #).

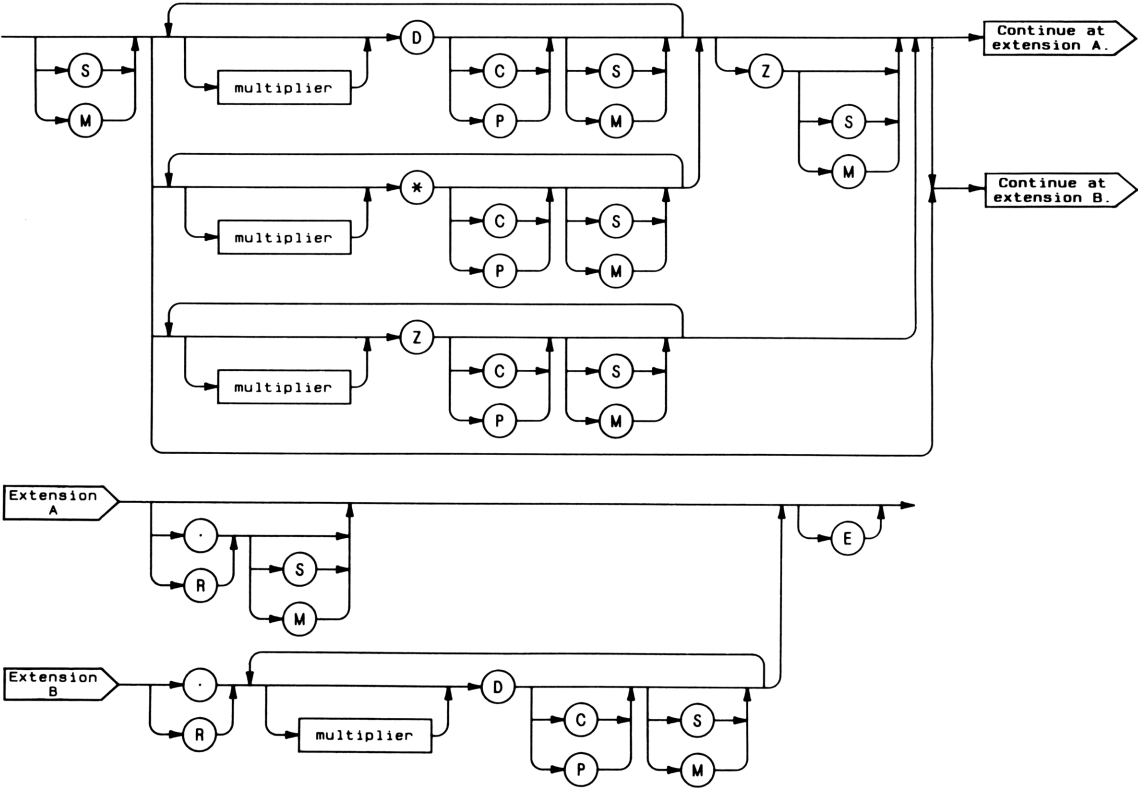
- | | |
|----------------------|--|
| $n \times$ | Displays/prints a blank. |
| $n \text{ "chars."}$ | Displays/prints the characters contained in the quotes. Any accompanying multiplier must precede the leading quote(s). |

- String Field Symbol:

- | | |
|---------------|--|
| $n \text{ A}$ | Displays/prints a string character. Generates trailing blanks if the specified number of characters is greater than the number available in the corresponding string. If the field specifier becomes exhausted while the corresponding string still contains characters, those characters are ignored. |
|---------------|--|

IMAGE (continued)

- Numeric Field Symbols:



Note: Editing symbols may precede or follow any IMAGE character (except #).

IMAGE (continued)

- Digit specifiers:

- $n \text{ D}$ Displays/prints one digit. Replaces a leading zero with a blank. If the number is negative, but no sign symbol is specified, the minus sign occupies a leading digit position. If a sign is displayed, it “floats” to the left of the leftmost digit. Any editing symbols preceding the first D specifier are also “floated” to the left of the leftmost digit (or sign, if appropriate).
- $n \text{ *}$ Same as for D , except leading zeroes are replaced with “*”. Since blanks are not filled, no symbols float to the leftmost digit.
- $n \text{ Z}$ Same as for $*$, except leading zeroes are displayed.

- Sign specifiers:

- S Displays/prints the number’s sign (+ or –).
 - M Displays/prints the number’s sign if negative, and a blank if positive.
- No more than one S or M is allowed per numeric field.

- Radix specifiers:

- . Displays/prints a decimal point radix.
- R Displays/prints a comma for the European radix.

- Digit separators:

- C Displays/prints a comma as a digit separator.
- P Displays/prints a decimal point for the European digit separator.

- Exponent specifier:

- E Displays/prints E with a sign and a three-digit exponent. (Leading zeroes in the exponent are displayed.)

Field Specifiers. `IMAGE` items are grouped into logical field specifiers, with each output item formatted by a new field. There are two types of field specifiers:

- Output Field: Consists of numeric, string or special specifiers, all requiring items from the output list.
- Editing Symbol Field: Consists of blanks or quoted characters that do not require an output item. A special case of this type is the null field, which consists of either a pair of adjacent commas or a pair of parentheses that do not enclose any characters.

IMAGE (continued)

Each field terminates when it encounters a delimiter. Delimiters indicate that formatting is completed for an output item, whether or not the entire item was used. The chart to the right illustrates the five field delimiters.

Field Delimiters	
Delimiter*	Usage
,	The usual way to separate two field specifiers.
<i>n</i> /	Page control character (output EOL sequence).
<i>n</i> @	Page control character (output form feed).
<i>n</i> (Left parenthesis.
)	Right parenthesis.
*The <i>n</i> indicates that the symbol can be preceded by a multiplier.	

When the HP-71 executes `DISP USING` or `PRINT USING`, it sequentially accesses the items in the output list; one for each output field in the `IMAGE` list. If the computer encounters a field that does not require output items (editing symbol field), the field is acted upon without accessing the output list.

The processing of field specifiers stops when the computer encounters an output field having no matching output item. If the output fields are exhausted while output items remain, the fields are reused, beginning with the first field.

IMAGE Overflow. An `IMAGE` overflow results when a numeric item requires more digit places to the left of the decimal point than are provided by the field specifier. This occurrence is reported in the same manner as math overflows; that is, as an error or as a warning, depending upon the value of the `OVF` trap. With either `DEFAULT OFF` or `TRAP(OVF)=0`, the `IMAGE` overflow is reported as an error, thus halting execution. With `DEFAULT ON` or `DEFAULT EXTEND`, or with `TRAP(OVF)` set to 1 or 2, the `IMAGE` overflow is reported as a warning. After the warning is displayed, the numeric field is filled with `*` symbols and execution continues.

A minus sign takes a digit place if you do not specify `M` or `S`, and can generate unexpected overflows of the field specifier. If the number contains more digits to the right of the decimal point than the field specifier, it is rounded to fit the field specifier.

String Field Output. If a string is longer than the field specifier, it is truncated, and the rightmost characters are not sent out. If the string is shorter than the specified field, trailing blanks are used to fill out the field.

IMAGE (continued)

IMAGE Syntax Details. The HP-71 checks the syntax of an `IMAGE` statement only when `DISP USING` or `PRINT USING` execute that `IMAGE` statement. Only those parts necessary to exhaust the output list are checked, since not all of the field specifiers may be used if there are fewer items in the output list. Thus, a syntax error near the end of the format string is not reported if there is no output item to access it. In addition, since the computer processes field specifiers sequentially, several items may be generated before a syntax error is reported.

Typing for Readability. You can type `IMAGE` symbols in upper- or lowercase characters. A space imbedded anywhere in an `IMAGE` statement is ignored, except where the space is imbedded in a quoted series of characters.

Carriage Control Symbol. Only one occurrence of the `#` carriage control symbol is allowed in any format string, and only as the first nonblank character in that list. The carriage control symbol must be followed by a delimiter.

Parentheses. Although parentheses allow repeated output with multipliers, they always function as delimiters for a new field. When building the format string, you should include parentheses only when needed for field delimiters. For example, in each of the following program segments, the `DISP USING` operations are the same, except for the parentheses appearing in one of the two `IMAGE` statements:

- Program segment:

```
10 DISP USING 100; "LUNCH","FRIDAY"
20 DISP USING 200; "LUNCH","FRIDAY"
100 IMAGE AA3"."AA , 2X
200 IMAGE AA3(".)AA , 2X
```

Program output:

```
LU...NC FR...ID
LU...FR
```

- Program segment:

```
10 DISP USING 100; 5280, 1760
20 DISP USING 200; 5280, 1760
100 IMAGE 3ZCZZZ, 2X
200 IMAGE 3(ZCZZZ), 2X
```

IMAGE (continued)

Program output:

```
005,280 001,760
5,2801,760
```

- Program segment:

```
10 DISP USING 100; 1.25,6.50
20 DISP USING 200; 1.25,6.50
100 IMAGE "$"DDD.DD , 2X
200 IMAGE ("$")DDD.DD , 2X
```

Program output:

```
$1.25      $6.50
$ 1.25    $ 6.50
```

Multipliers. Multipliers are integer values in the range 1 to 9999. A multiplier of 1 is ignored in all cases (that is, no error will be generated in cases where a multiplier is not allowed). Only the following types of symbols allow a multiplier.

- Editing symbols: `%, "chars."`.
- Page control symbols: `/, @`.
- Symbols specifying individual characters in an output item: `D, *, Z` and `A`.
- Left parenthesis `(`.

For a description of the preceding characters, refer to “Using IMAGE symbols to Control Output” on page 137.

You can replicate any field by enclosing it in parentheses with a preceding multiplier. Thus, since the symbols `K, H, B`, and `^` are fields specifying entire output items, they must be enclosed in parentheses in order to be replicated. For example:

```
100 IMAGE 3(4D.2D,2X,5A),2/
200 IMAGE 2(B),4X,2Z3ZZ.D/25'=' /4(K)
```

The Special Output Symbols H, K, B, and ^. A special output specifier, because it uses an entire item, comprises an entire field. `H, K`, and `^` can be used to generate string or numeric items; `B` can be used only with numeric items.

The `K` and `H` symbols generate an item in the current display format, except that no leading or trailing blanks are generated. `K` and `H` differ only in the type of radix symbol for numeric output. `K` uses the decimal point, whereas `H` uses the comma (European radix).

IMAGE (continued)

E, which is used to format a numeric item, is equivalent to sending out CHR\$ of the number. The numeric item n must be in the range

$$-1048575 \leq n \leq 1048575.$$

The ^ symbol allows output items to be skipped. Any such item is still evaluated, which allows the HP-71 to execute a function without displaying or printing the returned value. This is useful if the evaluated function changes the state of the machine, such as changing the display device.

Examples Using H, K, B, and ^.

Program segment:

```
10 DISP USING 100; 583.5,247.3,"June",20
20 DISP USING 200; "Blue","Red","Hat","Shirt"
30 DISP USING 300; 65,66,67,CHR$(65),CHR$(66),CHR$(67)
100 IMAGE H,3X,K
200 IMAGE K,X,^
300 IMAGE 3(B),"=",3(A)
```

Program output:

```
583,5      247.3June      20
Blue Hat
ABC=ABC
```

For the next example, assume a plug-in ROM provides the PEN statement to change ink colors on a printer.

Program segment:

```
10 PRINT USING 100; FN(X(B,C), A
20 STOP
100 IMAGE ^, "Account #",6Z,X,"Updated"
200 DEF FN(X(B,C)
205 REM PEN 1 = black ink; PEN 2 = red ink
210 IF B>=C THEN PEN 1 ELSE PEN 2
220 FN(X)=B-C
230 END DEF
```

IMAGE (continued)

Printer output with B=33, C=22, and A=118042:

```
Account #118042 Updated          (in black ink)
```

Printer output with B=18, C=22, and A=118042:

```
Account #118042 Updated          (in red ink)
```

Editing symbols. The symbol `%` and quoted characters can be imbedded within any other field specifier without delimiters.

Editing Example.

Program segment:

```
10 DISP USING 100;2592, "Tues"
100 IMAGE DDD" dollars and "DD" cents.",2XK"day"
```

Program output:

```
25 dollars and 92 cents. Tuesday
```

Numeric Specifiers. The symbol `D` sends out a digit, with leading zeroes replaced by blanks.

The `*` symbol generates a digit, with each leading zero replaced by `*`.

The `Z` symbol generates a digit, with leading zeroes shown.

Numeric Specifier Example.

Program segment:

```
10 DISP USING 100; 12.4, 10.1, 6
100 IMAGE DDDDD,2X,****,2X,ZZZZZ
```

Program output:

```
12 ***10 00006
```

Digit output before the radix can be specified with a `D`, `*`, or `Z`, but not a mixed set of these symbols. The only exception is that a `Z` can always occupy the unit digit's place. This unit digit `Z` is only necessary with decimal output, since integer output will always show a digit in the unit place.

Sign specifiers (`S` and `M`) can be imbedded anywhere within a numeric field. Only one `S` or `M` is allowed in each numeric field.

IMAGE (continued)

If **S** or **M** is not specified, and the number is negative, then the minus sign will take up one digit position before the radix.

Radix specifiers (**.** and **R**) can only be followed by **D** symbols if digits are to be shown after the decimal point. Only one decimal (**.**) or **R** is allowed in each numeric field.

Digit separators (**C** and **F**) can appear anywhere in a numeric field. They cannot be adjacent to (or separated only by editing symbols from) other digit separators, radices, exponent specifiers, or delimiters. If a **C** or **F** appears when leading zeroes are being output, the symbol causes output as shown below:

Program segment:

```
10 DISP USING 100;2
20 DISP USING 200;2
30 DISP USING 300;2
100 IMAGE DC3DC3D.3DCD
200 IMAGE *C3*C3*.3DCD
300 IMAGE ZC3ZC3Z.3DCD
```

Program output:

```
      2.000,0
*****2.000,0
0,000,002.000,0
```

When used with **D** symbols, a digit separator within leading zeroes is replaced with a blank.

When used with ***** symbols, a digit separator within leading zeroes is replaced with a *****.

When used with **Z** symbols, a digit separator within leading zeroes appears as specified.

The **D** symbol allows output to “float” past blanks to the leftmost digit of the number, or to the radix indicator. The allowable floating specifiers are **S**, **M**, **X** and quoted characters that precede all digit specifiers in the field. These characters will float over any blank digit positions or separator positions before the first nonzero digit is encountered. When the first **D** is encountered, the floating capability is lost for any following **S**, **M**, **X**, or quoted characters. However, an implied negative sign always floats past all blanks.

IMAGE (continued)

Examples.

Program segment:

```
10 DISP USING 100; 1234.56, -65.8
20 DISP USING 200; 1234.56, -65.8
30 DISP USING 300; 1234.56, -65.8
100 IMAGE "$"6D.DD, X, "$"6D.DD
200 IMAGE "$"DC4D.DD, X, M"$"5D.DD
300 IMAGE "$"2DC3D.DD, X, D"$"4DZ.DD
```

Program output:

```
$1234.56      $-65.80
$1234.56      -$65.80
$1,234.56    $  -65.80
```

At least one digit specifier must precede the E symbol. The maximum exponent value allowable in DISP USING or PRINT USING output is ± 999 . This is only obtainable through display form manipulation, since, for normalized numbers, the maximum exponent value obtainable through arithmetic operations is ± 499 .

Program segment:

```
10 DISP USING 100; 1E-480
20 DISP USING 100; 1E-481
100 IMAGE 520DE
```

Program output:

```
100...(519 zeroes total)...000E-999
WRN L20:IMAGE Ovfl
***** (Generates 525 * symbols.)
```

NaN and Inf. Numeric field specifiers display the numeric values NaN and Inf if there are at least three numeric specifiers in the field (four in the case of -Inf). Numeric specifiers include the symbols D, Z, *, S, M, C, P, ., R, and E. For NaN and Inf output, each specifier corresponds to one position, except that E corresponds to five positions. Where there are insufficient positions to print or display the characters, the result depends on the current trap setting. That is, if the OVF trap is set to 0, the HP-71 displays the IMAGE Ovfl (error 47) message. Otherwise, the computer generates a warning and fills the numeric field with * symbols.

IMAGE (continued)

In the numeric field, `NaN` and `Inf` characters are right-justified, and will not be divided by editing symbols. All excess positions in the numeric field are filled with blanks, regardless of the type of symbol you use. (For example, `Z` and `*` symbols are filled with blanks when generating `NaN` or `Inf` output.) Generating `NaN` or `Inf` output with `D` symbols disables the floating characteristics of all editing symbols.

Program segment:

Note: In the following program, line 10 sets the IEEE traps to allow `NaN`, `Inf`, and error on overflow. Assigning the trap value to `T` in each case simply avoids displaying the value returned by `TRAP`. The variable `T` is otherwise not used in the program.

```

10 T=TRAP(IVL,2)@T=TRAP(DVZ,2)@T=TRAP(OVF,0)
20 SFLAG(-1) !SET QUIET TO SUPPRESS WARNINGS
30 DISP USING 100;SQR(-1),1/0
40 DISP USING 110;-123,-Inf
50 DISP USING 120;NaN
100 IMAGE Z,D,3".",3DC3D.2D
110 IMAGE "{S4D":"2D}"
120 IMAGE 2D

```

Program output:

```

      NaN...      Inf
      [-1:23][[-Inf:]
ERR L50:IMAGE Ovf1

```

Related Keywords

`DISP`, `DISP USING`, `ENDLINE`, `ENG`, `FIX`, `PRINT USING`, `SCI`, `STD`.

INF

INF (*infinity*) returns the machine representation of positive infinity.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
IF X=INF THEN INFINITE !BRANCHING
FOR I=1 TO INF STEP 5 !INFINITE LOOP
Y=TAN(2*X) !IF X=45 DEG,Y=INF
```

Comments

Any (non-NaN) number x is less than or equal to Inf . Most arithmetic functions accept Inf as an argument and process it according to standard arithmetic rules. For further information, refer to the descriptions of the individual functions in this manual, or to your *HP-71 Owner's Manual*.

Operations that cause an overflow can result in an infinite value if you set the overflow trap (OVF) to 2. Also, operations that set the divide-by-zero (DVZ) exception create Inf or $-\text{Inf}$ if you set the DVZ trap to 2.

Related Keywords

EPS, MAXREAL, MINREAL, TRAP.

INPUT

INPUT enables you to assign values to program variables from the keyboard.

■ Statement

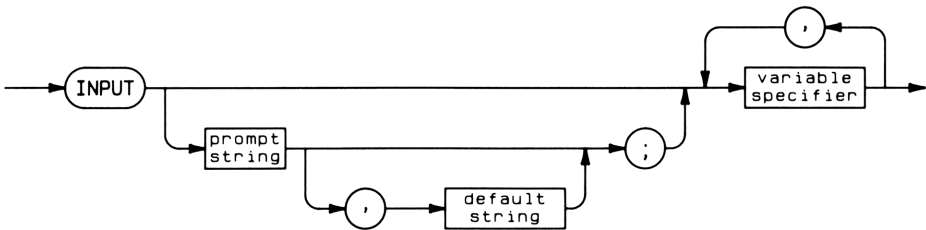
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
INPUT "NAME: ",N$,N$
INPUT "Duration,Frequency?";T,F
INPUT X,Y,A$C1,1J
```

Input Parameters

Item	Description	Restrictions
prompt string	Quoted string. Default: "?"	Cannot contain the same quote character as that used to delimit the string.
default string	String expression. Default: Null string.	None.
variable specifier	Numeric variable specifier or string variable specifier.	None.

INPUT (continued)

Comments

You can assign values from the keyboard to any numeric or string variable, substring, or array element.

Prompts. Executing `INPUT` displays the input prompt (?). If the last `DISP` or `DISP USING` statement was terminated (EOL suppressed) with a comma or semicolon for a delimiter, the ? prompt is appended to the specified display message. If the last such statement did not terminate (EOL *not* suppressed) with a comma or semicolon delimiter, the ? prompt appears on a line by itself.

Responding to Prompts. You can respond to a prompt in the following ways:

- Enter a list containing one or more numeric expressions, string expressions, or unquoted strings in any combination. The individual items must be separated by commas and must match the `INPUT` list variables in number and type. To enter an unquoted null string, key in two consecutive commas.
- If the input statement specifies a default string, you can either accept the string by pressing `END LINE` or change the string by editing it, then pressing `END LINE`. When you execute an `INPUT` prompt containing a default string, the HP-71 displays this string immediately after the prompt, with the cursor positioned at the first character in the string. For example:

Program Segment:

```
100 INPUT "CURRENT YEAR: ","1984"; A$
```

Prompt:

```
CURRENT YEAR: 1984
      ▲
      Display
      Cursor
```

If the `INPUT` statement requires a string, but the item you key in is not a string, the computer interprets the item as an unquoted string.

If you provide an improper number of inputs or enter an item that cannot be interpreted as a numeric expression when `INPUT` requires a numeric input, the computer gives an error message and again prompts you for an input. If this occurs, the cursor appears on the input character at which the error was detected.

INPUT (continued)

When prompted by an `INPUT` statement, if you press `[CONT]`, the computer:

- Terminates the `INPUT` operation without changing the variables in the input list.
- Continues program execution with the next statement.

Executing INPUT and Other Operations Simultaneously. While the `INPUT` prompt is displayed, the following operating conditions are active:

- The command stack is always active during `INPUT` execution. (Pressing the command stack key — `[9] [CMDS]` — is not required in order to initiate command stack operation.) You can use the `[▲]` and `[▼]` keys to move up and down in the command stack. The same command stack is used for keyboard input regardless of whether the computer is in BASIC or CALC mode, or is executing an `INPUT` statement.
- If you press a direct execute user-defined key, the computer ignores the display and uses the input of that user-defined key as the response to the `INPUT` prompt.
- The `[VIEW]` key and the `[9] [ERRM]` key sequence are active during `INPUT` execution.
- Pressing `[ATTN]` either clears the input buffer (if it is not already clear) or pauses the program (if the input buffer is already clear). Thus, pressing `[ATTN]` twice in a row during execution of `INPUT` interrupts program execution. If you subsequently continue the program, the interrupted `INPUT` statement is reexecuted.
- Pressing `[RUN]`, `[SST]`, or `[CALC]` during `INPUT` execution has the same effect as pressing `[END LINE]`.
- The keyboard buffer allows you to “type ahead” in anticipation of a prompt if you know what input will be required by that prompt. As soon as the prompt appears, the key(s) held in the buffer are accepted as the input, and program execution continues.
- The HP-71 temporarily suspends any `ON TIMER` operations that come due during execution of an `INPUT` statement. After (1) you complete the input process and (2) the computer assigns all values (and unless a multiple-line user-defined function is invoked), any suspended `ON TIMER` operations are processed.
- An error can cause an `ON ERROR` branch. If there is no active `ON ERROR` statement, and if an input response results in an input error, the computer prompts you to reenter all variables in the input list.

The HP-71 automatically turns itself off after ten minutes of inactivity. To continue execution, press `[ATTN]` to turn the computer on, then press `[CONT]` to reexecute the `INPUT` statement.

Related Keywords

`LINPUT`, `WIDTH`.

INT

INT (*integer* \leq *argument*) returns the greatest integer that is less than or equal to the argument, and is identical to FLOOR.

☐ Statement

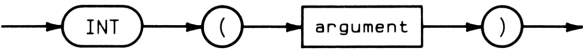
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A1=INT(X1)
IF X/2 = INT(X/2) THEN GOTO EVEN
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

If the value of the numeric expression is an integer, that value is returned. If the value of the expression is not an integer, INT returns the greatest integer value less than or equal to the expression. For example:

- INT(1.5) returns 1.
- INT(-1) returns -1.
- INT(-1.5) returns -2.

Related Keywords

CEIL, FLOOR, FP, IP.

INTEGER

INTEGER allocates memory for integer variables and arrays.

■ Statement

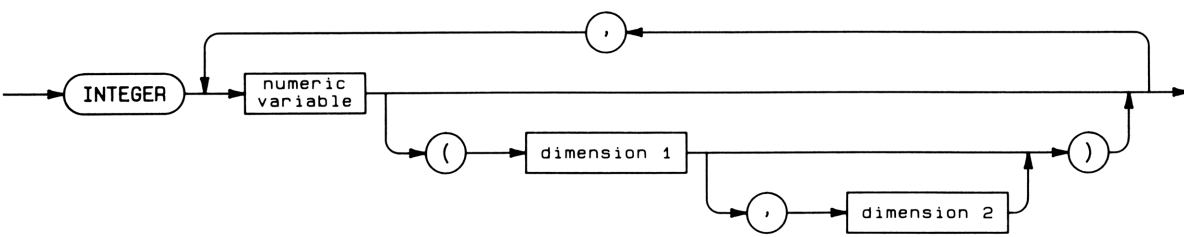
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

INTEGER I INTEGER J2, B(4,4) INTEGER P9(7,3)

Input Parameters

Item	Description	Restrictions
numeric variable	Letter followed by optional digit.	None.
dimension limit 1 dimension limit 2	Numeric expression rounded to an integer.	Current OPTION BASE setting to 65535.

Comments

INTEGER creates integer variables and arrays. Creation occurs upon execution of INTEGER. The dimension limits are evaluated at creation time. The lowest-numbered subscript in any dimension is 0 or 1, depending on the OPTION BASE setting when the array is created. All elements are initialized to zero.

INTEGER (continued)

If `INTEGER` specifies a simple numeric variable that already exists, the variable is reinitialized to zero. Array variables are redimensioned, but not reinitialized to zero (unless the data type is changed). If `INTEGER` expands an array, it also initializes all newly-created elements in the array. Notice that redimensioning does not necessarily preserve an element’s position within an array, but does preserve the sequence of elements within an array. Refer to “Declaring Arrays (`DIM`, `REAL`, `SHORT`, `INTEGER`)” in section 3 of the *HP-71 Owner’s Manual*.

The following table indicates the conditions that apply to `INTEGER` variables and arrays:

Integer Numeric Variables	
Initial Value of Variables	0
Integer Range	± 99999
Maximum Number of Array Dimensions	2
Maximum Dimension Limit	65535
Memory Usage in Bytes:	
• Simple Variable	9.5
• Array	$3 * (dim1 - base + 1) * (dim2 - base + 1) + 9.5$

Related Keywords

`DIM`, `REAL`, `SHORT`.

INX (*inexact*) returns the number of the inexact result flag (−4).

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF . . . THEN . . . ELSE



Examples

```
A=FLAG(INX,0)                                B=TRAP(INX,2)
```

Related Keywords

CFLAG, DEFAULT, DVZ, FLAG, IVL, OVF, SFLAG, TRAP, UNF.

IP

IP (*integer part*) returns the integer part of the argument.

☐ Statement

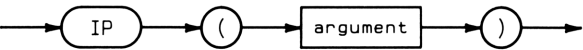
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A1=IP(X1)
PRINT "Integer part = ";IP(Y),
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

The returned value has the same sign as the argument. For example:

- IP(1.5) returns 1.
- IP(-1) returns -1.
- IP(-1.5) returns -1.

Related Keywords

CEIL, FLOOR, FP, INT.

IVL (*invalid operation*) returns the number of the invalid operation flag (−8).

- | | |
|--|--|
| <input type="checkbox"/> Statement | <input checked="" type="checkbox"/> Keyboard Execution |
| <input checked="" type="checkbox"/> Function | <input checked="" type="checkbox"/> CALC Mode |
| <input type="checkbox"/> Operator | <input checked="" type="checkbox"/> IF ... THEN ... ELSE |



Examples

```
IF FLAG(IVL) THEN STOP           A=TRAP(IVL,0)
```

Related Keywords

CFLAG, DEFAULT, DVZ, FLAG, INX, OVF, SFLAG, TRAP, UNF.

KEY\$

KEY\$ (*key string*) returns a string representing the oldest key or keystroke combination currently held in the key buffer, and removes that key or keystroke combination from the buffer.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A$=KEY$
```

Comments

The key buffer can contain up to 15 keys or keystroke combinations. The format in which the key data is returned is the same as that for DEF KEY, KEYDOWN, and KEYDEF\$. If the key buffer is empty then the null string is returned.

The string returned for a given key is determined as follows:

- If there is a single ASCII character that uniquely identifies the key, KEY\$ returns this character. For example, Q identifies the [Q] key and q identifies the [g]-shifted [Q] key.
- If the key is an [f]- or [g]-shifted key, and the key's primary function is uniquely identified by a single ASCII character, then KEY\$ returns a two-character string. This string consists of f or g followed by the corresponding primary character. For example, gQ is the [g]-shift of the [Q] key.
- If neither of the above apply, KEY\$ returns # followed by the decimal-numbered key code for that key. For example, if you press [RUN] during execution of the following program, the HP-71 displays #46.

```
10 DELAY 0,0

20 FOR I=1 TO 100
30 DISP KEY$;
40 NEXT I
```

Suppresses display time (because any keys held in the display buffer at the start of a DISP display are cleared).

Adds the key designation to the current display.

The LC statement does not affect the returned string.

KEY\$ (continued)

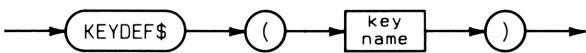
Related Keywords

DEF KEY, KEYDEF\$, KEYDOWN, PUT.

KEYDEF\$

KEYDEF\$ (*key definition string*) returns the redefined value of a key.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A$=KEYDEF$("Q")
IF KEYDEF$("#43")C10 = "U" THEN DISP "Attn key has been
redefined"
```

Input Parameters

Item	Description	Restrictions
key name	String expression.	Less than five characters.

Comments

The returned string has the same format as is seen by pressing VIEW .

KEYDEF\$ uses the same format as DEF KEY to specify a key name. The first character in the returned string indicates the type of definition for the key. The following list describes the possible first characters:

- ; : Nonterminating.
- Blank: Terminating.
- ! : Direct execute.
- U : Key is not redefined.

The remaining characters in the string describe the key redefinition text.

If a specified key has not been redefined, KEYDEF\$ returns the message “Unassigned.”

KEYDEFS (continued)

Related Keywords

DEF KEY, FETCH KEY.

KEYDOWN

KEYDOWN returns either a 0 or a 1, depending on whether a key is being pressed.

☐ Statement

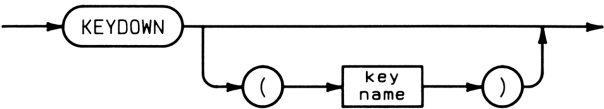
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
IF NOT KEYDOWN THEN DISP "All keys up"  
IF KEYDOWN("7") THEN CALL MOVEUP @ CALL MOVELEFT
```

Input Parameters

Item	Description	Restrictions
key name	String expression.	Less than five characters. Also refer to "Comments," below.




Comments

KEYDOWN enables you to test for either any key being pressed or a specific key being pressed.

- To test for any key being pressed, use KEYDOWN without any key parameter. The and keys are *not* ignored by this KEYDOWN option.
- To test for a specific key being pressed, use KEYDOWN with a parameter specifying the desired key. Parameter options are as follows:
 - Key name expressed as a character: For unshifted keys identified by a symbol, such as a letter or the key, enter a quoted string containing only that character. Where a shifted or unshifted key enters a symbol having a corresponding ASCII code, you can specify that key by using CHR\$(*n*), where *n* is the ASCII code for that symbol.

KEYDOWN (continued)

- **Key name identified by key number:** If the first character of a multiple-character key name is #, KEYDOWN interprets the string as a key number. You can use this method to specify any unshifted key. The key numbers are as follows:

Key Numbers	Description	Keys
1 through 56	Unshifted keys	 ,  , . . . 

The preceding parameters use the same formats as those used with DEF KEY. If you use the null string for a KEYDOWN parameter, the HP-71 returns zero.

The HP-71 accepts KEYDOWN (without a parameter) in CALC mode.

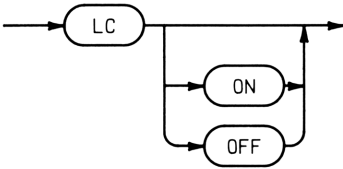
Related Keywords

DEF KEY, KEY#, KEYDEF#, PUT.

LC

LC (*case lock*) toggles between the uppercase lock and the lowercase lock on the keyboard.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF . . . THEN . . . ELSE



Examples

LC

LC ON

LC OFF

Comments

During usual keyboard input the primary alpha keys enter upper case letters and the [9]-shifted alpha keys enter lower case letters. The LC statement is used primarily for switching these assignments, as follows:

- Executing LC ON sets system flag -15, which assigns the lowercase characters to the corresponding primary alpha keys and assigns the uppercase characters to the [9]-shifted alpha keys.
- Executing LC OFF clears system flag -15, which assigns the uppercase characters to the corresponding primary alpha keys and assigns the lowercase characters to the [9]-shifted keys.
- Executing LC switches the current setting of system flag -15, which exchanges the current uppercase and lowercase key assignments for the primary and [9]-shifted alpha keys. You can switch this same flag from the keyboard by pressing the [LC] key.

LEN (*string length*) evaluates the specified string expression and returns its length.

☐ Statement

☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

L=LEN(A\$&B\$I,JJ)

IF LEN(A\$)=5 THEN 600

Input Parameters

Item	Description	Restrictions
string expression	Refer to the Glossary.	None.

Comments

Leading, trailing, and enclosed spaces are included in the returned value. For example, executing LEN (" Nathan Meyers ") returns 15.

LET

LET is the assignment statement, which is used to assign values to variables.

■ Statement

□ Function

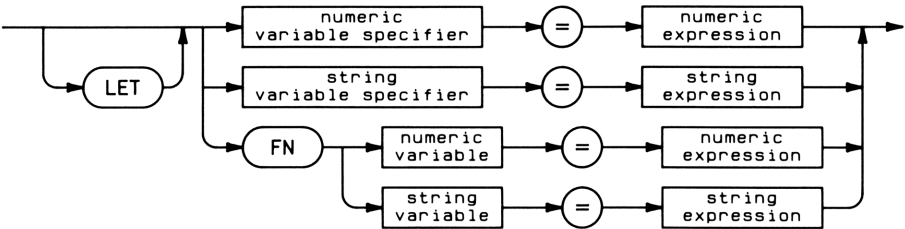
□ Operator

■ Keyboard Execution

■ CALC Mode*

■ IF ... THEN ... ELSE

* You can use only the implicit form of LET in CALC mode—for example, A=7. That is, the explicit form of LET—for example, LET A=7—is not allowed in CALC mode.



Examples

LET A=3*B+7

A\$="STEVEN A,"

Q\$C3,5J="MULTIMETER"

M(4,3)=4*PI*R^2

B7\$(I)="TITLES"

LET FNF=Q/P+3

Input Parameters

Item	Description	Restrictions
numeric variable specifier	Refer to Glossary.	None.
numeric expression	Refer to Glossary.	None.
string variable specifier	Refer to Glossary.	None.
string expression	Refer to Glossary.	None.
numeric variable	Refer to Glossary.	None.
string variable	Refer to Glossary.	None.

LET (continued)

Comments

If the specified variable does not exist, it is created when the HP-71 executes the corresponding `LET` statement. If the variable is an element of a nonexistent array, the HP-71 creates that array (where the maximum allowed indices are 10 and minimum allowed indices correspond to the current `OPTION BASE` setting). A string variable created by `LET` has a default maximum length of 32 characters. Thus, larger maximum lengths must be explicitly dimensioned.

A numeric variable of type `SHORT` causes rounding (according to the current round-off setting) to five significant decimal digits. However, the calculation itself is performed in 12-digit arithmetic. A numeric variable of type `INTEGER` causes rounding to an integer (according to the *integer rounding* method). If, after rounding, the result has more than five digits, the stored value will be ± 99999 or `INF`, depending on the current trap settings for the math exception flags.

In addition to assigning a numeric result to the specified variable, the HP-71 stores the result in the register reserved for use by `RES`. (The value stored for use by `RES` is the result calculated prior to any rounding for `SHORT` or `INTEGER`.)

In a user-defined function having multiple statements, the function name itself is a legal variable. Therefore, whatever value is in this variable when the HP-71 finishes executing the corresponding function is the value returned by that function.

A string value replaces the current value of the specified string variable. If the new value exceeds the string variable's maximum length, the replacement is not performed, and a `String Ovfl` (error 37) condition occurs.

The following illustrates how the HP-71 handles string assignments:

Statement: `A$="ABCDEFGHJKLMNOP"`

Output: `ABCDEFGHJKLMNOP`

Replaces any earlier `A$` string value.

Statement: `A$[5]="TUVW"`

Output: `ABCDTUVW`

Replaces portion of `A$` that begins with position 5. (`TUVW` becomes the new end-of-string, starting at position 5.)

Statement: `A$[5,10]="EFGHIJK^^QRS"`

Output: `ABCDEFGHJKLMNOP`

Inserts string between positions 4 and 5. Notice that second substring index is less than the first (by an arbitrary amount), which causes the HP-71 to insert the substring into the existing string instead of overwriting part of that string.

LET (continued)

Statement: `A#[12,13]="LMNOP"`

Output: ABCDEFGHIJKLMNOPQRSTUVWXYZ

Statement: `A#[2,22]="-"`

Output: A-W

Statement: `A#[5]="123"`

Output: A-W 123

Statement: `A$=""`

Output: —Null—

Replaces a two-character substring with a five-character string.

Replaces a 21-character substring with a 1-character string.

Uses a “blank-filling” technique that allows the 123 string to be appended.

Sets `A$` to null. Note that a null value differs from a blank space value.

Related Keywords

INTEGER, REAL, RES, SHORT.

LINPUT

LINPUT (*line input*) assigns an entire line from the display to a string variable.

■ Statement

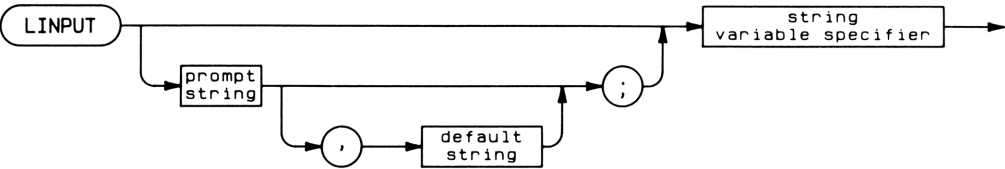
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF...THEN...ELSE



Examples

```
LINPUT "City, State: ", "Whittier, CA");C$
LINPUT L$
LINPUT "Enter line:");T$
```

Input Parameters

Item	Description	Restrictions
prompt string	Quoted string. Default: "?"	Cannot contain delimiting character.
default string	String expression. Default: Null string.	None.
string variable specifier	Refer to Glossary.	None.

Comments

LINPUT is similar to the INPUT statement in that it causes the HP-71 to pause execution, activate the keyboard, and allow you to enter a variable. LINPUT assigns the resulting line to the specified string variable. LINPUT is distinguished from INPUT in that it allows punctuation, whereas INPUT treats punctuation as delimiters. (Unlike the INPUT statement, the string you enter with LINPUT is interpreted literally, and not as a string expression.)

LINPUT (continued)

You can respond to a LINPUT prompt in either of the following ways:

- Enter any number of characters.
- Enter an “execute only” string. (That is, a string that has been used with a colon to redefine a key.) Notice that if you enter an execute only string, the HP-71 ignores any other characters that may already be in the display.

For information concerning prompts and default inputs, refer to the comments provided in the **INPUT** keyword entry.

Related Keywords

INPUT.

LIST

LIST displays the specified BASIC or KEY file.

■ Statement

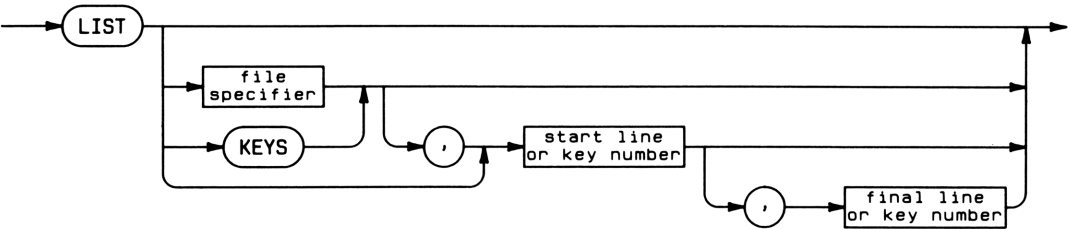
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

`LIST`
`LIST KEYS,15,168`
`LIST 10`
`LIST MECH65,100,200`

`LIST MEAN:PORT(2)`
`LIST A$, 225`
`LIST 50,85`

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default: Current file.	File name with optional device specifier.
start line or key number	Integer constant identifying a program line or key number. Default: First program line or key assignment in file.	1 through 9999.
final line or key number	Integer constant identifying a program line or key number. Default: Start line or key number, if specified; otherwise, last program line or key assignment in file.	Start line or key number through 9999.

LIST (continued)

Comments

General Operation. Specifying a file that is not a BASIC or KEY file generates an `Invalid File Type` (error 63) condition.

Executing `LIST KEYS` results in a listing of the current key assignment file, `keys` (if present).

Specifying a single line or key parameter lists only that line or key assignment. If you specify a range and the HP-71 does not find the start line or key number, but does find a higher-numbered line or key number within the specified range, the listing begins with that higher-numbered line or key. Executing `LIST` without specifying any line or key numbers generates a listing of the entire file.

The current `DELAY` setting determines how long the computer displays each line.

The current `WIDTH` setting determines the width of the display line.

Interrupting a Listing. To halt a listing and display the cursor, simply press `ATTN`.

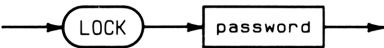
Related Keywords

`DELAY`, `WIDTH`, `PLIST`. See also the descriptions of `FETCH` and the `▲`, `▼`, `9 ▲`, and `9 ▼` keystrokes in your *HP-71 Owner's Manual*.

LOCK

LOCK specifies a password to provide for security against unauthorized use of your HP-71. Anyone who turns on the computer will be prevented from using it until they enter the password.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

LOCK "Abc"

LOCK N\$

LOCK ""

Input Parameters

Item	Description	Restrictions
password	String expression.	0 through 8 characters.

Comments

The password can be any combination of up to eight letters, numbers, spaces, and symbols.

Note: Because the lock is *absolute*, it is recommended that you choose an easy-to-remember password. If you cannot enter the correct password, the HP-71 will not respond to your instructions. If this situation occurs, you can regain control of the computer only by resetting the computer, which clears both the memory and the clock.

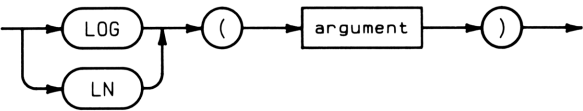
Using LOCK. After initially executing LOCK with a password, each time you turn on the HP-71, you are prompted by the message password?. To unlock the computer, type in the exact password that you specified using LOCK, then press ENDLINE. The HP-71 then displays the cursor and you can continue with your work. Any attempt to unlock the computer by entering an invalid password simply turns off the computer.

Deactivating LOCK. To deactivate LOCK, remove the password by entering LOCK with a null argument, as shown in the rightmost statement example, above. (Ensure that there are no spaces between the quotes. If a space is included, it will be interpreted as a new password.)

LOG (LN)

LOG returns the natural logarithm (base *e*) of the argument.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
T = LOG(X/A) + T1
PRINT "Natural log of";Y;"=";LOG(Y)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	Refer to “Comments,” below.

Comments

- LOG is subject to the following restrictions:
- Attempting to compute the natural logarithm of zero results in a LOG(0) (error 12) condition.
 - Attempting to compute the natural logarithm of a negative number results in a LOG(neg) (error 13) condition.

Related Keywords

EXP, EXPM1, LOGP1, LOG10.

LOGP1

LOGP1 (*logarithm of (argument +1)*) returns $\ln(1 + x)$.

☐ Statement

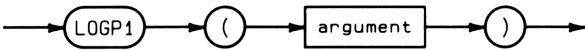
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
X=LOGP1(1.2345E-10)
PRINT "(1+R)^N =" ; EXP(N*LOGP1(R))
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	Must be greater than -1 .

Comments

You can use LOGP1 to eliminate large relative errors in certain calculations. For instance, LOGP1 (1.2345E-10) is accurate to 12 significant digits, but LN(1+1.2345E-10) becomes LN (1.000000000012) (because the HP-71 uses 12 digits). This results in a large relative difference from the accurate answer, since $\ln(1.00000000012)$ is quite different from $\ln(1+1.2345E-10)$.

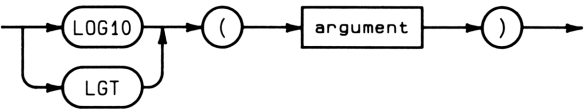
Related Keywords

EXP, EXPM1, LOG.

LOG10 (LGT)

LOG10 returns the logarithm (base 10) of the argument.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

D=20*LOG10(V)

PRINT TAB(LGT(X));"*

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	Refer to “Comments,” below.

Comments

The following exceptions can occur:

- Attempting to compute the logarithm of zero results in a Log(0) (error 12) condition.
- Attempting to compute the logarithm of a negative number results in a LOG(neg) (error 13) condition.

Related Keywords

EXPONENT, LOG.

LR (*linear regression*) specifies the current linear regression model and computes the intercept and slope for that model. The computed intercept and slope are returned in optional variables.

■ Statement

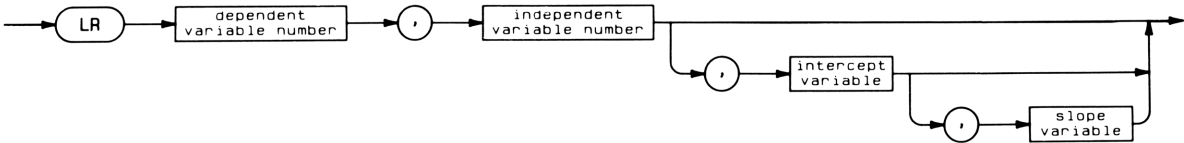
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

LR 1,3

LR 1,2,A(1),A(2)

LR 2,1,X,Y

Input Parameters

Item	Description	Restrictions
dependent variable number	Numeric expression rounded to an integer.	Zero through the current <code>STAT</code> array dimension.
independent variable number	Numeric expression rounded to an integer.	One through the current <code>STAT</code> array dimension.
intercept variable	Numeric variable specifier. (Refer to Glossary.)	None.
slope variable	Numeric variable specifier. (Refer to Glossary.)	None.

LR (continued)

Comments

LR specifies the linear regression of the first (dependent) variable on the second (independent) variable. The intercept and slope are stored in the intercept and slope variables, if present.

Related Keywords

ADD, CLSTAT, CORR, DROP, MEAN, PREDV, SDEV, STAT, TOTAL.

MAX (*maximum*) returns the larger of two values.

☐ Statement

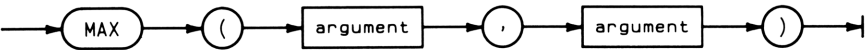
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
M = MAX(A,B)
PRINT "Maximum = ";MAX(A,MAX(B,C))
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Related Keywords

MIN.

MAXREAL

MAXREAL (*maximum real number*) returns 9.999999999999E499, which is the overflow threshold—the maximum positive finite number that the HP-71 can represent.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
U=TRAP(UNF,2)@R=EXPONENT(MAXREAL)-EXPONENT(MINREAL)@TRAP(UNF,U)
```

Comments

The HP-71's exponent range is -510 through 499 ($R=1009$).

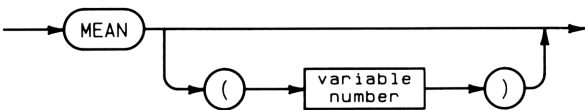
Related Keywords

EPS, INF, MINREAL.

MEAN

MEAN returns the sample mean of the specified variable in the current statistical array.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A1=MEAN(X1)
```

```
PRINT "Mean = ";MEAN
```

Input Parameters

Item	Description	Restrictions
variable number	Numeric expression rounded to an integer. Default: 1.	Zero through the current STAT array dimension.

Related Keywords

```
ADD, CLSTAT, CORR, DROP, LR, PREDV, SDEV, STAT, TOTAL.
```

MEM

MEM (*memory*) returns the number of available bytes in either main RAM or a memory device.

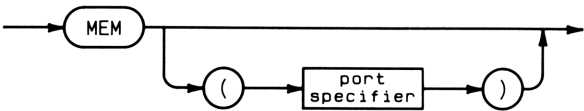
- ☐ Statement

☒ Function

☐ Operator
- ☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

MEM

MEM+4*2^10

DIM A(MEM/8-20)

MEM (1)

Input Parameters

Item	Description	Restrictions
port specifier	<p>Numeric expression truncated to two digits after the decimal point. Interpreted as <i>P.dd</i>, where:</p> <p><i>P</i> = <i>port number</i>.</p> <p><i>dd</i> = <i>device number</i>.</p> <p>Default: Returns current MEM value for main RAM.</p>	<p>$0 \leq P \leq 5$</p> <p>$0 \leq dd \leq 15$</p>

Comments

The following can help you plan how to use available memory.

Variable Type	Bytes Used per Simple Variable	Bytes Used per Array* Variable	Precision
INTEGER	9.5	3	± 99999
SHORT	9.5	4.5	5-digit
REAL	9.5	8	12-digit
STRING	11.5 + MAXIMUM LENGTH	2 + MAXIMUM LENGTH	—
*Any array has a 9.5-byte overhead.			

MERGE

MERGE integrates a portion of the file you specify into either the current BASIC file or the system `keys` file, depending on the type of the specified file.

■ Statement

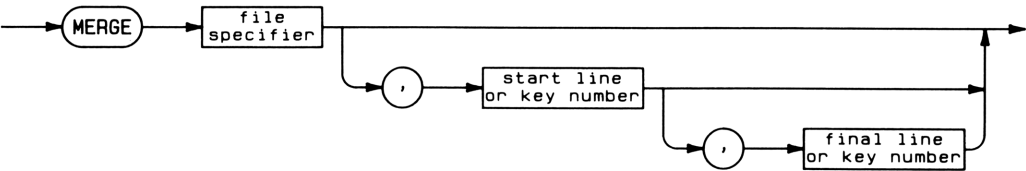
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF...THEN...ELSE



Examples

MERGE A:PORT

MERGE B\$,5

MERGE KEYFILE1,100,135

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string.	File name with optional device specifier.
start line or key number	Integer constant identifying a program line or key number. Default: First program line or key assignment in file.	1 through 9999.
final line or key number	Integer constant identifying a program line or key number. Default: Start line or key number, if specified. Otherwise last line or key assignment in file.	Start line or key number through 9999.

MERGE (continued)

Comments

If you specify a BASIC file, it is merged into the current file. If you specify a KEY file, it is merged into the system `keys` file. Any other file type results in an `Invalid File Type (error 63)` condition. If the specified file is a KEY file and the system `keys` file does not already exist, `MERGE` creates it.

`MERGE` inserts all line numbers or key assignments into their proper positions in the destination file. If a line or key number in the file to be merged already exists in the destination file, the line or key number in the source file replaces the corresponding program line or key assignment in the destination file. In a BASIC file, you can prevent any duplicate line numbers from being replaced in the destination file by first renumbering either of the files to ensure that there are no common line numbers.

Merging a BASIC file by executing `MERGE` in a running program performs the merging operation, then terminates program execution, releasing local environments and clearing all program control information.

Merging a file does not alter the file being merged. Only the current or `keys` file is altered.

Related Keywords

`RENUMBER.`

MIN

MIN (*minimum*) returns the smaller of two values.

☐ Statement

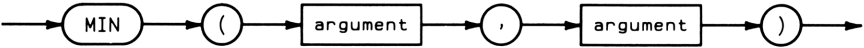
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
M=MIN(A,0)
PRINT "Minimum = ";MIN(A,MIN(B,C))
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Related Keywords

MAX.

MINREAL

MINREAL returns 0.000000000001E-499, which is the smallest positive number that the HP-71 can represent. To return this number, set the underflow trap to 2 before executing MINREAL(TRAP(UNF, 2)). Otherwise, executing MINREAL generates an underflow exception.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
U=TRAP(UNF,2)@W=1-EXPONENT(MINREAL/EPS)@TRAP(UNF,U)
```

Comments

The value MINREAL returns is one example of a denormalized number. (For a discussion of denormalized numbers, refer to “Denormalized Numbers and -0” on page 341.) Executing MINREAL sets the underflow exception flag (flag -5) if TRAP(UNF) is set to 0 or 1.

Related Keywords

EPS, INF, MAXREAL.

MOD

MOD(*x*, *y*) (*modulo*) returns a remainder that is defined by the expression $x - y * INT(x/y)$.

☐ Statement

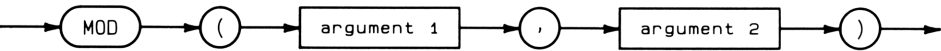
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
IF MOD(K,3)=1 THEN PRINT K      W=X-MOD(Y,Z)
```

Input Parameters

Item	Description	Restrictions
argument1	Numeric expression	Refer to “Comments,” below.
argument2	Numeric expression.	Refer to “Comments,” below.

Comments

MOD is similar to RED and RMD, and has the following properties:

- MOD(*x*, *y*) is periodic in *x* with period $|y|$.
- MOD(*x*, *y*) lies in the interval $[0,y)$ for $y > 0$ and $(y,0]$ for $y < 0$.
- MOD can set the Inexact Result flag (INX). When this occurs, it indicates that the value returned by MOD requires rounding in order to fit into a REAL variable. For example, MOD($-E-20,360$) is $(360 - 1E-20)$, which cannot be represented exactly with 12 digits of precision.

If either $x = Inf$ or $y = 0$, an Invalid Arg (error 11) condition occurs.

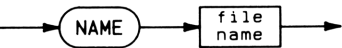
Related Keywords

RED, RMD.

NAME

NAME names the system `workfile`.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

NAME A\$&B\$

NAME "FILE1"

NAME ABC1

Input Parameters

Item	Description	Restrictions
file name	String expression or unquoted string.	Any valid file name.

Comments

While `workfile` is the current file you can easily access it. However, once you move the edit pointer away from `workfile`, there is no way to reference the file without redesignating it as the current file (by executing `EDIT` `END LINE`). Assigning a name to `workfile` enables you to reference it without designating it as the current file.

NAN

NAN (*not-a-number*) returns the value of a Signaling NaN.

- | | |
|--|--|
| <input type="checkbox"/> Statement | <input checked="" type="checkbox"/> Keyboard Execution |
| <input checked="" type="checkbox"/> Function | <input checked="" type="checkbox"/> CALC Mode |
| <input type="checkbox"/> Operator | <input checked="" type="checkbox"/> IF ... THEN ... ELSE |



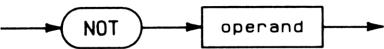
Examples

```
A=NAN
```


NOT

NOT performs the logical NOT of its operand.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input checked="" type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A=P AND NOT Q OR Q AND NOT P      IF NOT Q THEN GOTO 100
```

Input Parameters

Item	Description	Restrictions
operand	Numeric expression.	Subject to operator precedence.

Comments

Operands used with NOT are considered logically false if zero and logically true if nonzero. The table to the right indicates the range of results for NOT.

The precedence of NOT in relation to the HP-71's other operators is described under "Precedence of Operators" on page 317.

Operand	Result
True	0
False	1

Related Keywords

AND, EXOR, OR.

NUM

NUM (*number*) returns the ASCII character code for the first character of a string.

☐ Statement

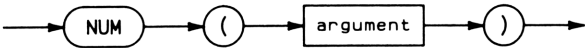
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

NUM("X")

IF NUM(B\$)=R THEN C\$=C\$&B\$

Input Parameters

Item	Description	Restrictions
argument	String expression.	None.

Comments

NUM returns a decimal number. For example, executing NUM("A") returns 65. If a specified string is null, NUM returns 0.

NUM is the inverse of the CHR\$ function.

Related Keywords

CHR\$.

OFF, OFF ERROR/TIMER

OFF turns off the HP-71.

OFF ERROR disables any previous ON ERROR statement, thus returning the computer to its default method of error reporting.

OFF TIMER # deactivates the corresponding ON TIMER # statement.

■ Statement

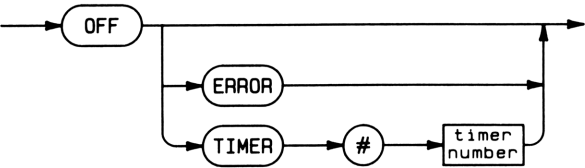
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

OFF

OFF ERROR

OFF TIMER

OFF TIMER #1

OFF TIMER #N

#T1*X/3

Input Parameters

Item	Description	Restrictions
timer number	Numeric expression rounded to an integer.	Timer number 1, 2, or 3.

OFF, OFF ERROR/TIMER (continued)

Comments

The OFF Statement. OFF is equivalent to BYE. Executing OFF in a program, then later turning on the HP-71 causes program execution to automatically resume with the statement immediately following the OFF statement. If you execute OFF from the keyboard, any statements concatenated after OFF are not executed when the HP-71 is turned on again.

The OFF ERROR Statement. In addition to OFF ERROR, any one of the following four operations disables a previously executed ON ERROR statement:

- Executing END.
- Executing the last statement of a program.
- Editing a program.
- Running another program.

Timer Control of Program Execution. If the HP-71 activates a timer by executing ON TIMER, then subsequently executes OFF, the computer turns itself off as described above. However, when the timer expires, the computer turns itself on, services the timer, and resumes program execution.

The OFF TIMER Statement. There are three timers, numbered 1, 2, and 3. When rounded, an OFF TIMER number must equal an integer from 1 to 3.

An OFF TIMER # statement deactivates the corresponding ON TIMER # statement. No further interrupts from that timer will occur until you reactivate it.

Any of the four operations listed above that disables an ON ERROR statement also deactivates *all three timers*.

Related Keywords

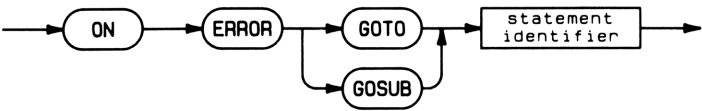
- OFF: BYE.
- OFF ERROR: ON ERROR.
- OFF TIMER: ON TIMER #.

ON ERROR GOSUB/GOTO

`ON ERROR` causes the computer to execute the specified subroutine or branch when an error occurs during program execution.

☒ Statement
☐ Function
☐ Operator

☐ Keyboard Execution
☐ CALC Mode
☒ IF ... THEN ... ELSE



Examples

`ON ERROR GOSUB 1200`
`ON ERROR GOSUB "ERROR"`
`ON ERROR GOSUB E$`

`ON ERROR GOTO 3000`
`ON ERROR GOTO "EXIT"`
`ON ERROR GOTO E1$`

Input Parameters

Item	Description	Restrictions
statement identifier	Line number or label of a program statement.	Any valid line number or label reference.

Comments

This keyword allows you to design specialized error-handling routines into your programs.

`ON ERROR` is a local declaration that, when executed, activates an `ON ERROR` condition that traps program errors in the following manner:

- An `ON ERROR GOSUB` condition transfers execution to the specified *statement* in the same way as for a subroutine call. Thus, when execution subsequently encounters a `RETURN` statement, execution transfers to the first statement following the one in which the error occurred.
- An `ON ERROR GOTO` condition unconditionally transfers execution to the specified *statement* in the same way as for a program branch.

ON ERROR GOSUB/GOTO (continued)

ON ERROR is subject to the following operating conditions:

- Regardless of the number of errors that occur, an ON ERROR declaration remains active for the program that executed it until replaced with another ON ERROR declaration or disabled with an OFF ERROR statement.
- ON ERROR and OFF ERROR act locally, affecting only the program or subprogram in which they appear. When one program calls another, the computer temporarily suspends any active ON ERROR declaration in the first program until execution returns to that first program.

If an ON ERROR GOSUB routine itself contains an error, program execution halts. If an ON ERROR GOTO routine contains an error, an infinite loop can occur between the statement in error and the routine specified by ON ERROR GOTO.

Executing OFF ERROR deactivates ON ERROR.

When using ON ERROR it is sometimes helpful to also use ERRL, ERRN, or ERM\$ in the specified ON ERROR subroutines or branches, which produces the following results:

ERRL: Returns the most recent line number in error.

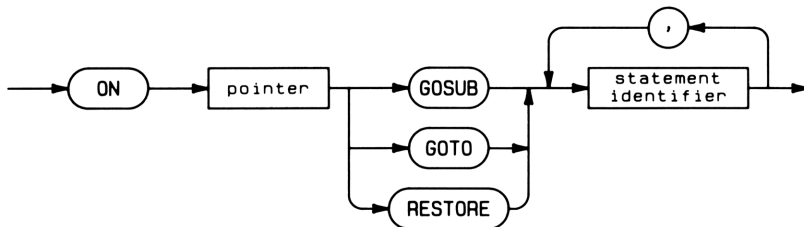
ERRN: Returns the most recent error number.

ERM\$: Returns the error message associated with the most recent error number.

ON...GOSUB/GOTO/RESTORE

ON ... GOSUB and ON ... GOTO transfer program execution to the destination specified by GOSUB or GOTO. ON ... RESTORE resets the READ data pointer to the closest DATA statement following the selected statement.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF ... THEN ... ELSE



Examples

```

ON X1 GOSUB 100,200,300
ON P2 GOSUB CALC1,CALC2,CALC3,CALC4
ON Q*Y/Z GOSUB G1$(Q*Y/Z),G1$(Q*Y/Z),G1$(Q*4/Z),X$
ON Y5 GOTO 1000,2000,3000
ON Q1*Y/Z GOTO Q1$,Q2$,Q3$,E$
ON P GOTO COMP1,COMP2,COMP3,COMP4,COMP5
ON D RESTORE 150,250,350
ON E1 RESTORE DATA1,DATA2,DATA3,DATA4,DATA5
ON Q1*Y/Z RESTORE D$(1),D$(2),D$(3),E$

```

ON...GOSUB/GOTO/RESTORE (continued)

Input Parameters

Item	Description	Restrictions
pointer	Numeric expression rounded to an integer.	1 through the number of statement identifiers listed.
statement identifier	Line number or label of a program statement.	Any valid line number or label reference. If more than one is listed, use , to separate.

Comments

The ON ... GOSUB and ON ... GOTO Statements. `ON ... GOSUB` executes any of one of several subroutines that you have listed by statement identifier in an `ON ... GOSUB` statement. The HP-71 uses the current value of the pointer in `ON ... GOSUB` to select a statement identifier from the list. That is, executing `ON ... GOSUB` causes the computer to:

1. Determine the specified pointer value.
2. Use the pointer value as a position in the `ON ... GOSUB` address list. A pointer value of 1 corresponds to the first statement identifier; 2 corresponds the the second statement identifier, and so on.
3. Execute the subroutine that begins at the statement specified by the statement identifier listed in the selected position in the `ON ... GOSUB` list.

Following an `ON ... GOSUB` statement, when a running program encounters a `RETURN` statement, program execution returns to the statement following that `ON ... GOSUB` statement.

`ON ... GOTO` transfers execution to another location in program memory in the same way as `ON ... GOSUB`, but has no provision for a subsequent return of execution.

The ON...RESTORE Statement. `ON ... RESTORE` specifies a statement in the same way as `ON ... GOSUB` and `ON ... GOTO`. If that statement is a `DATA` statement, the computer then resets the `READ` data pointer to the first item in that `DATA` statement. If the specified statement is not a `DATA` statement, the computer resets the `READ` data pointer to the first item in the first `DATA` statement that follows the specified statement.

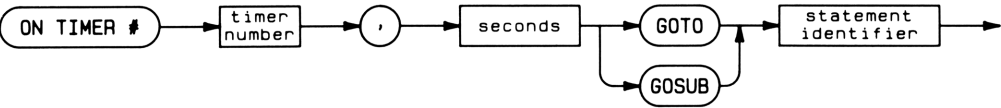
Related Keywords

`DATA.`

ON TIMER

ON TIMER # enables one of three individual timers to interrupt a program at the specified time interval and cause the specified branching to occur.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☐ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

```
ON TIMER #1,10 GOSUB 2000
ON TIMER #N,I GOSUB "TIMER"
ON TIMER #T*(X1-X2),I*7/32 GOTO T#
```

Input Parameters

Item	Description	Restrictions
timer number	Numeric expression rounded to an integer.	1 through 3.
seconds	Numeric expression.	1/32 second through 134,217,727 seconds.
statement identifier	Line number or label of a program statement.	Any valid line number or label reference.

ON TIMER # (continued)

Comments

The timer number must be either 1, 2, or 3. The time interval must be specified in seconds. Time intervals less than the minimum or greater than the maximum are set to the minimum or maximum, respectively.

When the HP-71 executes `ON TIMER #`, the appropriate timer is activated. When that timer expires, program execution transfers to the specified line. Where more than one timer has expired, the expired timers are serviced in numeric order. If the timer expires during execution of a program statement, the transfer does not occur until the computer has completed execution of that statement.

Executing the `OFF TIMER #` statement deactivates the timer previously activated by the corresponding `ON TIMER #` statement. Any of the following operations deactivates all three timers:

- Executing `END`, `END ALL`, or `STOP`.
- Executing the last statement of a program.
- Editing a program.
- Running a program.

Timer Subroutines. When the timer set by an `ON TIMER ... GOSUB` statement expires, program execution transfers to the indicated statement and continues. The computer terminates the subroutine when it executes a `RETURN` statement. Execution then returns to the statement following the last statement executed before the timer interruption occurred, and resumes. Simultaneously, the subject timer is reset to the time interval you originally specified in the `ON TIMER ... GOSUB` statement.

Timer Branches. When the timer set by an `ON TIMER ... GOTO` statement expires:

- Program execution transfers to the indicated statement and continues.
- The subject timer is immediately reset.

Timers continue to run after a program is suspended, but the interrupt does not cause the specified branching until the program is resumed.

Timer Operating Details. If a `BYE` or `OFF` statement within a program turns off the HP-71 after a timer has been activated, when that timer expires, the computer turns itself on and:

1. Executes the subroutine or branch specified by the timer.
2. When `RETURN` is encountered after `ON TIMER # ... GOSUB` execution, the computer resumes program execution with the first statement following the `BYE` or `OFF` statement.

ON TIMER # (continued)

Timers are global. Thus, after a program has called a subprogram, any timers activated by the calling program and expiring during execution of the subprogram are not acted upon until execution returns to the calling program. Also, a timer activated within a subprogram supersedes any other timer that was activated earlier by another program.

Related Keywords

OFF TIMER #.

OPTION ANGLE/BASE/ROUND

OPTION ANGLE selects the global unit of measure for expressing angles. OPTION BASE specifies the subscript lower bound(s) for subsequently dimensioned arrays. OPTION ROUND selects the roundoff setting.

■ Statement

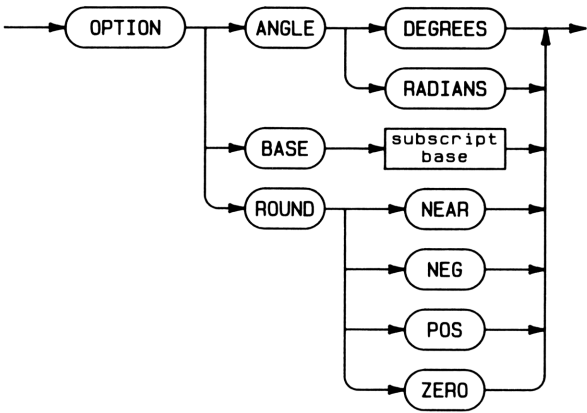
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

`OPTION ANGLE DEGREES
OPTION ROUND NEAR
OPTION BASE 0
OPTION ROUND NEG`

`OPTION BASE B
OPTION ANGLE RADIANS
OPTION ROUND POS
OPTION ROUND ZERO`

Input Parameters

Item	Description	Restrictions
subscript base	Numeric expression rounded to an integer.	0 or 1.

OPTION ANGLE/BASE/ROUND (continued)

Comments

Any `OPTION BASE` setting has a global effect. That is, an `OPTION BASE` setting has the same effect regardless of whether it is executed in a program or subprogram, or from the keyboard.

The `OPTION ANGLE` Statement. When executed subsequent to `OPTION ANGLE`, all functions that return an angle do so in the specified units, and all subsequently executed operations that use parameters representing angles will interpret angles in the specified units.

Selecting radians setting turns on the **RAD** annunciator and sets system flag `-10`. Selecting degrees setting clears the **RAD** annunciator and clears system flag `-10`.

The `OPTION BASE` Statement. Any array you create or redimension after executing `OPTION BASE` will have the subscript lower bound(s) specified in that `OPTION BASE` statement. However, subscript lower bounds for previously dimensioned arrays will not be changed. That is, changing the `OPTION BASE` setting after creating an array does not change the array's option base. (The lowest-numbered element in any array has a subscript of either 0 or 1, depending upon the current `OPTION BASE` setting.)

Executing `OPTION BASE 0` clears flag `-16`. Executing `OPTION BASE 1` sets flag `-16`.

The `OPTION ROUND` Statement. Executing `OPTION ROUND NEAR` selects rounding to the nearest machine value. Where there is a choice between two values, the even value is selected. `OPTION ROUND POS` selects rounding upward (towards $+\infty$). `OPTION ROUND NEG` selects rounding downward (towards $-\infty$). `OPTION ROUND ZERO` selects rounding towards zero (rounds the absolute value downward).

`OPTION ROUND` sets or clears system flags `-11` and `-12` as shown in the table to the right.

OPTION ROUND	Flag - 11	Flag - 12
NEAR	Clear	Clear
ZERO	Clear	Set
POS	Set	Clear
NEG	Set	Set

Related Keywords

- `OPTION ANGLE`: DEGREES, FLAG, RADIANS.
- `OPTION BASE`: DIM, FLAG, INTEGER, REAL, SHORT.
- `OPTION ROUND`: FLAG.

OR

The `OR` binary operator performs the logical OR of its two operands.

☐ Statement

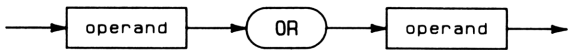
☐ Function

☒ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
IF Z=5 OR A=C THEN GOSUB 100      Q=R OR S
```

Input Parameters

Item	Description	Restrictions
operand	Numeric expression.	Subject to operator precedence.

Comments

Operands used with `OR` are considered logically false if zero, and logically true if nonzero. The table to the right indicates the range of results for `OR`.

`OR` and `EXOR` have the same level of precedence, which is the lowest of all HP-71 operators. (Refer to “Precedence of Operators” on page 317.)

Operand		Result
Left	Right	
False	False	0
False	True	1
True	False	1
True	True	1

Related Keywords

`AND`, `EXOR`, `NOT`.

OVF

OVF (*overflow*) returns the number of the overflow flag (−6).

- | | |
|--|--|
| <input type="checkbox"/> Statement | <input checked="" type="checkbox"/> Keyboard Execution |
| <input checked="" type="checkbox"/> Function | <input checked="" type="checkbox"/> CALC Mode |
| <input type="checkbox"/> Operator | <input checked="" type="checkbox"/> IF ... THEN ... ELSE |



Examples

```
IF FLAG(OVF) THEN GOSUB OVRFLW
A=TRAP(OVF,2)
```

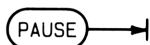
Related Keywords

CFLAG, DEFAULT, DVZ, FLAG, INX, IVL, SFLAG, TRAP, UNF.

PAUSE

`PAUSE` suspends program execution.

- | | |
|---|--|
| <input checked="" type="checkbox"/> Statement | <input type="checkbox"/> Keyboard Execution |
| <input type="checkbox"/> Function | <input type="checkbox"/> CALC Mode |
| <input type="checkbox"/> Operator | <input checked="" type="checkbox"/> IF ... THEN ... ELSE |



Examples

```
PAUSE
```

Comments

In a running program, `PAUSE` suspends program execution. Subsequently executing `CONT` or pressing either the `CONT` key or the `SST` key resumes execution at the statement immediately following the `PAUSE` statement.

Executing `END`, `STOP`, or `END ALL`, running a program, or editing a program clears the “suspended” status (that is, releases local environments and clears all program control information maintained in the computer).

Note: A suspended program retains any memory allocations it has already established for local variables, pending subprogram calls, subroutine levels, `GOSUB` statements and `FOR ... NEXT` statements.

Attempting to execute `PAUSE` from the keyboard results in an `Illegal Context` (error 79) condition.

Related Keywords

`CONT`, `END`, `STOP`, and the `ATTN` and `SST` keys.

PEEK\$

PEEK\$ returns the contents of a specified section of memory.

☐ Statement

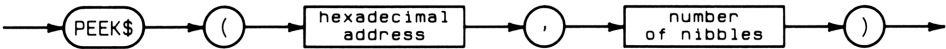
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

A\$ = PEEK\$("AF30",6)

DISP PEEK\$(A\$,I)

Input Parameters

Item	Description	Restrictions
hexadecimal address	String expression containing hexadecimal digits.	Up to five uppercase or lowercase digits.
number of nibbles	Numeric expression rounded to an integer.	0 through 524,287.

Comments

Specifying memory that is located in a private file results in a File Protect (error 61) condition.

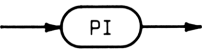
Related Keywords

ADDR\$, DTH\$, HTD, POKE.

PI

PI returns a 12-digit value (3.14159265359) representing π .

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
C=PI*D
FOR X=0 TO PI STEP 0.1 @ PRINT SIN(X) @ NEXT X
```

PLIST

PLIST displays the specified BASIC or KEY file on the print device.

■ Statement

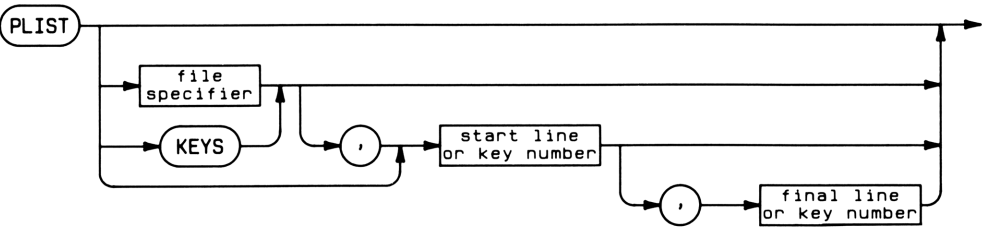
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
PLIST
PLIST KEYS
```

```
PLIST CAT$(I),10,99
PLIST 100,500
```

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default: Current file.	File name with optional device specifier.
start line or key number	Integer constant identifying a program line or key number. Default: First program line or key assignment in file.	1 through 9999.
final line or key number	Integer constant identifying a program line or a key number. Default: Start line or key number, if specified; otherwise, last program line or key assignment in file.	Start line or key number through 9999.

PLIST (continued)

Comments

Operation. The `PLIST` operation is identical to that of `LIST`, except the output goes to the print device instead of to the display device.

Specifying a file that is not a `BASIC` or `KEY` file generates an `Invalid File Type` (error 63) condition.

Executing `PLIST KEYS` results in a listing of the current key assignment file, `keys`.

Specifying a single line or key parameter lists only that line or key assignment. If you specify a range and the HP-71 does not find the start line or key number, but does find a higher-numbered line or key number within the specified range, the listing begins with that higher-numbered line or key. Executing `PLIST` without specifying any line or key numbers generates a listing of the entire file.

If the print device is the display, the current `DELAY` setting determines how long the computer displays each line.

The current `PWIDTH` setting determines the width of the printed line.

Interrupting a Listing. To halt a listing and display the cursor, simply press ATTN.

Related Keywords

`DELAY`, `ENDLINE`, `LIST`, `PWIDTH`.

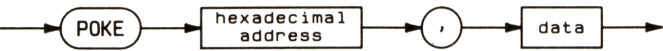
POKE

POKE writes to memory at the specified hexadecimal address.

CAUTION

Executing POKE can cause a Memory Lost condition. Also, executing POKE for some areas of memory can result in a condition from which the only methods of recovery are either to execute INIT3 or to remove the batteries. INIT3 destroys all files in main RAM. Removing the batteries causes a loss of memory in all of RAM.

- Statement
- Function
- Operator
- Keyboard Execution
- CALC Mode
- IF ... THEN ... ELSE



Examples

```
POKE "AF30","56DE"      POKE A$,D$
```

Input Parameters

Item	Description	Restrictions
hexadecimal address	String expression containing hexadecimal digits.	Up to five uppercase or lowercase digits.
data	String expression containing hexadecimal digits.	No restriction on number of characters in data.

Comments

If there is a nonhexadecimal digit in the data string, POKE writes to memory all digits preceding the nonhexadecimal digit, then generates an Invalid Arg (error 11) condition.

The amount of data written is limited only by the amount of available memory. Attempting to use POKE within a secure or private file results in a File Protect (error 61) condition.

POKE (continued)

Related Keywords

ADDR\$, DTH\$, HTD, PEEK\$.

POP cancels the pending return of program execution from the current subroutine.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
POP
```

Comments

POP is similar to RETURN except that it causes program execution to continue at the instruction following the POP statement instead of at the instruction following the calling GOSUB statement. (That is, POP cancels the pending return established by the last GOSUB statement.)

Related Keywords

```
GOSUB, RETURN.
```

POS

POS (*substring position*) returns the position of a given substring within a string.

☐ Statement

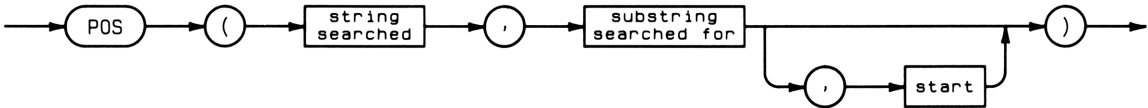
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
POS("WARTHOG","HOG")
IF POS(A$,B$)=5 THEN GOSUB 100
POS("MISSISSIPPI","I",3)
```

Input Parameters

Item	Description	Restrictions
string searched	String expression. Numeric expression rounded to an integer. Default: 1.	None. If the rounded value is less than or equal to zero, POS used the default value of "1."
substring searched for		
start character number		

Comments

POS searches within the first string for the second string. If the second string is found within the first, the starting character position of the second string is returned.

If the search is to start at a position other than the first character, use the optional third parameter to specify the starting position. In the above examples, POS("MISSISSIPPI","I",3) returns 5, which represents the result of a search beginning at character 3 (S) for the character I.

POS (continued)

POS returns zero when the substring cannot be found. For example:

- `POS("WARTHOG", "HOG")` returns 5.
- `POS("WARTHOG", "PIG")` returns 0.

PREDV

PREDV (*predicted value*) returns the predicted value (based upon the current statistical array) of the dependent variable, the last LR statement, and the value of the independent variable specified as the argument.

☐ Statement

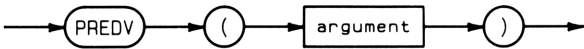
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
Y = PREDV(X)
PRINT "Prediction = ";PREDV(X)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

PREDV must be preceded by execution of LR to specify the dependent and independent variables.

Related Keywords

ADD, CLSTAT, CORR, DROP, LR, MEAN, SDEV, STAT, TOTAL.

PRINT

PRINT causes the print items to be sent to the print device.

■ Statement

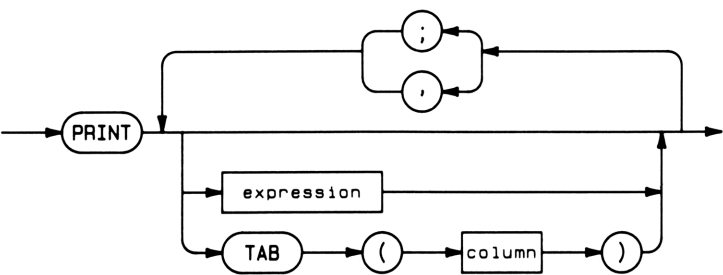
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
PRINT I;TAB(I+5);"*"  
PRINT "Hello ";A$;".","Welcome"  
PRINT
```

Input Parameters

Item	Description	Restrictions
expression	Numeric or string expression.	None.
column	Numeric expression rounded to an integer.	Greater than zero.

PRINT (continued)

Comments

Operation. PRINT operates in a manner similar to that of DISP. However, PRINT is not affected by WIDTH statements (and, conversely, DISP is not affected by PWIDTH statements).

You can use ENDLINE to alter the string that PRINT appends to the end of each print line. (The default string is the carriage return/line feed.)

Use of TAB. TAB positions PRINT (and DISP) output to begin at the column you specify. If the current column position is beyond the specified TAB column, the computer first moves to the next line, then positions itself to the specified column. If the column position value exceeds the current line width, the computer reduces the position value by a multiple of the line width (in a manner similar to the RED function), then moves to the reduced column position.

Related Keywords

DISP, ENDLINE, PRINT USING, PRINT #, PWIDTH.

PRINT USING

PRINT USING causes the print list to be sent to the print device in a user-specified image format.

■ Statement

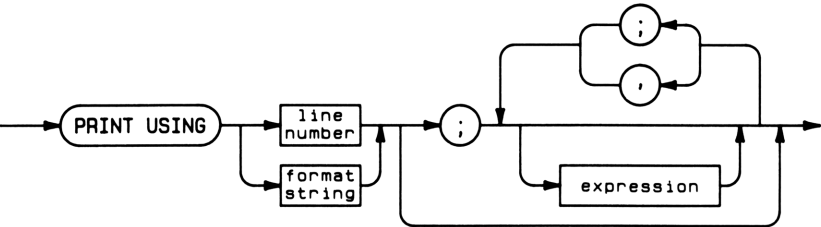
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
PRINT USING 200; X, 345.99
PRINT USING S$; A$, 1.25+X
PRINT USING "#,K,2(XX,3D.D)"; "Output=",Y,2.34E2
```

Input Parameters

Item	Description	Restrictions
line number	Integer constant identifying a program line.	1 through 9999.
format string	String expression	Refer to the IMAGE keyword entry.
expression	Numeric or string expression.	None.

PRINT USING (continued)

Comments

`PRINT USING` uses a format string to format output items. If there are no output items, there may or may not be any output to the printer, depending on the items in the format string.

If `PRINT USING` references a line number, an `IMAGE` statement must be the first statement in that line. When executed from the keyboard, the computer searches for an `IMAGE` statement at the referenced line number in the *current* file.

If `PRINT USING` contains a string expression for the image, that expression must evaluate to a valid format string, as described in the **IMAGE** keyword entry. For example:

```
PRINT USING "2X,'hello',X,8A,'!'; N$
PRINT USING '8A,5X,"$"3D.DD'; "Profit=", P
```

The following program provides a further `PRINT USING` illustration:

```
10 S$= '2X," & "Today's" & "',8A,"$"2D.DD'
20 PRINT USING S$; " Special",2.95
    Today's Special $2.95
```

Uses string expressions for images.

References string expression.

Result of executing lines 10 and 20.

The items in an output list must be separated by commas or semicolons. However, unlike `PRINT`, these punctuation characters have no further meaning in `PRINT USING` statements.

Related Keywords

`DISP USING`, `IMAGE`, `PRINT`, `PWIDTH`.

PRINT

PRINT# writes data items to a data file associated with a specified channel number.

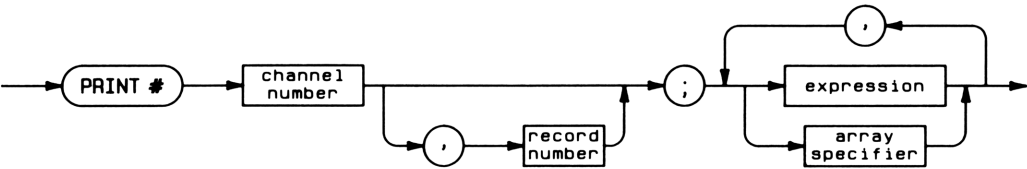
- Statement

□ Function

□ Operator
- Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
PRINT #3; A,B$,X+Y,"SUNDAY"           PRINT #18,20;A,B$,C
PRINT #F; A1(,),D$[3,39],G#()
```

Input Parameters

Item	Description	Restrictions
channel number	Numeric expression rounded to an integer.	0 through 255.
record number	Numeric expression rounded to an integer. The first record of the file is record 0.	0 through 1,048,575.
expression	Numeric or string expression.	None.
array specifier	Array name followed by an empty set of parentheses <code>()</code> or <code>(,)</code> which indicates that the entire array is to be written to the file.	None.

PRINT # (continued)

Comments

`PRINT #` writes data items to the file assigned to the specified channel number. The file type can be `DATA`, `TEXT`, or `SDATA`.

If you do not specify a record number, the data items are sequentially written to the specified file (termed *sequential access*), beginning with the current position of the data pointer within the file. The HP-71 automatically increases the file size if the end of the file is reached before all data is written to the file. If you specify a record number, the file pointer moves to that record and the data items are written. This process is termed *random access*, and can be used with all *data* file types except `TEXT` files. (Attempting to randomly access a `TEXT` file results in an `Invalid File Type` (error 63) condition. Also, for a file of type `DATA` or `SDATA`, specifying a record number greater than the last record number in the file generates an `End of File` (error 54) condition.)

The format of numeric and string values depends upon the types of files to which they are written.

For information concerning the amount of memory required for various HP-71 file types, refer to the information listed under “Files” in the “System Memory Requirements” table that begins on page 330.

TEXT Files. Every data item written to a text file forms an individual record. The computer writes an end-of-file mark when it completes execution of each `PRINT #` statement. (Only sequential access is allowed on a `TEXT` file—a record number must *not* be specified.) The computer converts numeric data to text form—according to the current display format—before writing the data to the file.

DATA Files. `PRINT #` uses eight bytes of memory to write each numeric value to the file, and uses three bytes plus one byte per character for each string. Sequential access starts at the current file position, may cross record boundaries, and writes an end-of-file mark after the last data item written. Random access writes data items to the specified record number, then writes an end-of-record mark. (Random access does not cross record boundaries.) Thus, for random access, all of the data must fit into one record. Otherwise, a `Record Ovfl` (error 29) condition results.

SDATA Files. You can write only numeric data to an `SDATA` file. Each value uses eight bytes of memory.

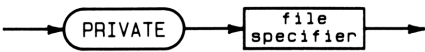
Related Keywords

`ASSIGN #`, `CREATE`, `READ #`, `RESTORE #`.

PRIVATE

PRIVATE limits access to the specified file and restricts changes in its protection.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

```
PRIVATE A:PORT(3)      PRIVATE "BCD"      PRIVATE A$
```

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string.	File name with optional device specifier.

Comments

A private file may not be altered, listed, or otherwise read. This means that you cannot use it as the destination file in a MERGE statement or otherwise modify it. Because of the permanent results of executing PRIVATE, this statement requires that you always specify the name of the file you want to make private.

Note: As an added precaution, only an *unsecured* file can be made private.

You can execute and purge an unsecured file that is private. The catalog listing for an unsecured private file appears with a P in the protection field. A secured private file can only be executed. In a catalog listing, a secured private file appears with an E in the protection field. Once you make a file private, the file remains private. Thus, its protection can only oscillate between the protection types P and E. Also, you can designate as private, only the files that you can execute. That is, only executable BASIC and BIN (binary) files can be designated as private.

Related Keywords

SECURE, UNSECURE.

PROTECT

PROTECT write-protects one track of a magnetic card.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF . . . THEN . . . ELSE



Examples

```
PROTECT
```

Comments

PROTECT is available only when the HP 82400A Card Reader is installed in your HP-71. When you execute PROTECT, the computer prompts you for a card and then allows you to pull the card through the card reader once. This operation, which places a write-protect code on the track accessed by the card reader, protects that track from being overwritten by a subsequent COPY operation. To protect the card's other track, turn the card around, reexecute PROTECT, and again pull the card through the card reader.

When a track has been protected, you cannot write over it unless you first execute an UNPROTECT operation for that track.

Related Keywords

```
UNPROTECT.
```

PURGE

PURGE deletes a file (or files) from RAM.

■ Statement

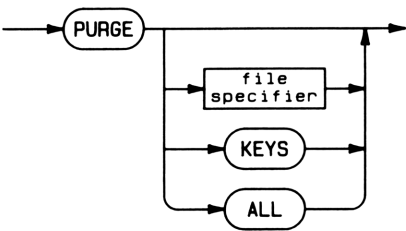
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF . . . THEN . . . ELSE



Examples

```
PURGE "TESTFILE:PORT"
PURGE KEYS
PURGE "TQ"
```

```
PURGE ALL
PURGE STRESS7
PURGE
```

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default: Current file.	File name with optional device specifier.

Comments

After you purge a file, you cannot access the information it contained. If you purge the *current* file, `workfile` becomes the current file.

PURGE (continued)

Purging the current file during program execution:

1. Halts the program.
2. Clears the program from memory.
3. Releases local environments.
4. Clears all internally-maintained program control information.
5. Designates `workfile` as the current file.

Attempting to purge the current file when (1) the `workfile` does not exist, and (2) there is not enough memory to create the `workfile` results in the `Insufficient Memory (error 24)` condition.

Executing `PURGE ALL` clears all unsecured files from main RAM. Executing `PURGE KEYS` purges the system `keys` file.

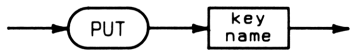
Related Keywords

`SECURE`, `UNSECURE`.

PUT

PUT enters a key code (specified by the key's string argument) into the key buffer.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

PUT A\$

PUT "fQ"

PUT "#43"

Input Parameters

Item	Description	Restrictions
key name	String Expression.	Less than five characters.

Comments

This statement adds the specified key to the end of the list of keys currently held in the buffer. (The buffer can hold up to 15 keys.)

The key name is specified in the same format as that for DEF KEY. If the buffer is full, executing PUT leaves the buffer unchanged.

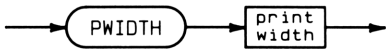
Related Keywords

DEF KEY, KEY\$.

PWIDTH

PWIDTH (*print width*) defines the line length of PRINT statements.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

```
PWIDTH INF          PWIDTH 20
```

Input Parameters

Item	Description	Restrictions
print width	Numeric expression rounded to an integer.	PWIDTH interprets a value less than 1 as 1, and a value greater than 255 as infinity.

Comments

PWIDTH specifies a line width that is used during execution of subsequent PRINT statements to determine the output format.

Using TAB in PRINT statements reduces the TAB argument modulo the PWIDTH argument.

Related Keywords

PLIST, PRINT, WIDTH.

RAD

`RAD` (*degrees to radians conversion*) converts arguments expressed in degrees to radians.

☐ Statement

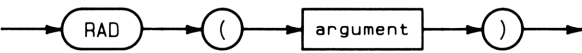
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
RAD(X+Y)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

The conversion constant is accurate to 15 digits, which often produces more accurate results than a conversion that does not use `RAD`.

Related Keywords

```
DEG, DEGREES, RADIANS.
```

RADIANS

RADIANS selects Radians mode, which specifies radians as the unit of measure for expressing angles. It is a short form of the OPTION ANGLE RADIANS statement.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
RADIANS                                IF A=1 THEN RADIANS
```

Comments

RADIANS turns on the RAD annunciator.

After executing RADIANS:

- Any function that returns an angle does so in radian units.
- Instructions that use angles for parameters will interpret such parameters in radian units.

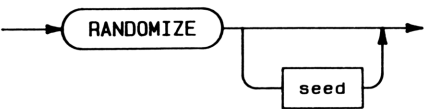
Related Keywords

DEGREES, OPTION.

RANDOMIZE

RANDOMIZE specifies a seed for the RND function.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF . . . THEN . . . ELSE



Examples

RANDOMIZE RANDOMIZE R

Input Parameters

Item	Description	Restrictions
seed	Numeric expression. Default: Current value in clock register.	None.

Comments

The random number series generated by executions of the RND function depends upon the starting seed value you specify with RANDOMIZE. If you specify 0 for a seed, the subsequent RND sequence will consist of all zeroes. Otherwise, RANDOMIZE uses the given seed’s absolute value or the contents of a clock register. RANDOMIZE is a global declaration. That is, executing RANDOMIZE in a subprogram affects the subsequent RND series in the calling program.

Related Keywords

RND.

READ

READ assigns values from DATA statements to variables.

■ Statement

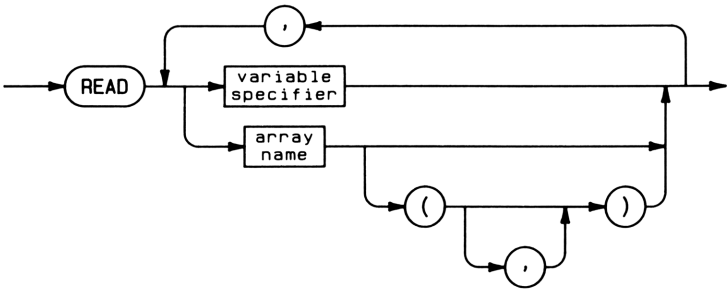
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

READ A,B,X(A)

READ Z\$[3,3],C

READ A(,),B()

Input Parameters

Item	Description	Restrictions
variable specifier	Numeric variable specifier or string variable specifier.	None.
array name	Name of a numeric or string array. Indicates that the entire array is to be read.	The number of dimensions must match the empty subscripts that follow, if specified.

READ (continued)

Comments

The numeric items stored in `DATA` statements are read into the numeric variables in a `READ` statement. If they are not in the correct form (that is, if they are not valid numeric expressions) then an error results. A string variable may read what appears to be a numeric item, as long as it is dimensioned large enough to contain the characters. When the current program context changes due to execution of `RUN` or `CALL`, the `DATA` pointer is reset. The first `READ` statement in a program accesses the first item in the first `DATA` statement in that program unless you used `RESTORE` to specify a different `DATA` statement as the starting point. Successive `READ` operations access successive `DATA` items, progressing sequentially through the `DATA` statements as necessary. Trying to read past the end of the last `DATA` statement causes an error. The `RESTORE` statement may be used to alter the order in which `READ` accesses `DATA` statements.

A valid `DATA` expression may contain any function calls or variable references that are allowed in any other expression. This feature may be useful for reading computed constants or special characters that are normally unavailable from the keyboard, such as `DATA SQR(2*PI)` and `CHR$(27)`.

Related Keywords

`DATA`, `READ#`, `RESTORE`.

READ

READ # reads data items from a data file associated with a specified channel number.

■ Statement

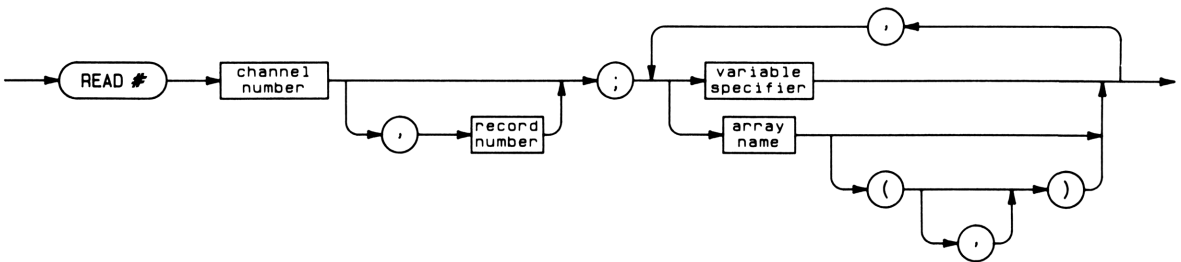
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
READ #3;0,R,D$,P(),S1$(  
READ #7,20;I,A(I),X
```

```
READ #H;Z(,)
```

Input Parameters

Item	Description	Restrictions
channel number	Numeric expression rounded to an integer.	1 through 255.
record number	Numeric expression rounded to an integer. The first record of the file is record 0.	0 through 1,048,575.
variable specifier	Numeric variable specifier or string variable specifier.	None.
array name	Name of a numeric or string array. Indicates that the entire array is to be read.	The number of dimensions must match the empty subscripts that follow, if specified.

READ # (continued)

Comments

READ # reads data items from the file assigned to the specified channel number. The file's type can be DATA, TEXT, or SDATA. The data type of each item in the **READ #** read list must match the type of corresponding data item in the data file.

If you specify a record number, the file is positioned to that record and the data items are read. This is termed *random access*. For a file of type DATA or SDATA, specifying a record number greater than the number of records in the file causes an **End of File** (error 54) condition.

If you do not specify a record number, the computer begins reading data items from the data pointer's current position in the file. This is termed *sequential access*.

TEXT Files. Every record is assumed to contain a single data item represented as a string of characters. If a data item is to be read into a numeric variable, the computer attempts to evaluate the ASCII string as a numeric expression in a manner similar to that of the **VAL** function. If the specified record number is greater than the number of records in the file, the file pointer is set to the end of the file.

DATA Files. For random access, the number of items in the read list must not exceed the number of data items in the record.

SDATA Files. Each record of this file type can contain a single numeric or string data item. Note that strings are limited to six characters.

Related Keywords

ASSIGN #, ASSIGN, CREATE, PRINT #, RESTORE #.

REAL

REAL allocates memory for real variables and arrays.

■ Statement

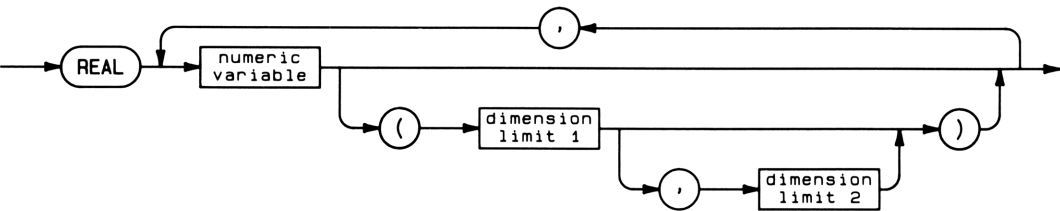
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

REAL X

REAL 04(7,3)

REAL N, P(4,4)

Input Parameters

Item	Description	Restrictions
numeric variable	Letter followed by an optional digit.	None.
dimension limit 1	Numeric expression rounded to an integer.	Current OPTION BASE setting through 65535.
dimension limit 2		

Comments

REAL creates real variables and arrays. Creation occurs upon execution of REAL. The dimension limits are evaluated at creation time. The lowest-numbered subscript in any dimension is 0 or 1, depending on the OPTION BASE setting when the array is created. All elements are initialized to zero.

REAL (continued)

If `REAL` specifies a simple numeric variable that already exists, the variable is reinitialized to zero. Array variables are redimensioned, but not reinitialized to zero (unless the data type is changed). If `REAL` expands an array, it also initializes all newly-created elements in the array. Notice that redimensioning does not necessarily preserve an element's *position* within an array, but does preserve the *sequence* of elements within an array. (Refer to “Declaring Arrays (`DIM`, `REAL`, `SHORT`, `INTEGER`)” in section 3 of your *HP-71 Owner's Manual*.)

The following table indicates the conditions that apply to `REAL` variables and arrays:

REAL Numeric Variables

Initial Value	0
Numeric Precision	12 Decimal Digits
Exponent Range	± 499
Maximum No. of Array Dimensions	2
Maximum Dimension Limit	65535
Memory Usage in Bytes	
• Simple Variable	9.5
• Array	$8 * (dim1 - base + 1) * (dim2 - base + 1) + 9.5$

Related Keywords

`DIM`, `INTEGER`, `OPTION BASE`, `SHORT`, `DESTROY`.

RED

RED(*x*, *y*) (*reduction*) returns a remainder defined by the expression

$$x - y * n$$

where *n* is the nearest integer to *x*/*y*. If *x*/*y* lies exactly between two integers, RED uses the even integer.

☐ Statement

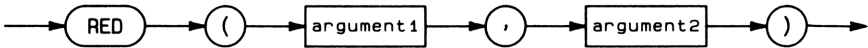
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A=RED(X,360)
```

```
IF RED(K,3)=1 THEN PRINT K
```

Input Parameters

Item	Description	Restriction
argument1 argument2	Numeric expression.	Refer to “Comments,” on the facing page.

RED (continued)

Comments

RED is the remainder function defined by the IEEE Floating Point Standard.* RED has the following properties:

- $\text{RED}(x, y)$ is periodic in x with period $|y|$, except at multiples of y , where the period is $2|y|$.
- $\text{RED}(x, y)$ lies in the interval $[-|y|/2, |y|/2]$.
- RED generates exact answers (that is, no rounding is necessary for the 12-digit result).

RED is useful for angle reductions, since

$$-180 \leq \text{RED}(x, 360) \leq 180$$

and the result is exact. For example, $\text{RED}(-1\text{E}-20, 360) = -1\text{E}-20$, exactly.

If argument1 is Inf or argument2 is equal to zero, the Invalid Arg (error 11) condition results.

Related Keywords

MOD, RMD.

* The remainder function is referred to as *REM* in the IEEE Floating Point Standard.

REM

REM (*remarks*) enables you to document your programs with comments.

■ Statement

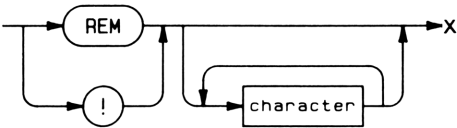
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
REM Program to compute average speed
! Program to compute average speed
IF A THEN GOSUB 750 ! Subroutine for depth factor.
```

Input Parameters

Item	Description	Restrictions
character	Any character.	None.

Comments

When program execution encounters REM, it assumes that any remaining information in that line consists of remarks, and not executable keywords. Thus, in the following example, the @ DISP B\$ is ignored:

```
10 IF B$=C$ THEN PURGE B$ @ REM PURGED B$ @ DISP C$
```

When you want to enter a comment in a program line without using the @ REM... construction, use the ! exclamation point. (Wherever REM is allowed, ! is also allowed. Notice, however, that ! does not require use of the @ character.)

RENAME

RENAME changes the name of a file.

■ Statement

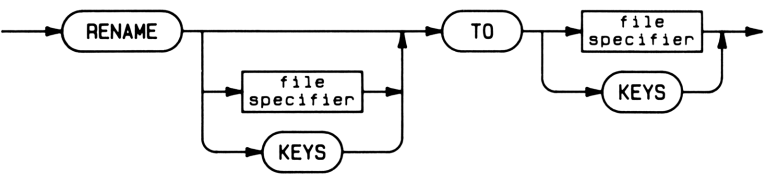
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF... THEN... ELSE



Examples

RENAME TEST1:PORT(2) TO TEST2

RENAME TO STRESS6

RENAME KEYS TO "KEYS"

RENAME KEYF32 TO KEYS

RENAME A\$ TO B\$

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default: Current file.	File name with optional device specifier. Proposed file name must not already exist on the device containing the designated file.

Comments

You can use `RENAME TO file name` to name the `workfile` when the `workfile` is the current file.

`RENAME KEYS TO file name` renames the current key assignment file, which deactivates any active key assignments. `RENAME file name TO KEYS` designates the specified `KEY` file to become the current key assignment file. If the specified file is not in main RAM, an error occurs.

RENAME (continued)

A device specifier for the file you rename may be given in either file specifier (that is, before or after the `TO`). If both file specifiers include device specifiers, then the device in the first file specifier (whether implicit or explicit) is always interpreted as the device containing the file to be renamed. In such cases the second device specifier is ignored. An example of an implicit device specifier is:

```
RENAME TO A:PORT
```

The preceding example implicitly specifies the device containing the current file. Thus, `RENAME` ignores the `:PORT` device specifier.

RENUMBER

RENUMBER rennumbers the lines in a program file.

■ Statement

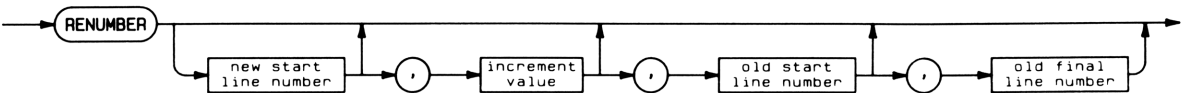
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

RENUMBER

RENUMBER 10,5

RENUMBER 300,10,100

Input Parameters

Item	Description	Restrictions
new start line number	Integer constant. Default: 10.	1 through 9999.
increment value	Integer constant. Default: 10.	1 through 9999.
old start line number	Integer constant. Default: Start of file.	1 through 9999.
old final line number	Integer constant. Default: End of file.	1 through 9999.

RENUMBER (continued)

Comments

RENUMBER operations are subject to the following:

- If the HP-71 cannot find the *old start line number*, it locates the next successive line number and begins renumbering from that line.
- If, during renumbering, RENUMBER reaches line 9999 before the entire program is renumbered, the computer automatically renumbers the program lines using 1 for the increment between lines. In this case, if the new start line number was not specified, it defaults to 1.
- If a renumbered line is referenced by a program statement such as GOTO or GOSUB, the computer automatically changes that reference to reflect the new line number.

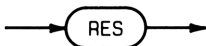
Note: Where RUN, LIST, or RENUMBER are program statements referring to specific line numbers, any execution of RENUMBER does not adjust those references to correspond to the new line numberings.

RES

RES (*result*) returns the value of the most recently executed numeric expression.

- ☐ Statement
- ☒ Function
- ☐ Operator

- ☒ Keyboard Execution
- ☒ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

```
SIN(RES)
```

```
RES^2+(1-RES)^2
```

```
IF RES > .5 THEN DISP M$
```

Comments

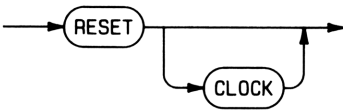
The HP-71 stores the result of each numeric assignment and calculator statement in a register reserved for use by **RES**. This occurs prior to any rounding that may take place due to a **SHORT** or **INTEGER** destination variable. **RES** recalls the value currently stored in this register. **RES** can be particularly useful in iterative computations because it is much faster than variable references.

In **CALC Mode**, the computer implicitly executes **RES** whenever a null subexpression is completed. Thus, **SIN()** returns the sine of the previous result, and **MAX(,A)** returns the larger of the previous result and variable **A**.

RESET, RESET CLOCK

RESET resets user and system flags to their default settings. RESET CLOCK nullifies the effect of executing EXACT.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
RESET                                RESET CLOCK
```

Comments

The RESET Statement. RESET clears all user flags (0 through 63) and the system flags that can be set or cleared by users (flags -1 through -32). Reset also sets all math traps to their DEFAULT ON settings.

The RESET CLOCK Statement. RESET CLOCK clears the Exact flag (flag -46) and the system clock's adjustment factor.

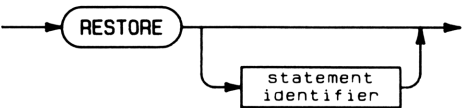
Related Keywords

```
RESET: CFLAG, SFLAG.  
RESET CLOCK: AF, EXACT.
```


RESTORE

RESTORE specifies which DATA statement will be used by the next READ operation.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

RESTORE

RESTORE 100

RESTORE ARRAY2

Input Parameters

Item	Description	Restrictions
statement identifier	Line number or label of a program statement. Default: First DATA statement in the program.	Any valid line number or label reference.

Comments

If RESTORE specifies a line that does not contain a DATA statement, the computer uses the first DATA statement following the specified line. RESTORE can only refer to lines within the current program. If no line number is specified, the next READ statement uses the first DATA statement in the current program or subprogram.

Related Keywords

DATA, READ.

RESTORE

RESTORE # sets the file pointer associated with the specified channel number to the indicated record number.

■ Statement

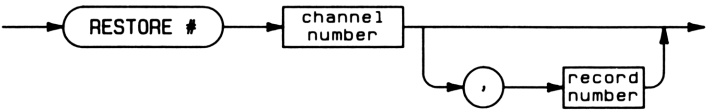
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

RESTORE #2

RESTORE #3,20

RESTORE #1,99999

Input Parameters

Item	Description	Restrictions
channel number	Numeric expression rounded to an integer.	1 through 255.
record number	Numeric expression rounded to an integer. The first record of the file is record 0. Default: Beginning of the file.	0 thru 1,048,575.

Comments

For a file of type DATA or SDATA, specifying a record number greater than the number of records in the file results in an End of File (error 54) condition. For a TEXT file, if the record number does not exist, RESTORE # sets the file pointer to the end of the file, which causes the next record written to the file to be appended to the current end of the file.

Related Keywords

ASSIGN #, PRINT #, READ #.

RETURN

`RETURN` returns program execution to the statement following the invoking `GOSUB`.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

Program Segment:

```
300 A = B
310 GOSUB 700
320 DISP A
. .
. .
. .
700 DISP "B =";
710 INPUT B
720 RETURN
```

Comments

In the above example, the `GOSUB` on line 310 transfers execution to the subroutine beginning at line 700. The `RETURN` at line 720 terminates subroutine execution and transfers program execution to line 320.

Related Keywords

`GOSUB`, `POP`.

RMD

RMD(*x*,*y*) (*remainder*) returns the remainder defined by the expression $x - y * IF(x/y)$.

☐ Statement

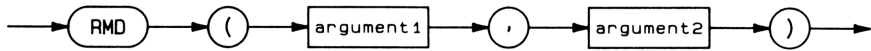
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
M=RMD(M,60)
```

Input Parameters

Item	Description	Restrictions
argument1 argument2	} Numeric expression.	Refer to Comments, below.

Comments

RMD is the remainder function defined by the ANSI BASIC Standard,* and has the following properties:

- RMD(*x*,*y*) is *not* periodic in *x* across $x = 0$.
- RMD(*x*,*y*) lies in the interval
 - $[0,|y|)$ for $x \geq 0$.
 - $(-|y|,0]$ for $x \leq 0$.
- For $x,y > 0$, $RMD(x,y) = MOD(x,y)$.
- Returns results that are exact. (RMD never sets the INX flag. No rounding needs to be done because the result is never greater than 12 digits.)

If either argument1 equals Inf or argument 2 equals zero, the Invalid Arg (error 11) condition results.

* The remainder function is referred to as REM in the ANSI BASIC Standard.

Related Keywords

DIV, MOD, RED.

RND

`RND` (*random number*) returns the next real number in a pseudo-random number sequence and updates the random number seed.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
R=(RND+RND)/2                                PRINT "random = ";RND
```

Comments

The HP-71 uses a linear congruential method and a seed value to generate a random number value (*r*), which always lies in the range $0 \leq r < 1$. Each succeeding execution of `RND` returns an *r*-value computed from a seed based upon the previous `RND` value.

Note: If the seed is zero, the `RND` sequence will likewise be zero.

You can change the seed by executing `RANDOMIZE`.

Related Keywords

```
RANDOMIZE.
```

RUN

RUN executes a BASIC or binary program.

■ Statement

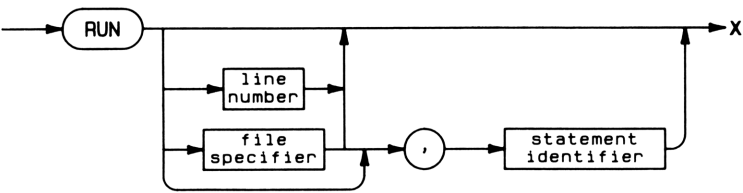
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

RUN

RUN ;CARD

RUN ,START5

RUN TEST3:PORT(1),3500

RUN 100

RUN PROG1

RUN ;CARD,30

RUN ,L\$

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default: Current file.	File name with optional device specifier or ;CARD.
line number	Integer constant.	1 to 9999.
statement identifier	Line number or label of a program statement.	Any valid line number or label reference.

RUN (continued)

Comments

If an external device contains the specified file, it is first copied into main RAM, then executed. When you enter a file specifier and execute `RUN`, that file becomes the current file.

If you enter a line number or label reference after a file specifier, `RUN` begins program execution at that line or statement. Otherwise, `RUN` begins execution at the first statement of the program. If the computer does not find the specified line number, but does find a higher-numbered line, `RUN` begins execution with that line. If you enter only a line number or label reference (and no file name), `RUN` begins executing the current program from the specified statement.

Note: The label reference *must* be preceded by a comma to distinguish it from a file specifier. The comma preceding the line number is optional.

Effect of RUN on Suspended or Running Programs. Executing `RUN` from within a running program causes the computer to halt the current program, switch to the specified program (which becomes the “current file”), and run the new program.

If you suspend a program (which turns on the **SUSP** annunciator), subsequently executing `RUN` turns off the **SUSP** annunciator, closes all files, releases local environments, clears all program control information with respect to the suspended program, and executes the specified program, as described above.

The `[RUN]` Key. This key is a direct execute key. It causes program execution to begin with the first statement of the current program.

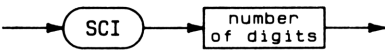
The Effect of RUN on Current Memory Status. `RUN` closes all files, releases local environments and clears all program control information associated with a prior suspended program. (This is equivalent to `END`.)

Related Keywords

`CALL`, `CHAIN`, `COPY`.

SCI (*scientific format*) sets the scientific display format (SCI mode) and the number of significant digits to be displayed (or printed).

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF ... THEN ... ELSE



Examples

```
SCI 0                                IF X < 1 THEN SCI 11
```

Input Parameters

Item	Description	Restrictions
number of digits	Numeric expression rounded to an integer.	0 through 11. SCI interprets a value less than 0 as 0, and a value greater than 11 as 11.

Comments

Display format statements control the format setting for displaying numbers. The display setting remains in effect until you execute another SCI, FIX, ENG, or STD statement.

SCI display format displays values in scientific notation to $d + 1$ significant digits, where d is the specified number of digits. The value appears as

(Sign) Mantissa E (Sign) Exponent

where, for normalized numbers,

$1 \leq \text{Mantissa} < 10$

and, for denormalized numbers,

$\text{Mantissa} < 1.$

SCI (continued)

Related Keywords

ENG, FIX, STD, STR\$.

SDEV

SDEV (*standard deviation*) returns the sample standard deviation for the specified variable in the current statistical array.

☐ Statement

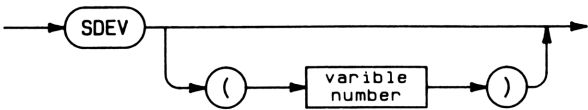
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A1=SDEV(X1)
PRINT "Standard Deviation = ";SDEV
```

Input Parameters

Item	Description	Restrictions
variable number	Numeric expression rounded to an integer. Default: 1.	Zero through the current STAT array dimension.

Comments

The sample standard deviation calculation uses $n - 1$ as the denominator, where n is the sample size. For information concerning statistical arrays, refer to the “Mathematical Discussion of HP-71 Statistical Arrays,” page 334.

Related Keywords

ADD, CLSTAT, CORR, DROP, LR, MEAN, PREDV, STAT, TOTAL.

SECURE

SECURE protects a file from being altered or purged.

■ Statement

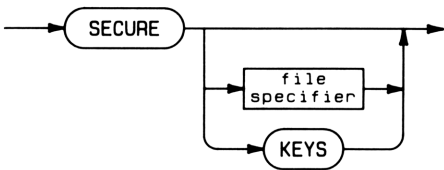
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

SECURE

SECURE KEYS

SECURE A#

SECURE "TESTDVC:PORT(2)"

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default: Current file.	File name with optional device specifier.

Comments

If a secured file is not private:

- In a catalog listing the file appears with an S in the file protection field.
- The file can be read, listed, and executed.
- The file cannot be altered, purged, or declared private.

A file that is both secure and private appears in a CAT listing with an E in the file protection field. Access to such files is limited to execution only. To remove a file from secure status, refer to the UNSECURE keyword entry.

SECURE (continued)

Related Keywords

PRIVATE, UNSECURE.

SETDATE

SETDATE sets the system clock to the specified date.

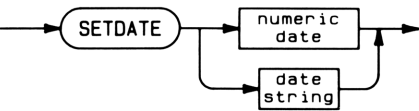
- ☒ Statement

☐ Function

☐ Operator
- ☒ Keyboard Execution

☐ CALC Mode

☒ IF . . . THEN . . . ELSE



Examples

SETDATE 82188

SETDATE 1982305

SETDATE "83/07/21"

SETDATE "1984/11/05"

SETDATE D

SETDATE D\$

Input Parameters

Item	Description	Restrictions
numeric date	Numeric expression rounded to an integer.	Must be of the form YYDDD or YYYYDDD, where YY or YYYY = year and DDD = day-in-year. For leap year: DDD is in the range of 1 through 366. For other years, DDD is in the range of 1 through 365.
date string	String expression.	Must be a valid date of the form YY/MM/DD or YYYY/MM/DD.

SETDATE (continued)

Comments

You can specify either a year and the day number of the day in that year, or a date string. The HP-71 uses either *YY* or *YYYY* formats for year inputs, *MM* for month inputs, and *DD* for day inputs.

For numeric date inputs and for date string inputs, if you use the two-digit year format, and if:

- $60 \leq YY \leq 99$, then $YY = YY + 1900$.
- $0 \leq YY \leq 59$, then $YY = YY + 2000$.

If you use the date string format that specifies a four-digit year, the computer interprets the year as entered. Although `DATE` and `DATE$` display only the last two digits of the year portion of a date, all four year digits are maintained internally.

Related Keywords

`DATE`, `DATE$`, `SETTIME`.

SETTIME

SETTIME sets the time on the system clock.

■ Statement

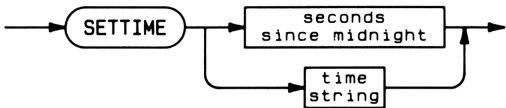
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

SETTIME 3*60*60 + 15*60 + 12

SETTIME 100,34

SETTIME "15:30:17"

SETTIME T

SETTIME T#

Input Parameters

Item	Description	Restrictions
seconds since midnight	Numeric expression rounded to an integer.	0 <= seconds < 86400.
time string	String expression.	Must be of the form HH:MM:SS representing time < 24 hours.

SETTIME (continued)

Comments

Hours (*HH*) must always be entered as two digits, and in the 24-hour clock format shown in the table to the right.

Hour a.m./p.m.	<i>HH</i>
12 Midnight	00
1 a.m.	01
.	.
.	.
.	.
12 Noon	12
1 p.m.	13
2	14
.	.
.	.
.	.
10	22
11	23

The HP-71 uses `SETTIME` inputs in two ways:

1. The time you enter is used to set the clock time.
2. The difference between the time currently maintained by the clock and the time you enter is interpreted as two entities, as follows, for interpreting the error correction factor:
 - The portion of the difference that is a multiple of 30 minutes is interpreted as an absolute adjustment, such as that used for a time zone or daylight savings change, and is not used as an error correction input.
 - The portion of the difference that is not a multiple of 30 is used to automatically determine and store an error correction factor that is applied the next time you execute `EXACT`. The HP-71 measures this portion from the nearest 30-minute increment and applies the appropriate sign.

For example, if the current HP-71 clock time is 10:03:17 and you execute `SETTIME "10:05:30"` to correct the clock, the HP-71's clock is set to 10:05:30 and an error factor is computed on the basis of $5^m 30^s - 3^m 17^s$, or $+2^m 13^s$, and is automatically stored.

SETTIME (continued)

The `SETTIME` error correction factor is cumulative between executions of `EXACT`. This allows you to execute `SETTIME` as many times as you wish before executing `EXACT`.

The resolution of the clock system is 1/512 second. A numeric input for `SETTIME` can specify fractional seconds; a string input cannot.

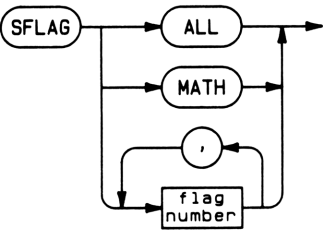
Related Keywords

`ADJABS`, `ADJUST`, `AF`, `EXACT`, `TIME`, `TIME$`.

SFLAG

SFLAG (*set flag*) sets user and/or system flags specified by keyword or by a flag number list.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF . . . THEN . . . ELSE



Examples

SFLAG ALL

SFLAG MATH

SFLAG 1,INX,OVF

Input Parameters

Item	Description	Restrictions
flag number	Numeric expression rounded to an integer.	−32 through 63.

Comments

SFLAG sets flags as follows:

- SFLAG ALL sets all user flags (0 through 63).
- SFLAG MATH sets the math exception flags.
- SFLAG with a flag number list sets the system and user flags specified by the integer-rounded values of the numeric expressions in the list. SFLAG cannot set system flags numbered less than −32.

Related Keywords

CFLAG, DVZ, FLAG, INX, IVL, OVF, UNF.

SGN

SGN(*X*) (*sign* (*x*)) returns -1 if *x* is less than zero; 0 if *x* equals zero; and 1 if *x* is greater than zero.

☐ Statement

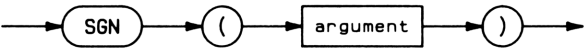
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

Y=SGN(X)*Y

PRINT "X'=";SGN(X1)*ABS(X)

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

The table to the right shows the values returned by SGN.

x	SGN(x)
> 0	1
= 0	0
< 0	-1
= NaN	NaN

Related Keywords

ABS, CLASS.

SHORT

SHORT allocates memory for short (5-digit precision) variables and arrays.

■ Statement

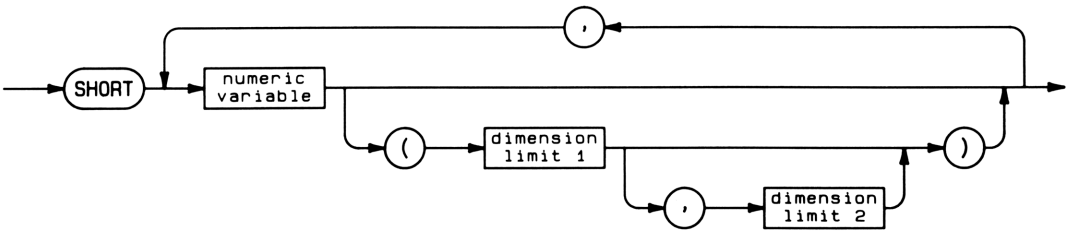
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF... THEN... ELSE



Examples

SHORT T

SHORT R7(7,3)

SHORT X2, M(4,4)

Input Parameters

Item	Description	Restrictions
numeric variable	Letter followed by optional digit.	None.
dimension limit 1	Numeric expression rounded to an integer.	Current OPTION BASE setting through 65535.
dimension limit 2		

Comments

SHORT creates short variables and arrays. Creation occurs upon execution of SHORT. The dimension limits are evaluated at creation time. The lowest-numbered subscript in any dimension is 0 or 1, depending on the OPTION BASE setting when the array is created. All elements are initialized to zero.

SHORT (continued)

If `SHORT` specifies a simple numeric variable that already exists, the variable is reinitialized to zero. Array variables are redimensioned, but not reinitialized to zero (unless the data type is changed). If `SHORT` expands an array, it also initializes all newly-created elements in the array. Notice that redimensioning does not necessarily preserve an element’s position within an array, but does preserve the sequence of elements within an array. (Refer to “Declaring Arrays (`DIM`, `REAL`, `SHORT`, `INTEGER`)” in section 3 of the *HP-71 Owner’s Manual*.)

The following table indicates the conditions that apply to `SHORT` variables and arrays:

SHORT Numeric Variables

Initial Value	0
Numeric Precision	5 Decimal Digits
Exponent Range	± 499
Maximum No. of Array Dimensions	2
Maximum Dimension Limit	65535
Memory Usage in Bytes:	
• Simple Variable	9.5
• Array	$4.5 * (Dim1 - Base + 1) * (Dim2 - Base + 1) + 9.5$

Related Keywords

`DESTROY`, `DIM`, `INTEGER`, `OPTION BASE`, `REAL`.

SHOW PORT

SHOW PORT displays the type and size, in bytes, of all the plug-in memory devices and independent RAM in your HP-71. SHOW PORT is nonprogrammable.

■ Statement

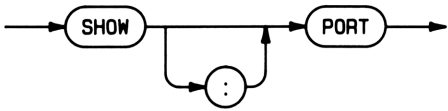
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

SHOW PORT

SHOW :PORT

Comments

For each memory device SHOW PORT displays the port number, the device size in bytes, and the device type number. Memory type numbers range from 0 through 15:

Device Type	Device Type Number
Independent RAM	1
ROM	2

Refer to FREE PORT for port numbering information.

SIN

SIN (*sine*) returns the sine of its argument.

☐ Statement

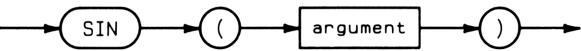
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
Y=R*SIN(T)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	Must be finite value.

Comments

SIN first reduces the argument modulo 360 (when in Degrees setting) or modulo $2\cdot\pi$ (when in Radians setting). SIN assumes that the argument is expressed according to the current angular setting. In Radians setting, SIN uses a 31-digit representation of π to increase accuracy. Also:

- $SIN(+0) = +0$
- $SIN(180 * n) = [(-1)^n] * 0$; where $n = 1,2,3,...$ (Degrees mode).
- $SIN(-x) = -SIN(x)$; for example, $SIN(-180) = +0$.

Related Keywords

```
ACOS, ACS, ANGLE, ASIN, ASN, ATAN, ATN, COS, DEG, DEGREES, OPTION, RAD, RADIANS, TAN.
```


SQR (SQRT)

SQR (*square root*) returns the square root of the argument.

☐ Statement

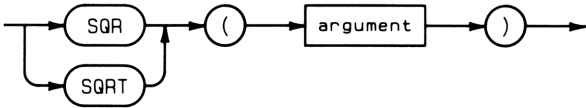
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

T=SQR(2*S/G)

PRINT "C=";SQR(A^2+B^2)

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	The argument must be greater than or equal to zero. Refer also to "Comments," below.

Comments

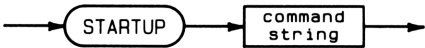
If the argument is less than zero, then the SQR(*neg*) (error 10) condition occurs.

Note: Because -0 is non-negative ($-0=0$), $SQR(-0)=-0$.

STARTUP

STARTUP defines a command string to be executed when you turn on the HP-71.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF . . . THEN . . . ELSE



Examples

```
STARTUP ""
STARTUP "RUN MONITOR"
STARTUP "WIDTH INF @ DELAY .5, INF"
```

Input Parameters

Item	Description	Restrictions
command string	String expression.	0 through 95 characters.

Comments

When specifying a STARTUP string, you can use any string of instructions that you can otherwise execute from the keyboard. To enter a multistatement string, use @ to concatenate the statements in the same way that you would when executing multistatement commands from the keyboard or in a program.

STARTUP Operation. When you execute STARTUP, the HP-71 stores the specified command string as it is typed, without checking for syntax errors. The computer maintains only one STARTUP string at any given time. When you switch the computer off, then on, if the string is error-free, the STARTUP string is executed. Otherwise, the computer displays an error message. If the computer is in CALC mode when you turn it off, or if BYE or OFF turns off the computer during program execution, the STARTUP string is not executed when you turn the computer on again.

Clearing a STARTUP String. Once you specify a STARTUP string, it remains active until cleared. To clear a STARTUP string, execute STARTUP with a null string (STARTUP "").

STAT

STAT (*statistics*) either creates and dimensions a statistical array to the appropriate size for a specified number of variables, or designates a previously dimensioned statistical array as the current statistical array.

■ Statement

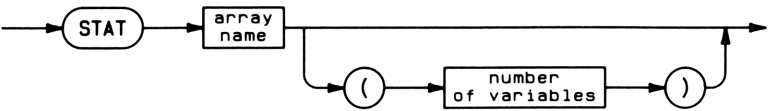
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

STAT A(4) STAT B

Input Parameters

Item	Description	Restrictions
array name	Numeric variable.	None.
number of variables	Numeric expression rounded to an integer.	0 through 15.

Comments

STAT allows you to simultaneously store several statistical arrays and to analyze them in any order.

Before performing statistical calculations, you must specify both the array to use for accumulating summary statistics and, if the array did not already exist, the number of statistical variables (coordinates) for each data point.

Creating an Array. STAT specifies the current statistical array and the number of variables per data point. You can specify the number of variables as any numeric expression whose rounded value is in the range of 0–15.

The HP-71 uses the number of variables to automatically allocate enough space for the array. The array has an OPTION BASE setting of zero, *regardless of the current OPTION BASE setting.*

STAT (continued)

Selecting an Existing Array. In an existing statistical array the number of variables is already set. Thus, you need not specify the number of variables when using `STAT` to select an existing statistical array.

Related Keywords

`ADD`, `CLSTAT`, `CORR`, `DROP`, `LR`, `MEAN`, `PREDV`, `SDEV`, `TOTAL`.

STD

`STD` (*standard*) selects the HP-71's standard BASIC format for displaying numbers.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
STD
```

```
IF FLAG(D)=1 THEN STD
```

Comments

The `STD` format conforms to ANSI Minimal BASIC Standard X3J2, and produces the following results when displaying or printing a number:

- Numbers that can be represented exactly as integers with 12 or fewer digits are displayed without a decimal point or exponent.
- Numbers that can be represented exactly with 12 or fewer digits, but not as integers, are displayed with a decimal point but no exponent. Leading zeroes to the left of the decimal point and trailing zeroes in the fractional part are omitted.
- Zero is displayed as 0.
- All other numbers are displayed in the following format:

(Sign) Mantissa E (Sign) Exponent

where the value of the mantissa is in the range $1 \leq x < 10$, and the exponent is represented by one to three digits. (A denormalized number has a mantissa that is less than 1 and an exponent of -499 .) Trailing zeroes in the mantissa and leading zeroes in the exponent are omitted.

STD (continued)

The following table provides examples of numbers displayed in STD format:

Number	Displayed As	Representable With 12 Digits?
10^{11}	100000000000	Yes-integer.
10^{12}	1.E12	No.
10^{-12}	.000000000001	Yes.
1.2×10^{-11}	.000000000012	Yes.
1.23×10^{-11}	1.23E-11	No.
12.345	12.345	Yes.

Any display setting remains in effect until changed by another FIX, SCI, ENG, or STD statement.

Related Keywords

ENG, FIX, SCI, STR\$.

STOP

STOP operates in the same way as END to end a subprogram, user-defined function, or a program.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
STOP                                IF X<0 THEN STOP
```

Comments

A program can contain more than one STOP statement.

Executing STOP from the keyboard releases a suspended program.

Related Keywords

END, PAUSE.

STR\$

STR\$ (*numeric-to-string conversion*) returns a string representation of the value of the argument.

☐ Statement

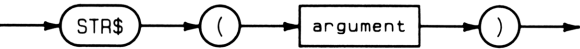
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF . . . THEN . . . ELSE



Examples

```
A$=STR$(N)                                DISP "X=";STR$(X)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	None.

Comments

The returned string is in the current display (or print) format, except that a positive value represented as a string is shifted one space to the left. STR\$ formats infinity as Inf, and NaN as NaN.

Standard (STD) Display Format. The STD format conforms to ANSI Minimal BASIC Standard X3J2, and is enabled by executing STD. (Refer to the STD keyword entry.)

Fixed-Precision (FIX) Display Format. The value is displayed rounded to *n* places past the decimal point, where *n* is specified by FIX *n*. Where the number of digits displayed by this method would exceed 12, the value is displayed in SCI *n* format. Where the nonzero value rounded to *n* places past the decimal point would be zero, the value is displayed in SCI *n* format.

STR\$ (continued)

Scientific (SCI) Display Format. The value is displayed in scientific notation to $n + 1$ significant digits, where n is specified by `SCI n`. The value appears as:

(Sign) Mantissa E (Sign) Exponent

where $1 \leq \text{mantissa} < 10$ for normalized numbers, and the mantissa is less than 1 for denormalized numbers.

Engineering (ENG) Display Format. The value is displayed to $n + 1$ significant digits, where n is specified by `ENG n`. The value appears as:

(Sign) Mantissa E (Sign) Exponent

where $1 \leq \text{mantissa} < 1000$, and the exponent is divisible by 3. If the value has an exponent of -499 , it is displayed in `SCI n` format.

Related Keywords

`DISP`, `ENG`, `FIX`, `PRINT`, `SCI`, `STD`.

SUB

SUB (*begin subprogram*) is the first statement in a subprogram and can specify the subprogram's formal parameters.

- Statement

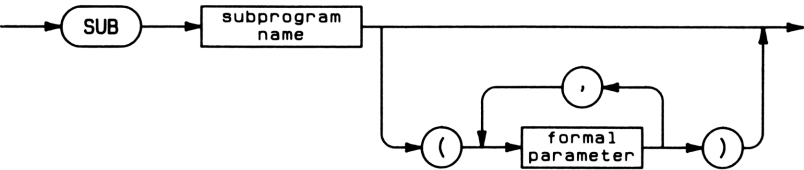
☐ Function

☐ Operator

☐ Keyboard Execution

☐ CALC Mode

☐ IF ... THEN ... ELSE



Examples

SUB PLOT(X,Y,D())

SUB A(N,B\$,#8)

SUB PAYROLL

Input Parameters

Item	Description	Restrictions
subprogram name	Unquoted string consisting of letters and digits, and starting with a letter.	Up to eight characters.
formal parameter	Simple or array variable name, or channel number preceded by #.	Channel number must be integer constant in the range 1 to 255.

Comments

SUB identifies the start of a subprogram. An ENDSUB, another SUB statement, or the end of the program file terminates the subprogram and returns program execution to the statement following the CALL statement that invoked the subprogram.

Parameter Passing. Actual parameters (those in the CALL parameter list) and formal parameters (those in the SUB parameter list) must match in type and number. Parameters can be passed to a subprogram by value or by reference. The distinction is specified in the CALL statement. Array variables passed by reference must be specified by the variable name, followed by an empty set of parentheses to specify the number of dimensions. (For example, a two-dimensional array, A, appears as A(,)).

SUB (continued)

All variables used in the subprogram, except those passed as parameters, are local to the subprogram. If a subprogram takes no parameters, then the channel numbers it uses are those of the calling program. Any channel number specified in the formal parameter list must be a constant.

Subprograms in Files. A BASIC program file can contain more than one subprogram. Also, a subprogram can reside in the same file as a main program.

If a subprogram is in the same file as the main program it must follow the main program. If there is more than one subprogram in a file, the subprograms must be placed consecutively. Any statement placed between two subprograms is ignored during program execution.

Binary Subprograms. You can write a subprogram in BASIC or in assembly code. `CALL` can invoke either subprogram type.

How the HP-71 Searches for Subprograms. If no file specifier is given in the `CALL` statement, the subprogram search begins with the current file. If the subprogram is not found, the search continues through the files in main RAM, then through the files on plug-in memory devices and independent RAMs. If the subprogram is not found in any file, and if the `CALL` statement lists no parameters, the HP-71 then begins searching for a *Program* having the specified name.

Related Keywords

`CALL`, `END SUB`.

TAN

TAN (*tangent*) returns the tangent of its argument.

☐ Statement

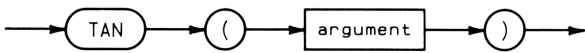
☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
Z=TAN(T)
```

Input Parameters

Item	Description	Restrictions
argument	Numeric expression.	Must be finite value.

Comments

TAN first reduces the argument modulo 360 (when in Degrees setting) or modulo $2 \cdot \pi$ (when in Radians setting). TAN assumes that the argument is expressed according to the current angular setting. In Radians setting, TAN uses a 31-digit representation of π for increased accuracy.

- `TAN(+0)` = +0.
- `TAN(180 * n)` = +0; $n=1,2,3,\dots$ (Degrees mode).
- `TAN(-x)` = -TAN(x).

Related Keywords

```
ACOS, ACS, ANGLE, ASIN, ASN, ATAN, ATN, COS, DEG, DEGREES, OPTION, RAD, RADIANS, SIN.
```

TIME

TIME returns the time of day expressed in seconds since midnight (00^h 00^m 00^s).

☐ Statement

☒ Function

☐ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A=TIME
```

```
DISP TIME
```

Related Keywords

TIME\$.

TIMES

TIME\$ (*time string*) returns the time of day as a string in the format *HH:MM:SS*.

☐ Statement

☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF . . . THEN . . . ELSE



Examples

```
A$=TIME$          DISP TIME$
```

Comments

The TIME\$ function uses the 24-hour format shown in the table to the right.

Hour a.m./p.m.	HH
12 Midnight	00
1 a.m.	01
.	.
.	.
.	.
12 Noon	12
1 p.m.	13
.	.
.	.
.	.
11	23

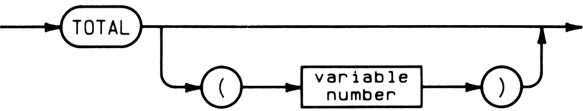
Related Keywords

TIME.

TOTAL

TOTAL returns the total of the specified variable in the current statistical array.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
A1 = TOTAL(X1)                                PRINT "Total = ";TOTAL
```

Input Parameters

Item	Description	Restrictions
variable number	Numeric expression rounded to an integer. Default: 1.	Zero through current STAT array dimension.

Comments

Because variable “0” of a statistics array accumulates the number of coordinate inputs, executing TOTAL(0) returns the number of data points currently summarized in an array.

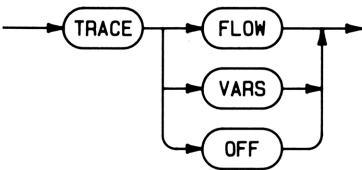
Related Keywords

```
ADD, CLSTAT, CORR, DROP, LR, MEAN, PREDV, SDEV, STAT.
```

TRACE FLOW/VARS/OFF

TRACE FLOW reports changes in the flow of a running program. TRACE VARS reports all variable assignments in a running program. TRACE OFF turns off all TRACE operations.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

TRACE FLOW

TRACE VARS

TRACE OFF

Comments

TRACE FLOW traces program execution during branches from the current statement, as when GOTO or GOSUB is executed.

TRACE VARS traces the changes in the values of all program variables.

Note: TRACE VARS does not trace the effects of DESTROY and variable redimensioning.

TRACE OFF turns off any active TRACE operations.

Trace operates globally. Thus, if a subprogram sets Trace mode, but does not subsequently clear it, the mode remains active when execution returns to the calling program.

TRACE VARS and TRACE FLOW can be active at the same time.

TRANSFORM

TRANSFORM is used to transform BASIC program files into TEXT files for interchange purposes, or to transform them back into BASIC program format.

- Statement

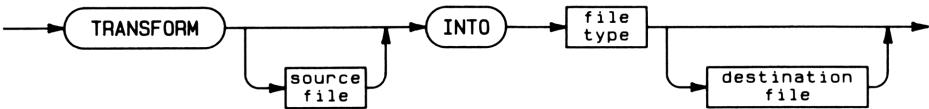
□ Function

□ Operator

■ Keyboard Execution

□ CALC Mode

■ IF ... THEN ... ELSE



Examples

```
TRANSFORM LIF1PROG:PORT INTO BASIC PROG
TRANSFORM A# INTO BASIC
TRANSFORM INTO TEXT
TRANSFORM "INTO" INTO TEXT
```

Input Parameters

Item	Description	Restrictions
source file	File specifier of file from which lines to be reformatted will be read. Default: Current file.	String expression or unquoted string containing a file name with an optional device specifier.
file type	File type specifying format of destination file.	Must be BASIC or TEXT (LIF1).
destination file	File specifier of the file into which lines to be reformatted will be written. Default: Source file.	String expression or unquoted string containing a file name with an optional device specifier.

TRANSFORM (continued)

Comments

The HP-71 TEXT file type can be referred to as LIF1, and is in fact identical to the LIF1 file type generated by the HP-75 computer. The HP-71 interprets and decompiles LIF1 as TEXT. For example, if you enter

```
100 TRANSFORM "AAA" INTO LIF1
```

then list this line, the HP-71 displays

```
100 TRANSFORM "AAA" INTO TEXT.
```

Transformation In-Place. If you omit the destination file, or if it is the same as the source file, the source file itself is transformed into the specified file type. If the file is already of the desired type, no action is taken. If the file is the `workfile`, it is renamed to `WORKFILE`, and a new `workfile` is created. The HP-71 allows in-place transformation only for files residing in RAM.

Transformation Out-of-Place. If the destination file is not the source file, a transformation destination file is created (it cannot already exist). If the source file is already of the desired type, a simple copy occurs. If the destination file is to be created on an external device, the HP-71 makes a preliminary pass through the source file to determine the file size needed for the destination file. The computer then creates the destination file and writes the transformed data, line by line, to the destination file. Any warning messages displayed during the preliminary pass will be displayed again when the file is actually transformed.

BASIC-To-TEXT Transformations. A BASIC-to-TEXT transformation converts (“decompiles”) a BASIC program file into equivalent lines of ASCII text. The starting line number of each statement is decompiled as a four-digit number with leading zeroes supplied. A decompiled line exceeding 120 characters generates the `Line Too Long` (error 65) condition.

TRANSFORM (continued)

TEXT-To-BASIC Transformations. TRANSFORM generally uses the same statement entry rules that the HP-71's operating system uses for statements entered from the keyboard, with certain exceptions and restrictions. TRANSFORM accepts a maximum line length of 120 characters instead of the 95-character keyboard limit. Longer lines are truncated to 120 characters, and generate the `Line Too Long` (error 65) condition. Restrictions are that every line must begin with a line number, and that an implied `DISP` statement may not begin with an expression that itself begins with a variable. For example:

```
100 3*SIN(A)
```

will be interpreted as

```
100 DISP 3*SIN(A)
```

but the statement

```
100 A
```

generates a syntax error.

Note: A transformation can take up to several minutes to complete, depending upon the size of the file being transformed.

Error Handling During Transformation. Errors detected during a transformation are handled automatically. If the error is a *recoverable* error (that is an error that *will not* prevent completion of the transformation), a warning message is displayed in the following format:

```
TFM WRN Lnnnn: ...
```

The *nnnn* indicates the line number of the source file line on which the error exists, and “...” indicates the warning message. If the computer cannot identify a line number for the source line, *nnnn* indicates the sequential number of the line in the source file. After the transformation is complete, the `Syntax` error message is generated if one or more recoverable errors occurred during the transformation.

When transforming a TEXT file to BASIC, a line having a valid line number but an invalid BASIC statement is a recoverable error and results in a warning. In this case the line is converted to a BASIC remark statement by preceding it with the characters “! ? ”. A BASIC-to-TEXT transformation removes these characters. Thus, original TEXT lines that cause an entry error when transformed from TEXT to BASIC will be restored when transformed back to TEXT.

TRANSFORM (continued)

If an error is *unrecoverable* (that is, an error that *will* prevent completion of the transformation), the computer displays the warning message

```
TFM WRN:Transform Failed
```

and aborts the transformation. If the transformation is out-of-place, the destination file is purged. If the transformation is in-place, the file is restored to its original format. After the HP-71 completes either operation, an error message identifying the cause of the unrecoverable error appears in the display.

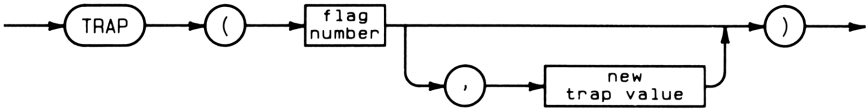
CAUTION

An error condition that prevents completion of an in-place transformation causes the computer to perform a reverse transformation on the portion of the file already changed in order to restore the file to its original format. If the reverse transformation is aborted by another error, such as insufficient memory, the file is rendered unrecoverable and is *automatically purged*. Note that in low memory conditions with extended keywords supplied by external ROMS or LEX files, there is a remote possibility that despite all system safeguards an error that prevents completion of a transformation operation can occur during the inverse transformation. *For this reason, in-place transformation is not recommended in conditions of limited memory or on a file for which you do not have a backup copy.*

TRAP

TRAP returns the current value of the trap for the specified flag number, and optionally allows you to supply a new value.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```

T=TRAP(UNF,1) ! set UNF trap to 1
T=TRAP(OVF,J) ! set OVF trap to J
IF TRAP(DVZ)=0 THEN RETURN ! test DVZ trap
O=TRAP(IVL) ! save IVL trap value
O=TRAP(IVL,O) ! restore IVL trap value
  
```

Input Parameters

Item	Description	Restrictions
flag number	Numeric expression rounded to an integer.	−8 through −4
new trap value	Numeric expression rounded to an integer.	0 through 15. (Refer to “Comments” on the next page.)

TRAP (continued)

Comments

You can use `TRAP` to set, test, save, and restore individual trap values. If an optional new trap value is specified, the trap is then assigned the new value. Otherwise, the trap is left unchanged. For information concerning trap values, refer to your *HP-71 Owner's Manual*.

A trap value in the range of 3 through 15 is treated in the same way as a trap value of 0.

Related Keywords

`DEFAULT`, `DVZ`, `INX`, `IVL`, `OVF`, `UNF`.

UNF (*underflow*) returns the underflow flag number (−5).

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
IF FLAG(UNF) THEN A=0           A=TRAP(UNF,2)
```

Related Keywords

CFLAG, DEFAULT, DVZ, FLAG, INX, IVL, OVF, SFLAG, TRAP.

UNPROTECT

UNPROTECT removes the write-protection from one track of a magnetic card.

<input checked="" type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF . . . THEN . . . ELSE



Examples

UNPROTECT

Comments

UNPROTECT is available only when the HP 82400A Card Reader is installed in your HP-71. When you execute UNPROTECT, the computer prompts you for a card and then allows you to pull the card through the card reader once. If the track accessed by the card reader was previously write-protected by PROTECT, the UNPROTECT operation removes the write protection. To remove the write-protection from the card's other track, turn the card around, reexecute UNPROTECT, and again pull the card through the card reader.

When you remove write-protection from a track, you can then use COPY to write over that track.

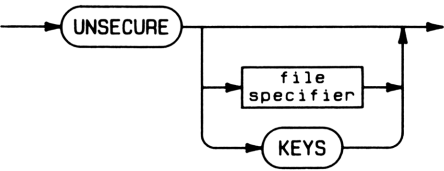
Related Keywords

PROTECT.

UNSECURE

UNSECURE clears the file access restriction that is set by SECURE.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

UNSECURE KEYS

UNSECURE "TESTFILE:PORT(2)"

UNSECURE

UNSECURE CAT\$(N)

UNSECURE FILE2

Input Parameters

Item	Description	Restrictions
file specifier	String expression or unquoted string. Default: Current file.	File name with optional device specifier.

Comments

UNSECURE KEYS reverses the effects of SECURE on the current key assignment file, keys.

You can purge an unsecured file. If such a file is not private, it may be read, listed, and altered without restriction.

Note: All files are unsecured at creation.

Related Keywords

PRIVATE, SECURE.

UPRC\$

UPRC\$ (*uppercase conversion*) converts all lower case letters in a string to their uppercase counterparts.

☐ Statement

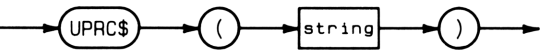
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
A$=UPRC$(A$)
```

Input Parameters

Item	Description	Restrictions
string	String expression.	None.

Comments

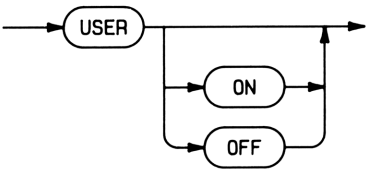
UPRC\$ returns a string that is identical to its argument except that all lowercase letters contained in the string are converted to uppercase.

This function is useful when you wish to allow input in lowercase, but require that such input be converted to uppercase before determining its value.

USER

`USER` activates or deactivates the User keyboard, in which user-defined key assignments are active.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF ... THEN ... ELSE



Examples

`USER ON`

`USER OFF`

`USER`

Comments

When the User keyboard is inactive, all keys correspond to those indicated on your HP-71's keyboard, and the **USER** mode annunciator is not displayed. When User mode is active, any user-defined key assignments maintained in the `keys` file are active and the **USER** annunciator is displayed. For a description of how to redefine keys, refer to the **DEF KEY** keyword entry.

Executing `USER` without specifying `ON` or `OFF` switches the current status of User mode to its opposite status.

Activating User mode sets system flag `-9`; deactivating User mode clears flag `-9`.

Related Keywords

`DEF KEY.`

VAL

VAL (*string-to-numeric conversion*) converts to a numeric value any numeric expression within a string expression.

☐ Statement

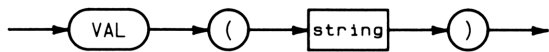
☒ Function

☐ Operator

☒ Keyboard Execution

☐ CALC Mode

☒ IF ... THEN ... ELSE



Examples

D=VAL<DATE\$[7]>
X=VAL<"A*B+2">

A=VAL<A\$>
A=VAL<"FNA<PI*2">>

Input Parameters

Item	Description	Restrictions
string	String expression.	Must contain a valid numeric expression.

Comments

- The parameter string is entered as a numeric expression.
- Any characters following the first valid numeric expression are ignored. For example, VAL("123B7") would return 123.
 - Any numeric expression is allowed including those that return Inf or NaN.

If VAL cannot interpret the string as a numeric expression, an error results. For example, VAL("234*") which the computer interprets as an incomplete expression, causes an Invalid Expr (error 80) condition.

Related Keywords

STR\$.

VER\$

VER\$ (*version string*) returns a string indicating the versions of the system ROMs built into your HP-71. VER\$ also returns the names of any LEX files present in the computer's memory or in a plug-in ROM.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input checked="" type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
DISP VER$
```

Comments

This function can provide useful information to Service personnel should your computer ever need service.

The default string returned by VER\$ indicates the version of your computer's system ROM set. The default string uses the following format:

```
HP71:10000
```

where each character in the string following the colon indicates the version of each of the computer's four system ROMs.

If there are any LEX files in main RAM, plug-in ROMs or independent RAMs, the HP-71 appends to the default string a separate string identifying each of these files. The format for any appended string is:

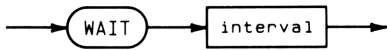
```
" name [ : version identifier ] "
```

The number of characters in the VER\$ string depends upon the number of LEX files in the machine.

WAIT

WAIT causes the computer to wait for the specified number of seconds.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

WAIT 5 WAIT X WAIT .3

Input Parameters

Item	Description	Restrictions
interval	Numeric expression giving number of seconds to wait.	0 through 19,531,250,000,000. Both integer and fractional values allowed.

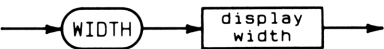
Comments

To interrupt the waiting period, press the ATTN key.

WIDTH

WIDTH defines the line length for the DISP and LIST statements.

- ☒ Statement
- ☐ Function
- ☐ Operator
- ☒ Keyboard Execution
- ☐ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

```
WIDTH INF           WIDTH 20
```

Input Parameters

Item	Description	Restrictions
display width	Numeric expression rounded to an integer.	1 to infinity. Arguments exceeding 255 are interpreted as infinite. Arguments less than 1 are interpreted as 1.

Comments

WIDTH affects only the displayed output. WIDTH does not set an absolute length on stored lines, but rather sets the number of characters for any line you display. If a stored line has more characters than specified by the current WIDTH setting, the line is split into two or more lines when included in DISP and LIST operations. For example, if you execute WIDTH 10, then execute

```
DISP"ABCDEFGHIJKLMNQRST"
```

the HP-71 displays the string as

```
ABCDEFGHIJ
```

and

```
KLMNQRST.
```

WIDTH affects input prompt displays in the same way.

WIDTH (continued)

Using `TAB` in `DISP` statements reduces the `TAB` function value modulo display width.

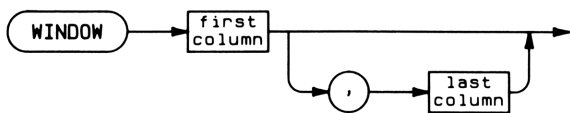
Related Keywords

`DISP`, `INPUT`, `LIST`, `PWIDTH`.

WINDOW

WINDOW sets the display window size and location.

■ Statement	■ Keyboard Execution
□ Function	□ CALC Mode
□ Operator	■ IF ... THEN ... ELSE



Examples

```
WINDOW 5,20           WINDOW 1
```

Input Parameters

Item	Description	Restrictions
first column	Numeric expression rounded to an integer.	1 through 22.
last column	Numeric expression rounded to an integer. Default: 22.	First column to 22.

Comments

A display window specified by WINDOW contains the “free portion” of the display that you can access and edit. Any portions of the display outside of the current display window are protected fields and cannot be edited. Any scrolling is thus performed in the free portion of the window. Any dot pattern in a protected field remains unchanged until one of the following occurs:

- Another WINDOW command clears the protected field in which the dot pattern is located.
- A GDISP statement changes the dot pattern.
- You perform an INIT 1 reset operation. (Refer to “Verifying Proper Operation” in appendix A of the HP-71 Owner’s Manual.)

If the current window is [1,22] then the entire LCD is reserved for the display. If the value of the first column specifier is greater than 1, WINDOW locks six dot columns into the display for each display character position not in the window. The effective size of the LCD is reduced by that amount, and scrolling occurs within the reduced window.



The @ concatenation character joins statements, which enables you to enter more than one statement in a program line or keyboard instruction.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input type="checkbox"/> CALC Mode
<input type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
SFLAG-25 @ BEEP 250,1 @ CFLAG-25
INPUT "DIAMETER?"; D @ C=PI*D @ FIX 4 @ DISP C @ STD
```

Comments

Using multistatement lines instead of single-statement lines conserves memory. (Refer to the “BASIC” entry under “Files” in the “System Memory Requirements” chart on pages 330 through 332.)

Most HP-71 statements can be used in multistatement lines without restriction. However, AUTO, CONT, FETCH, and RUN cannot be followed by an @.

The & concatenation operator joins strings.

- | | |
|--|--|
| <input type="checkbox"/> Statement | <input checked="" type="checkbox"/> Keyboard Execution |
| <input type="checkbox"/> Function | <input type="checkbox"/> CALC Mode |
| <input checked="" type="checkbox"/> Operator | <input type="checkbox"/> IF ... THEN ... ELSE |



Examples

Program segment:

```

10 A$="AMPERSAND: "
20 C$="CONCATENATOR"
30 DISP A$ & " STRING" & C$
  
```

Program output:

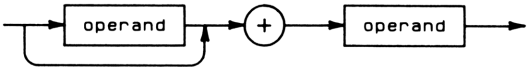
```

AMPERSAND:  STRING  CONCATENATOR
  
```



The `+` (*addition*) binary operator returns the sum of its operands. When used as a unary operator, `+` returns the value of its operand.

- | | |
|--|--|
| <input type="checkbox"/> Statement | <input checked="" type="checkbox"/> Keyboard Execution |
| <input type="checkbox"/> Function | <input checked="" type="checkbox"/> CALC Mode |
| <input checked="" type="checkbox"/> Operator | <input checked="" type="checkbox"/> IF ... THEN ... ELSE |



Examples

S=S + A

PRINT "sum =";A + B

Input Parameters

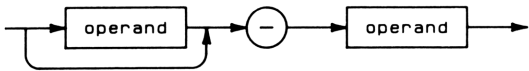
Item	Description	Restrictions
operand	Numeric expression.	Subject to operator precedence. Also, refer to "Comments," below.

Comments

Executing `Inf + (-Inf)` results in the `Inf-Inf` (error 15) condition.

The $-$ (*minus*) unary operator reverses the sign of its operand. When used as a binary operator, $x - y$ is defined as $x + (-y)$.

- ☐ Statement
- ☐ Function
- ☒ Operator
- ☒ Keyboard Execution
- ☒ CALC Mode
- ☒ IF ... THEN ... ELSE



Examples

A=-A

PRINT "Difference =";X1-X2

Input Parameters

Item	Description	Restrictions
operand	Numeric expression.	Subject to operator precedence. Also, refer to “Comments,” below.

Comments

The $-x$ operation reverses the sign of x . (This is true even for ± 0 and for NaN.)

`Inf - Inf` results in an `Inf-Inf` (error 15) condition.



The \times (*multiplication*) binary operator returns the product of its operands.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input checked="" type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

Y=2*X+1

PRINT "X squared =";X*X

Input Parameters

Item	Description	Restrictions
operand	Numeric expression.	Subject to operator precedence. Cannot multiply Inf by zero.

Comments

(Inf * 0) and (0 * Inf) results in an Inf*0 (error 16) condition.



The \div (*division*) binary operator returns the quotient of its operands.

☐ Statement

☐ Function

☒ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

C=X/R

PRINT "cost=";X/1.05

Input Parameters

Item	Description	Restrictions
dividend divisor	} Numeric expression.	Subject to operator precedence. Cannot divide by zero or divide Inf by Inf.

Comments

- The following exceptions can occur:
- $0 \div 0$ results in an error 7 condition.
 - $n \div 0$ results in an error 8 (\div Zero) condition.
 - $Inf \div Inf$ results in an error 14 condition.

Related Keywords

DIV.

The [^] (*exponentiation*) binary operator returns the first operand raised to the power given by the second operand.

☐ Statement

☐ Function

☒ Operator

☒ Keyboard Execution

☒ CALC Mode

☒ IF ... THEN ... ELSE



Examples

```
R2 = X^2 + Y^2
PRINT "N-th Root of x = ";X^(1/N)
```

Input Parameters

Item	Description	Restrictions
base power	Numeric expression.	Subject to operator precedence. Also, refer to "Comments," below.

Comments

The following exceptions can occur:

- A negative value raised to an non-integer power results in a `Neg^Non-int` (error 9) condition.
- The value of 1 raised to an `Inf` power results in a `1^Inf` (error 17) condition.
- Zero raised to a negative power results in a `0^neg` (error 5) condition.
- The operations `0^0` and `Inf^0` give warnings, but return the default value 1 unless `DEFAULT OFF` is active. (Refer to the **DEFAULT OFF/ON** keyword entry.)

The `%` (*percent*) binary operator returns x percent of y for the operation $x \% y$.

<input type="checkbox"/> Statement	<input checked="" type="checkbox"/> Keyboard Execution
<input type="checkbox"/> Function	<input checked="" type="checkbox"/> CALC Mode
<input checked="" type="checkbox"/> Operator	<input checked="" type="checkbox"/> IF ... THEN ... ELSE



Examples

```
T=R%S                                PRINT "10% of Total =" ;10%T
```

Input Parameters

Item	Description	Restrictions
<div>percent argument</div>	Numeric expression.	Subject to operator precedence. Also, refer to “Comments,” below.

Comments

The $x \% y$ operation is defined by

$$x \% y = (x/100) * y.$$

Executing `0 % Inf` or `Inf % 0` results in the `Inf*0` (error 16) condition.

System Characteristics

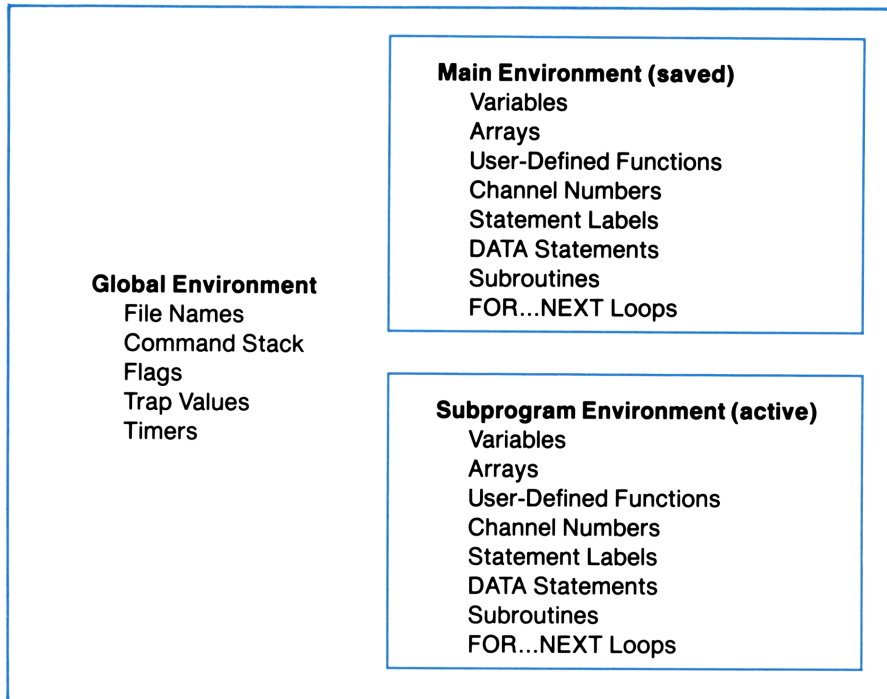
Scope of Environments

An environment is the set of variables, pointers, flags, and other information within which a program or a subprogram operates. Three types of environments can exist in the HP-71:

1. *Global environment.* This can be accessed by every program and subprogram and from the keyboard. For example, a flag set in one subprogram can be tested in another subprogram.
2. *Main environment.* This is the default environment. It is active when a subprogram isn't running or execution is not halted in a subprogram. The main environment consists of those variable items and channel numbers that are saved when a subprogram is called. Executing `END ALL` returns the computer to the main environment.
3. *Subprogram environment.* This is similar to the main environment, except that it is created when a subprogram is called. Running a subprogram saves the main environment. A subprogram's environment is saved if it calls another subprogram. When the second subprogram ends, its environment is erased and the first subprogram's environment is restored. When the first subprogram ends, its environment is erased and the main environment is restored.

The illustration on the facing page indicates the components of the global, main, and subprogram environments.

Global, Main, and Subprogram Environments



A user-defined function can be used only by the main program or subprogram in which the function is created. A user-defined function created by a program or a subprogram cannot be used by another program or subprogram. Also, the `DATA` statements contained within a program or subprogram can be read only by `READ` statements.

Channel numbers listed as parameters in a subprogram's `SUB` statement are not local to that subprogram. Channel numbers not listed in a `SUB` statement are local unless the `SUB` statement contains no other parameters, in which case the channel numbers in the calling environment are shared with those of the subprogram's environment.

Variables

A simple variable and an array variable cannot use the same identifier.

Note: In the following information, any optional parameter is indicated by a pair of square brackets, as shown below:

required parameter[*optional parameter*]

Simple Numeric Variables

- Identifier: *letter*[*digit*]
- Types: REAL (default type)
SHORT
INTEGER
- Examples: A, P3

Numeric Array Variables

- Identifier: *letter*[*digit*](*subscript*[, *subscript*])
- Types: REAL (default type)
SHORT
INTEGER
- Examples: A(1), N(5,10)

Simple String Variables

- Identifier: *letter*[*digit*]*
- Examples: A*, T3*
- Default string size: 32.
(Refer to the **DIM** keyword entry.)

String Array Variables

- Identifier: *letter*[*digit*]*(*subscript*)
- Examples: B*(3), Z9*(4)
- Default element size: 32. (Refer to the **DIM** keyword entry.)

Array Bounds and Referencing

The default lower bound on array subscripts at memory reset (memory lost) is 0. (Refer to the **OPTION BASE** keyword entry.)

The default upper bound on array subscripts is 10. (DIM, REAL, SHORT, or INTEGER declarations specify an upper bound.)

The maximum upper bound on array subscripts is 65535.

You can reference entire arrays in PRINT #, READ #, READ, and CALL statements by specifying only the array name. Also, you can use an array name with parentheses and no subscripts. For example, you can use T(,) instead of the array name alone.

Math Reference

Precedence of Operators

The table below lists HP-71 operators in their order of precedence. The first line indicates the highest precedence. Where an expression contains two or more operators having the same level of precedence, those operators will be evaluated in the left-to-right order in which they occur within the expression.

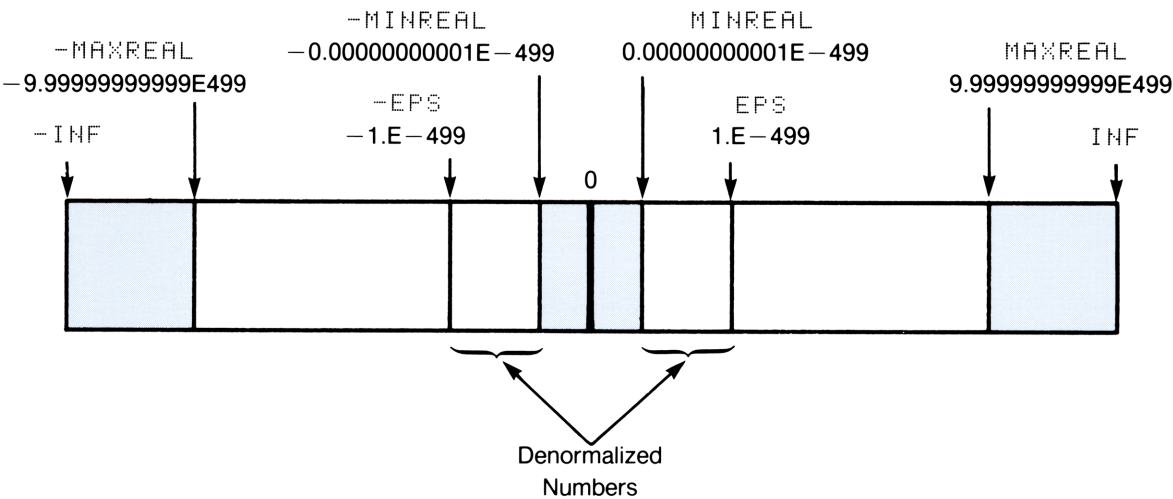
Performed First	<code><...></code> (Nested parentheses are evaluated from the inside out.)
	Functions (such as <code>SIN</code> , <code>RND</code> , etc.)
	<code>^</code>
	unary <code>+</code> , unary <code>-</code> , <code>NOT</code>
	<code>*</code> , <code>/</code> , <code>DIV</code> , <code>%</code>
	<code>+</code> , <code>-</code> , <code>&</code>
	<code><</code> , <code>=</code> , <code>></code> , <code>#</code> , <code>?</code> , <code><=</code> , <code>>=</code> , <code><></code>
	<code>AND</code>
	<code>OR</code> , <code>EXOR</code>
Performed Last	

Numeric Precision

Type	Precision	Maximum Value
REAL	12 digits	$\pm 9.9999999999 \times 10^{\pm 499}$
SHORT	5 digits	$\pm 9.9999 \times 10^{\pm 499}$
INTEGER	5 digits	± 99999

Range of Numbers

The shaded areas in the following illustration indicate intervals that do not contain any HP-71 representable numbers. `INF`, `MAXREAL`, and `MINREAL` are functions that return the endpoints of these intervals. `MAXREAL` and `EPS` return the overflow and underflow thresholds respectively. The HP-71 representable numbers whose magnitudes lie between `EPS` and `MINREAL` are “denormalized.” That is, they have the minimum exponent, -499 , but contain one or more leading zeroes in the mantissa.



Numeric Expressions

A numeric expression can include any of the following:

- A numeric constant.
- A numeric variable.
- A numeric function.

A numeric expression can also be any of the preceding forms, combined by operators (arithmetic, relational, or logical) or pairs of parentheses.

System Flags

A memory reset (memory lost) clears all of the following flags except flag -61.

Flag Number	Effect When Set	Set/Clear by User
-1	Warning messages suppressed.	Yes
-2	Beeper is off.	Yes
-3	Continuous on.	Yes
-4	Inexact result (INX).	Yes
-5	Underflow (UNX).	Yes
-6	Overflow (OVF).	Yes
-7	Division by zero (DVZ).	Yes
-8	Invalid operation (IVL).	Yes
-9	User keyboard is active.	Yes
-10	Angular setting is radians.	Yes
-11, -12	Round-off setting.	Yes
-13, -14	Display format.	Yes
-15	Lowercase lock.	Yes
-16	Base option 1.	Yes
-17 to -20	Number of display digits.	Yes
-25	Beep set to loud.	Yes
-26	Don't prompt.	Yes
-46	Exact flag.	No
-57	AC annunciator on.	No
-60	Alarm annunciator on.	No
-61	BAT annunciator on.	No
-62	PRGM annunciator on.	No
-63	SUSP annunciator on.	No
-64	CALC annunciator on.	No

Keyboard and Display Control

Input Keystrokes

[A] through [Z]	} Uppercase and lowercase letters.
[9][a] through [9][z]	
[0] through [9]	
[SPC]	Space.
[.]	Period. Used as a decimal point in numbers.

, , , ,
 ,

,

,

,

Arithmetic symbols: addition (plus), subtraction (minus), multiplication (asterisk), division (slash), exponentiation (circumflex), and percent.

Comma. Used to separate items in commands, statements, and functions; and to separate keyboard responses to the `INPUT` statement. Also used to space between items displayed.

Parentheses. Used to key in expressions and to dimension variables.

Exclamation mark. Used for end-of-line comments in program statements.

Double and single quotation marks. Used to enclose strings.

Number sign. Used to specify file numbers of BASIC files in `ASSIGN #`, `PRINT #`, `RESTORE #`, and `READ #` statements. Also used to assign timer numbers in `ON TIMER #` and `OFF TIMER#` statements and for inequality in relational tests.

Dollar sign. Used to specify string variables and string functions.

Ampersand. Used to concatenate string expressions.

Opening and closing brackets. Used to dimension string variables and to specify substrings.

Statement separator. Used between statements to form multistatement lines.

Semicolon. Used as a delimiter in `PRINT`, `DISP`, `INPUT`, `PRINT #`, and `READ #` statements, and as a “typing aid” specifier in key redefinitions.

Equals. Used to assign values to variables and to test for equality.

Less than. Used in relational tests.

Greater than. Used in relational tests. Also used as the BASIC prompt.

Left brace and right brace.


Colon. Used as a delimiter and as an “execution only” specifier in key redefinitions.

Question mark. The default prompt for the `INPUT` statement. Also used as a relational operator.

Editing Keystrokes

(gold key)

The f-shift key. Used to access f-shifted keywords, commands, and functions.

 (blue key)

The g-shift key. Used to access lowercase letters, characters indicated in blue on the keyboard, and certain editing features.

Control character prefix. Used to display certain characters from the HP-71 character set.



Lowercase lock. Switches keyboard between uppercase and lowercase.



Insert/Replace. Switches between insert cursor and replace cursor.

Backspace. Backspaces the cursor one character position and deletes the character at the cursor's new position.

Delete character. Deletes the character at the cursor position and shifts all succeeding characters one position to the left.

Delete line. Deletes all characters from the cursor position to the end of the line.

Next port catalog. Used during `CAT ALL` and `CAT:PORT` operations to access the catalog on the next port in sequence.

 , 


Left and right arrows. Moves the cursor to the left or right across the display or, if no cursor, scrolls the display window to the left or right.

  ,  

Far left and far right arrows. Moves cursor to the far left and the far right of the display. If no cursor, shifts the display window to the far left or far right.



 , 

Up and down arrows. Moves the display up and down through BASIC program files, through the system and mass storage catalogs, and through the command stack.

  ,  

Far up and far down arrows. Moves display to first and last line of a BASIC program file and catalog, and to oldest and most recent entry in the command stack.

System Keystrokes

 , 

Attention. Turns on the display, clears the display, and interrupts running programs.

Turns the display off. Memory and clock system remain active.

Reset. When pressed simultaneously, causes a reset to occur. The computer prompts for one of three initialization settings.

Typing aid for `EDIT` function. If pressed during a multiple file catalog operation, causes the currently displayed file to become the current file.

Typing aid for `FETCH` function.



End line. Causes an expression, statement, or command in the display to be evaluated, stored, or executed.

RUN	Run program. Runs the current program when in BASIC mode.
f SST	Single step. Displays and executes the next program statement in the current file.
9 CMDS	Command stack. Displays most recent instruction executed from the keyboard and sets display to command stack. When in the command stack, this key sequence returns you to the BASIC prompt.
9 ERRM	While held down, displays most recent error message.
9 1 USER	Toggles user keyboard for one shifted or unshifted keystroke.

Escape Keystrokes

9 CTRL 9 I	Generates ASCII character 27 (escape). The HP-71 responds to the escape sequences listed under “HP-71 Display Escape Code Sequences” on page 328.
--------------------------	---

Display Control

DELAY	Delay function. Controls the rate at which information is sent to and scrolled across the display.
PWIDTH	Sets line length of output to printer devices.
WIDTH	Sets line length of output to display devices.
WINDOW	Sets number of characters in display window.

CALC Mode Keystrokes

f SST RUN	Causes evaluation of the rightmost operator or function.
9 BACK	Backup execution. For the current expression, restores the last operator or function that was evaluated.
▲ , ▼	Command stack scroll up, down. Scrolls through command stack. While in CALC mode, unnecessary to press 9 CMDS to enter command stack.
END LINE	Causes expression to be evaluated and the results of numeric expressions to be stored in the RES variable.

HP-71 Character Set and Character Codes

The following table shows the HP-71 character set. Where keystrokes are shown to the right of a character, you can use either those keystrokes or the `CHR#` function to display the character. Where no keystrokes are shown to the right of a character, you can use only the `CHR#` function to display that character. (In most cases, the default display characters in the right side of the table are duplicates of the display characters in the left side of the table.)

Note: You can use the `CHARSET` statement to change the display character symbol for one or more of the default display characters corresponding to character codes 128 through 255.

Hex	Binary		Decimal (CHR\$) Character Code	Display Character	Keystrokes	Decimal (CHR\$) Character Code	Default Display Character
	Left	Right					
00	0000	0000	0	-None-		128	Space
01	0000	0001	1	▯		129	▯
02	0000	0010	2	⌘		130	⌘
03	0000	0011	3	⌥		131	⌥
04	0000	0100	4	⌘		132	⌘
05	0000	0101	5	⌥		133	⌥
06	0000	0110	6	⌘		134	⌘
07	0000	0111	7	⌥		135	⌥
08	0000	1000	8	-Blank-		136	⌘
09	0000	1001	9	⌘		137	⌘
0A	0000	1010	10	-Blank-		138	⌘
0B	0000	1011	11	⌥		139	⌥
0C	0000	1100	12	⌘		140	⌘
0D	0000	1101	13	-Blank-		141	⌘
0E	0000	1110	14	⌥		142	⌥
0F	0000	1111	15	⌘		143	⌘
10	0001	0000	16	⌘		144	⌘
11	0001	0001	17	⌥		145	⌥
12	0001	0010	18	⌘		146	⌘
13	0001	0011	19	⌥		147	⌥
14	0001	0100	20	⌘		148	⌘
15	0001	0101	21	⌥		149	⌥
16	0001	0110	22	⌘		150	⌘
17	0001	0111	23	⌥		151	⌥
18	0001	1000	24	⌘		152	⌘
19	0001	1001	25	⌥		153	⌥
1A	0001	1010	26	⌘		154	⌘
1B	0001	1011	27	-Blank-		155	⌘
1C	0001	1100	28	⌥	-None-	156	⌥
1D	0001	1101	29	⌘		157	⌘
1E	0001	1110	30	⌥		158	⌥
1F	0001	1111	31	⌘	-None-	159	⌘
20	0010	0000	32	Space		160	Space
21	0010	0001	33	!		161	!
22	0010	0010	34	"		162	"
23	0010	0011	35	#		163	#

(Continued on next page.)

(Continuation of Character Set/Code Table)

Hex	Binary		Decimal (CHR#) Character Code	Display Character	Keystrokes	Decimal (CHR#) Character Code	Default Display Character
	Left	Right					
24	0010	0100	36	\$	[g][\$]	164	\$
25	0010	0101	37	%	[g][%]	165	%
26	0010	0110	38	&	[g][&]	166	&
27	0010	0111	39	'	[g][']	167	'
28	0010	1000	40	([f][]	168	(
29	0010	1001	41)	[f][)]	169)
2A	0010	1010	42	*	[f][*]	170	*
2B	0010	1011	43	+	[f][+]	171	+
2C	0010	1100	44	,	[f][,]	172	,
2D	0010	1101	45	-	[f][-]	173	-
2E	0010	1110	46	.	[f][.]	174	.
2F	0010	1111	47	/	[f[/]	175	/
30	0011	0000	48	0	[0]	176	0
31	0011	0001	49	1	[1]	177	1
32	0011	0010	50	2	[2]	178	2
33	0011	0011	51	3	[3]	179	3
34	0011	0100	52	4	[4]	180	4
35	0011	0101	53	5	[5]	181	5
36	0011	0110	54	6	[6]	182	6
37	0011	0111	55	7	[7]	183	7
38	0011	1000	56	8	[8]	184	8
39	0011	1001	57	9	[9]	185	9
3A	0011	1010	58	:	[g][:]	186	:
3B	0011	1011	59	;	[g][;]	187	;
3C	0011	1100	60	<	[g][<]	188	<
3D	0011	1101	61	=	[g][=]	189	=
3E	0011	1110	62	>	[g][>]	190	>
3F	0011	1111	63	?	[g][?]	191	?
40	0100	0000	64	@	[g][@]	192	@
41	0100	0001	65	A	[A]	193	A
42	0100	0010	66	B	[B]	194	B
43	0100	0011	67	C	[C]	195	C
44	0100	0100	68	D	[D]	196	D
45	0100	0101	69	E	[E]	197	E
46	0100	0110	70	F	[F]	198	F
47	0100	0111	71	G	[G]	199	G
48	0100	1000	72	H	[H]	200	H
49	0100	1001	73	I	[I]	201	I
4A	0100	1010	74	J	[J]	202	J

(Continued on next page.)

(Continuation of Character Set/Code Table)

Hex	Binary		Decimal (CHR\$) Character Code	Display Character	Keystrokes	Decimal (CHR\$) Character Code	Default Display Character
	Left	Right					
4B	0100	1011	75	K		203	K
4C	0100	1100	76	L		204	L
4D	0100	1101	77	M		205	M
4E	0100	1110	78	N		206	N
4F	0100	1111	79	O		207	O
50	0101	0000	80	P		208	P
51	0101	0001	81	Q		209	Q
52	0101	0010	82	R		210	R
53	0101	0011	83	S		211	S
54	0101	0100	84	T		212	T
55	0101	0101	85	U		213	U
56	0101	0110	86	V		214	V
57	0101	0111	87	W		215	W
58	0101	1000	88	X		216	X
59	0101	1001	89	Y		217	Y
5A	0101	1010	90	Z		218	Z
5B	0101	1011	91	[219	[
5C	0101	1100	92	\	-None-	220	\
5D	0101	1101	93]		221]
5E	0101	1110	94	^		222	^
5F	0101	1111	95	_	-None-	223	_
60	0110	0000	96	`	-None-	224	`
61	0110	0001	97	a		225	a
62	0110	0010	98	b		226	b
63	0110	0011	99	c		227	c
64	0110	0100	100	d		228	d
65	0110	0101	101	e		229	e
66	0110	0110	102	f		230	f
67	0110	0111	103	g		231	g
68	0110	1000	104	h		232	h
69	0110	1001	105	i		233	i
6A	0110	1010	106	j		234	j
6B	0110	1011	107	k		235	k
6C	0110	1100	108	l		236	l
6D	0110	1101	109	m		237	m
6E	0110	1110	110	n		238	n
6F	0110	1111	111	o		239	o
70	0111	0000	112	p		240	p
71	0111	0001	113	q		241	q

(Continued on next page.)

(Continuation of Character Set/Code Table)

Hex	Binary		Decimal (CHR\$) Character Code	Display Character	Keystrokes	Decimal (CHR\$) Character Code	Default Display Character
	Left	Right					
72	0111	0010	114	r	[g][r]	242	r
73	0111	0011	115	s	[g][s]	243	s
74	0111	0100	116	t	[g][t]	244	t
75	0111	0101	117	u	[g][u]	245	u
76	0111	0110	118	v	[g][v]	246	v
77	0111	0111	119	w	[g][w]	247	w
78	0111	1000	120	x	[g][x]	248	x
79	0111	1001	121	y	[g][y]	249	y
7A	0111	1010	122	z	[g][z]	250	z
7B	0111	1011	123	{	[g][{]	251	{
7C	0111	1100	124		-None-	252	
7D	0111	1101	125	}	[g][}]	253	}
7E	0111	1110	126	~	-None-	254	~
7F	0111	1111	127	␣	-None-	255	␣

Control Characters

Characters 0 through 31 in the preceding table are also special control characters defined by the American Standard Code for Information Interchange (ASCII). These characters (also known as control codes) are primarily used in data communications to control peripheral devices. The following table shows the standard ASCII definitions for those characters.

Note Except for the four special control character codes marked with an asterisk, none of the characters represented by the following codes perform any control function in the HP-71. Also, adding 128 to the character code for any of these four special control characters displays the symbol for that code instead of performing the indicated control function.

Decimal (CHR#) Character Code	Mnemonic	Control Function
0	NUL	The Null Character
1	SOH	Start of Header
2	STX	Start of Text
3	ETX	End of Text
4	EOT	End of Transmission
5	ENQ	Enquiry
6	ACK	Acknowledge
7	BEL	Ring Bell
8*	BS	Backspace
9	HT	Horizontal Tab
10*	LF	Linefeed
11	VT	Vertical Tab
12	FF	Form Feed
13*	CR	Carriage Return
14	SO	Shift Out
15	SI	Shift In
16	DLE	Data Link Escape
17	DC1	Device Control 1
18	DC2	Device Control 2
19	DC3	Device Control 3
20	DC4	Device Control 4
21	NAK	Negative Acknowledge
22	SYN	Synchronous Idle
23	ETB	End of Transmission Block
24	CAN	Cancel Line
25	EM	End of Medium
26	SUB	Substitute
27*	ESC	Escape
28	FS	File Separator
29	GS	Group Separator
30	RS	Record Separator
31	US	Unit Separator

HP-71 Display Escape Code Sequences

To enter an escape (ESC) sequence, press 9CTRL 9I followed by the key corresponding to the desired sequence.

Press 9 CTRL 9 I and:	Effect
Q	Inserts cursor.
N	Inserts cursor (with wrap-around).
R	Replaces cursor.
C	Moves cursor right.
D	Moves cursor left.
H	Homes cursor.
J	Clears display.
K	Deletes through end of line.
9 >	Turns cursor on.
9 <	Turns cursor off.
E	Resets display.
P	Deletes character.
O	Deletes character (with wrap-around).
%	Sets cursor position in video monitor. Refer to following paragraph.
9 CTRL C	Moves cursor to right of rightmost character.
9 CTRL D	Moves cursor to leftmost character.

To reposition the cursor to a specific row and column in a video monitor, use the `CHR$(27) & "%"` form of the escape sequence with the `%` symbol as follows:

```
DISP CHR$(27) & "%" & CHR$(column) & CHR$(row) END LINE
```

Using the key sequence 9CTRL 9I 9% (as indicated in the preceding table) instead of `CHR$(27) & "%"` causes the cursor to move to the column position specified by the ASCII character code for the first key you press after 9%.

Reset Conditions

The following conditions exist in the HP-71 when you first install the batteries, or whenever a Memory Lost condition occurs or you execute a memory reset— (

Condition	After the Machine Reset or After the Batteries Are First Installed
Display:	
WIDTH	96
PWIDTH	96
CONTRAST	9
WINDOW	1,22
DELAY	.5, .125
LC (lowercase lock)	Off.
User Keyboard	Not Active.
Alternate Character Set	None.
Flags:	
All user flags.	Clear.
System flags	—1 through —32 Clear.
Files:	
File Catalog	One file, the system workfile, which is the current file.
File Access	No files are open.
keys File	Nonexistent.
Numeric Settings:	
Display Format	STD
Angular Setting	DEG
Random Number Seed	999500333083533
DEFAULT	ON
Variables and Arrays	None allocated.
OPTION BASE	0
Round-off Setting	Nearest.
Password:	
LOCK	Set to null string.
ENDLINE Setting	CR/LF.
Clock:	
SETTIME	00:00:00
SETDATE	0000/01/01
Accuracy Factor (AF)	0

(Continued on next page.)

Reset Conditions (continued)

Condition	After the Machine Reset or After the Batteries Are First Installed
Programming:	
Current File	System workfile.
AUTO mode	Off.
Timers	Off.
TRACE	Off.
Warning Messages	Displayed.
Continuous ON	Not set.
BEEP	Set to ON and LOW.
STARTUP String	None.

System Memory Requirements

Item	Memory Required
Alternate Character Set	3½ bytes, plus 6 bytes per character.
Command Stack	15 bytes, plus 1 byte for each character in each command in the stack, including carriage return.
Data File Channels	31½ bytes for each file opened, plus 2½ bytes for each new channel number. (Up to 64 files can be opened at once.)
Expressions	
Constants:	
One-Digit Integer Constant	1 byte.
Multi-Digit Integer Constant	1 byte plus ½ byte for each digit.
Constant With Fractional Part or Exponent > 11	2½ bytes plus ½ byte for each digit.
String Constants	2 bytes plus 1 byte per character.

(Continued on next page.)

(Continuation of System Memory Requirements Table)

Item	Memory Required
Variable References: <ul style="list-style-type: none">• Simple Variable• Array Variable	1 byte per character in variable name. 1½ bytes, plus 1 byte per character in array name, plus memory requirement for subscript expression(s).
Function References: <ul style="list-style-type: none">• User-Defined Functions• System Functions	1½ bytes, plus 1 byte per character in function name, plus memory requirement for parameter expression(s), if present. 1 or 3½ bytes, plus memory requirement for parameter expression(s), if present.
Operators: <ul style="list-style-type: none">• Relational Operators• Other Operators	1½ bytes. 1 byte.
Substring References	1½ bytes plus memory requirement for index expression(s).
Files	
BASIC	24½ bytes for the empty file. Also, in addition to the memory used for the statement(s) you enter in a line, any single-statement line uses four bytes, and any multiple-statement line uses four bytes plus two more bytes for each concatenated statement. (Thus, by using a multiple-statement line instead of two or more single-statement lines, you conserve two bytes per concatenated statement.)
TEXT	18½ bytes plus 2 bytes per line plus 1 byte per character plus 2 bytes for end-of-file.
DATA	22½ bytes plus record size in bytes times number of records. String data items written to the file use 3 bytes plus 1 byte per character plus 3 bytes per crossing of a record boundary. Numeric data items occupy 8 bytes.
KEY	18½ bytes plus 3 bytes per definition, plus 1 byte per character in assigned string(s).

(Continued on next page.)

(Continuation of System Memory Requirements Table)

Item	Memory Required
LEX	18½ bytes plus file's data size. Requires an additional 5½ bytes of system overhead for each LEX file in memory.
BIN	24½ bytes plus file's data size.
SDATA	18½ bytes plus 8 bytes for each record.
Plug-In ROM or RAM Device	5 bytes per device plus 5½ bytes of system overhead for each LEX file, if any.
Subprogram Calls	39½ bytes plus 9½ bytes for each numeric parameter, plus 9½ bytes for each string parameter passed by reference, plus, for each string parameter passed by value, 2 bytes plus one byte per character.
User-Defined Function Calls	50 bytes plus 9½ bytes per numeric parameter plus, for each string parameter, 2 bytes plus one byte per character. The function name itself requires an added amount of memory (as if the name were a passed parameter).
Subroutine Calls	3 bytes each.
FOR...NEXT Loop Configuration	20½ bytes for each loop.
Pre-allocated System RAM	755 bytes minimum. Add 5 bytes for every ROM, RAM, or memory-mapped I/O device plugged in.
Variables	
Simple Numeric	9½ bytes, regardless of type.
Simple String	11½ bytes, plus 1 byte per character in a maximal string. (For example, a default length simple string variable consumes 43½ bytes, regardless of how many characters the variable contains.)
Numeric Array	9½ bytes, plus: <ul style="list-style-type: none">• 8 bytes per REAL number.• 4½ bytes per SHORT number.• 3 bytes per INTEGER number.
String Array	9½ bytes, plus 2 bytes for each element, plus the maximum element string length times the number of elements.

Memory Usage During Evaluation of Expressions

The HP-71 stores expressions in postfix form. During evaluation, the computer copies operands to a stack (maintained by the operating system), which temporarily consumes user-available memory. Each numeric operand requires eight bytes; each string operand requires eight bytes, plus one byte for each character in the string.

Substrings are extracted by a postfix operation which requires the entire parent string to be present on the stack. For this reason, it may be preferable to store related data fields as smaller elements of a string array instead of grouping these fields in a single, large string from which substrings are extracted.

Mathematical Discussion of HP-71 Statistical Arrays

Matched Samples

A sample is a collection of observations of a random variable. A matched sample consists of one or more samples where each observation in a sample is matched with an observation in each of the other samples. Each sample has the same number of elements, which is denoted by N . $NVAR$ will denote the number of variables (samples). A matched sample data set can then be visualized as a table with N rows and $NVAR$ columns.

Sample Set of Matched Data

Element	Variable Number							
	1	2	...	j	...	k	...	$NVAR$
1	$x(1,1)$	$x(1,2)$...	$x(1,j)$...	$x(1,k)$...	$x(1,NVAR)$
2	$x(2,1)$	$x(2,2)$...	$x(2,j)$...	$x(2,k)$...	$x(2,NVAR)$
.
.
.
i	$x(i,1)$	$x(i,2)$...	$x(i,j)$...	$x(i,k)$...	$x(i,NVAR)$
.
.
.
N	$x(N,1)$	$x(N,2)$...	$x(N,j)$...	$x(N,k)$...	$x(N,NVAR)$

Each row of this table represents a point in $NVAR$ -dimensional space and will be called a data point. A data point can be considered an observation or realization of an $NVAR$ -dimensional random variable, resulting in N such realizations.

Summary Statistics

For the purposes of performing statistical operations and functions, the HP-71 does not need to store the entire data set. Instead, the computer reduces, or summarizes, the data in the following way. Let $x(ij)$ represent the entry in row i and column j for $i=1,2,\dots,N$ and $j=1,2,\dots,NVAR$. The summary statistics are then:

$$\begin{aligned} &N \\ &NVAR \\ &T(j) = \sum_i x(ij) \quad j=1,2,\dots,NVAR \\ &S(jk) = \sum_i [x(ij) - T(j)/N][x(ik) - T(k)/N] \quad j,k=1,2,\dots,NVAR \end{aligned}$$

$T(j)$ represents the sums of the columns and the $S(jk)$ represents the sums of squares and cross-products of the mean-adjusted variables.

Some calculators and handheld computers with statistical functions accumulate the sums of squares and sums of cross-products of the unadjusted variables,

$$\sum_i x(ij)x(ik),$$

rather than the $S(jk)$. Three advantages to using the mean-adjusted sum $S(jk)$ are:

1. It reduces the potential for loss of significance errors when the data points have non-zero means.
2. Calculations based on mean-adjusted values are faster than those based on the unadjusted ones.
3. It is easier to use sample means, variances, and correlations as inputs in place of the original data.

The `STAT` statement reserves space for these summary statistics by dimensioning a statistical array. This array has one dimension and has length $(NVAR+1)(NVAR+2)/2$. $NVAR$ is stored in the statistical array's internal representation. The other statistics are stored as

$$(N, T(1), S(11), T(2), S(12), S(22), \dots, S(NVAR, NVAR)).$$

Multiple matched samples can be stored simultaneously and analyzed in any order by using more than one statistical array.

The `STAT` statement specifies the current statistical array. The HP-71 contains only one current statistical array at a time. All other statistical arrays are preserved, however, like any array or simple variable, up to the limit of memory. A noncurrent statistical array can be made current by specifying it with a `STAT variable name` statement.

Recursive Calculation of Statistics

A data point $V = (V(1), \dots, V(NVAR))$ is “added” to or “dropped” from the current data set using the `ADD` and `DROP` statements, respectively. The *NVAR* numeric expressions in an `ADD` or `DROP` statement are evaluated and are interpreted as the coordinates of the data point. If less than *NVAR* numeric expressions are included, the missing trailing expressions are assumed equal to zero.

`ADD` updates the summary statistics according to:

```
If N < 0 then display ERR:Invalid Statistic and stop.
For k = 0 to NVAR
  For j = 1 to k (skip if k = 0)
    If N=0 then S(jk): = 0
      else S(jk): = S(jk)+(N*V(j)-T(j))(N*V(k)-T(k))/[N(N+1)]
  Next j
  T(k): = T(k)+V(k)
Next k
N: = N+1
End.
```

`DROP` updates the summary statistics according to:

```
If N < 0 or 0 < N < 1 then display ERR:Invalid Statistic and stop.
If N = 0 then display ERR:Invalid Stat Op and stop.
For k = 0 to NVAR
  For j = 1 to k (skip if k = 0)
    If N = 1 then S(jk): = 0
      else S(jk): = S(jk)-(N*V(j)-T(j))(N*V(k)-T(k))/[N(N-1)]
  Next j
  T(k): = T(k)-V(k)
Next k
N: = N-1
End.
```

The `CLSTAT` statement sets the elements of the current statistical array to zero. This prepares the statistical array for accumulating statistics for another data set.

Simple Linear Regression

The simple linear regression model is:

$$X(j) = a + b * X(k) + e,$$

where $X(j)$ is the dependent variable, $X(k)$ is the independent variable, a and b are constants to be determined (estimated), and e represents random errors (uncorrelated with zero mean and with unknown but constant variance). The constants a and b are determined by the method of least squares. That is, they are chosen to minimize the residual sum of squares:

$$\sum_i [x(ij) - a - b * x(ik)]^2.$$

The solution is:

$$b = S(jk)/S(kk), \text{ and}$$

$$a = [T(j) - b * T(k)]/N.$$

Notice that a (constant) random variable equal to one and having the coefficient a is implicitly present in the regression model. This interpretation can be quite useful when adding and dropping variables (or terms) to or from multiple linear regression models.

The mean-adjusted sum of squares for this constant variable and any mean-adjusted sum of cross-products involving this variable are zero. The sum of the values for this variable is N . Therefore, no additional summary statistics need be accumulated in order to implicitly include this variable in the data set.

For these reasons, this random variable (numbered zero) will always be considered present in a data set, and zero will be considered a valid variable number for all statistical statements and functions, except where explicitly stated otherwise.

The `LR` statement specifies the dependent and independent variables. Any variable number in $[0, NVAR]$ is allowed to be the dependent variable, and any variable number in $[1, NVAR]$ is allowed to be the independent variable.* These variable numbers are stored by the computer.

If one or two variable names are included in the `LR` statement, the estimated intercept a is stored in the first variable and the estimated slope b is stored in the second variable, if appropriate.

* For a definition of “variable number” refer to the table on page 334.

The IEEE Proposal for Handling Math Exceptions

Introduction

The IEEE Radix Independent Floating-Point Proposal divides all of the floating-point “exceptional events” encountered in calculations into five classes of *math exceptions*: invalid operation, division by zero, overflow, underflow, and inexact result. Associated with each math exception is a flag that is set by the HP-71 whenever an exception is encountered. These flags remain set until you clear them. Each of these flags can be accessed by its number or by its name.

Setting and Clearing Math Exception Flags

You can clear and set the math exception flags in the same way as any flag, except that flag names can be used as well as flag numbers.

Examples. Both of these statements clear the invalid and inexact flags.

```
CFLAG IVL,INX
CFLAG -8,-4
```

The keyword `MATH` specifies all math exception flags as a group. It can be used with the `SFLAG` and `CFLAG` statements.

The Five Math Exception Flags

The following describes the conditions that set each of the math exception flags:

- **IVL.** *Invalid* operations are those for which no real value can reasonably be returned as a result. Some examples that set the IVL flag are `SQR(-1)`, `LOG(-1)`, `ACOS(2)`, and `0/0`.
- **OVZ.** The *division by zero* flag is set whenever finite operands give rise to an exact infinite result. Typical examples: `1/0`, `LN(0)`, `TAN(90)` in degrees, and `0^(-5)`.
- **OVF.** The *overflow* flag is set when:
 1. The finite 12-digit rounded result r of an operation satisfies $|r| > \text{MAXREAL}$, such as $r = 1\text{E}499*10$.
 2. A `SHORT` variable is assigned a finite value r such that the five-digit rounded value r' satisfies $|r'| > 9.9999\text{E}499$, such as $r = 9.99996\text{E}499$.

3. An INTEGER variable is assigned a finite value r which, when rounded to an integer r' , satisfies $|r'| > 99999$, such as $r = 100000$.
 4. A decimal string representing a value greater than MAXREAL is converted to a numeric value, such as VAL("1E500").
- UNF. The *underflow* flag is set when:
 1. The 12-digit rounded result r satisfies $0 < |r| < 1\text{E}-499$ and will not be represented exactly in its destination. For instance, $1\text{E}-499/3$ sets the UNF flag, although $1\text{E}-499/100$ depends on the default value dictated by the TRAP(UNF) setting. If TRAP(UNF) = 2, then the denormalized result .001E-499 is exact, and UNF is not set. (The next topic covers default values.)
 2. A SHORT variable is assigned a value r such that the five-digit rounded result r' satisfies $0 < |r'| < 1\text{E}-499$ and will not be represented exactly in its SHORT destination.
 3. A decimal string representing a number less than EPS is converted to a numeric value.
 - INX. Generally, the *inexact* flag is set if and only if the rounded real result of an operation is not identical to the exact result. This is true for most HP-71 functions, including all functions specified by the IEEE Proposal (+, -, *, /, SQR, and RED). For example, $1/3$, SQR(5) and $1 + 1\text{E}-100$ all set INX while $1/4$, SQR(16), and $1 + 1\text{E}-10$ do not set INX.

However, for some compound operations, such as X^Y , and for most statistics operations, it is impractical to determine exactness in all cases. Thus, for these compound and statistics operations, the following is true: if INX is not set, the result is exact, but some exact results may (overcautiously) set INX. This is also the rule for any sequence of HP-71 operations. If INX is not set by a sequence of operations, then no rounding errors have occurred, and you can be sure that your result is exact. But if INX was set by any of the operations, it is not necessarily true that the result is inexact. For example, the statement $Y = \text{SQR}(5) / \text{SQR}(X)$ sets INX even when $X = 5$. This is because the HP-71 reacts to the rounding errors committed in SQR(5) and SQR(X), but, of course, does not detect the fact that they compensate each other. Although some exact calculations will set INX, these cases have been kept to a minimum so that the flag will retain its usefulness in more complicated calculations.

Extended Default Values

When exceptions occur, it is sometimes desirable to continue the calculation by using a default value for the result. The HP-71 extends the normal range of floating-point values to include, as proposed by the IEEE, special default results which can be used with relative safety for each exception.

By setting the appropriate traps with the TRAP function or with the DEFAULT statement, you can select the action of continuing the calculation with a choice of default values or of suspending the program with an error message.

A `TRAP` value of `2` causes all math exceptions to return the new default values introduced by the IEEE Proposal.

Default Values with `TRAP` Value of `2`
and `OPTION ROUND NEAR` Active

Exception	Default Value	Comment
<code>IVL</code>	<code>NaN</code>	Not a Number, explained below.
<code>DVZ</code>	<code>Inf</code> or <code>-Inf</code>	} Mathematically exact <i>infinity</i> , explained below.
<code>OVF</code>	<code>Inf</code> or <code>-Inf</code>	
<code>UNF</code>	<code>0</code> , <code>-0</code> , or a Denormalized Number	Explained below.

Not a Number

When an invalid operation (`IVL`) occurs, the IEEE default result is the value “Not a Number,” displayed as `NaN`. This value conveys the information that an invalid operation has occurred and the result of the calculation is not a number.

Example. When you enter the following three concatenated statements, the display will first show the current trap value associated with the `IVL` flag, whatever it happens to be. This example assumes the current trap value is `1`. Then the two following statements will display the outputs shown.

Input/Result

<code>TRAP(IVL,2) @ Z=0/0 @ DISP Z</code>	
<code>1</code>	<code>TRAP(IVL,2)</code> returns the current trap value and sets <code>2</code> as the new trap value for <code>IVL</code> .
<code>WRN:0/0</code>	<code>Z=0/0</code> returns a warning, since the current <code>IVL</code> trap value is now <code>2</code> .
<code>NaN</code>	<code>DISP Z</code> returns the value of <code>Z</code> .

If you do not wish to preserve an `IVL` trap value of `2`, execute `TRAP(IVL,1)` or `TRAP(IVL,0)`.

A `NaN` value propagates through the usual math operations so that it generally appears in the final result. For example, using `Z` from the above example, `3 * Z + 1` returns the original `NaN` with no exceptions being raised. In the event that two `NaN`s enter one operation (for example, `Z + Z`), then the result will also be `NaN`. Exceptions to this are logical operations and numeric comparisons, which always return `0` or `1`.

`NaN` can be stored in a `SHORT` or `INTEGER` precision variable.

Since `NaN` has no ordering with the other `REAL` numbers, when `NaN`s are involved in certain comparisons the `IVL` flag will be set. Refer to the topic “The Unordered Comparison Operator” on page 343.

The `NaN`s described above are called Quiet `NaN`s. There is also another type of `NaN` called a Signaling `NaN`. This `NaN` is created directly by the user when the `NaN` function is executed, and at this time no exception results. Later, a Signaling `NaN` will set the `IVL` flag whenever it is first encountered in a math operation. The Signaling `NaN` then becomes a Quiet `NaN`. The Signaling `NaN` can be used to initialize any uninitialized data so that the `IVL` flag will be set whenever this data unexpectedly enters into a calculation.

Infinity

When the `DVZ` or `DVF` exception occurs, the IEEE Proposal specifies default values `Inf` and `-Inf`. These behave like mathematical infinity in subsequent operations. For example, `Inf/1E200 = Inf` and `-2/0 = -Inf`.

The `INF` function returns the value `Inf`. `Inf` can be stored in an `INTEGER`, `SHORT`, or `REAL` variable.

Denormalized Numbers and `-0`

On many computers, the simple test for $x=y=0$ does not guarantee that $x=y$. For, when $|x|$ and $|y|$ are small enough to be near the underflow threshold, $x-y$ may underflow to 0 without having equality. For instance, with `DEFAULT ON` active, `1.2345E-497 - 1.234E-497` produces a `WRN:Underflow` warning, followed by 0.

The IEEE Proposal specifies that gradual underflow be implemented. This requires that small results be denormalized to the minimum exponent of the number system. Gradual underflow is implemented when `DEFAULT EXTEND` is in force. With `DEFAULT EXTEND` active, the above example `1.2345E-497 - 1.234E-497` does not underflow. Rather, the HP-71 returns the denormalized result `0.05E-499`.

Here's another example: `1.2345E-499*1E-9` displays, after the warning, the rounded, denormalized result `0.00000000123E-499`, with `OPTION ROUND NEAR` active. The exponent of the normalized form of this result is given by the `EXPONENT` function. Executing `EXPONENT(RES)` gives `-508`.

When the magnitude of the rounded result is smaller than $1*10^{-510}$ (`OPTION ROUND NEAR` active), it underflows to 0 and the `UNF` flag is set. To retain more information about the underflow, the value 0 preserves the sign of the underflowed value. For instance, a result of $-1*10^{-600}$ would underflow to `-0`.

This extra information in the sign of zero can be useful for carefully written programs designed to handle exceptions like `1/(-0)` automatically. Otherwise, in unexceptional calculations, `0` and `-0` behave in the same way. They even compare equally, so that `-0=0` is true and `-0<0` is false.

The sign of zero will normally be preserved through functions $F(x)$, where $F(0)=0$. For example, `SQR(0)` returns `0` and `SQR(-0)` returns `-0` (recall that `-0` is not less than `0`).

Classes of Numbers

The inclusion of `TRAP 2` default values for math exceptions extends the normal range and type of numbers. This extended range is divided into 12 classes as shown in the table below.

Classes of Numbers	
Value or Range of x	Class of x
<code>0</code> or <code>-0</code>	1 or -1
Denormalized ($\text{MINREAL} \leq x < \text{EPS}$) ($-\text{MINREAL} >= x > -\text{EPS}$)	2 -2
Normalized ($\text{EPS} \leq x < \text{Inf}$) ($-\text{EPS} >= x > -\text{Inf}$)	3 -3
<code>Inf</code> or <code>-Inf</code>	4 or -4
Quiet $\pm \text{NaN}$	5 or -5
Signaling $\pm \text{NaN}$	6 or -6

Examples. The value returned by `CLASS` identifies the class of numbers to which its argument belongs. The sign of the result indicates the sign of the argument. The returned values are shown in `STD` display format with `DEFAULT EXTEND` set. While you can execute `STD` and `DEFAULT EXTEND` only in `BASIC` mode, you can execute `CLASS` in either `BASIC` or `CALC` mode.

```

CLASS(-EPS/100)           Returns -2.
CLASS(4.378E35)           Returns  3.
CLASS(-4773)              Returns -3.
CLASS(INF)                Returns  4.
```


The Unordered Comparison Operator

The introduction of NaN to represent the result of an invalid operation requires a new comparison operator, `?`, to complement the usual relational operators `<`, `=`, and `>`. The comparison `X?Y` is true if and only if one (or both) of `X` and `Y` is NaN. With any comparison of `X` and `Y` exactly one of four conditions is true:

1. `X` is less than `Y`.

(`X<Y` is true).
2. `X` is equal to `Y`.

(`X=Y` is true).
3. `X` is greater than `Y`.

(`X>Y` is true).
4. `X` is unordered with `Y`.

(`X?Y` is true).

One consequence of NaN is that `X<>Y` is no longer identical to `X#Y`, as the following table shows:

Differences Between the Operators `<>` and `#`

Comparison	Equivalent To
a. <code>X<>Y</code>	<code>X<Y OR X>Y</code>
b. <code>X#Y</code>	<code>X<Y OR X>Y OR X?Y</code>
c. <code>NOT(X=Y)</code>	<code>X<Y OR X>Y OR X?Y</code>

If either `X` or `Y` is NaN, *a* is false but *b* is true. Note that if `X` is NaN, then the comparison `X=X` is false. In fact, the comparison `X=X` can be used to test if `X` is a NaN. The `CLASS` function (page 49) can also do this test.

The invalid (IVL) flag is set for numerical comparisons whenever the expressions being compared are unordered and the relational operator being used to compare the expressions contains `<` or `>` but not `?`. That is, the IVL flag is set when the expressions are “unexpectedly” unordered. Note that even though the flag is set, no NaN is created.

Example. Assume the current trap value for the invalid flag is 1. This example assigns a trap value of 2 for the invalid flag so the HP-71 will respond to an invalid operation with a warning and a default value rather than an error and an operation halt.

Input/Result

```
TRAP(IVL,2) @ 0=0/0 @ R=5 END LINE  
  
1  
  
WRN:0/0
```

The HP-71 displays the current trap value for the invalid flag.

Next, the computer warns that an invalid operation has occurred and sets the IVL flag.

Input/Result

```
DISP Q [END LINE]
NaN
```

```
C=Q<>R @ DISP C [END LINE]
```

```
WRN:Unordered
```

```
0
```

The result of $0/0$ is “Not a Number,” since the trap value for the IVL flag is 2.

Display the answer to the question: is Q less than or greater than R ?

The computer sets the IVL flag and warns that a comparison involving at least one NaN has been attempted.

C is assigned 0, not NaN. The comparison is false. Since Q is a NaN, the only true comparison would be one involving the unordered comparison operator, $?$.

Table of Comparisons (X Compared to Y)

The table shown below illustrates the ordering relationships among values represented by \pm infinity, finite positive and negative numbers, \pm zero, and NaN (not-a-number). The symbols used in the table are described in the box to the right. In the table itself:

<	Less Than
=	Equal To
>	Greater Than
?	Unordered

n = a real negative number.
 p = a real positive number.

True Results of Comparing X to Y

		Y						
		-Inf	<i>n</i>	-0	+0	<i>p</i>	+inf	NaN
X	-Inf	=	<	<	<	<	<	?
	<i>n</i>	>	=	<	<	<	<	?
	-0	>	>	=	=	<	<	?
	+0	>	>	=	=	<	<	?
	<i>p</i>	>	>	>	>	=	<	?
	+Inf	>	>	>	>	>	=	?
	NaN	?	?	?	?	?	?	?

Example using the Relational Operator Table. Suppose that you wish to evaluate $x \leq y$, where $x = -37.3$ and $y = -0$. Since the x -value corresponds to row 2 (***n***) of the table and the y -value corresponds to column 3(***-0***) of the table, the tabular result is ***<***. Thus, the table indicates that, in this case, $x \leq y$ is true.

Glossary

Introduction

This glossary describes terms used in both the *HP-71 Reference Manual* and the *HP-71 Owner's Manual*. If a term you are looking for is not described in the glossary, refer to the *HP-71 Owner's Manual* index.

Special Symbols

&	String concatenator.
*	Used to close a channel. (Refer to the ASSIGN# keyword entry on page 23.)
␣	Separates statements in multistatement lines.
!	Allows end-of-line comments in program lines.
?	Two special uses: <ul style="list-style-type: none">• The default <code>INPUT</code> prompt.• Used as a relational operator. (See unordered.)
~~	Displayed in lieu of a line number to indicate an execution error in a non-BASIC program.
E	Two uses: <ul style="list-style-type: none">• Exponent—precedes a power of 10 in a floating point number.• Indicates a private and secure file in a catalog entry (an “execute only” file).
P	Indicates a private file in a catalog entry.
S	Indicates a secure file in a catalog entry.

Terms

A

accessory: Devices or modules that plug directly into the HP-71, such as the HP 82400A Magnetic Card Reader or the HP 82420A 4K Memory Module.

accuracy factor: See **adjustment factor**.

active environment: The currently accessible local environment.

actual parameter: An expression or variable that is passed as an argument to a subprogram or function. In a `CALL` statement, an actual parameter can also be a channel number (preceded by #). See also **formal parameter**.

address space: The range of memory locations the computer can access. The HP-71 address space is 1,048,576 four-bit locations.

adjustment factor: The amount of correction applied to the internal clock of the HP-71. The adjustment factor is expressed as the number of seconds the HP-71 waits before adding (or subtracting) one second to correct a slow (or fast) clock. Sometimes referred to as *accuracy factor*. Refer to the **EXACT** entry in the Keyword Dictionary.

adjustment period: The period delimited by successive executions of `EXACT`. The computer uses this period to calculate the adjustment factor.

alternate character set: A set of user-defined characters that are represented by the ASCII character codes 128 through 255. The HP-71 alternate character set is entirely user-definable.

annunciators: Symbols that appear in the left and right ends of the display window to indicate certain machine conditions.

ANSI BASIC Standard: A standard for the BASIC language that was developed by the American National Standards Institute. Two standards have been developed. The first was ANSI X3.60-1978 and is referred to as “ANSI Minimal BASIC.” The second, referred to as the Level 1 standard, was proposed by ANSI subcommittee X3J2. As of the original printing of this manual, the second standard was not yet adopted.

argument: A parameter.

arithmetic operator: One of the elementary operators: $+$, $-$, $*$, $/$, $^$, $\%$, and `DIV`.

array: A variable containing an ordered collection of values. Each value is termed an *element*. An element is specified by an array name followed by a list of one or two subscripts enclosed in parentheses. (The rules governing numeric and string array names are the same as those for numeric and string variable names.) All elements of an array are of the same data type. See also **numeric array** and **string array**.

array base: The lowest-numbered subscript that can be used to reference an element of an array. The base of an array is automatically set at the time that you create the array and is determined by the current `OPTION BASE` setting (1 or 0). The HP-71 default `OPTION BASE` setting is 0.

array element: One of the values in an array. An array element is referenced by array name and element *subscript(s)*.

ASCII: The American Standard Code for Information Interchange. This is a standard used by the computer industry to represent characters by numeric values. This code enables different types of computers to exchange information. Each HP-71 character, either built-in or user-defined, corresponds to a decimal code in the range 0 through 255.

assignment statement: A `LET` statement or an implied `LET` statement used to give a variable a value.

available memory: The part of RAM that is not currently being used to hold files, variables, arrays, or any system control information. (`MEM` returns the amount of available memory in main RAM, in a specified plug-in memory device, or in an independent RAM.)

B

base: See **array base**.

base part: See **mantissa**.

BASIC: Beginner's All-purpose Symbolic Instruction Code. BASIC is the HP-71's programming language. Also refers to a file type.

BASIC mode: The computer mode in which you can write BASIC programs and perform most keyboard operations. It is distinct from CALC mode.

binary operator: An operator that performs its operation on two expressions. It is placed between the two expressions. The following binary operators are available:

Arithmetic	Relational	Logical
+	#	AND
*	=	OR
-*	<>	EXOR
DIV	<	
/	>	
^	<=	
%	>=	
	?	
* The "-" operator is a unary operator that can be used as a binary operator.		

See also **unary operator**.



bit: The smallest unit of memory, equivalent to one binary digit. A bit can have one of two values: 0 or 1.

bit pattern: The pattern formed in the HP-71 display by a group of binary digits (bits). Each bit represents one dot on the display. (The bit pattern is described under “Defining Alternate Characters” on page 132 in the *HP-71 Owner’s Manual*.)

branch: To transfer program execution to a specified program statement.

byte: A standard unit of memory equivalent to eight binary digits (bits). Depending on the value of each bit, a byte can have a value of 0 through 255. Each ASCII character occupies one byte of memory. See also **bit** and **nibble**.

C

CALC mode: A mode in which you can perform keyboard calculations and view intermediate results. Pressing   switches the computer between CALC mode and BASIC mode.

calling environment: The local environment of a program or subprogram that has referenced a user-defined function or called a subprogram. When a program or subprogram calls a subprogram, the calling environment is maintained in an inactive status until the called subprogram ends. See also **active environment**, **environment**, **global environment**, **local environment**, and **main environment**.



calling program: A program that executes another program or subprogram using the `CALL` statement.

calling statement: A statement that transfers execution to a subroutine (`GOSUB`) or a subprogram (`CALL`). When a subroutine or a subprogram ends, program execution returns to the statement following the calling statement.

CARD: A keyword that specifies the HP 82400A Magnetic Card Reader as a device. This keyword is used in `COPY` or `CAT` statements.

card file: A file that resides on one or more magnetic cards.

carriage-return (CR): A control character (character code 13) that causes the cursor to return to the left edge of the display.

carriage-return/line-feed (CR/LF): A sequence of two characters normally generated by the termination of keyboard entries and `PRINT` and `DISP` statements. The  and  keys send carriage-return/line-feeds to display devices.

catalog entry: A display line of information showing the name, protection, type, length, creation date and time, and port (if any) of a file in memory or on a mass storage medium.

channel: A path through which the computer stores and retrieves information from a data file. A channel is assigned a number and is associated with a file when you execute `ASSIGN#`.

channel number: A number assigned to a channel at the time the channel is associated with a specific data file.

character: An elementary symbol, such as a letter, numeral, punctuation mark, or other special symbol that can be displayed on the HP-71. The HP-71 has 128 predefined ASCII characters and 128 user-definable characters.

character code: The numeric value, ranging from 0 through 255, associated with a character. For example, the character code for `A` is 65. (Refer also to “HP-71 Character Set and Character Codes,” pages 322 through 326.)

character delay: The length of time between successive horizontal scrolls in the display. When the computer displays a line longer than 22 characters, it scrolls the line through the display one character at a time, from right to left. The character delay is the length of time a portion of that line is displayed before the computer scrolls it one character. This value can be set using the `DELAY` statement.

CHARSET string: A group of characters whose character codes represent a bit pattern for a user-defined character set. This string is an argument the user supplies to the `CHARSET` statement.

close a file: To dissociate a file from a channel. You can close a file by executing `ASSIGN#`.

command stack: A portion of main RAM that stores the five most recent commands entered from the keyboard. You can access the command stack by pressing `[9]` `[END LINE]`.

concatenate: To join string expressions (with `&`) or to join statements (with `@`) on a single line. (Some statements cannot be followed by `@`. The Keyword Dictionary entries for such statements include this information.)

conditional branch: A type of conditional execution in which a program branch occurs as a result of a conditional test.

conditional execution: The execution of a statement or statements based on the outcome of a conditional test.

conditional test: A test based upon a logical expression or an implied comparison between two values, as in `IF` or `ON...GOTO/GOSUE/RESTORE` statements.

continuous on: A condition in which the automatic shut-off feature of the computer is disabled. This condition can be set on the HP-71 by setting flag -3.

contrast value: The argument used by the `CONTRAST` statement to adjust the viewing angle of the display.

control character: One of 32 characters that controls the operation of a printer or display. The character codes for the control characters are 0 through 31.

CR: See **carriage return**.

CR/LF: See **carriage return/line feed**.

current file: The file that receives the program lines you type into the computer. Also, the current file is the default file for the following operations:

- Pressing the `[RUN]` key.
- Executing `RUN` without specifying a file name.
- Performing most file operations when no file is specified.

Note: The “current file” designation can be changed by `RUN`, `EDIT`, `CHAIN`, `CALL`, `PURGE`, `TRANSFORM`, `CLAIM PORT`, and `FREE PORT`.

current key assignments: The key definitions in the system `keys` file. These definitions are the assignments that become active when the User keyboard is active (**USER** annunciator displayed).

current line: The program line at which the computer is positioned. You can display the current line by executing `FETCH` with no argument. If a program is paused, the line containing the suspend statement (the statement at which execution will resume if you press `[f][CONT]`) is the current line. When you run a program, the current line (as displayed by executing `[f][FETCH]` or `FETCH` without an argument) does not change unless the program is paused by `[ATTN]`, `PAUSE`, or `[SST]`, or if an error or warning condition occurs. If the program is paused, the current line contains the statement at which execution will resume if you press `[f][CONT]`. When a running program is suspended by `PAUSE`, the current line is the line containing the statement after the `PAUSE`.

current statistical array: The statistical array most recently selected by executing `STAT`. You can define several statistical arrays, but you can perform statistical operations on only the current statistical array.

cursor: A blinking symbol that indicates the point on the display line at which characters can be entered or deleted.

D

data item: A numeric expression, string expression, or unquoted string contained in a `DATA` statement.

data pointer: The computer's internal mechanism for indicating the next `DATA` item to read. You can reposition the data pointer using `RESTORE`. See also **file pointer**, **pointer**, and **program pointer**.

data type: A category in which a data item falls. HP-71 data types are:

- String
- Real
- Integer
- Short

Also, a simple variable differs in data type from an array variable.

debug: To locate and correct errors (particularly logical errors) in a program.

default setting: A setting (such as the display contrast value) that the HP-71 uses until you specify a different setting.

default value: A value supplied by the HP-71 in either of the following cases:

- If an optional parameter is not specified in a statement or function.
- If an improper operation has occurred, thus requiring a substitute value so that execution can continue.

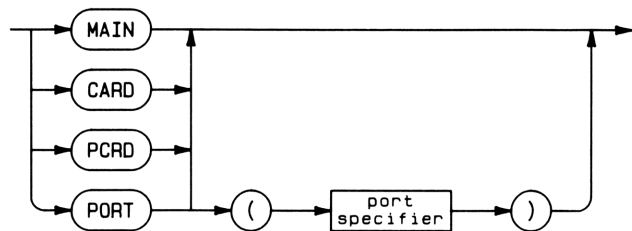
degrees setting: The `OPTION ANGLE` setting used for expressing trigonometric function arguments and results in degrees.

delimiter: A character, such as a comma, that separates items in an input list or separate arguments supplied to a statement.

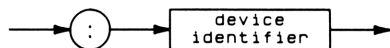
denormalized number: A floating point number that has a true exponent less than -499 . (Refer to the section entitled "The IEEE Proposal For Handling Math Exceptions"—page 338—for more information about denormalized numbers.) See also **gradual underflow**.

destination file: The file that is created or modified as the result of a copy, merge, or transform operation.

device identifier: A special system device word. `MAIN` refers to main RAM; `CARD` and `PCRD` refer to the optional magnetic card reader; `PORT` refers to plug-in memory and independent RAM. The `PORT` device word can be followed by an optional port specifier for referencing a particular plug-in port.



device specifier: A colon (:) followed immediately by a device identifier.



digit: One of the following characters: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, for decimal numbers. Also includes A, B, C, D, E, F, for hexadecimal numbers.

dimension: Used as a noun to refer to either the maximum length of a string variable or the number of elements in an array. If an array's base is 0, the number of elements is equal to the maximum subscript value plus 1. If an array's base is 1, the number of elements is equal to the maximum subscript value. "Dimension" is also used as a verb to describe the execution of `DIM` to set the dimension of a string variable or an array.

direct execute key: A key or keystroke combination that, when pressed, executes one or more instructions without altering the display. Refer also to the `DEF KEY` entry in the Keyword Dictionary.

display format: The way that numeric information appears in the display. Refer to the `FIX`, `SCI`, `ENG`, and `STD` entries in the Keyword Dictionary.

display line: The line currently in the HP-71 display. It can be up to 96 characters in length, minus zero or more character positions for the prompt. The display *window* displays up to 22 characters at a time and can scroll back and forth along the display line.

dummy array variable: A formal parameter that is indicated by a variable name followed by `()` for one dimension or `(,)` for two dimensions.

E

- editing:** Using keyboard operations to add, delete, or alter information in either the display or a file.
- end-of-file mark:** An internal marker placed by the computer to indicate the end of information in certain files.
- end-of-line sequence:** The sequence of characters indicating the end of a line. The sequence usually consists of a carriage return and a line feed, and is sometimes called the *end-of-line indicator*. You can use the `ENDLINE` statement to set an end-of-line sequence for the `PRINT` and `PLIST` statements.
- entry slot:** The left-hand opening of the card reader slot. Magnetic cards are inserted through this opening.
- environment:** The combination of the local and global environment that can be accessed by a program or subprogram. See also **active environment**, **calling environment**, **global environment**, **local environment**, and **main environment**.
- EOL indicator:** See **end-of-line sequence**.
- EPS:** The smallest normalized number representable on the HP-71: $1.0E-499$.
- error condition:** A condition in which the computer cannot perform an operation. An error condition results in an error message. For further information, see section 9, Error Conditions, in your *HP-71 Owner's Manual*.
- error message:** A message the computer associates with a given error. You can view the most recent error message using `ERRM#` or `[9][ERRM]`. In some cases, a warning message occurs instead of an error message. For further information, see section 9, Error Conditions, in your *HP-71 Owner's Manual*.
- escape character:** The control character whose character code is 27. Used as the first character in an escape sequence. This character can be generated on the HP-71 by pressing `[9][CTRL][9][I]` or by evaluating `CHR#(27)`.

escape sequence: A sequence of characters, the first being the escape character, that controls the operation of a display or printer. Escape sequences that control the HP-71 display are listed on page 328.

evaluate: To compute the value of an expression. The result is always a string or numeric value.

exception flag: See **math exception flag**.

exit slot: The right-hand opening of the card reader slot. Magnetic cards exit through this opening.

exponent: See **floating point number**.

expression: See **numeric expression** and **string expression**.

external medium: See **mass storage medium**.

F

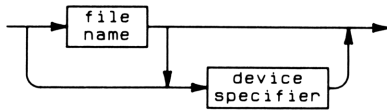
field specifier: A sequence of one or more symbols used in a format string to indicate the printer or display format for a data item.

file: A set of lines or other data in memory or on a mass storage medium. A file can be manipulated as a single unit and has a unique name.

file name: A string of one to eight characters that specifies a file in memory. The first character must be a letter. Remaining characters may be any combination of letters and digits. The HP-71 automatically changes lowercase letters to uppercase letters. In general, file names can be specified by string expressions or unquoted strings. A blank character terminates a file name. See also **reserved word**.

file pointer: An internal mechanism the computer uses to indicate the position in an open file where the next read or write operation on that file will occur.

file specifier: An unquoted string or a string expression that, when evaluated, indicates the name of a file and, optionally, the device in which the file is located.



file type: The characteristic format of a file. A file type is indicated by an unquoted string. The HP-71 recognizes the following file types:

- BASIC (BASIC program).
- BIN (machine language program).
- DATA (fixed-length record data file).
- KEY (key definition file).
- LEX (language extension file).
- LIF1 (same format as TEXT).
- SDATA (serial data file from the HP-41).
- TEXT (variable length record file of character data).

final character position: The position of the last character to be replaced by an assignment to a string variable. Also refers to the last character of a substring. A value greater than the variable's current string length is interpreted as equal to the current string length. A value less than the start character position is interpreted as specifying a null substring immediately preceding the start character position. The default value is the current string length. See also **start character position**.

flag: An internal system variable that can have two possible values, 0 and 1, which correspond to the "clear" and "set" states. Refer to the **FLAG**, **CFLAG**, and **SFLAG** entries in the Keyword Dictionary.

floating-point number: A number represented internally by the HP-71 in the decimal format

$$m * 10^e$$

where m is the mantissa, which is a 12-digit number in the range

$$1 \leq |m| \leq 9.999999999999$$

for normalized numbers, and

$$|m| < 1$$

for denormalized numbers;

and where e is the exponent, which is a 3-digit integer in the range

$$0 \leq |e| \leq 499.$$

formal parameter: A variable name that appears in a `SUB` or `DEF FN` statement. A formal parameter either references or assumes the value of an actual parameter passed to it. You can specify a channel number as a formal parameter in a `SUB` statement by using `#` followed by an integer constant in the range of 1 through 255.

format string: The characters in an `IMAGE`, `DISP USING`, or `PRINT USING` statement that specify the formatting of information sent to display and printer devices.

free a port: To use `FREE PORT` to set aside a portion of main RAM as independent RAM.

function: A built-in routine that can operate on arguments and produces a single string or numeric value. A *user-defined* function is a function that is defined in a BASIC program using the `DEF FN` statement.

G-H

global environment: The set of files, flags, and other settings that are accessible to any program, subprogram, or user-defined function. See also **active environment**, **calling environment**, **environment**, **main environment**, and **local environment**.

gradual underflow: A process by which numbers too small to be represented in normal floating point format are represented instead by the minimum allowable exponent and a mantissa less than 1 (for example, 0.0123E-499). These values are called *denormalized numbers*.

hexadecimal address: A location in memory specified by a base 16 number.

hexadecimal digit: A digit 0 through 9 or a letter A through F.

HH:MM:SS: Hours, minutes, and seconds.

hierarchy: A prescribed order in which data is ordered or operations are performed. Refer also to the information under “Precedence of Operators” on page 317.

HP-IL: Hewlett-Packard Interface Loop. A means of controlling peripherals used by the HP-71 and other computers.

I

Identifier: A string expression or unquoted string that identifies a file and/or device.

IEEE default value: The default values specified by the IEEE Floating Point standard. These values can be supplied by the HP-71 when math exceptions occur. (Refer to the section entitled “The IEEE Proposal For Handling Math Exceptions” for further information about IEEE default values.)

immediate execute key: A key or keystroke combination that, when pressed, displays an instruction, then executes it. Compare with **direct execute key**. Refer also to the **DEF KEY** entry in the Keyword Dictionary.

independent RAM: RAM that is separated from main RAM by executing `FREE PORT`. An independent RAM contains files and is referenced by its port number.

inexact result: A numeric result that cannot be exactly represented in the HP-71 floating point format, such as an irrational number or a number having a repeating decimal.

INF: A function used to return the HP-71 representation for infinity (`Inf`).

initialize a variable: To assign an initial value to a variable. In the HP-71, all numeric variables and array elements are initialized to zero when created by executing `DIM`, `REAL`, `SHORT`, and `INTEGER`. All string variables and array elements are initialized to the null string when created by executing `DIM`.

input: To enter data from the keyboard or from a data file. Also refers to the data itself.

integer value: A number that has no fractional part.

instruction: A generic term for all operations that can be performed on the HP-71, including any statement, function, or operator (with its corresponding operands).

interchange file: A file of type `TEXT` (or `LIF1`) written to a mass storage medium and used to interchange information between the HP-71 and other computers.

interface: The circuitry that connects a computer to other devices and enables them to function together.

I/O: Abbreviation for *input/output*, indicating an operation that either receives input from a device or sends output to a device. The HP-71's built-in input and output devices are the keyboard and display. The card reader is both an input and an output device.

K

key definition: The current functionality of a key or keystroke combination, which can be to display a string of characters and/or to execute one or more instructions. Refer also to the **DEF KEY** entry in the Keyword Dictionary.

key name: A string expression that identifies a key. A single-character string refers to the key that enters that character. (Letters can be uppercase or lowercase.) A two-character string in which the first character is upper- or lowercase "f" or "g" refers to the f- or g- shift of the specified key. A string beginning with # and followed by one to three digits refers to a key number. For further information, refer to the **DEF KEY** entry in the Keyword Dictionary.

keyboard buffer: An area of main RAM in which the computer stores keystrokes until it can process them. The keyboard buffer can contain up to 15 keystrokes.


keyboard execution: To perform HP-71 operations from the keyboard, as opposed to operations performed by a running program.

KEYS: A reserved word that specifies the system `keys` file. When used, it must be an unquoted string.

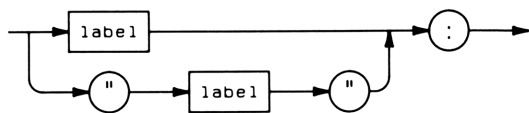
keys file: The system file that contains the current key redefinitions. This file is named `keys`. To reference it, use the `KEYS` keyword.

keyword: A word having a predefined meaning in the HP-71. Usually a statement, function, or operator.

L

label: A quoted or unquoted string, immediately followed by a colon, that identifies a program statement. A label can precede any statement within a BASIC program, and can follow a line number, an , or another label. That is, multiple labels can identify the same statement.

A label can consist of one to eight characters. The first character must be a letter. Each of the remaining characters can be either a letter or a digit. The HP-71 converts all letters in program labels to uppercase and places the label within single quotes.



label reference: A string expression or unquoted string used in GOTO, GOSUB, and other statements to transfer execution to the statement immediately following the corresponding label. A label reference contains the same string as its corresponding label. See also **label**.

language extension (LEX) file: A type of file used to define new keywords and to extend HP-71 capabilities. LEX files can be maintained on mass storage media and in the computer's memory.

least significant bit: The bit contributing the smallest amount to the value of a binary number. In the owner's manual and the reference manual, it refers to the rightmost bit of a binary number. See also **most significant bit**.

left justify: To write a string beginning at the leftmost column of a display or print field.

LEX file: See **language extension (LEX) file**.

LIF: See **logical interchange format (LIF)**.

line editing: Adding to, modifying, or deleting characters in the display.

line-feed: A control character (decimal code 10) that causes the HP-71 to advance to a new display line.

line number: An integer constant in the range 1 through 9999 that identifies a program line.

local environment: The environment that the HP-71 defines for exclusive use by a specific main program or subprogram. See also **active environment**, **calling environment**, **environment**, **global environment**, and **main environment**.

logical error: An error in a program's design. A logical error can result from using a faulty algorithm or from simply keying in a program improperly. Logical errors aren't detected by the computer, but are evident from faulty program output.

logical expression: A numeric expression used in a conditional test. Evaluated as true if nonzero, and false if zero. Usually includes relational operators. Refer also to the **IF ... THEN ... ELSE** entry in the Keyword Dictionary.

logical interchange format (LIF): A mass media format that is common to several Hewlett-Packard computers. This format enables different computers to interchange programs and information.

logical operator: An operator that returns a logical value (0 or 1). The logical operators on the HP-71 are **OR**, **AND**, **EXOR**, and **NOT**.

loop counter: The simple numeric variable in a **FOR ... NEXT** loop that controls the number of loop iterations.

looping: Repeatedly executing a series of statements, usually until a specified condition is satisfied.

M

main environment: The local environment of a main program. The main environment is also active when no program is running or suspended. See also **active environment**, **calling environment**, **environment**, **global environment**, and **local environment**.

main program: A program that is not called by another program or subprogram, but is typically executed by **[RUN]**, **RUN**, or **CHAIN**.

main RAM: The memory used by the computer to create the main environment or subprogram environments. (The computer uses part of this memory to maintain operating information that is not directly accessible to users.) Main RAM is distinct from independent RAM. (Main RAM is referenced by **MAIN**. Creating a file either when no device is specified or when **MAIN** is specified causes the HP-71 to create the file in main RAM. Refer to "Device Names" in your *HP-71 Owner's Manual*.)

mantissa: The normalized part of a number displayed or printed with an exponent in scientific display format. For example, the number `1.573E17` has `1.573` as its mantissa and `17` as its exponent of 10. See also **floating point number**.

mass storage device: An I/O device such as the HP 82400A Magnetic Card Reader or an HP 82161A Cassette Drive that you can use to copy files between memory and mass storage media.

mass storage medium: A magnetic card, cassette, or disc on which you can store computer files.

math exception: An event that occurs during evaluation of a numeric expression and is in one of the following categories:

- Invalid operation.
- Division by zero.
- Overflow.
- Underflow.
- Inexact.

Depending upon the trap value that corresponds to the exception, the HP-71 can either treat such events as errors or supply a default value for the expression.

math exception flag: A flag that the HP-71 sets whenever a math exception occurs. The five math exception flags are:

- `IVL` (invalid operation).
- `DVZ` (division by zero).
- `OVF` (overflow).
- `UNF` (underflow).
- `INX` (inexact result).

MAXREAL: The largest finite value the HP-71 can represent, which is: `9.999999999999E499`.

memory reset: A condition in which the memory of the computer is cleared of all programs, data, and other information. This happens when you execute `INIT:3` (in which case independent RAM is protected) or remove the batteries for an extended period (independent RAM is not protected). To execute an `INIT:3`, refer to “Memory Reset, BASIC Mode, and the BASIC Prompt” in section 1 of your *HP-71 Owner's Manual*.

message number: A number that identifies an error, warning, or instructional message. Error messages and their corresponding numbers are listed on pages 380 through 392.

MINREAL: The smallest positive value the HP-71 can represent, which is the denormalized number 0.00000000001E-499. Refer to the **MINREAL** entry in the Keyword Dictionary.

mode: A condition of the computer that determines which operations can be performed. For example, a BASIC program can be edited in BASIC mode only; it cannot be edited in CALC mode.

module: A device that fits into one of the HP-71 ports and extends its memory or its capabilities.

most significant bit: The bit contributing the greatest amount to the value of a binary number. Often referred to as the *leftmost bit*.

multiple-statement function: A user-defined function that contains more than one program statement.

multistatement line: A line containing two or more BASIC statements concatenated by the `␣` symbol.

N

NaN: The HP-71 function that returns a Signaling NaN. When entered from the keyboard and used in an expression, it signals the computer to set the `IVL` flag and supplies the value `NaN` (not-a-number).

NaN: An abbreviation for *Not-a-Number*. This is a default value supplied for invalid operations (operations that set the `IVL` flag) when the `IVL` exception trap is set to 2. NaN is supplied for such invalid operations as 0/0.

nested loop: A `FOR...NEXT` loop contained within another such loop.

nested subroutine: One subroutine that is invoked by another subroutine.

nibble: One-half of a byte; equivalent to four bits. The HP-71 individually addresses nibbles in memory.

normalized number: A number represented internally by the HP-71 in the decimal format $m * 10^e$

where m is the mantissa, which is a 12-digit number in the range

$$1.000000000000 \leq |m| \leq 9.999999999999,$$

and e is the exponent, which is a 3-digit number in the range

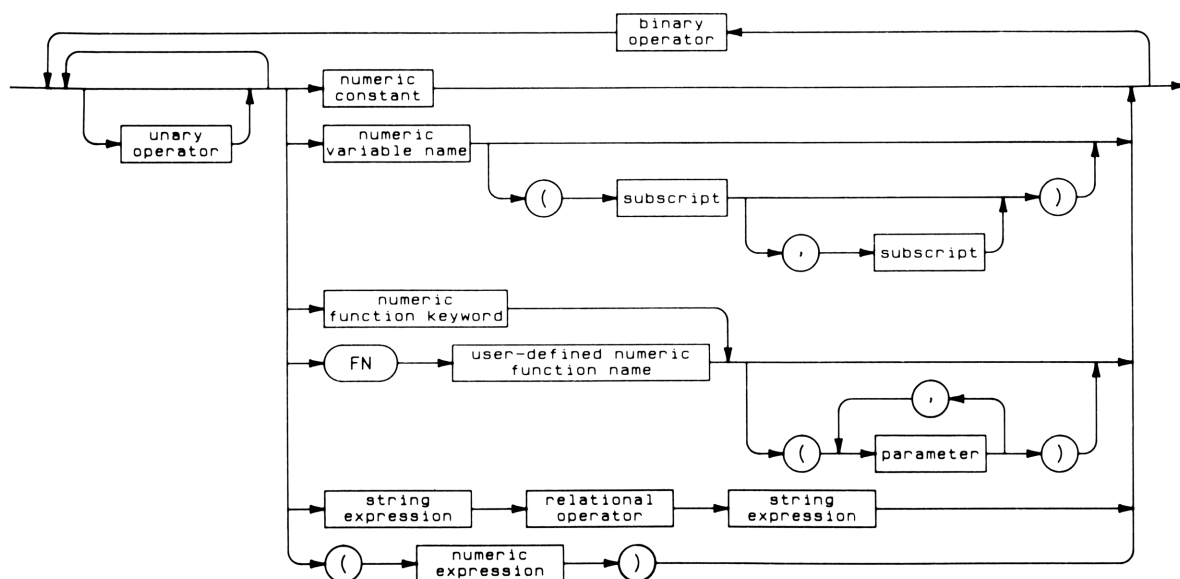
$$-499 \leq e \leq 499.$$

null string: String of zero length, specified by `' '` or `''`.

numeric array: An array containing elements that are numeric values.

numeric constant: A fixed numeric value within the range the HP-71 can represent.

numeric expression: A valid combination of values and operators that produce a numeric result. The following diagram describes the syntax of a numeric expression.

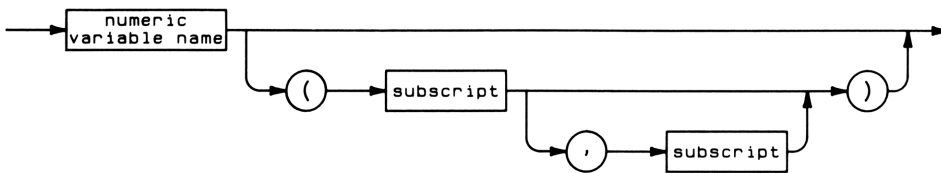


Note: The \wedge binary operator cannot be followed immediately by a binary operator.

numeric function: An operation (that is, an HP-71 keyword or a user-defined function) that, given the appropriate type, number, and range of arguments, returns a single numeric value. Some numeric function keywords, like `RES`, require no arguments.

numeric variable name: A letter or letter-digit combination that represents a location in memory where you can store numeric information.

numeric variable specifier: Designates a simple numeric variable or a numeric array element that is to receive a new value, as in `READ`, `READ#`, `INPUT`, `LINPUT`, `LET`, and `LR`.



O

open a file: To associate a file with a channel number. Opening a file enables the computer to read and write information to that file. A file is opened by executing `ASSIGN#`.

operand: A numeric or string value upon which an operation is performed.

operating system: A collection of built-in programs that control the overall operation of the computer, performing such tasks as interpreting and executing BASIC programs, assigning places in memory for files, processing keyed-in information, controlling the display, and performing calculations.

operator: A symbol that combines or compares the values of two expressions. *Arithmetic*, *relational*, and *logical* operators result in a numeric quantity; the *string* operator (`&`) results in a string quantity.

output: Information that the computer sends to a device such as a printer or display.

P

parameter: The numeric or string information acted on by a keyword. Also, numeric or string information used by a function to determine the function's value. Parameters passed to a subprogram can be passed by reference or by value. See **reference parameter** and **value parameter**.

pass by reference: To pass a variable as an actual parameter to a subprogram so that it can be altered.

pass by value: To pass an expression value as an actual parameter to a subprogram or user-defined function. A variable is passed to a subprogram by value only if it is enclosed in parentheses (which ensures that it cannot be altered by the subprogram.) All parameters passed to user-defined functions are passed by value.

password: A string expression specified in the `LOCK` statement. A password prevents unauthorized use of the HP-71. Refer to the **LOCK** entry in the Keyword Dictionary.

peripheral: Any external device on a standard interface controlled by the HP-71.

pointer: An internal mechanism the computer uses to indicate the next piece of information to process. The HP-71 uses pointers to access information such as data items, files, and program lines. See also **data pointer**, **file pointer**, and **program pointer**.

port specifier: A number of the form *m.nn* or *m* that identifies one of the computer's ports. The value for *m* is in the range from 0 through 5; *nn* is in the range from 00 through 15. (Values of *nn* less than 10 must be expressed with two digits; for example 09, 08, . . . 01.) If the port specifier is contained within an unquoted string, the specifier can be designated by a numeric expression.

precision: The number of significant digits a computer uses when it computes and stores a numeric value. The HP-71 performs computations in 15-digit precision and stores in 12-digit precision.

precedence of operators: The order in which mathematical operations are performed by the computer, based on the types of operators in an expression. Refer to "Precedence of Operators" on page 317.

predefined function: Any functions that are defined for you by either the HP-71 or by a language extension file present in the computer.

private file: A file on a magnetic card or other external medium that can be copied into memory and run, but which cannot be changed or recopied. (A private file in memory can be run and/or purged, but cannot be copied.)

program: A set of instructions that performs some computing task and controls the input, processing, and output of data. You can store programs in BASIC files and in BIN (binary) files.

program line: One line of a BASIC file; contains a line number and one or more statements.

program pointer: The mechanism used to identify the next function or statement to be executed in a BASIC program. See also **pointer**.

program scope: The environment of the current program or subprogram. This environment determines which variables and statement identifiers can be referenced.

program unit: A program, subprogram, or user-defined function that performs a specified task.

prompt: The symbol that appears at the left edge of the display (the right edge when in CALC mode) to indicate readiness for user input. Also, a text string used in an `INPUT` statement to indicate that the user should key in some information.

protected field: A portion of the display or a display line that cannot be overwritten. The `WINDOW` statement defines protected fields in the 22-character display window. Escape sequences can define protected fields in the 96-character display line.

R

radians setting: The `OPTION ANGLE` setting used for expressing trigonometric function arguments and results in radians.

radix mark: The mark, such as a decimal point, that separates the integer portion of a number from its fractional part.

RAM: See **random access memory**.

random access memory (RAM): Nonpermanent memory circuits in which a computer can read and write. RAM requires a constant source of power to retain its memory. Compare with **read-only memory (ROM)**.

random file access: The process of reading from or writing to a specified record in a data file.

random seed: A number used by `RND` to generate a random number. You can set the random seed using `RANDOMIZE`.

read-only memory (ROM): Permanent memory from which a computer can only read. ROM retains its memory when disconnected from a power source.

real value: A numeric value represented as a floating point number.

record: The smallest addressable unit of a data file.

recursion: A process or procedure that is defined in terms of itself.

recursive subprogram: A subprogram that calls itself.

reference parameter: A parameter that can be changed by the subprogram to which it is passed. See also **parameter** and **value Parameter**.

relational operator: A binary operator that compares two arguments and returns a 0 or a 1 based on the outcome of the comparison. The following relational operators are available: `<`, `=`, `>`, `<=`, `>=`, `<>`, `#`, and `?`.

reserved word: A word that has a predefined meaning in the HP-71 and cannot be used as a file name unless the word is part of a string expression. The reserved words are: `TO`, `ALL`, `KEYS`, `CARD`, and `INTO`. The reserved words are a subset of the HP-71 keywords.

ROM: See **read-only memory**.

ROM-based file: Files residing in read-only memory.

round: To adjust the least significant digits of a number according to digits that were either truncated or cannot be represented due to limits of precision.

rounding error: The error resulting from rounding a quantity by deleting the less significant digits and applying some rule of correction to the part retained. For example, 2.6641 can be rounded to 2.664 with a rounding error of .0001.

round-off setting: Any one of four settings that determine how the computer rounds numbers in arithmetic operations. Refer to the **OPTION ROUND** entry in the Keyword Dictionary.






routine: A program, program segment, user-defined function, or subprogram that performs a specific task and supports the execution of a larger program.

run-time error: An error that occurs during execution of a program or individual keyboard instruction and is not due to a syntax error. See also **logic error**.

S

scope: The range of operation or accessibility of a program, statement, user-defined function, file, setting, or other information.

scroll: Apparent horizontal movement of characters across the display or apparent vertical movement of program lines, command stack items, or catalog entries through the display.



scroll key: The , ,  and  keys, and their -shifted counterparts.

SDATA file: A data file with a fixed record length of eight bytes.

secure file: A file that cannot be purged, altered, or declared private.

sequential file access: The procedure of reading from or writing to successive records in a data file.

setting: A flag or operating condition (such as the `OPTION ANGLE RADIANS` setting) that is part of the global environment.

shifted keystroke: A keystroke that is preceded by pressing the - or -shift key.

short value: A numeric value that is represented according to the `SHORT` format.

signaling NaN: A NaN entered from the keyboard. Such a NaN, when used in a subsequent math operation, causes the HP-71 to set the `IVL` flag.

simple numeric variable: A variable name (a single letter or a letter and a digit) representing a memory location in which a single numeric value has been stored.

simple string variable: A variable name (a single letter followed by `$` or a letter and a digit followed by `$`) representing a memory location in which a single string value has been stored.

single step: To execute a program one statement at a time using `[F] [SST]`. Also, to evaluate an expression one operation at a time while in CALC mode.

source file: A file that contains information that the computer either reads or transfers to another file.

start character position: The position of the first character to be replaced by an assignment to a string variable. Also refers to the first character of a substring. Values less than 1 are interpreted as 1, and the default value is 1. For an assignment, the value cannot exceed the string variable's maximum length. See also **final character position**.

start-up string: A string that is specified in the `STARTUP` statement and executed as a command when the computer turns on.

statement: A BASIC instruction that can be executed in a running program or from the keyboard. Statements can be concatenated with `:` to form a multistatement line.

statement identifier: A line number or label reference that specifies a program statement.

statistical array: A numeric, one-dimensional array in which summary statistics are accumulated. Statistical arrays are created using the `STAT` statement.

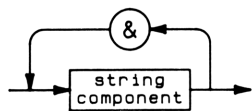
string: An arbitrary sequence of characters that is not regarded as a number. In general, you can specify such a sequence as a string expression or as an unquoted series of characters.

string array: A one-dimensional array having elements that are strings.

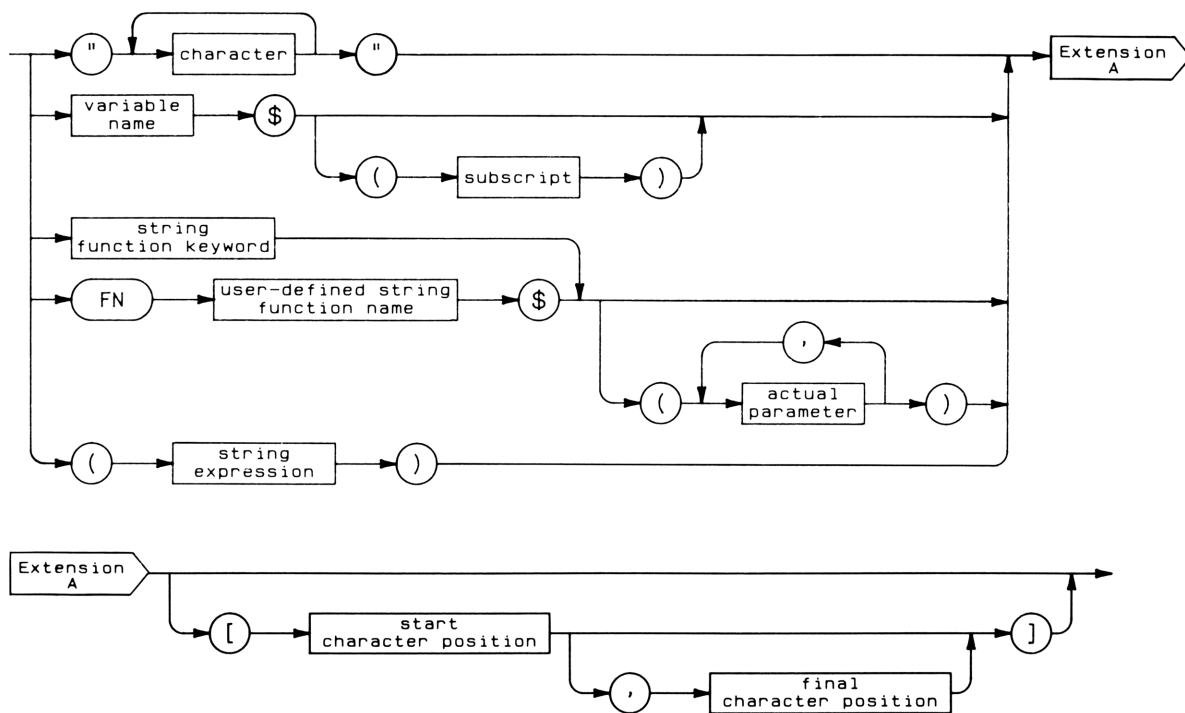
string constant: An arbitrary sequence of characters delimited by quotation marks (`' ... '` or `" ... "`). Also called a *quoted* string.

string expression: A valid combination of string components that produces a string result. The following two diagrams describe string expression syntax and components.

String Expression:



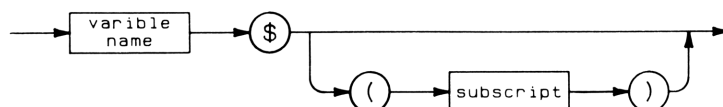
String Component:



string function: An operation (that is, an HP-71 keyword or a user-defined function) that, given the appropriate type, number, and range of arguments, returns a single string value. Some string function keywords, like `TIME$`, require no argument.

string variable name: A letter or a letter-digit combination followed by \$, that represents a location in memory where you can store character information.

string variable specifier: Designates a simple string variable or a string array element (or a portion of that string) that is to receive a new string value, as in READ, READ#, INPUT, LINPUT, and LET. The portion of the destination string variable to be replaced is specified by the *start character position* and the *final character position*. The computer always begins the substitute string at the *start character position*.

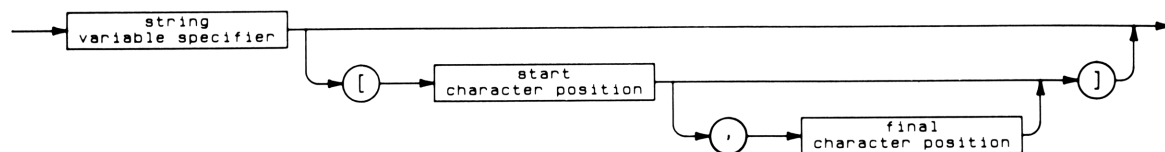


subprogram: A set of program lines that forms a routine that is independent of any main program within the file in which it's stored. A subprogram is delimited by the SUB and END SUB statements.

subroutine: A program segment that begins execution as a result of a GOSUB statement and that returns control when it executes a RETURN statement.

subscript: A number that specifies the row (or column) location of an array element. A subscript value must be less than or equal to the array dimension limit, and greater than or equal to the array base.

substring: A portion of a string variable; contains zero or more contiguous characters. To specify a substring, follow a string variable specifier with a numeric expression (or numeric expressions) enclosed in brackets []).



suspend statement: When a program is suspended, the statement at which program execution will resume when you press ☐ f ☐ CONT .

suspended program: A program that has been interrupted without affecting its program control information. The **SUSP** annunciator is displayed when a program is suspended. A suspended program can be resumed by pressing ☐ f ☐ CONT .

syntax: The rules governing the spelling of keywords, variable names, operators, file names, etc., and the construction of statements and functions.

system file: A file that is designated by the computer for a special use. Two such files are `workfile` and `keys`. System files can be identified by their lowercase spelling in a catalog entry.

system flag: A flag used by the computer to indicate a condition, such as a low battery or an overflow in a calculation. System flags are numbered from -1 through -64 .

system message: A message displayed by the computer to give instructions.

T

tab: To move the cursor to a specified column.

TAB: A keyword that moves the `DISP` or `PRINT` position ahead to a specified column.

text: An arbitrary collection of characters.

TEXT file: A file composed of textual information in a form that can be transmitted to other computers. In the HP-71, the TEXT file type is identical to the LIF1 file type on the HP-75.

timer number: An integer from 1 through 3 that is used in an `ON TIMER#` statement to specify a system timer.

track: One of two read/write channels on a magnetic card.

trap: To detect certain exceptional conditions when they occur in calculations and to take an action based on the type of exception and the setting of its corresponding trap value. Refer to “The IEEE Proposal for Handling Math Exceptions” section that begins on page 338.

trap action: The actual computer response to errors as determined by the trap value.

trap value: A value associated with the five math exception flags to indicate how the computer responds to the errors that set those flags. A trap value can be 0, 1, or 2.

trappable error: Any error that, when it occurs, allows use of an `ON ERROR` branch or subroutine to take some action other than halting the program; an error that either sets one of the five math exception flags or can cause a branch to occur during an `ON ERROR` condition.

truncate: To cut off a portion of a string or number. For example, the `IP` function truncates a number at its decimal point. No rounding occurs.

typewriter key: A key that normally displays a letter, digit, or other symbol.

typing aid: A key or keystroke combination that enters a string of characters when pressed, as if they had been typed in. Typing aids can be user-defined. Refer to the **DEF KEY** entry in the Keyword Dictionary.

U-V

unconditional branch: A branch that occurs every time the statement is executed, as when program execution encounters a `GOTO` statement.

unary operator: An operator that performs its operation on one operand. It is placed in front of the operand. The following unary operators are available.

- `-` : Reverses sign of its operand.
- `+` : Identity operator. (This is a binary operator that can be used as a unary operator.)
- `NOT` : Logical complement.

See also **binary operator**.

underflow: A condition in which the computer cannot represent a very small result within its normal range of precision. Some results that cause an underflow condition can be represented as denormalized numbers.

unquoted string: A series of characters, regarded as a string, that is not enclosed in quotes.

unordered: A relation in which a value is neither greater than, equal to, or less than another value, as when a NaN is compared to another value. An unordered relation is tested by the `?` operator.

user-defined function: A numeric or string function that a user defines in a program by means of `DEF FN` and `END DEF` statements.

user-defined function name: `FN` followed by a letter and an optional digit. A user-defined string function name always ends with `▯`.

user flag: A flag whose use is determined by the user. The user flags on the HP-71 are those numbered 0 through 63.

user keyboard: The set of key redefinitions in the current `keys` file that are activated by pressing `f` `USER` .

value parameter: A parameter that cannot be changed by the subprogram to which it is passed. Value parameters include numeric expressions, string expressions, and variable names enclosed in parentheses. See also **Parameter** and **reference parameter**.

variable name: Consists of a letter followed by an optional digit, and is used to identify a variable. String variable names end with `▯`. See **numeric variable specifier** and **string variable specifier**.

variable number: An integer identifying a particular statistical variable.

verify: To check a magnetic card to ensure that information was properly written to it.

W-X

warning condition: A condition in which either a math result needs a default value or the computer has detected an undesirable hardware condition (such as low batteries).

work file: A system file existing as a BASIC scratch file in main RAM.

▯FN: An abbreviation used in either an error message or a displayed line to refer to a function in a plug-in ROM or language extension (LEX) file when the ROM or LEX file has been removed from the HP-71.

▯WORD: An abbreviation used in an error message or a displayed line to refer to a keyword defined in a plug-in ROM or language extension (LEX) file when the ROM or LEX file has been removed from the HP-71.

Errors, Warnings, and System Messages

This section contains the error, warning, and system messages related to the HP-71 computer. For messages related to devices that can be connected to the HP-71, please refer to the manuals for those devices.

Introduction

This section contains two listings:

1. An alphabetically-ordered listing of error, warning, and system message names, with their corresponding error numbers. You can use this listing to determine the number of any message.
2. A numerically-ordered listing of error, warning, and system messages, with a description of each message.

An *error message* begins with `ERR:` and names the error type. In most cases, an error halts execution, which indicates that the computer cannot perform a pending operation. A *warning* begins with `WRN:` and names the warning type. A warning indicates that a default value has been substituted for a value that the computer could not compute, and does not halt execution. Some messages can be either error messages or warning messages, depending upon the arithmetic *trap values*. An error or warning message displayed during program execution indicates the line number causing the error or warning, such as `ERR L30:` (error at line 30) and `WRN L95:` (warning at line 95).

A *system message* is usually a procedural message, such as a prompt for card reader operations. For a description of error and warning conditions, refer to your *HP-71 Owner's Manual*.

Alphabetical Message Listing

Messages beginning with nonalphabetical characters are placed at the end of this listing.

Message	Number
Cat: Align then ENDLN.....	96
Chnl# Not Found	41
Configuration.....	26
Data Type	31
Device Not Found	64
End of File	54
Excess Chars	78

Message	Number
EXPONENT (0).....	3
File Exists	59
File Not Found.....	57
File Open	62
File Protect.....	61
File Too Big	74
FN Not Found	33

(Continuation of Message Table)

Message	Number
FOR w/o NEXT	42
Illegal Access	60
Illegal Context	79
IMAGE Ovfl	47
Inexact	21
Inf*0	16
Inf-Inf	15
Inf/Inf	14
Inf^0	18
Insufficient Memory	24
Invalid AF	27
Invalid Arg	11
Invalid Expr	80
Invalid File Type	63
Invalid Filespec	58
Invalid IMAGE	45
Invalid Key	85
Invalid Parm	81
Invalid Stat Array	51
Invalid Stat Op	53
Invalid Statistic	52
Invalid TAB	48
Invalid Transform	55
Invalid USING	46
Invalid Var	83
Line Too Long	65
LOG(0)	12
LOG(neg)	13
Low Battery	22
Missing Parm	82
Module Pulled	25
Neg^Non-int	9
NEXT w/o FOR	43
No Data	32
Not This File	67
Numeric Input	38
Operand Expected	86
Operator Expected	87
Overflow	2
Parameter Mismatch	36
Precedence	84

Message	Number
PROT: Align then ENDLN	94
Pull <i>nnn</i> of <i>mmm</i>	89
Pull Card	90
Quote Expected	77
R/W Error	70
Read: Align then ENDLN	93
Record Ovfl	29
RTN w/o GOSUB	44
Signaled Op	19
SQR(neg)	10
Stmt Not Found	30
String Ovfl	37
Sub Not Found	49
Subscript	28
Syntax	75
System Error	23
TAN=Inf	4
TFM WRN <i>Lnnn</i> :	88
Too Fast	71
Too Few Inputs	40
Too Many Inputs	39
Too Slow	72
Transform Failed	56
Trk <i>nnn</i> Done	97
Underflow	1
Unknown Card	69
Unordered	20
Unpr: Align then ENDLN	95
Var Context	50
Verify Fail	68
Vfy: Align then ENDLN	92
Write Protected	66
Wrong Name	73
Wrt: Align then ENDLN	91
XFN Not Found	34
XWORD Not Found	35
> Expected	76
/Zero	8
0/0	7
0^0	6
0^neg	5
1^Inf	17

Numerical Message Listing and Descriptions

Math Errors (1 through 21)

Error Number	Message and Condition
1	<code>Underflow</code> The magnitude of a result is too small to fit exactly into its destination format. Sets UNF (underflow) flag (−5).
2	<code>Overflow</code> The magnitude of a result is too large to fit into its destination format. Sets OVF (overflow) flag (−6).
3	<code>EXPONENT(0)</code> Indicates attempt to compute the exponent of a zero argument. Sets DVZ (division-by-zero) flag (−7).
4	<code>TAN=Inf</code> <code>TAN</code> argument is an odd integral multiple of 90 degrees. Sets DVZ (division-by-zero) flag (−7).
5	<code>0^neg</code> x^y , where $x = 0$ and $y < 0$. Sets DVZ (division-by-zero) flag (−7).
6	<code>0^0</code> x^y , where $x = y = 0$. Does not set an exception flag.
7	<code>0/0</code> x/y or $x \text{ DIV } y$, where $x = y = 0$. Sets IVL (invalid operation) flag (−8).
8	<code>/Zero</code> $x \text{ DIV } y$ or x/y , where $0 < x < \text{Inf}$ and $y = 0$. Sets DVZ (division-by-zero) flag (−7).
9	<code>Neg^Non-int</code> x^y , where $x < 0$, $0 < y < \text{Inf}$, and y is noninteger. Sets IVL (invalid operation) flag (−8).
10	<code>SQR(neg)</code> Attempt to compute the square root of a negative number. Sets IVL (invalid operation) flag (−8).
11	<code>Invalid Arg</code> An argument of a function, operation, or statement has the correct data type, but lies outside the domain of definition of that function, operation, or statement. In some circumstances sets IVL (invalid operation) flag (−8).

(Continued on next page.)

Math Errors (continued)

Error Number	Message and Condition
12	<p><code>LOG(0)</code></p> <ul style="list-style-type: none"> • <code>LGT</code>: Attempt to compute the logarithm of 0. • <code>LOGP1</code>: If argument equals -1. <p>Sets <code>DVZ</code> (division-by-zero) flag (-7).</p>
13	<p><code>LOG(neg)</code></p> <p>Attempt to compute the logarithm of a negative number.</p> <p><code>LOGP1</code>: If argument is less than -1.</p> <p>Sets <code>IVL</code> (invalid operation) flag (-8).</p>
14	<p><code>Inf/Inf</code></p> <p>An operation requires the division of an infinite dividend by an infinite divisor. Sets <code>IVL</code> (invalid operation) flag (-8).</p>
15	<p><code>Inf-Inf</code></p> <p>An operation requires the subtraction of an infinite value from another infinite value of like sign. Sets <code>IVL</code> (invalid operation) flag (-8).</p>
16	<p><code>Inf*0</code></p> <p>An operation requires multiplying an infinite argument by a zero argument. Sets <code>IVL</code> (invalid operation) flag (-8).</p>
17	<p><code>1^Inf</code></p> <p>Attempt to raise $+1$ or -1 to an infinite power. Sets <code>IVL</code> (invalid operation) flag (-8).</p>
18	<p><code>Inf^0</code></p> <p>Attempt to raise an infinite argument to the power of zero. Does not set an exception flag.</p>
19	<p><code>Signaled Op</code></p> <p>An input (parameter or data) is a signaling NaN. Sets <code>IVL</code> (invalid operation) flag (-8).</p>
20	<p><code>Unordered</code></p> <p>A comparison involves at least one NaN and either a <code><</code> or <code>></code> operator without the <code>?</code> operator. Occurs with <code>MAX</code> or <code>MIN</code> if either or both arguments are NaNs. Sets <code>IVL</code> (invalid operation) flag (-8).</p>
21	<p><code>Inexact</code></p> <p>The result of a function or operation may not be exact. Sets <code>INX</code> (inexact result) flag (-4).</p>

System Errors (22 through 27)

Error Number	Message and Condition
22	<div> Low Battery <p>During a card reader operation, warns that the battery level is low and continued operation may cause incorrect results.</p> </div>
23	<div> System Error <p>The integrity of information in memory has been disrupted such that the computer cannot process the information. This can be caused, for example, by deleting a calling program while its corresponding subprogram is executing. At this point, due to the memory disruption, the HP-71 may not operate properly until you execute an <code>INIT 3</code>.</p> <div> <div>CAUTION</div> <p>Executing <code>INIT 3</code> clears <i>all</i> of the HP-71's main RAM. Refer to your <i>HP-71 Owner's Manual</i>.</p> </div> </div>
24	<div> Insufficient Memory <p>Generally, insufficient memory available to complete an operation. Specifically, indicates insufficient memory for the following functions to perform their indicated operations:</p> <ul style="list-style-type: none"> • <code>ASSIGN#</code>: Creating the I/O buffer needed to open the file. • <code>CHAIN</code>: Copying in the specified file. • <code>CHARSET</code>: Storing the alternate character set. • <code>COPY</code>: Creating the destination file. When copying to a card, building the card header in main RAM. • <code>DIM</code> • <code>INTEGER</code> • <code>SHORT</code> • <code>REAL</code> <div> } <p>Creating a variable with the specified dimensions.</p> </div> • <code>STAT</code>: Creating a statistical array for the specified number of variables. • <code>EDIT</code>: Creating a file that does not already exist. • <code>FOR...NEXT</code>: There is not enough memory to save the needed <code>FOR...NEXT</code> information. • <code>FREE PORT</code>: There is not enough memory to remove RAM from main RAM. • <code>RUN</code>: The file specified is on an external device and there is not enough memory to copy the file into main RAM. • <code>PURGE</code>: Attempting to purge the current file when the <code>workfile</code> does not exist and there is not enough memory to create the <code>workfile</code>. </div>

(Continued on next page.)

System Errors (continued)

Error Number	Message and Condition
25	<p>Module Pulled</p> <p>A module was removed while the computer was turned on. Doing so causes a system reconfiguration and, if any ROMs have moved in address space, causes an automatic EDIT. The system workfile is now the current file.</p>
26	<p>Configuration</p> <p>A configuration error has occurred, such as too many ROMs for the address space or insufficient memory to build all of the configuration tables. The computer should operate satisfactorily, but will not find all of the plug-in memory and I/O devices.</p>
27	<p>Invalid AF</p> <p>An invalid value was specified for the clock adjustment factor. (Executing either AF (<i>parameter</i>) or EXACT with an adjustment factor whose absolute value lies between 0 and 10.)</p>

Program Errors (28 through 56)

Error Number	Message and Condition
28	<p>Subscript</p> <p>An array subscript is out of range.</p>
29	<p>Record Ovfl</p> <ul style="list-style-type: none"> • PRINT #: The data items in a print list cannot fit into the specified record. • READ #: A random access read from a data file has specified a record that has fewer data items than the number of items in the read list.
30	<p>Stmt Not Found</p> <p>A statement identifier that is not in the current program scope has been referenced. For example, occurs when FETCH references a label that is not in the current file.</p>
31	<p>Data Type</p> <p>The HP-71 cannot use a variable, result, or quantity as it has been specified. Can be generated, for example, by referencing an existing variable as an array.</p>
32	<p>No Data</p> <p>Data for READ cannot be found.</p>
33	<p>FN Not Found</p> <ul style="list-style-type: none"> • END DEF: A function was not being evaluated when the computer executed END DEF. • FN: Attempting to execute a user-defined function that is not defined in the current program.

(Continued on next page.)

Program Errors (continued)

Error Number	Message and Condition
34	<p>XFN Not Found</p> <p>The language extension (LEX) file required to execute a <i>function</i> cannot be found.</p>
35	<p>XWORD Not found</p> <p>The language extension (LEX) file required to execute a <i>statement</i> cannot be found.</p>
36	<p>Parameter Mismatch</p> <p>The number or type of the actual parameters in a CALL statement or function reference does not match the number or type of the formal parameters in the corresponding SUB or DEF FN statement.</p>
37	<p>String Ovfl</p> <p>Either a string result is too long for the given destination or a STARTUP string exceeds 95 characters in length.</p>
38	<p>Numeric Input</p> <p>INPUT requires numeric input. (Reprompts you for input.)</p>
39	<p>Too Many Inputs</p> <p>Too many items entered in response to an INPUT statement. (Reprompts you for input.)</p>
40	<p>Too Few Inputs</p> <p>Too few items entered in response to an INPUT statement. (Reprompts you for input.)</p>
41	<p>Chnl# Not Found</p> <p>The specified channel number has not been assigned by an ASSIGN # statement.</p>
42	<p>FOR w/o NEXT</p> <p>The HP-71 has encountered a FOR statement that is not going to be executed, and cannot find a corresponding NEXT statement. This can occur, for example, where a statement such as FOR I = 1 TO -5 STEP .5 occurs without a corresponding NEXT statement.</p>
43	<p>NEXT w/o FOR</p> <p>A NEXT statement has been encountered for which a corresponding FOR statement cannot be found.</p>
44	<p>RTN w/o GOSUB</p> <p>A RETURN has been encountered for which there is no corresponding GOSUB statement.</p>

(Continued on next page.)

Program Errors (continued)

Error Number	Message and Condition
45	<p>Invalid IMAGE</p> <p>An image list has one or more errors, as follows:</p> <ul style="list-style-type: none"> • Carriage return symbol not followed by a delimiter. • Unrecognized character in the format string. • Illegal use of image symbols. For example: <ul style="list-style-type: none"> • D in a string field. • Mixed D, *, and Z symbols in a numeric field. • Adjacent C or P. • C or P beginning or ending a numeric field. • C or P adjacent to radix or exponent specifier. • Specifying more than one S or M. • Leading S or M with no following digit or radix specifier. • Leading . or R with no following D. • Omitting the closing quote on a quoted string. • Specifying a multiplier for a symbol that does not allow one. • Specifying a zero multiplier or a multiplier greater than 9999. • Following a multiplier with a comma or closing parenthesis. • Specifying a multiplier for a unit digit's Z. • No matching closing parenthesis. • Closing parenthesis without opening parenthesis.
46	<p>Invalid USING</p> <p>Either DISP USING or PRINT USING statement references a line number that has no IMAGE statement, or the output item is not of the type specified in the image list. (For example, a numeric output field may be matched to a string item.)</p>
47	<p>IMAGE Ovfl</p> <p>A field in an image list does not have enough digits specified to the left of the decimal point. This can also occur when the number is negative and the implied negative sign reduces the number of specified digits by occupying the leftmost digit. Also occurs when an E symbol in a numeric field would result in a displayed exponent having more than three digits.</p>

(Continued on next page.)

Program Errors (continued)

Error Number	Message and Condition
48	<code>Invalid TAB</code> TAB column rounds to a value less than 1. A warning condition occurs and a value of 1 is supplied.
49	<code>Sub Not Found</code> CALL: The called subprogram cannot be found.
50	<code>Var Context</code> An attempt has been made to explicitly or implicitly create or destroy a variable within a multiple-statement user-defined function, where that variable is already defined as either a destination in an assignment statement or as a function parameter.
51	<code>Invalid Stat Array</code> No valid statistical array is currently defined.Sometimes occurs when redimensioning an existing array within a user-defined function when the pending assignment statement's destination is a variable element in the array.
52	<code>Invalid Statistic</code> The sample size is invalid for ADD, DROP, LR, MEAN, PREDV, or SDEV, or a sum of squares is less than zero for CORR, LR, PREDV, or SDEV.
53	<code>Invalid Stat Op</code> Can be caused by any of the following: <ul style="list-style-type: none">• The variable list length is greater than the number of variables for ADD or DROP.• The sample size is zero for DROP, LR, MEAN, PREDV, or SDEV.• The sample size is 1 for SDEV.• A sum of squares is zero for CORR, LR, or PREDV.
54	<code>End of File</code> The specified record number is greater than the last record number in the file. In a text file this can also occur if a line length header points beyond the end of the file.
55	<code>Invalid Transform</code> Generated by the TRANSFORM statement if the desired transformation from the source file type to the destination file type is not allowed.
56	<code>Transform Failed</code> TRANSFORM failed due to an unrecoverable error, such as insufficient memory.

File and Device Errors (57 through 65)

Error Number	Message and Condition
57	<p>File Not Found</p> <p>The specified file does not exist. (Can be generated upon return from a subprogram if execution of the calling program file cannot be resumed because it has been purged or transformed.)</p>
58	<p>Invalid Filespec</p> <p>File specifier contains an invalid name or an invalid device specifier. The specified file resides in a device that is not appropriate for the attempted operation.</p>
59	<p>File Exists</p> <ul style="list-style-type: none"> • A file with the same name already exists on that device. • RUN specified an external file which cannot be copied into main memory for execution due to a duplicate filename. • COPY or TRANSFORM cannot operate on system file <code>workfile</code> or <code>keys</code> because <code>WORKFILE</code> or <code>KEYS</code> already exists on the specified device.
60	<p>Illegal Access</p> <p>Either an attempt to alter, purge, or declare private a file that resides in ROM, or an attempt to POKE into a restricted part of memory.</p>
61	<p>File Protect</p> <ul style="list-style-type: none"> • Attempt to purge, alter, or declare private a file that is secure. • Privacy violation. Attempt to alter a private file or to access such a file through LIST, FETCH, MERGE, PEEK#, or AUTO. A private file may only be executed, purged, or copied from an external medium into memory. • Attempt to copy into the HP-71 a private file of unrecognized type. • DEF KEY attempted when system <code>keys</code> file is secured.
62	<p>File Open</p> <p>Attempt to open more than 64 files or attempt to open a file that is already open. Can be generated, for example, by attempting to execute <code>ASSIGN #</code> to open a file to a channel number when that file is already open to another channel number.</p>

(Continued on next page.)

File and Device Errors (continued)

Error Number	Message and Condition
63	<div>Invalid File Type</div> <ul style="list-style-type: none">• Specified file is not of appropriate type for the operation.• Attempt to execute <code>CONT</code> or <code>FETCH</code>, or to edit a line when the current file type is not <code>BASIC</code>.• Attempt to run a file that is not <code>BASIC</code> or <code>BINARY</code> in type.• Copying to a card when the source file type is not recognized by the operating system.• Copying from a card when the source file type is not recognized by the operating system and is not within the standard file type range.
64	<div>Device Not Found</div> <ul style="list-style-type: none">• Specified device cannot be found.• Attempt to either free a port that does not contain main RAM or to claim a port that is not an independent RAM.• Attempt to create a file on an unspecified port when no independent RAM with enough memory is available.
65	<div>Line Too Long</div> <ul style="list-style-type: none">• Attempt to display a line that exceeds 95 characters.• A line exceeds 120 characters. (Occurs as a <code>TRANSFORM</code> warning.)• Attempt to enter a statement having an internal representation exceeding 127 bytes.• In <code>CALC</code> mode, either the expression being entered has more than 95 characters or there is no memory available.

Card Reader Errors (continued)

Error Number	Message and Condition
66	<p>Write Protected</p> <p>The computer encountered a track that has been write-protected by <code>PROTECT</code> and therefore cannot be written on.</p>
67	<p>Not This File</p> <p>The computer has attempted to read a track which is not in the same file as other tracks already read.</p>
68	<p>Verify Fail</p> <p>The computer failed to verify the card against the data which was written to it.</p>
69	<p>Unknown Card</p> <p>The computer does not recognize the card format.</p>
70	<p>R/W Error</p> <ul style="list-style-type: none"> • A wait of more than 7.5 seconds after the <code>Pull</code> message has occurred. • Checksum error in card header or data field. • Hardware data error.
71	<p>Too Fast</p> <p>Card pulled too fast.</p>
72	<p>Too Slow</p> <p>Card pulled too slowly.</p>
73	<p>Wrong Name</p> <p>During a <code>READ</code> operation the name on the card does not match the name specified in the <code>COPY</code> command.</p>
74	<p>File Too Big</p> <p>During a <code>WRITE</code> operation the source file is larger than 65535 bytes.</p>

Syntax Errors (75 through 88)

Error Number	Message and Condition
75	<div>Syntax<ul style="list-style-type: none">• Missing keyword; incorrect (or missing) characters in a statement.• Valid line number or statement label not given for one of the following:<ul style="list-style-type: none">• GOTO• GOSUB• RUN <i>file specifier</i>• TRANSFORM: A syntax error occurred during the transformation (accompanied by a warning message).• DIM: More than one subscript in a string array.• Mandatory expressions not entered in ON TIMER statement.</div>
76	<div>> Expected<ul style="list-style-type: none">• At end of parameter list for DEF, SUB, or CALL.• After port specification of FREE or CLAIM.• Array variable in PRINT, DISP, STAT, REAL, SHORT, INTEGER, DIM.• Dummy array variable in SUB parameter list.</div>
77	<div>Quote Expected<p>Missing closing quote in INPUT and DATA statements.</p></div>
78	<div>Excess Chars<ul style="list-style-type: none">• Extra characters at the end of an otherwise legal statement.• Statement followed by @ when not allowed.• DEF FN: More than 14 parameters specified in parameter list.</div>
79	<div>Illegal Context<ul style="list-style-type: none">• Statement not programmable.• Statement not keyboard executable.• Statement not legal in an IF construct.</div>

(Continued on next page.)

Syntax Errors (continued)

Error Number	Message and Condition
80	<p>Invalid Expr</p> <ul style="list-style-type: none"> • Syntax error in an expression. • Assignment with type mismatch (string on one side, numeric on other). • Quoted label declaration without a closing quote. (Statement interpreted as an implied DISP.) • File command interpreted as an unquoted label declaration due to no spaces; for example: 10 PURGE:PORT, where PURGE is interpreted as a label declaration, and the computer interprets PORT as an implied DISP statement.
81	<p>Invalid Parm</p> <p>Improper input is entered for an expected parameter.</p> <ul style="list-style-type: none"> • Line number expected in LIST, MERGE, AUTO, DELETE, RENUMBER, and DISP/PRINT USING. • Invalid keyword in OPTION or DEFAULT statement. • Invalid parameter in SUB parameter list.
82	<p>Missing Parm</p> <p>No input received for expected parameter:</p> <ul style="list-style-type: none"> • Line number expected in LIST, MERGE, AUTO, DELETE, RENUMBER, and DISP/PRINT USING. • Missing keyword in OPTION or DEFAULT statement. • Missing parameter in SUB parameter list.
83	<p>Invalid Var</p> <p>Either an array variable has been specified where a simple variable is required, or a variable of the wrong data type has been specified.</p>
84	<p>Precedence</p> <p>After single-step in CALC mode, an operator has been entered that has a higher precedence than the operator that was executed by the last single-step operation.</p>
85	<p>Invalid Key</p> <p>In CALC mode:</p> <ul style="list-style-type: none"> • A <i>system</i> typing aid begins with a space. Does not apply to user-defined typing aids. • User-defined key that is an immediate execute key definition instruction (specified with : when defining).

(Continued on next page.)

Syntax Errors (continued)

Error Number	Message and Condition
86	Operand Expected In CALC mode an operator is input when an operand is required.
87	Operator Expected In CALC mode an operand is input when an operator is required.
88	TFM WRN Lnnn: <i>message</i> The syntax error message occurred on source file line <i>nnn</i> during execution of TRANSFORM. This warning condition is reported during the transformation, but does not halt execution.

Card Reader Messages (89 through 97)

Message Number	Message and Condition
89	Pull <i>nnn</i> of <i>mmm</i> Prompt for all write and verify pulls, and for the second and any subsequent pulls needed to read a file.
90	Pull Card Prompt for first read pull, and for CAT, PROTECT, and UNPROTECT.
91	Wrt: Align then ENDLN Prompt for write (COPY TO CARD) operation.
92	Vfy: Align then ENDLN Prompt for write-verify operation.
93	Read: Align then ENDLN Prompt for read (COPY CARD) operation.
94	Prot: Align then ENDLN Prompt for PROTECT operation.
95	Unpr: Align then ENDLN Prompt for UNPROTECT operation.
96	Cat: Align then ENDLN Prompt for CAT CARD operation.
97	Trk <i>nnn</i> Done After verifying or reading each track.

HP-71 Exception Flag Summary

The following summarizes the conditions under which the HP-71 sets any of the five exception flags.

IVL (Invalid Operation)

The IVL exception (sets flag -8) occurs when an operand has the appropriate data type, but its value is a signaling NaN and/or is invalid for the operation to be performed.

DVZ (Division By Zero)

The DVZ exception (sets flag -7) occurs when a finite operand produces an exact infinite result ($+\text{Inf}$ or $-\text{Inf}$). The actual result depends on the `TRAP` setting.

OVF (Overflow)

The OVF exception (sets flag -6) occurs when the magnitude of a result is too large to fit into its destination format.

UNF (Underflow)

The UNF exception occurs when the magnitude of a nonzero result is less than $1\text{E}-499$. If `TRAP (UNF)` $\neq 2$, the underflow exception flag (flag -5) is set whenever the underflow exception occurs. Otherwise, the HP-71 sets this flag only if the underflow exception occurs and the result cannot be exactly represented in the destination's denormalized format.

INX (Inexact Result)

The INX exception (sets flag -4) occurs when a result may be inexact. If the HP-71 does not set the inexact result flag, the result is exact. For several functions (`+`, `-`, `*`, `/`, `SQR`, and `RED`), the flag is set only if the result is inexact. However, for some compound functions like `^` and the statistics functions, the computer sometimes sets the flag for results that are actually exact.

HP-71 Keyword Index and Summary

Some keywords appear in more than one category.

Keyword	Page	Description
Program Entry/Editing		
AUTO	26	Numbers lines automatically.
DELETE	77	Deletes program line(s) from current file.
EDIT	91	Assigns "current file" status to specified file.
FETCH	110	Displays any line of current program.
LIST	173	Displays listing of specified lines in a file.
NAME	191	Names the work file.
PLIST	211	Prints listing of specified lines in a file.
PRIVATE	225	Limits access to file and restricts changes in its protection.
REM (!)	242	Enables entry of comments in program lines for program documentation.
RENUMBER	245	Renums lines in current file.
SECURE	260	Protects file from being altered or purged.
TRANSFORM	289	Transforms BASIC file to TEXT file, or reverse.
UNSECURE	297	Clears file access restriction set by SECURE.
@	306	Appends a statement in a multiple-statement line.
Program Execution		
CALL	31	Transfers program execution to subprogram.
CHAIN	42	Purges current file, copies specified file into main RAM, and executes that file.
CONT	52	Continues execution of suspended program.
RUN	255	Executes a BASIC or binary program.

Keyword	Page	Description
Program Control		
BYE	30	Turns computer off.
CALL	31	Transfers program execution to subprogram.
CHAIN	42	Purges current file, copies specified file into main RAM, and executes that file.
DEF FN	67	Indicates beginning of user-defined function definition.
END	93	Terminates a subprogram, user-defined function, or program.
END DEF	94	Causes normal return from a multiple-statement user-defined function.
END SUB	94	Causes normal return from subprogram invoked by CALL statement.
FN	116	Transfers program execution to specified user-defined function.
FOR...NEXT	118	Defines loop that is repeated until loop counter exceeds specified value.
GOSUB	129	Transfers program execution to subroutine.
GOTO	131	Transfers program execution to specified statement.
IF...THEN...ELSE	134	Provides conditional execution.
OFF	195	Turns computer off.
OFF ERROR	195	Disables any previous ON ERROR statement.
OFF TIMER	195	Deactivates corresponding ON TIMER # statement.
ON ERROR GOSUB	197	Executes specified subroutine when an error occurs.
ON ERROR GOTO	197	Executes specified branch when an error occurs.
ON TIMER #	201	Interrupts program at specified time and causes specified branching to occur.
ON...GOSUB	199	Transfers program execution to selected subroutine.
ON...GOTO	199	Transfers program execution to selected statement or line.
ON...RESTORE	199	Selects which DATA statement will be used by next READ statement.
PAUSE	208	Suspends program execution.
POP	215	Cancels pending return of program execution from current subroutine.

Keyword	Page	Description
RETURN	251	Returns program execution to statement following invoking GOSUB.
STOP	279	Terminates a subprogram, user-defined function, or program.
SUB	282	Identifies beginning of subprogram.
WAIT	302	Causes program execution to wait for specified number of seconds.
Debugging		
CONT	52	Continues execution of suspended program.
DEFAULT	72	Sets math exception traps to specific values.
ERRL	100	Returns line number of most recent error or warning.
ERRM\$	101	Returns message text of most recent error or warning.
ERRN	102	Returns error number of most recent error or warning.
ON ERROR GOSUB	197	Executes specified subroutine when an error occurs.
ON ERROR GOTO	197	Executes specified branch when an error occurs.
PAUSE	208	Suspends program execution.
TRACE	288	Traces program execution and variables in a running program.
Storage Allocation		
CLAIM PORT	48	Returns independent RAM to main RAM status.
DESTROY	78	Deletes variables and arrays from memory.
DIM	79	Allocates memory for string or REAL variables and arrays.
FREE PORT	122	Switches a portion of main RAM to independent RAM status.
INTEGER	155	Creates INTEGER variables and arrays.
MEM	184	Returns number of bytes available in memory.
OPTION BASE	204	Specifies subscript lower bounds for arrays.
REAL	238	Creates REAL variables and arrays.
SHORT	269	Creates SHORT variables and arrays.
SHOW PORT	271	Displays type and size of all plug-in memory devices and independent RAMs.
STAT	275	Selects or creates a statistical array.

Keyword	Page	Description
Logical and Relational Operators		
AND	19	Performs logical And of its operands.
EXOR	105	Performs logical Exclusive Or of its operands.
NOT	193	Performs logical Not of its operand.
OR	206	Performs logical Or of its operands.
=	317	Performs Equality test on its operands.
#	317	Performs Inequality test on its operands.
<>	317	Performs Less Than or Greater Than test on its operands.
<	317	Performs Less Than test on its operands.
<=	317	Performs Less Than or Equal To test on its operands.
>	317	Performs Greater Than test on its operands.
>=	317	Performs Greater Than or Equal To test on its operands.
?	317	Performs Unordered Comparison test on its operands.
Arithmetic Operators		
+	308	Addition.
-	309	Subtraction.
*	310	Multiplication.
/	311	Division.
DIV	87	Divides one argument by another and returns integer portion of quotient.
^	312	Exponentiation.
%	313	Percent.
General Math		
ABS	10	Returns absolute value of its argument.
CEIL	40	Returns smallest integer greater than or equal to specified argument.
CLASS	49	Returns value indicating class of argument.
DVZ	90	Returns divide-by-zero flag number (-7).
EXPONENT	108	Returns exponent of its normalized argument.
FACT	109	Returns factorial of non-negative integer argument.
FLOOR	115	Returns greatest integer less than or equal to argument.

Keyword	Page	Description
FP	121	Returns fractional part of numeric value.
INT	154	Returns greatest integer less than or equal to argument.
INX	157	Returns inexact result flag number (−4).
IP	158	Returns integer part of argument.
IVL	159	Returns invalid operation flag number (−8).
LET	168	Assigns value to variable.
MAX	181	Returns larger of two values.
MIN	188	Returns smaller of two values.
MOD	190	Returns remainder of modulo reduction.
OPTION ROUND	204	Selects roundoff setting.
OVF	207	Returns overflow flag number (−6).
RANDOMIZE	233	Specifies a “seed” for the RND function.
RED	240	Returns remainder of argument reduction.
RES	247	Returns value of most recently executed numeric expression.
RMD	252	Returns remainder of division.
RND	254	Returns next real number in a pseudo-random number sequence and updates current seed.
SGN	268	Returns −1, 0, or 1 if argument is less than zero, equal to zero, or greater than zero, respectively.
SQR	273	Returns square root of argument.
SQRT	273	Alternate spelling for SQR.
UNF	295	Returns underflow flag number (−5).
Logarithmic Operations		
EXP	106	Returns the number $e = 2.718281828 \dots$ raised to power given by argument.
EXPM1	107	Returns value of $e^{\text{argument}} - 1$.
EXPONENT	108	Returns exponent of its normalized argument.
LGT	178	Alternate spelling for LOG10.
LN	176	Alternate spelling for LOG.
LOG	176	Returns natural logarithm (base e) of argument.
LOGP1	177	Returns $\ln(1 + \text{argument})$.
LOG10	178	Returns logarithm (base 10) of argument.

Keyword	Page	Description
Trigonometric Operations		
ACOS	11	Returns arccosine of its argument.
ACS	11	Alternate spelling for ACOS.
ANGLE	20	Returns polar angle determined by (x,y) coordinate pair.
ASIN	22	Returns arcsine of its argument.
ASN	22	Alternate spelling for ASIN.
ATAN	25	Returns arctangent of its argument.
ATN	25	Alternate spelling for ATAN.
COS	59	Returns cosine of its argument.
DEG	73	Converts argument in radians to degrees.
DEGREES	74	Sets unit of measure for expressing angles to degrees.
OPTION ANGLE	204	Selects unit of measure for expressing angles.
RAD	231	Converts arguments expressed in degrees to radians.
RADIANS	232	Sets unit of measure for expressing angles to radians.
SIN	272	Returns sine of its argument.
TAN	284	Returns tangent of its argument.
Statistics		
ADD	12	Adds coordinates of a data point to data set represented by summary statistics in current statistical array.
CLSTAT	51	Clears all elements in current statistical array.
CORR	58	Returns sample correlation coefficient between a specified pair of variables.
DROP	88	Removes coordinates of a data point from the data set represented by summary statistics in current statistical array.
LR	179	Specifies current linear regression model and computes intercept and slope for that model.
MEAN	183	Returns sample mean of specified variable.
PREDV	218	Returns predicted value of dependent variable.
SDEV	259	Returns standard deviation of specified variable.
STAT	275	Selects or creates statistical array.
TOTAL	287	Returns total of specified variable.

Keyword	Page	Description
Constants		
EPS	99	Returns HP-71's smallest positive, normalized number (1.0 E−499).
INF	150	Returns machine representation of positive infinity.
MAXREAL	182	Returns maximum positive finite number that the HP-71 can represent (9.999999999999E499).
MINREAL	189	Returns smallest positive number that HP-71 can represent (0.000000000001E−499).
NAN	192	Returns Signaling NaN.
PI	210	Returns 12-digit value representing π .
Strings		
&	307	Concatenation operator.
CHR\$	47	Converts numeric value into ASCII character.
LEN	167	Returns length of specified string.
NUM	194	Returns ASCII numeric code for first character of string.
POS	216	Returns position of given substring.
STR\$	280	Returns string representation of value of argument.
UPRC\$	298	Converts lowercase letters to uppercase.
VAL	300	Converts a numeric expression within a string expression to a numeric value.
VER\$	301	Indicates versions of system ROMs and LEX files.
Input/Output		
ASSIGN #	23	Associates symbolic channel number with specified file and opens that file.
BEEP	28	Causes specified tone to sound.
BEEP OFF	28	Disables beeper.
BEEP ON	28	Enables beeper.
CONTRAST	54	Adjusts display contrast.
COPY	55	Copies information from source file to destination file.
CREATE	60	Creates a data file.
DATA	62	Contains data that can be read by READ.
DELAY	75	Sets line and character scroll rates in display.

Keyword	Page	Description
DISP	82	Displays numeric and string data.
DISP USING	84	Displays items according to specified format.
DISP\$	86	Returns string containing all readable characters in display.
ENDLINE	96	Specifies end-of-line sequence used in PRINT and PLIST statements.
ENG	97	Selects engineering display format.
FIX	112	Selects fixed display format.
GOISP	124	Sets specified dot pattern in display.
GOISP\$	127	Returns 132-character string reflecting dot pattern in display.
IMAGE	136	Controls format of displayed and printed output.
INPUT	151	Enables assigning values to program variables from keyboard.
KEYDOWN	164	Returns 0 or 1, depending on whether key is being pressed.
LC	166	Selects between uppercase and lowercase lock on keyboard.
LINPUT	171	Assigns display line to string variable.
LIST	173	Displays listing of specified lines in a file.
ON...RESTORE	199	Selects which DATA statement will be used by next READ statement.
PLIST	211	Prints on print device a listing of specified lines in a file.
PRINT	219	Causes print list to be sent to print device.
PRINT USING	221	Causes print list to be sent to print device according to specified format.
PRINT #	223	Writes data items to data file in memory.
PUT	229	Enters a specified key code into key buffer.
PWIDTH	230	Defines line length of PRINT and PLIST statements.
READ	234	Assigns values from DATA statements to variables.
READ #	236	Reads data items from data file.
RESTORE	249	Specifies which DATA statement will be used by next READ operation.
RESTORE #	250	Sets specified file pointer to indicated record number.
SCI	257	Selects scientific notation display format.
STD	277	Selects standard BASIC display format for numbers.

Keyword	Page	Description
TAB	{ 82 219	Moves DISP or PRINT position ahead to specified column. (Refer to the DISP or PRINT keyword entry.)
UPRC\$	298	Converts lowercase letters to uppercase.
USER	299	Activates or deactivates user-defined key assignments.
WIDTH	303	Defines line length for DISP and LIST statements.
WINDOW	305	Sets display window size and location.
Graphics		
GDISP	124	Sets specified dot pattern in display.
GDISP\$	127	Returns a 132-character string reflecting dot pattern in display.
File Management		
ADDR\$	13	Returns string representing hexadecimal address of specified file.
CAT	35	Gives catalog of file information.
CAT\$	38	Returns catalog information for a specified file.
CLAIM PORT	48	Returns independent RAM to main RAM status.
COPY	55	Copies information from source file to destination file.
CREATE	60	Creates a data file.
EDIT	91	Assigns "current file" status to specified file.
FREE PORT	122	Switches a portion of main RAM to independent RAM status.
MEM	184	Returns number of bytes available in memory.
MERGE	186	Merges all or part of file into another file.
NAME	191	Names system workfile.
PRIVATE	225	Limits access to file and restricts changes in its protection.
PROTECT	226	Write-protects one track of a magnetic card.
PURGE	227	Deletes file from RAM.
RENAME	243	Changes name of file.
SECURE	260	Protects file from being altered or purged.
SHOW PORT	271	Displays type and size of all plug-in memory devices and independent RAMs.
TRANSFORM	289	Transforms BASIC files into TEXT files, or the reverse.
UNPROTECT	296	Removes the write-protection from one track of a magnetic card.
UNSECURE	297	Clears file access restriction set by SECURE.

Keyword	Page	Description
Time and Date		
ADJABS	14	Performs an absolute adjust on system clock.
ADJUST	15	Changes clock time and specifies clock speed correction.
AF	17	Returns current value of clock accuracy factor and gives option of setting new adjustment factor.
DATE	65	Returns current clock date as an integer (YYDDD).
DATE#	66	Returns current clock date in year/month/day format.
EXACT	103	Calibrates system clock and tells HP-71 that time currently stored is the correct time.
RESET CLOCK	248	Nullifies effect of executing EXACT.
SETDATE	262	Sets date on system clock.
SETTIME	264	Sets time on system clock.
TIME	285	Returns time of day in seconds since midnight.
TIME#	286	Returns time of day in HH:MM:SS format.
System Settings and Flags		
CFLAG	41	Clears specified user and/or system flags.
DEFAULT	72	Sets math exception traps to specific values.
DEGREES	74	Selects degrees as unit of measure for angles.
DELAY	75	Sets line and character scroll rates in display.
DVZ	90	Returns divide-by-zero flag number (−7).
FLAG	114	Returns current value (0 or 1) of specified flag, and optionally selects new flag setting.
INX	157	Returns inexact result flag number (−4).
IVL	159	Returns invalid operation flag number (−8).
OPTION ANGLE	204	Specifies unit of measure for expressing angles.
OPTION BASE	204	Specifies subscript lower bounds for arrays.
OPTION ROUND	204	Specifies round-off setting.
OVF	207	Returns overflow flag number (−6).
RADIANS	232	Selects radians as unit of measure for angles.
RESET	248	Resets user and system flags and traps to their system default settings.
SFLAG	267	Sets specified user and/or system flags.

Keyword	Page	Description
TRAP	293	Returns trap for specified flag number and optionally selects new trap setting.
UNF	295	Returns underflow flag number (−5).
Customization, Keyboard, and Display Control		
ADDR\$	13	Returns string representing hexadecimal address of specified file.
CHARSET	43	Specifies alternate character set in ASCII code range of 128 through 255.
CHARSET\$	46	Returns string representing current alternate character set.
CONTRAST	54	Adjusts display contrast.
DEF KEY	69	Assigns character string to specified key.
DELAY	75	Sets line and character scroll rates in display.
DTH\$	89	Converts decimal number to string representing its five-digit hexadecimal value.
FETCH KEY	111	Displays specified key assignment for editing.
FIX	112	Sets fixed display format and number of fractional digits to be displayed.
HTD	133	Converts string argument representing hexadecimal number to decimal number.
IMAGE	136	Controls format of displayed and printed output.
KEY	69	Assigns character string to specified key.
KEY\$	160	Returns and deletes oldest key or keystroke combination from keyboard buffer.
KEYDEF\$	162	Returns redefined value of a key.
KEYDOWN	164	Returns 0 or 1, depending on whether key is being pressed.
LC	166	Selects between uppercase and lowercase lock on keyboard.
LOCK	175	Sets password. Causes HP-71 to prompt for that password the next time computer is turned on.
PEEK\$	209	Returns contents of specified section of memory.
POKE	213	Writes to memory at specified hexadecimal address.
PUT	229	Enters a specified key code into key buffer.
STARTUP	274	Defines command string to be executed when HP-71 is turned on.
USER	299	Activates or deactivates user-defined key assignments.
WINDOW	305	Sets display window size and location.

Subject Index

Page numbers in **bold** type indicate primary references; page numbers in standard type indicate secondary references. Because the manual mentions some indexed topics only in a secondary sense, such topics are listed without any primary references. Also, because the HP-71 keywords are listed alphabetically in the Keyword Dictionary, main keyword entries are not indexed.

A

- Abort, transformation, 292
- Absolute value, 10
- Access
 - random, **224, 237**
 - sequential, **224, 237**
- Accuracy, π , **284**
- Accuracy factor, default, 329
- Actual parameter, 68, 116
- Addition, **308**
- Address, hexadecimal, 89, 133, 209, 213
- Adjust clock, **15**
- Adjust clock, absolute, 14
- Adjustment
 - factor, **17, 104**
 - interval, **17**
 - period, clock, **103-104**
- Allocating memory, **79, 155, 156**
- Altering a file, protection, **260**
- Alternate character set, 43-45, 46, **323-326**, 330
 - deactivating, **45**
 - memory, **45**
- Ampersand, **307**
- Angle, unit of measure, **204-205**
- Angular setting, 11, 20, 22, 25, 59
- Angular setting default, **74**
- Annunciator
 - flags, 319
 - RAD**, 205, 232
 - SUSP**, 110, 256, 53
- ANSI
 - BASIC Standard, 252
 - minimal BASIC, 83
 - minimal BASIC Standard X3J2, 277, 280
- Antilogarithm, 106, 107
- Argument
 - normalized, 108
 - pairs, invalid, 20
- Array, 155, 156
 - bounds, **316**
 - creating, **169**
 - deleting, **78**
 - integer, **155-156**
 - lower bounds, 204, 205
 - pass by reference, 32
 - real, 79-80, 238-239
 - real string, 80
 - redimensioning, 80, 156, 239
 - SHORT, 269-270
 - statistical, 12, 88, 218, 259, 275, 276, 287, **334-337**
 - statistical default `OPTION BASE` setting, 275, 276
 - string, 333
 - variable, 315, 316
 - variable, parameter passing, 282
- ASCII, 61, 326
 - character, 47
 - character code, 43, 328
 - string, 237
- Assigning variables, **151-153**
- Assignment
 - statement, **168-170**
 - variable, **234-235**
- At (@) symbol, 274
- At (@) symbol, with `IMAGE`, **136**
- `ATTN`, during input, **153**
- Automatic
 - line numbering, **26**
 - startup, **274**
 - timeout, **153**

B

Base 10, logarithm, **178**
 BASIC file, 225
 BASIC file, memory requirements, 331
 BASIC-to-TEXT transformation, **290**
 Batteries, 329
 BEEP, default setting, **330**
 BIN file, 225
 BIN file, memory requirements, 332
 Binary
 codes, 323-326
 operator, 206, **308-313**
 program, 255
 representation, display, **124-126**
 subprogram, 283
 Bit pattern, 127, **124-126**
 Bit pattern, locking, 126
 Blank, 8
 leading, 63, 82
 trailing, 63, 82
 use in inputs, **9**
 Branch, 197
 on error, 197
 timer, **201-203**
 tracing, **288**
 unconditional, 131, 135
 Buffer
 display, 124
 input, 153
 key, 160, 229
 B~~Y~~E, operation with timer, **202**

C

CALC, during input, **153**
 mode, 247, 274
 mode, assignment statement, 168
 Calendar date, **262-263**
 Calibrate, clock, 15, **103-104**
 Call
 function, 235
 subroutine, 197
 Calling program, 63, 233
 Card
 copying, 226
 file, private, 56
 magnetic, 226, 296
 protection, 296
 reader, 36, 123, 226
 verifying **56**
 Carriage return, 138, 220
 Carriage return/line feed, 96. See also *CR/LF*.

Case lock, **166**
 Catalog
 listing, 260
 purged, 42
 string form, 38
 unsecured private file, 225
 Channel, 23, 24, 32, 33
 Channel number, 224, 236, 237, 250, 315
 Character
 code, 124, 125, **322-326**
 rate, **75**
 readable, 86
 set, **322-326**
 set, alternate, 43-45, 46, 330
 set, standard, 43
 width, 44
 Characters, control, **326-327**
 C~~H~~R\$, 125, 127, 139, 145, 164, 194,
 with character codes, 322-326
 with escape sequence, 328
 C~~L~~A~~S~~S, 342
 Classes of numbers, **342**
 Clearing a flag, **114**
 Clock, **103, 248, 262-266**
 absolute adjust, 14
 adjustment
 calibrate, 15
 default settings, **329**
 factor, **17, 103-104**
 interval, **17**
 resolution, **16, 266**
 setting, 103
 speed, 14, 15
 stored correction, 15
 time, 14
 Codes, binary, 323-326
 Codes, hexadecimal, 323-326
 Columns, display, 44
 Command stack, during input, **153**
 Command stack memory requirements, 330
 Commands, **5**
 Comments, **242**
 Comparison table, math exceptions, **344**
 Comparisons, unordered (?), **343**
 Concatenation, 30, 34, 129, 130, 196, 274
 I~~M~~A~~G~~E, 136
 operator, 5
 statements, **306**
 string, **307**
 Conditional execution, **134-135**
 Congruential method, linear, 254

Constants, memory requirements, 330
 Continuing a program, 208
 Control characters, 326-327. See also **CTRL**.
 Conversion
 degrees to radians, 231
 numeric-to-string, 280
 string-to-numeric, 300
 Coordinate pair, 20
 Coordinates, of data point, 12
 Copying a card, 226
 Counter, loop, 118, 119
 CR/LF, 96, 220
CTRL, 328
CTRL keystrokes, 323. See also *Control characters*.
 Current file, 91, 227, 228, 256
 Current file, assigning, 36
 Customizing the keyboard, 69

D

Data
 assignments, 234-235
 file, 60, 61, 223, 224, 236, 237, 250
 file channels, memory requirements, 330
 memory requirements, 331
 point, 334, 336
 pointer, 199, 200, 237
 reading, 234-237, 249
 type, changing, 239
 Date, setting, 262-263
 Deactivating timers, 196
 Deassigning keys, 70
 Debugging, 52
 Decompile, TEXT/LIF1 file, 290
 DEFAULT, math exceptions, 339
 DEFAULT EXTEND, 341
 Default
 angular setting, 74
 environment, 314
 settings, 329-330
 value, math exceptions, 339-343
 Degrees setting, 11
 Degrees-to-radians conversion, 231
 Deleting
 file, 227-228
 line number, 27
 variables and arrays, 78
 Delimiter, 9, 143, 171
 field, 142
 IMAGE, 146, 147
 Denormalized number, 98, 189, 257, 277, 317, 341-342
 Dependent variable, 179, 180, 218
 Destination file, 55-57, 187, 225, 290
 Deviation, standard, 259
 Device chain, 123
 Device number, 48
 Direct execute key, 162, 256
 Direct execute user-defined key, during input, 153
 Direct execution, 69, 70, 111,
 DISP, implied, 135
 Display, 124-126, 127, 128
 buffer, 124
 character set, 322-326
 columns, 44
 escape code sequences, 328
 format, 277-278
 format control, 136-149
 formatting, 83, 85
 line length, 303
 list, 84
 locking a bit pattern, 126
 rounding, 280
 setting, 112, 257
 setting, changing, 278
 setting, default, 329
 string, 86
 window, 305
 zones, 82
 Division-by-zero, 338, 393
 Division-by-zero flag, 108
 Dot pattern, 124-126, 127
 Dot pattern display, 305
 Duplicate names, subprograms, 33
 Duration, 28, 29
 DVZ, 338
 DVZ, 108, 150, 393

E

e, 106, 107, **176**
 Editing, 53, 91, 131
 End-of-file mark, 224
 End-of-line sequence, **96**, 138
 ENDLIN, default setting, **329**
 Endline string, **96**
 Engineering display format, **97-98**, 281
 Environment, 33, 34, 256, **314**
 global, 33
 local, 42, 228, 92
 Environment, maintaining, 314
 Environment, user-defined function, 117
 EOL, 152
ERRM, during input, 153
 Error
 clock precision, 103
 correction, accumulated, 16
 during input, 153
 during transformation, **291-292**
 handling, in programs, **197-198**
 IMAGE syntax, 143
 in subprogram, 100
 line number, finding, 100
 message, **378-392**
 message, defined, 378
 nonBASIC program, 100
 number, 102
 reporting, 195
 rounding, 339
 rounding, string, 101
 program, 197
 Escape code sequences, display, **328**
 European digit separator, 141
 European radix, 139, 141, 144
 Exact flag, 103
 Exception
 flag, 72
 flag, DVZ, 108
 flag, DVZ, summary, **393**
 math, comparison table, **344**
 traps, 72
 Exclusive or, 105
 Execute only string, 172
 Execution,
 suspend, 117, 130
 transferring, **251**
 Exponent
 DISP or PRINT USING maximum, **148**
 range, **182**
 Exponentiation, **312**
 Extended range computations, 108

F

Field delimiter, 142
 File
 access restriction, **260, 297**
 alter or purge protection, **260**
 automatic purge, during transformation, **292**
 backup, 292
 BASIC, 225
 BIN, 225
 creating, 91-92
 current, 36, 91, 227, 228, 256
 DATA, 224, 237, 250
 data, 223, 236
 default condition, **329**
 deleting, **227-228**
 destination, 55-57, 187, 225, 290
 header, 13, 77
 independent of main RAM, 122
 KEY, 243
 key assignment, 297
 keys, 212, 228, 299
 LEX, 57, 292, 301
 LIF1, 37, 57
 list, **173-174**
 listing, 225
 listing on printer, **211-212**
 memory requirements, 331
 merging, **186-187**
 pointer, 110
 private, 209, 213, **225**, 260
 private secured, 225
 protection, 36, **260**
 purging. *See deleting.*
 renaming, **243-244**
 reset, 48
 SDATA, 224, 237, 250
 source, 55-57, 187, 290
 TEXT, 224, 237, 250
 type unrecognized, 39
 unrecoverable, during transformation, 292
 unsecured, 225, 228, 297
 workfile, **191**
 Final value, loop, 119
 Fixed format, 112
 Fixed-precision display format, 280

Flag, 248
 -5, 295
 -10, 205
 -15, 166
 -46, 248
 clearing, 114
 default settings, 329
 divide-by-zero, 108, 90
 Exact, 103
 exception summary, 393
 inexact result, 104, 157, 190
 invalid operation, 159
 INX, 252
 math exception, 338
 numbers
 overflow (-6), 207
 set, clear, test, save, 114
 setting, 114, 267
 subprogram, 314
 system, 319
 underflow exception (-5), 189
 volume, 28

Floating specifiers, 147

FOR...NEXT loops, memory requirements, 332

Form feed, 138

Formal parameter, 33, 67, 68, 116

Format
 control, display and printer, 136-149
 display, 83, 85, 277-278
 engineering, 97-98
 fixed, 112
 image, 221
 Keyword Dictionary, 6-7
 scientific, 257-258
 string, 85, 138, 222

Fractional part, 121

Frequency, 28, 29

Function, call, 235

Function references, memory requirements, 331

G

Global
 angular settings, 74
 declaration, 233
 effect, settings, 205
 environment, 33, 314
 timer, 203
 tracing, 288
 variables, 68

Glossary, using, 9

GOTO, implied, 135

Gradual underflow, 341

Graphics, display, 124-128

H-I

Hexadecimal
 address, 13, 209, 213
 codes, 323-326
 conversion, 89
 to decimal conversion, 133

HP 82400A card reader, 296. See also *Card reader*.

HP Logical Interchange Format (See *LIF1*.)

HP-41, 61

HP-75, 61, 290

IEEE
 Floating Point Standard, 241
 Math Exceptions, 338-345
 remainder function, 241
 traps, 149

IMAGE, location in program, 137

Image format, 221

Immediate execution, 70, 111

Implied
 DISP, 135
 GOTO, 135
 negative, 147

In-place transformation, 290, 292

Independent RAM, 122

Independent variable, 179, 180, 218

Inexact, 157
 flag, 104, 190
 result, 339, 393

INF, 169

Inf, 280, 300, 310, 311, 312, 313, 341
 in IMAGE, 148-149
 loop control value, 119
 subtraction, 309

Infinite loop, 119, 198

Infinity, 341

Infinity, positive, 150

Initial value, loop, 119

Initialize, 155, 238, 239, 269
 data, signalling NaN, 341
 loop, 119
 numeric array, 155

Input, line, 171-172

Insufficient Memory, 122

INTEGER, 169

INTEGER variable, 247

Integer, greatest, 115

Integer, part, 158

Intercept, 179, 180

Interchanging files, 289

Invalid operation, **338, 393**
 Invalid operation flag, **159**
 Inverse transformation, 292
 INX, **339**
 INX flag, 190, 252
 IVL, **338, 393**
 IVL trap value, loop control, **119**

K

KEY file, 211, 243
 KEY file, memory requirements, 331
 Key
 assignment, **70, 111, 243**
 assignment file, 297
 buffer, 160, 229
 code, 229
 deassigning, **70**
 definition, **162**
 direct execute, 162, 256
 file, list, 173, 174
 nonterminating, 162
 renaming, **243**
 terminating, 162
 test, **164, 165**
 Keyboard, concatenation, 196
 Keyboard control, 129
 keys, 297
 keys file, **70, 186, 212, 228, 299**
 Keystroke combinations, **319-322**
 Keyword
 combined entry, 5
 defined, 5
 dictionary, how to use, 5
 dictionary format, **6-7**
 finding entries, 5
 index, using, **9**
 middle, 5
 operators, 5
 plug-in-module, 37
 related, 6

L

LCD, 305
 Leading blank, 63
 Length of string expression, **167**
 LEX file, 57, 292, 301
 LEX file memory requirements, 332
 LIF1, 57, 60, 61
 LIF1 file, 37, 290
 Limits of numerical representation, 318
 Line
 feed, 138, 220
 input, **171-172**
 length, **303**
 length, maximum, during transformation, 291
 length, printer, **230**
 renumbering, 187
 Line number, 131
 automatic, **26**
 deleting, **27**
 Line rate, **75**
 Lines, renumbering, **245-246**
 Linear congruential method, 254
 Linear regression, **179, 180**
 Linear regression, simple, **337**
 List display, 84
 List halt, **174**
 Listing file, 225
 Listing on a printer, **211-212**
 Local environment, 33, 34, 42, 228
 Lock, case, **166**
 Locking a bit pattern, 126
 Locking the HP-71, **175**
 Logarithm, natural, **176, 177**
 Logarithm base 10, **178**
 Logical NOT, **193**
 Logically false, 19
 Loop, **118, 119, 122, 127, 128**
 Loop, infinite, 198
 Loss of memory, **213**
 Lower bounds, subscript, 204, 205
 Lowercase, **8, 166, 298**

M

Machine-representable numbers, 50
 Magnetic card, 36, 226, 296. See also *Card reader*.
 file size, 37
 HP-75 file, 37
 overhead, 37
 track, 36
 Main
 environment, 314
 program, variables, 130
 RAM, 122
 Matched sample, 334
 Math exception, 338-345
 comparison table, 344
 CFLAG, 338
 flag, 267
 traps, 72
 Math operators, 5
 Math trap, 248
 Maximum real number, 182
 Maximum value, 181
 Mean, sample, 183
 Memory 184-185
 allocating, 79, 155, 156
 allocating, suspended program, 208
 conserving, 306, 333
 contents, 209
 DATA files, 224
 external, 184
 insufficient during transformation, 292
 loss, 213, 329
 reclaiming, 78
 releasing, 93, 94, 95
 reset, 100, 101, 316, 329
 reset, flags, 319
 SDATA files, 224
 system requirements, 330-332
 type, 271
 use, user-defined function, 94
 writing to, 213
 Memory Lost, 122
 Memory Lost condition, 329
 Merging files, 186-187
 Message string, error or warning, 101
 Minimum of two values, 188
 Minimum real number, 189
 Minus, 309
 Mode
 CALC, 274
 radians, 232
 SCI, 257-258

Modulo, 190, 272, 284, 304
 Multiple-statement function definition, 94
 Multiple-statement line, 137
 Multiplication, 310
 Multistatement line, 306
 restriction, 306
 string, 274
 string or command, 274

N

NaN, 150, 192, 280, 300, 340-341, 343
 in IMAGE, 148-149
 loop control value, 119
 sign, 309
 Natural logarithm, 176
 Negative, implied, 147
 Nested parentheses, 138
 Nesting, IF...THEN...ELSE, 135
 NonBASIC program, error, 100
 Nonprogrammable function, 271
 Nonprogrammable statement, 5, 110, 111, 122, 26, 48, 52, 77, 91
 Nonterminating key, 162
 Normalized
 argument, 108
 number, 98, 257, 342
 number, smallest positive, 99
 Not-a-number, 340-341. See also *NaN*.
 Notation, scientific, 257
 Null, 80, 247, 274
 argument, 175
 string, 101, 133, 160, 194, 274
 string, KEYDOWN parameter, 165
 unquoted, 152
 Number
 classes, 342
 denormalized, 189
 machine-representable, 50
 Numeric
 expression, 318
 precision, 80
 settings, default, 329
 variable, creating, 169
 variable, REAL, 80, 239
 variable, SHORT, 270
 Numeric-to-string conversion, 280

O

Off, 30
 Off, operation with timer, 202
 On, continuous, default, 330
 Opening a file, 60
 Operator
 binary, 308-313
 concatenation, 5
 descriptions, location 5
 math, 5
 memory requirements, 331
 precedence, 105, 317
 relational, 5
 OPTION BASE setting, 80
 Out-of-place transformation, 290
 Output field, reusing, 142
 Output list, 138, 142
 Overflow, 108, 207, 338-339, 393
 Overflow, factorial, 109
 Overflow, IMAGE, 142
 Overflow, string, 169
 Overflow, threshold, 182, 317, 318
 OVF, 150, 338-339, 393
 OVF, trap, 142, 148

P

Padded file size, 37
 Parameter
 actual, 116
 formal, 33, 116
 passing, 31, 68, 116
 passing, subprograms, 282-283
 Pass by reference, 32
 Pass by value, 32
 Passing parameters, 31, 32
 Passing parameters by value, 68
 Password, 175
 Password, default, 329
 FEN, 145
 Pending return, 215
 Percent, 313
 Pi (π), accuracy, 272
 Pi (π), number of digits, 59
 Piston program, 127, 128
 Pointer
 data, 199, 200, 235, 237
 file, 110, 122, 250
 program, 200
 Polar angle, 20

Port, 122-123, 184, 271
 device number, 122, 123
 HPIL, 123
 search, 92
 Postfix form, 333
 Precedence, operators, 105, 317
 Precision, numeric, 80
 Predicted value, 218
 Print list, 221
 Printer
 format control, 136-149
 listing a file, 211-212
 width setting, 212
 Private, 36
 card file, 56
 file, 209, 213, 225, 260
 file, secured, 225
 Program
 binary, 255
 calling, 233
 debugging, 52
 editing, 53, 91
 entering, 91
 error handling, 197-198
 interruption, 256
 line, multiple-statement, 137
 scope, 117, 131
 suspend, 42, 52, 92, 93, 208, 279, 339
 suspend timer operation, 202
 transfer execution, 199, 200, 201-203
 Programming, default conditions, 330
 Prompt, display message, 152
 Prompt, input (?), 152
 Protected field, display, 305
 Protection, file, 36, 260
 Pseudo-random number, 254
 Purging a file, protection, 260

Q-R

Question mark (?) symbol, input, 152
 Question mark (?) symbol, unordered comparisons, 343
 Quiet NaN, 341, 342
 Quoted strings, 6
 Quotes, 6
 Quotes, matching, 6
 RAD annunciator, 74, 205, 232
 Radians mode, 232
 Radians setting, 11, 59
 Radix, 141, 144, 146, 147
 European, 139
 IEEE proposal, 338

RAM
 independent, 122
 main, 122, 184
 plug-in, memory requirements, 332
 removing, 122
 Random
 access, **224, 237**
 number, 254
 number seed, **233**
 Range, exponent, **182**
 Range of numbers, diagram, 318
 Read list, 237
 Readable character, 86
 Real number, maximum, **182**
 REAL numeric variables, 80, 190
 Record, 61, 224
 Record number, 224, 237, 250
 Recoverable error, during transformation, 291
 Recursion, 68
 Recursive calculation, statistics, **336**
 Redimensioning, **79, 288**
 Redimensioning an array, 156
 Reduction, **240-241**
 Reference, parameter passing, 282
 Regression, linear, **179, 180**
 Reinitialize, 80, 156, 239, 270
 Reinitialize, numeric or array variable, 156
 Related keywords, 6
 Relational operators, 5
 Relative errors, logarithms, **177**
 Releasing variables and memory, **93**
 REM, 137
 Remainder, **252**
 Remainder function, IEEE, 241
 Remark, 63, **242**
 Remark with IMAGE, **136**
 Renumbering lines, 187, **245-246**
 Replication, 138, 144
 Representable numbers, **317-318**
 Reset, 175, 305
 level three, 103
 memory, 100, 101, 329
 Restoring data pointer, **200**
 Return
 from function, **94**
 from subprogram, **94**
 pending, 215
 Reverse transformation, warning, **292**
 ROM
 plug-in, 145
 plug-in memory requirements, 332
 system, 301

Rounding, 341
 error, 339
 INTEGER variable, **169**
 SHORT variable, **169**
 Roundoff setting, **204, 205**
 RUN, 256
 RUN during input, **153**
 Run-time error, 102

S

Sample, 334
 correlation, 58
 mean, 183
 standard deviation, **259**
 SCI mode, **257-258**
 Scientific
 notation, 257
 display format, 281
 format, **257-258**
 Scope, program, 117, 131
 Scrolling, 75, 127, 128, 305
 SDATA file, 60, 61, 224, 237, 250
 SDATA file, memory requirements, 332
 Search order, subprograms, 33
 Secure, 36
 Secured private file, 225
 Seed, random number, **233, 254**
 Sequential access, **224, 237**
 Service, 301
 Setting, display, **112**
 Setting a flag, **114**
 Shifted keys, \bar{i} , \bar{q} , 160
 SHORT, 169
 numeric variables, 247, **270**
 precision, **269-270**
 Sign, floating, 141
 Sign of zero, 342
 Signalling NaN, 192, 341, 342
 Significance errors, 335
 Simple variable, 315, 316
 Single-step execution, 131
 Slope, 179, 180
 Smallest integer, 40
 Source file, 55-57, 187, 290
 Space, ignored in IMAGE, **143**
 Specifier, port, 122
 SST, during input, **153**
 Standard deviation, **259**
 Standard display format **277-278, 280**
 Start line number, 26

STARTUP string, default, **330**

Statement

- concatenation, **306**
- joining, **306**
- suspend, **131**

Statistical array, 259, 275, 276, 287, **334-337**
 default OPTION BASE setting, 275, 276
 specifying, **335**

Statistics, 88

- ADD, 336
- CLSTAT, 336
- DROP, 336
- linear regression, **337**
- LR, 337
- math exceptions, 339
- STAT, 335

STEP, **118, 119**

String

- array, 333
- assignments, **168-170**
- catalog, 38
- concatenation, **307**
- execute only, 172
- expression, length, **167**
- multistatement, 274
- null, 101
- overflow, 169
- time, **286**
- variable, creating, **169**
- variable length, 81

String-to-numeric conversion, **300**

Subprogram, 31, 32, 33, 34, 208, 233, 279, **282-283**,
 288, 315

- calls, memory requirements, 332
- DATA statements, 63
- data reading, 249
- duplicate names, 33
- ending, **94**
- environment, 314
- error, **100, 198**
- parameter list, 32
- passing parameters, 31, 32, **116, 282-283**
- recursion, 32
- search order, 33
- suspended, 34, **95**
- terminating, **95**
- timer operation, **203**

Subroutine, **129, 130, 200, 208, 215**

- call, 197
- call, memory requirements, 332
- on error, 197
- return, **251**
- suspended execution, **130**
- timer, **202**

Subscript, 155, 238

Subscript, array, 269, 316

Subscript, array, lower bounds, 204, 205

Substring, 216, 333

Substring, references, memory requirements, 331

SUSP annunciator, 53, 92, 110, 131, 256

Suspend

- program, 93, **208**
- statement, 131
- subprogram, 34, **95**
- user-defined function, **94**

Suspended

- execution, single-step, 131
- program, 42, 52, 256, 279
- program, timer operation, **202**

Symbolic channel number, 23

Symbols, special, **346**

Syntax

- diagrams, 6
- diagrams, how to read, 8
- error, 102
- error, IMAGE, 143

System

- catalog, 33, **262-266**
- configuration, change, 122
- flag, 114, 319
- messages, 378, **382-383**

T

TAB, 82, **83, 220, 230, 304**

Terminate program, subprogram, or user-defined
 function, **93**

Terminating key, 162

Test, loop, **119**

TEXT file, 60, 61, 224, 237, 250, 289-292

TEXT file, memory requirements, 331

TEXT-to-BASIC transformation, **290**

Threshold, overflow, 182

Threshold, underflow, 99, 341

Time
 base, 17
 error correction, **265-266**
 of day, **285, 286**
 setting, **264-266**
 string, **286**
 Timeout, automatic, 153
 Timer, 195, 196, **201-203**
 Timer, suspended during input, 153
 Tone, 28
 Track, magnetic card, 36, 226, 296
 Trailing blank, 63
 Transfer, program execution, **199, 200, 201-203**
 Transferring execution, **129-132**
 Transforming files, **289-292**
 Trap, 150, 378, 339
 IEEE, 149
 math, 248
 math exceptions, 72, 339, 340
 OVF, 142
 program errors, 197
 settings, 169
 underflow, 189
 values, 293, 294
 Trigonometric functions, spellings, 5
 Truncated string, 142
 Turn off, **195, 196**
 Typing aid, 70, 111

U

Unary operator, 308, 309
 UNASSIGNED, 111
 Unconditional branching, 135
 Underflow, 108, **295, 339, 393**
 exception, 189
 gradual, 341
 threshold, 99, 317, 318, 341
 UNF, **339, 393**
 Unlock, 175
 Unordered comparisons (?), **343-344**
 Unrecoverable error, during transformation, **292**
 Unsecured file, 225, 228, 297
 Uppercase, 8, **166**
 Uppercase conversion, **298**
 User
 annunciator, 299
 flag, 114
 keyboard, 70, 299
 mode, system flag, 299

User-defined function, 32, **116-117**, 279, 315
 calls, memory use, 94, 332
 ending, **94**
 precedence, 117
 same name, 117
 suspend execution, **94, 117**
 value, **94**
 variable, 169
 User-defined key assignments, 299

V

Value, parameter passing, 282
 Variable
 array, 315, 316
 assigning, **151-153**
 assignments, **234-235**
 deleting, **78**
 dependent, 179, 180, 218
 independent, 179, 180, 218
 integer, **155-156**, 247
 local, 208
 memory requirements, 332
 numeric, 316
 numeric, REAL, 80
 numeric, SHORT, **270**
 passing, 31, 32, 33
 real, 79-80, 238-239
 real string, 80
 reference, 235
 reference, memory requirements, 331
 redimensioned, 80
 releasing, **93**
 SHORT, 247, 269-270
 simple, 315, 316
 simple, numeric, 270
 string, **81**, 316
 string length, 81
 subprogram, local, 283
 subroutine access, **130**
 Verifying a card, **56**
 Version string, **301**
VIEW, 111
VIEW during input, **153**
 Viewing angle, 54
 Volume flag, 28

W

Waiting period, **302**
Warning, 100
 defined, **378**
 during transformation, 290, **291-292**
 message string, **101**
 number, **102**
Warnings, **378-381**
Window, display, **305**
Window, setting, 126
Workfile, 122
work file, 91, 48, **191**, 227, 228, 243, 290, 290
Wrap-around, **26**
Write protection, **226, 296**

Z

Zero, positive and negative, 49, 50
Zero, sign of, 342
Zeroes, leading, 133
Zone, display, 82

How To Use This Manual (page 5)
HP-71 Keyword Dictionary (page 10)
System Characteristics (page 314)
Mathematical Discussion of HP-71 Statistical Arrays (page 334)
IEEE Proposal for Handling Math Exceptions (page 338)
Glossary (page 346)
Errors, Warnings, and System Messages (page 378)
Keyword Index and Summary (page 394)
Subject Index (page 406)



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

European Headquarters
150, Route Du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva-Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL