# A Guide to the HP38G

# Math Menu

# and

# Programming

**Acknowledgments:**

Thanks to Charles Patton of the design team of the HP38G for his help, both by sharing an advance copy of the MATH menu specifications, and by providing answers to many of my questions regarding syntax and usage.  Also, many thanks to Mark Howell, Sam Gough, and Jill Lovorn for sharing their programs and expertise.

June 1995

# A Guide to the HP38G

# Math Menu

# and

# Programming

**Acknowledgments:**

Thanks to Charles Patton of the design team of the HP38G for his help, both by sharing an advance copy of the MATH menu specifications, and by providing answers to many of my questions regarding syntax and usage.  Also, many thanks to Mark Howell, Sam Gough, and Jill Lovorn for sharing their programs and expertise.

June 1995

# Contents

**1. MATH FUNCTIONS**

A catalog of all the built-in functions that don't appear on the keyboard of the HP38G, along with specific syntax and explanations of the arguments and the result returned by each function.

**2. PROGRAM COMMANDS**

A catalog of all the built-in commands that can be used on the HP38G (either on the HOME screen or from within a program) along with specific syntax and explanations of the arguments and action of each command.

**3. PROGRAM CONSTANTS**

A catalog of all the built-in constants which the HP38G uses to set variables.

**4. PROGRAM HINTS AND EXAMPLES**

A discussion of programming basics, along with a collection of examples of programs written for the HP38G.

# Introduction

This is reference material in addition to that found in the Quick Guide for the HP38G.

It includes a listing of all the different functions, commands, and constants available on the HP38G, along with some hints on programming.

The MATH key provides the access to the catalogs for all the functions and commands available that don't appear directly on the keyboard. When you press MATH, you'll see the following menu labels along the bottom of the screen:

**MTH** **CMDS** **CONS** **CANCL** **OK**

The first three of these labels each activate a different "double" menu.

The currently active menu is indicated by a little white square lit up on one of these three labels and also by a title bar you see at the top of the menu.

| | | |
|---|---|---|
| **MTH** | activates the menu titled | MATH FUNCTIONS |
| **CMDS** | activates the menu titled | PROGRAM COMMANDS |
| **CONS** | activates the menu titled | PROGRAM CONSTANTS |

# Notes on Terminology and Conventions

## What's the difference between a Function and a Command?

A Function always takes zero or more arguments and then returns exactly one result. For example, RAND takes no arguments and returns a random number between 0 and 1. COMB takes two arguments m and n and returns the number of combinations of n items that can be chosen from m objects: COMB(m,n) = m!/(n!(m-n)!). Functions can be used in any expression.

A Command also takes zero or more arguments, but it does not return a result. Instead, a command may affect the value of a variable (STO> is a good example) or it may perform some specific action (BEEP is an example). Commands are used in programs, but you can also execute them from the HOME screen.

For each function and command, the syntax is given showing the required punctuation (parentheses, comma's, semicolon's, etc.) and the order of any needed arguments. Some specific types of arguments are indicated as follows:

| | |
|---|---|
| <name> | a variable name |
| <expr> | any expression |
| <N#> | a positive integer or expression that has a positive integer value |
| <R#> | a real number or expression that has a real numeric value |
| <C#> | a complex number or expression that has a complex value |
| <#> | a real or complex number or expression |
| <{}> | a list or an expression that has a list value |
| <M#> | a matrix or an expression that has a matrix value |
| <V#> | a vector or an expression that has a vector value |

Other more descriptive abbreviations are sometimes used.

In this reference material you will find the following information provided for each math function or programming command:

| **Name** | **Syntax of usage (showing type and order of arguments)** | | |
|---|---|---|---|
| FCN or CMD | FCN(<arg1>,<arg2>...) or CMD <arg1>;<arg2>;... | | |
| | *Description of Arguments* | ----------> | *Description of Result or Action* |

Additional notes of explanation are also provided as needed.

## HELPWITH

If you type HELPWITH FCN or HELPWITH CMD on the HOME screen (where FCN or CMD is a specific function name or programming command) a box comes up which shows the syntax needed.

# 1. Math Functions

## Overview of the MATH FUNCTIONS Menu

| Calculus | Complex | Constant | Hyperb. | List |
|---|---|---|---|---|
| ∂ | ARG | e | ACOSH | CONCAT |
| ∫ | CONJ | i | ASINH | ΔLIST |
| TAYLOR | IM | MAXREAL | ATANH | MAKELIST |
|  | RE | MINREAL | COSH | ΠLIST |
|  |  | π | SINH | POS |
|  |  |  | TANH | REVERSE |
|  |  |  | ALOG | SIZE |
|  |  |  | EXP | ΣLIST |
|  |  |  | EXPM1 | SORT |
|  |  |  | LNP1 |  |

| Loop | Matrix | Polynom. | Prob. | Real |
|---|---|---|---|---|
| ITERATE | COLNORM | POLYCOEF | COMB | CEILING |
| RECURSE | COND | POLYEVAL | ! | DEG→RAD |
| Σ | CROSS | POLYFORM | PERM | FLOOR |
|  | DET | POLYROOT | RANDOM | FNROOT |
|  | DOT |  | UTPC | FRAC |
|  | EIGENVAL |  | UTPF | HMS→ |
|  | EIGENVV |  | UTPN | →HMS |
|  | IDENMAT |  | UTPT | INT |
|  | INVERSE |  |  | MANT |
|  | LQ |  |  | MAX |
|  | LSQ |  |  | MIN |
|  | LU |  |  | MOD |
|  | MAKEMAT |  |  | % |
|  | QR |  |  | %CHANGE |
|  | RANK |  |  | %TOTAL |
|  | ROWNORM |  |  | RAD→DEG |
|  | RREF |  |  | ROUND |
|  | SCHUR |  |  | SIGN |
|  | SIZE |  |  | TRUNCATE |
|  | SPECNORM |  |  | XPON |
|  | SPECRAD |  |  |  |
|  | SVD |  |  |  |
|  | SVL |  |  |  |
|  | TRACE |  |  |  |
|  | TRN |  |  |  |

| Stat-Two | Symbolic | Test | Trig |
|---|---|---|---|
| PREDX | = | < | ACOT |
| PREDY | ISOLATE | ≤ | ACSC |
|  | LINEAR? | = = | ASEC |
|  | QUAD | ≠ | COT |
|  | QUOTE | > | CSC |
|  | \| | ≥ | SEC |
|  |  | AND |  |
|  |  | IFTE |  |
|  |  | NOT |  |
|  |  | OR |  |
|  |  | XOR |  |

## Syntax Guide to MATH FUNCTIONS

### Calculus

∂

∂<name>(<expr>)
*x, expression* ----------> *derivative of the expression with respect to variable x*

∫

∫(<R#>,<R#>,<expr>,<name>)
*a,b,expression,x* ----------> *integral from a to b of expression with respect to x*

TAYLOR

TAYLOR(<expr>,<name>,<N#>)
*expression,x,n* ----------> *Taylor polynomial about x=0 of order n*

### Complex

ARG

ARG(<C#>)
*complex number* ----------> *argument of complex number*
*[i.e., if the complex number z has polar form (r,t), then ARG(z) = t]*

CONJ

CONJ(<C#>)
*complex number* ----------> *conjugate of complex number*

IM

IM(<C#>)
*complex number* ----------> *imaginary part*

RE

RE(<C#>)
*complex number* ----------> *real part*

### Constant

(These take no arguments; they are special numbers with special names.)

e

the base of the natural log (rounded to 2.71828182846)

i

(0,1) the standard square root of -1

MAXREAL

the constant 9.9999999E499
*(largest postive real number that can be represented on the HP38G)*

MINREAL

the constant 1.0000000E-499
*(smallest positive real number that can be represented on the HP38G)*

π

the constant π (rounded to 3.14159265359)

## Hyperb

| | | | |
|---|---|---|---|
| ACOSH | ACOSH(<#>) | | |
| | *x* | ----------> | *inverse hyperbolic cosine of x* |
| ASINH | ASINH(<#>) | | |
| | *x* | ----------> | *inverse hyperbolic sine of x* |
| ATANH | ATANH(<#>) | | |
| | *x* | ----------> | *inverse hyperbolic tangent of x* |
| COSH | COSH(<#>) | | |
| | *x* | ----------> | *hyperbolic cosine of x* |
| SINH | SINH(<#>) | | |
| | *x* | ----------> | *hyperbolic sine of x* |
| TANH | TANH(<#>) | | |
| | *x* | ----------> | *hyperbolic tangent of x* |
| ALOG | ALOG(<#>) | | |
| | *x* | ----------> | *10^x* |
| | NOTE: More accurate than the usual operation | | |
| EXP | EXP(<#>) | | |
| | *x* | ----------> | *e^x* |
| | NOTE: More accurate than the usual operation e^x. | | |
| EXPM1 | EXPM1(<#>) "exponential of # minus one" | | |
| | *x* | ----------> | *exp(x)-1* |
| | NOTE: More accurate near x=0 than with usual operations. | | |
| LNP1 | LNP1(<#>) logarithm of (# plus one)" | | |
| | *x* | ----------> | *ln(x+1)* |
| | NOTE: More accurate near x=0 than with usual operations. | | |

## List

CONCAT

CONCAT(<{}>,<{}>) Concatenate two lists.
*{X1,...,xn},* ----------> *{x1,...,xn,y1,...,ym}*
*{y1,...,ym}*

NOTE: Joins two lists end to end to make a new list.

ΔLIST

ΔLIST(<{}>) List of "first differences"
*{x1,x2,...,xn}* ----------> *{x2-x1,x3-x2,...xn-x(n-1)}*

ΠLIST

ΠLIST(<{}>) Product of elements
*{x1,...,xn}* ----------> x1*x2*...*xn

ΣLIST

ΣLIST(<{}>) Sum of elements
*{x1,...,xn}* ----------> *x1+...+xn*

MAKELIST

MAKELIST(<expr>,<name>,<#>,<#>,<#>)
*expression, variable name, starting value, end value, step size*
---------->
*{expr|(name=start), expr|(name=start+step), ..., expr|(name=end) }*

NOTE: Creates a list by evaluating the expression for each value of the variable name obtained by stepping from the starting value to the end value by the step size.

EXAMPLE: MAKELIST(X^2,X,1,5,2) creates {1,9,25}

POS

POS(<{}>,<N#>) Position of value in list
*{x1,...,xn} , y* ----------> *index value i such that xi = y, or 0 if no such i exists*

REVERSE

REVERSE(<{}>)
*list* ----------> *list in reverse order*

SIZE

SIZE(<{}>)
*list* ----------> *size of the list (# of elements)*

SORT

SORT(<{}>)
*list* ----------> *list sorted in increasing order*

## Loop

ITERATE

ITERATE(<expr>,<name>,<R#>,<N#>)
*expression,name,x0,n ----------> iterates the function expression in variable name n times starting with input x0*

EXAMPLE: ITERATE(X^2,X,5,3) results in 390625 because we have 5^2 = 25, 25^2 = 625, and 625^2 = 390625.

RECURSE

RECURSE(<name>,<expr>,<N#>,<N#>)
*index variable, Nth term, first term, second term*
*---------->*
*allows you to store a sequence definition from within a program or from the HOME screen.*

EXAMPLE: RECURSE(N, N^2, 1, 4) STO> U1(N)

Σ

Σ(<name>=<N#>,<N#>,<expr>)
*index n, starting index n1, ending index n2, expression for nth term*
*---------->*
*summation of expression in index variable n from n = n1 to n = n2*

## Matrix

COLNORM

COLNORM(<M#>)
*matrix [[M]] ----------> column-norm of M*

COND

COND(<M#>)
*matrix [[M]] ----------> condition number of M*

CROSS

CROSS(<V#>,<V#>)
*vectors [v1], [v2] ----------> cross-product of v1 and v2*

DET

DET(<M#>)
*matrix [[M]] ----------> determinant of M*

DOT

DOT(<V#>,<V#>)
*vectors [v1], [v2] ----------> dot-product of v1 and v2*

EIGENVAL

EIGENVAL(<M#>)
*matrix [[M]] ----------> [eigenvalues] (vector of eigenvalues of M)*

EIGENVV EIGENVV(<M#>)
*matrix [[M]]* ----------> *{[[eigenvectors]],[eigenvalues]}*
NOTE: list of matrix of eigenvectors and a vector of eigenvalues

IDENMAT IDENMAT(<N#>)
*integer n* ----------> *Creates an nxn identity matrix*

INVERSE INVERSE(<M#>)
*matrix [[M]]* ----------> *multiplicative inverse M^-1*
NOTE: This performs exactly the same code as the x^-1 key.

LQ LQ(<M#>)
*matrix [[M]]* ----------> *{ [[L]] [[Q]] [[P]] }*

LSQ LSQ(<M#>,<M#>)
*matrices [[B]], [[A]]* ----------> *matrix [[X]]*
Note: X is the least squares solution of A*X=B

LU LU(<M#>)
*matrix [[M]]* ----------> *{ [[L]] [[U]] [[P]] }*
NOTE: result is list of three matrices L, U, and P, where P*L*U=M and L is lower-triangular, U is upper-triangular with 1's on the main diagonal, and P is a permutation matrix

MAKEMAT MAKEMAT(<expr>,<n1#>,<n2#>)
*expression in I and J, n1 (number of rows), n2 (number of columns)*
*---------->*
*matrix M(I,J) where the (I,J) entry is the value of the expression*

QR QR(<M#>)
*matrix [[M]]* ----------> *{ [[Q]] [[R]] [[P]] }*
NOTE: result is list of three matrices Q, R, and P, where Q is orthogonal, R is triangular, and P is a permutation

RANK RANK(<M#>)
*matrix [[M]]* ----------> *computed rank of M*

ROWNORM ROWNORM(<M#>)
*matrix [[M]]* ----------> *row-norm of M*

RREF RREF(<M#>)
*matrix [[M]]* ----------> *reduced row-echelon form of M*

SCHUR

SCHUR(<M#>)
*matrix [[M]]* ----------> *{ [[Q]] [[U]] }*
NOTE: result is list of two matrices where Q*U*Q^H = M

SIZE

SIZE(<M#>)
*matrix [[M]]* ----------> *{n1, n2} (size of the matrix M)*

SPECNORM

SPECNORM(<M#>)
*matrix [[M]]* ----------> *spectral norm of M*

SPECRAD

SPECRAD(<M#>)
*matrix [[M]]* ----------> *spectral radius of M*

SVD

SVD(<M#>)
*matrix [[M]]---------->{ [[U]] [[V]] [S] }*
NOTE: result is a list of two matrices and a vector where U^H * M * V^H = diag(S) and U * diag(S) * V = M

SVL

SVL(<M#>)
*matrix [[M]]* ----------> *vector [S]*
NOTE: Result is the vector [S] described in SVD as above

TRACE

TRACE(<M#>)
*matrix [[M]]* ----------> *trace of M*

TRN

TRN(<M#>)
*matrix [[M]]* ----------> *matrix transpose of M*

## Polynom

POLYCOEF POLYCOEF(<V#>)

_[rN,...,r1]_ ----------> _[aN,...,a1,a0]_

_vector of roots_ _vector of coefficients_

NOTE: The r's are the roots of the polynomial aNx^N + a(N-1)*x^(N-1) + ... + a1x + a0

POLYEVAL POLYEVAL(<V#>,<#>)

_[aN,...,a1,a0], r_ ----------> _aNr^N+...+a1r+a0_

_vector of coefficients, real number_ _value of polynomial_

POLYFORM POLYFORM(<expr>,<name>,<name>...,)

_expression, x, y... ,_ ----------> _expression_

NOTE: Result is a polynomial in x whose coefficients are polynomials in y whose coefficients are...

EXAMPLE: POLYFORM(X^2*Y^2+X^2*Y,X)

results in the polynomial (Y^2+Y)*X^2

POLYROOT POLYROOT(<V#>)

_[aN,...,a0]_ ----------> _[rN,...,r1]_

_vector of coefficients_ _vector of roots_

NOTE: The r's are the roots of the polynomial aNx^N + a(N-1)*x^(N-1) + ... + a1x + a0

## Prob

COMB COMB(<N#>,<N#>) combinations

_integers n, k_ ----------> _n choose k_

$$\frac{(n*(n-1)*..(n-k+1))}{(1*2* . . . (k-1)*k}$$

! (<N#>)! factorial

_integer n_ ----------> _n!_

$(1*2*. . .*n)$

PERM PERM(<N#>,<N#>) permutations

_integers n, k_ ---------> $n*(n-1)*..(n-k+1)$

RAND RAND

_no arguments_ ----------> _random # between 0 and 1_

## **Prob** *Continued*

| | | | |
|---|---|---|---|
| UTPC | UTPC(<N#>,<R#>) | | |
| | *integer N, real X* | ----------> | *upper-tail chi-square distrib. Of order N evaluated at X* |
| UTPF | UTPF(<N#>,<N#>,<R#>) | | |
| | *integers N1, N2,* | ----------> | *upper-tail F-distrib. of real X order N1, N2, evaluated at X* |
| UTPN | UTPN(<R#>,<R#>,<R#>) | | |
| | *reals m, v, X* | ----------> | *upper-tail normal distrib. Of mean m, variance v evaluated at X* |
| UTPT | UTPT(<N#>,<R#>) | | |
| | *integer N, real X* | ----------> | *upper-tail Student's T-distrib. N of order N evaluated at X* |

## **Real**

| | | | |
|---|---|---|---|
| CEILING | CEILING(<R#>) | | |
| | *real number x* | ----------> | *least integer* $^{3}$ *x* |
| DEG→RAD | DEG→RAD(<R#>) | | |
| | *real number x* | ----------> | *convert x degrees to corresponding radians* |
| FLOOR | FLOOR(<R#>) | | |
| | *real number x* | ----------> | *greatest integer* $^{2}$ *to x* |
| FNROOT | FNROOT(<expr>,<name>,<R#>) | | |
| | *expression, x,x0* | -----------> | *r such than expr*$\vert$*(x=r) = 0. Search is started near x0.* |
| FRAC | FRAC(<R#>) | | |
| | *real number x* | -----------> | *fractional part of x* |
| | EXAMPLE: FRAC(23.438) returns .438 as result. | | |
| HMS→ | HMS→ (< expr>) | | |
| | *real number x* | -----------> | *convert hours/minutes/seconds format to decimal hours* |
| →HMS | →HMS(<expr>) | | |
| | *real number x* | -----------> | *converts decimal hours to hours/minutes/seconds format* |

| **Real** | *Continued* | | |
|---|---|---|---|
| INT | INT(<R#>) | | |
| | *real number x* | -----------> | *integer part of x* |
| | EXAMPLE: INT(23.438) returns 23 as result. | | |
| MANT | MANT(<R#>) | | |
| | *real number x* | -----------> | *mantissa of x* |
| MAX | MAX(<R#>,<R#>) | | |
| | *real numbers x,y* | -----------> | *maximum of x,y* |
| MIN | MIN(<R#>,<R#>) | | |
| | *real number x,y* | -----------> | *minimum of x,y* |
| MOD | <N#> MOD <N#> | | |
| | *integers m, n* | -----------> | *integer remainder when m is divided by n* |
| | EXAMPLE: 27 MOD 4 returns a result of 3 | | |
| % | %(<R#>,<R#>) | | |
| | *real numbers x,p* | ----------> | *x*p*0.01* |
| %CHANGE | %CHANGE(<R#>,<R#>) | | |
| | *real numbers x,y* | ----------> | *100*(y-x)/x* |
| %TOTAL | %TOTAL(<R#>,<R#>) | | |
| | *real numbers x,y* | ----------> | *100*y/x* |
| RAD->DEG | RAD->DEG(<R#>) | | |
| | *real number x* | ----------> | *convert x radians to degrees* |
| ROUND | ROUND(<R#>,<N#>) | | |
| | *real x,integer n* | ----------> | *round x to n digits* |
| SIGN | SIGN(<#>) | | |
| | *real or complex x* | ----------> | *x/ABS(x) if x _0 0 if x=0* |
| TRUNCATE | TRUNCATE(<R#>,<N#>) | | |
| | *real x,integer n* | ----------> | *truncate x to n digits* |
| XPON | XPON(<R#>) | | |
| | *real number x* | ----------> | *exponent of x when written in scientific notation* |

## Stat-Two

PREDX

PREDX(<name>,<#>)
*name,y*
*---------->*
*predicted value for the "independent" variable of the indicated statistical dataset, given the "dependent" value*

PREDY

PREDY(<name>,<#>)
*name,x*
*---------->*
*predicted value for the "dependent" variable of the indicated statistical dataset, given the "independent" value*

## Symbolic

=

This is an equational operator used to define equations like A+C=C/2, but it is not a predicate (like == in Test ).

ISOLATE

ISOLATE(<expr>,<name>)
*expression or equation, name of a specific variable*
*---------->*
*symbolic expression for specific variable in terms of the others*

NOTE: An expression is interpreted as an equation with the expression on one side and 0 on the other side of the =.

LINEAR?

LINEAR?(<name>)
*expression name* *---------->* *flag indicating whether the expression is linear*

QUAD

QUAD(<expr>,<name>)
*quadratic expression or equation, name of a specific variable*
*---------->*
*symbolic expression for the two complex roots*

NOTE: An expression is interpreted as an equation with the expression on one side and 0 on the other side of the =.

QUOTE

QUOTE(<expr>)
*expression* *---------->* *suppresses expression eval.*

NOTE: Single quote marks also work as in '<expr>'

(WHERE) |

|<expr> | (<name>=<#>,<name>=<#>,...) "substitution"
*expression ,x, p1, y, p2,...*
*---------->*
*expression evaluated where x=p1, y=p2, etc.*

## Test

| | | | |
|---|---|---|---|
| < | <R#> < <R#><br>*a, b* | ----------> | *1 if a<b, 0 otherwise* |
| ≤ | <R#> ≤ <R#><br>*a, b* | ----------> | *1 if a≤b, 0 otherwise* |
| = = | <R#> = = <R#><br>*a, b* | ----------> | *1 if a=b, 0 otherwise* |
| ≠ | <R#> ≠ <R#><br>*a, b* | ----------> | *1 if a≠b, 0 otherwise* |
| > | <R#> > <R#><br>*a, b* | ----------> | *1 if a>b, 0 otherwise* |
| ≥ | <R#> ≥ <R#><br>*a, b* | ----------> | *1 if a≥b, 0 otherwise* |
| AND | <R#> AND <R#><br>*a, b* | ----------> | *1 if both a and b ≠0,<br>0 otherwise* |
| IFTE | IFTE(<predicate>,<true-clause>,<false-clause>)<br>*a, expression1,<br>expression2* | ----------> | *expression1 if a≠0,<br>expression2 otherwise* |
| NOT | NOT <R#><br>*a* | ----------> | *1 if a=0,<br>0 otherwise* |
| OR | <R#> OR <R#><br>*a, b* | ----------> | *1 if a or b are non-zero,<br>0 otherwise* |
| XOR | <R#> XOR <R#> "exclusive or"<br>*a, b* | ----------> | *1 if a≠0 or b≠0, but not both,<br>0 otherwise* |

## Trig

ACOT ACOT(<#>)
*x* ----------> *inverse cotangent of x*

ACSC ACSC(<#>)
*x* ----------> *inverse cosecant of x*

ASEC ASEC(<#>)
*x* ----------> *inverse secant of x*

COT COT(<#>)
*x* ----------> *cotangent of x*

CSC CSC(<#>)
*x* ----------> *cosecant of x*

SEC SEC(<#>)
*x* ----------> *secant of x*

## Functions which are not in the MATH menu and not on the keyboard:

NEG NEG(<# or V# or M# or {} or expr or grob>)
NOTE: same functionality as -x key.

# 2. PROGRAM COMMANDS

## Overview of the PROGRAM COMMANDS Menu

| Aplet | Branch | Drawing | Graphic | Loop |
|---|---|---|---|---|
| CHECK | IF | ARC | DISPLAY→ | FOR |
| SELECT | THEN | BOX | →DISPLAY | = |
| SETVIEWS | ELSE | ERASE | →GROB | TO |
| UNCHECK | END | FREEZE | GROBNOT | STEP |
|  | CASE | LINE | GROBOR | END |
|  | IFERR | PIXOFF | GROBXOR | DO |
|  | RUN | PIXON | MAKEGROB | UNTIL |
|  | STOP | TLINE | PLOT→ | END |
|  |  | →PLOT | WHILE |  |
|  |  | REPLACE | REPEAT |  |
|  |  | SUB | END |  |
|  |  | ZEROGROB | BREAK |  |

| Matrix | Print | Prompt | Stat-One | Stat-Two |
|---|---|---|---|---|
| ADDCOL | PRDISPLAY | BEEP | DO1VSTATS | DO2VSTATS |
| ADDROW | PRHISTORY | CHOOSE | RANDSEED | SETDEPEND |
| DELCOL | PRVAR | DISP | SETFREQ | SETINDEP |
| DELROW | DISPTIME | SETSAMPLE |  |  |
| EDITMAT |  | EDITMAT |  |  |
| RANDMAT |  | FREEZE |  |  |
| REDIM |  | GETKEY |  |  |
| REPLACE |  | INPUT |  |  |
| SUB |  | MSGBOX |  |  |
| SCALE |  | WAIT |  |  |
| SCALEADD |  |  |  |  |
| SWAPCOL |  |  |  |  |
| SWAPROW |  |  |  |  |

Not In Any Menu

DEMO
ON-1
ON-PLOT
LIBEVAL (not documented in manual)
PINIT (not documented in manual)
SYSEVAL (not documented in manual)
VERSION (not documented in manual)
WSLOG (not documented in manual)

## Syntax Guide to PROGRAM COMMANDS

**Aplet** These commands control Aplets.

CHECK CHECK <N#>

*integer n* ----------> *checks the corresponding function in current aplet*

EXAMPLE: CHECK 3 means check F3 if the current aplet is Function.

*SELECT <ApLetname>*

SELECT

*ApLetname* ----------> *makes this ApLet the currently active one.*

UNCHECK UNCHECK <N#>

*integer n* ----------> *unchecks the corresponding function in current ApLet*

EXAMPLE: UNCHECK 3 means uncheck F3 if current ApLet is Function.

SETVIEWS SETVIEWS<prompt_1>;<program_name_1>;<view#_1>;
*<prompt_2>;<program_name_2>;<view#_2>;*
*...*
*<prompt_n>;<program_name_n>;<view#_n>;*

**Usage of the SETVIEWS command:**

Each triple prompt/program/view defines one line of the menu as follows:

<prompt> is a string to display in the menu.

<program_name> is name of a program to run if this line is selected.

<View#> is number of view to start after the program finishes running.

The prompt can be specified in two ways besides an explicit prompt string <prompt>. First, if <prompt> is an empty string (e.g., nothing appears between two semicolons), the program name is used as the prompt. Second, if both <prompt> and <program> are empty strings, and <view> is one of the default `special views', the normal prompt for that view (e.g., `Auto Scale') is used.

The program can prompt for information and display information, and it can modify all sorts of variables, and it can select another aplet, but it can't directly start a view. That happens only indirectly when the program is finished. If no program is desired, specify an empty string. The views are numbered as follows:

1. PLOT 2. SYMB 3. NUM
4. PLOT setup 5. SYMB setup 6. NUM setup
7. VIEWS 8. NOTE 9. SKETCH

In addition, five catalogs are available as views numbered 10 - 14, and if the ApLet has N default `special views', they are numbered views 15 through N +14.)

10. LIB 11. LIST 12. MATRIX
13. NOTEPAD 14. PROGRAM 15-? SPECIAL

If the prompt specified for a view is exactly the string `Reset', or 'Start', the associated program is considered the ApLet's special reset or start mechanism. Pressing the RESET or START softkey in the LIB catalog will run the special reset or start program.

It can be useful to associate a program with an ApLet even though the program isn't used directly in the menu of custom views. For example, the program may be a subroutine called by other programs. To accomodate this case, if <prompt> is exactly one space (" "), this triple is not presented in the menu.

**Branch** These commands work in various combinations to make one or more tests and to execute one clause of several. (See PART 4 for examples of branching structures)

IF
THEN
ELSE
END

CASE

IFERR

RUN RUN <progname>
*program name* ----------> *runs the specified program*

STOP STOP
*no arguments* ----------> *stops the current program*

**Drawing** These commands act on the display.

ARC

ARC <cent-x#>;<cent-y#>;<radius#>;<start#>;<end#>

*x,y,r,s,e* ----------> *draws arc with center (x,y), radius r, start s, and end e on the display*

BOX

BOX <x1#>;<y1#>;<x2#>;<y2#>

*x1,y1,x2,y2* ----------> *draws a box with opposite corners (x1,y1) and (x2,y2) in the display*

ERASE

ERASE

*no arguments* ----------> *causes the display to be erased*

FREEZE

FREEZE

*no arguments* ----------> *causes display to not be updated*

NOTE: This can be used to "freeze" the display the way it was when the program stopped running.

LINE

LINE <x-start#>;<y-start#>;<x-end#>;<y-end#>

*x1,y1,x2,y2* ----------> *draws a line from (x1,y1) to (x2,y2) in the display*

PIXOFF

PIXOFF <x#>;<y#>

*x,y* ----------> *turns off pixel with indicated x,y coordinates in the display*

PIXON

PIXON <x#>;<y#>

*x,y* ----------> *turns on pixel with indicated x,y coordinates in the display*

TLINE

TLINE <x-start#>;<y-start#>;<x-end#>;<y-end#>

*x1,y1,x2,y2* ----------> *toggles a line from (x1,y1) to (x2,y2) in the display*

**Graphic** These program commands use grob-variables (G0,...,G9) as arguments.

DISPLAY→

DISPLAY→ <name>

*name* ----------> *stores display as a grob in name*

→DISPLAY

→DISPLAY <grob>

*grob* ----------> *puts grob in the display*

→GROB

→GROB <name>;<font#><object>

*name,f, obj* ----------> *creates a display grob using font number f and stores the result in name.*

NOTE: The text that is converted into a grob is specified by object, which can be either a grob-variable-name or a grob-expression. (A grob-expression is an expression which returns a grob.)

GROBNOT

GROBNOT <name>

*name* ----------> *replaces the grob in name with the "inverted" grob from name*

GROBOR

GROBOR <name1>;<pos>;<name2>

*name1,position, name2*

*---------->*

*replaces the grob in name1 with the bitwise OR with the grob in name2 starting at pos. (name2 can also be a grob-expression)*

GROBXOR

GROBXOR <name1>;<pos>;<name2>

*name1, position, name2*

*---------->*

*replaces the grob in name1 with the bitwise XOR with the grob in name2 starting at pos (name2 can also be a grob-expression)*

MAKEGROB

MAKEGROB <name>;<width> <height> <hex-data>

*name, width, height, hex-data*

*--------->*

*creates a GROB with the given width height and hex-data and stores it in name*

PLOT→

PLOT→ <name>

*name* ----------> *stores the Plot view display as a grob in name*

**Graphic** *Continued*

→PLOT

→PLOT <name>

*name* ----------> *puts grob from name into the Plot view display.*

NOTE: Plot view is erased first, and grob is placed in upper left corner of Plot view display.

ZEROGROB

ZEROGROB <name>;<width>; <height>

*name of grob variable, width, height*

---------->

*creates a blank GROB with given width and height and stores it in name*

*The following two commands also apply to matrices and lists.*

SUB

SUB <name>;<object>;<start> <end>

*name, obj, start, end* ----------> *stores the indicated subobject in name*

NOTE: SUB takes matrix, list, or grob arguments. The object can be a matrix-variable-name, a list-variable-name, a grob-variable-name, a matrix-expression, a list-expression, or a grob-expression.

REPLACE

REPLACE <name>;<start>;<object>

*name,start,object* ----------> *replaces the object in <name> with <object> starting at <start>*

NOTE: REPLACE takes matrix, list, or grob arguments. The object can be a matrix-variable-name, a list-variable-name, a grob-variable-name, a matrix-expression, a list-expression, or a grob-expression.

**Loop**

These commands are used in combinations to conditionally repeat a clause several times. (See **PART 4** for examples of looping structures.)

FOR

FOR ... = ... TO ... STEP ... END

DO

DO ... UNTIL ... END

WHILE

WHILE ... REPEAT ... END

BREAK

**Matrix** These commands store their results in specified matrix variables.

ADDCOL

ADDCOL <name>;<V#>;<N#>
*matrix name, vector of columndata, column position n*
---------->
*inserts the vector of columndata as column n of the matrix stored in name*

ADDROW

ADDROW <name>;<V#>;<N#>
*matrix name, vector of rowdata, column position n*
---------->
*inserts the vector of rowdata as column n of the matrix stored in name*

DELCOL

DELCOL <name>;<N#>
*matrix name, position n*
---------->
*deletes column n of the matrix stored in name*

DELROW

DELROW <name>;<N#>
*name, position n* ----------> *deletes row n of the matrix stored in name*

EDITMAT

EDITMAT <name>
*matrix name* ----------> *fires up the matrix editor on the specified matrix*

NOTE: Returns to program execution after you press OK

RANDMAT

RANDMAT <name>;<row#>;<col#>
*matrix name, number of rows n, number of columns m*
---------->
*creates a random matrix with n rows, m columns, and stores the result in <name>.*

REDIM

REDIM <name>;<size>
*redimensions the object in <name> to <size>*

REPLACE

REPLACE <name>;<start>;<object>

NOTE: See under **Graphic** for explanation.

SCALE

SCALE <name>;<factor>;<row#>
*matrix name, scale factor s, row number n*
---------->
*multiplies row n of the matrix stored in name by factor s*

**Matrix** *Continued*

SCALEADD

SCALEADD <name>;<factor>;<row#1>;<row#2>
*matrix name, scale factor s, row numbers n1,n2*
---------->
*adds the product of s and row n1 of the matrix stored in name to row n2*

SUB

SUB <name>;<object>;<start> <end>

NOTE: see under **Graphic** for explanation.

SWAPCOL

SWAPCOL <name>;<pos1>;<pos2>
*matrix name, column position n1, column position n2*
---------->
*swaps columns n1 and n2 of the matrix stored in name*

SWAPROW

SWAPROW <name>;<pos1>;<pos2>
*matrix name, row position n1, row position n2*
---------->
*swaps rows n1 and n2 of the matrix stored in name*

**Print** These commands print to an HP "Redeye" InfraRed printer.

PRDISPLAY

PRDISPLAY

NOTE: Prints the current display on the IR printer

PRHISTORY

PRHISTORY

NOTE: Prints the history "stack" of results from the HOME screen on the IR printer

PRVAR

PRVAR <name>

NOTE: Prints a variable's name and contents on the IR printer

PRVAR <name>;PROG

NOTE: Prints program's name and contents on the IR printer

PRVAR <name>;NOTE

NOTE: Prints a note's name and contents on the IR printer

**Prompt** These commands ask for information or provide information to the user.

---

BEEP

BEEP <freq#>;<time#>
*frequency f (cycles per second), duration time s (seconds)*
*---------->*
*emits a tone at frequency f for s seconds*

CHOOSE

CHOOSE <var-name>;<prompt>;<item-1>;...;<item-n>
*<var-name> is the name of the variable from which the number of the initially-highlighted item will be gotten and into which the number of the chosen item will be stored;*
*<prompt> is the top-of-box prompt or null, meaning no prompt;*
*<item-1> through <item-n> are the items to be displayed.*
*--------->*
*CHOOSE displays a choose box and sets the specified variable to real number 0 through n corresponding to whether the choose box is cancelled (0) or an item is chosen (1 through n).*

EXAMPLE: 3 STO> A CHOOSE A;
"AngleMode"; "Degrees("Degrees")";
"Radians("Radians")";
"Grads("Grads")"; "Current("HAngle")"

This displays a choose box with the prompt "Angle Mode" and three choices "Degrees(1)", "Radians(2)", "Grads(3)", "Current(??)" with "Grads(3)" (item 3) highlighted initially. All of the arguments but <var-name> are text items. (See Text Item Conventions below.)

DISP

DISP <N#>;<item>
*line n, text-item*
*---------->*
*display on line n of the display text that was created from the text item <item>*

NOTE: Uses same conventions as MSGBOX. (See Text Item Conventions below.)

DISPTIME

DISPTIME
*no arguments* ----------> *Displays current date and time.*

EDITMAT

EDITMAT <name>
*matrix name*
*---------->*
*fires up the matrix editor on the specified matrix; then returns to program execution with OK*

FREEZE

FREEZE
*no arguments* ----------> *Causes display to not be updated.*

NOTE: Freezes the display the way it was when the program stopped running.

GETKEY

GETKEY <name>

NOTE: Waits for a keystroke, then stores the corresponding keycode in the given name

INPUT

INPUT <name>;<title>;<label>;<help>;<default>
*name, title, label, help, default*
---------->
*1) prompts the user with title, label, and help,*
*2) initializes a command line with default, and*
*3) saves the resulting input in name.*

NOTE: <title>, <label>, and <help> are text-tems, and <default> is an expression and is evaluated. (See Text Item Conventions below.)

MSGBOX

MSGBOX <text-item>

NOTE: Displays a text-item in a message box.

WAIT

WAIT <time#>
*time in seconds* ----------> *waits s seconds*

**Text Item Conventions**

A text item is either a quoted string of text characters (for example, "ABC") or an expression which will be evaluated and turned into a string of characters (for example, SIN(0) is eqivalent to "0") or a sequence of these. The results of all of these will be concatenated together to form a single string of characters.

An explicit character string can be delimited by either double quotes ("ABC") or single quotes ('"new" improved!'). Sequences of explicit character strings will be concatenated ( "Alice's " '"new"' " ApLet is here" ). (Note that sequences of expressions may be interpreted using implicit multiplication.)

All this allows you to display labels and values. For example, with the text item "A:" SIN(0), the displayed string would be "A: 0". Null arguments are allowed.

**Stat-One** These commands are used for one-variable statistics.

DO1VSTATS

DO1VSTATS <datasetname>
_<datasetname> may be H1, H2, ..., or H5_
_---------->_
_calculates STATS using <datasetname> and stores results in the following variables: NΣ, TOTΣ, MEANΣ, PVARΣ, SVARΣ, PSDEV, SSDEV, MINΣ, Q1, MEDIAN, Q3, and MAXΣ._

RANDSEED

RANDSEED <R#>
_seed value s ----------> sets random number seed to s_

SETFREQ

SETFREQ <datasetname>;<definition>
_<datasetname> may be H1, H2, ..., or H5_
_--------->_
_defines <datasetname> frequency according to <definition> expression._

SETSAMPLE

SETSAMPLE <datasetname>;<definition>
_<datasetname> may be H1, H2, ..., or H5_
_---------->_
_Defines <datasetname> sample according to <definition> expression._

**Stat-Two** These commands are used for two-variable statistics.

DO2VSTATS

DO2VSTATS <datasetname>
_<datasetname> may be S1, S2,..., or S5_
_---------->_
_calculates STATS using <datasetname> and stores results in corresponding variables: MEANX, ΣX, ΣX^2, MEANY, ΣY, ΣY^2, ΣX*Y, CORR, COV, and FIT._

SETDEPEND

SETDEPEND <datasetname>;<definition>
_<datasetname> may be S1, S2, ..., or S5_
_---------->_
_Defines <datasetname> dependent according to <definition> expression._

SETINDEP

SETINDEP <datasetname>;<definition>
_<datasetname> may be S1, S2, ..., or S5_
_---------->_
_defines <datasetname> dependent according to <definition> expression._

## Program Commands Not Found In Any Menu

### *Documented In Manual*

DEMO

DEMO

NOTE: This displays the an animated demonstration of some of the HP38G's features.

ON-1

ON-1

NOTE: Pressing ON and 1 keys at the same time causes the current display information to be sent out through the cable connection.

ON-PLOT

ON-PLOT

NOTE: Pressing ON and PLOT keys at the same time causes the current display to be stored as a grob in G0.

### *Not Documented In Manual*

LIBEVAL

LIBEVAL <library_number>;<routine_number>

NOTE: This executes the specified rompointer.

PINIT

PINIT

NOTE: This is for port initialization.

RULES

SYSEVAL

SYSEVAL <address>

NOTE: This executes system object at specified address.

VERSION

VERSION

NOTE: This displays the version and HP copyright message in a message box.

WSLOG

WSLOG

NOTE: This displays the warmstart log.

# 3. PROGRAM CONSTANTS

Used for setting variables. All constants listed below (with the exception of StatMode constants) also appear in input forms.

The name in the **MATH** menu is always the same as the name that appears in the corresponding input form field. In some cases, the CHOOS-list that corresponds to the input form field will show longer names.

*The value of each constant is the same as its position in the* **MATH** *Menu (which is also the same as its position in the CHOOS-list).*

EXAMPLE: Degrees = 1, Radians = 2, and Grads = 3.

## Overview of the PROGRAM CONSTANTS Menu

| Angle | Format | SeqPlot |
|---|---|---|
| Degrees | Standard | Stairstep |
| Radians | Fixed | Cobweb |
| Grads | Sci | |
| | Eng | |
| | Fraction | |

| S1...5fit* | StatMode | StatPlot |
|---|---|---|
| Linear | Stat1Var | Hist |
| LogFit | Stat2Var | BoxW |
| ExpFit | | |
| Power | | |
| QuadFit | | |
| Cubic | | |
| Logist | | |
| User | | |

* constants in S1...5fit category are used to set S1fit,...,S5fit

## CHOOS-list Versions of the Constant Names

| Angle | Format | SeqPlot |
|---|---|---|
| Degrees | Standard | Stairstep |
| Radians | Fixed | Cobweb |
| Grads | Scientific | |
| | Engineering | |
| | Fraction | |

| S1...5fit | StatPlot |
|---|---|
| Linear | Histogram |
| Logarithmic | BoxWhisker |
| Exponential | |
| Power | |
| Quadratic | |
| Cubic | |
| Logistic | |
| User Defined | |

# 4. PROGRAM HINTS AND EXAMPLES

## What is a program?

At its very simplest, a program is just a sequence of commands, each of which performs an action. On the HP38G, a colon (:) is used to separate commands from one another, while a semicolon (;) is used to separate the arguments to a single command.

It is possible to write a program directly on the HOME screen in the Editline.

For example,

```
1 STO> A: 2 STO> B: 3 STO> C
```

followed by **ENTER** will return a result of 3 in Ans (the last command was 3 STO> C), but all three STO> commands were executed.

## The PROGRAM CATALOG

To save a program and give it a specific name, press **PROGRAM**, and you will see a title bar for the PROGRAM CATALOG. This contains a directory of all your programs (even including the Editline). The menu keys here are:

EDIT NEW SEND RECV RUN

To create a new program, press **NEW**. You are prompted for a name for the program. After you type in the name of your choice (it can be virtually anything you please and of any length), press **ENTER** or **OK**.

## The PROGRAM EDITOR

Now you are in the editor for your specific program, and the title bar should indicate the name of the program. For example, if you named your program MINE, then the title bar should read MINE PROGRAM at the top of the screen. The menu keys now are:

STO> SPACE A...Z BKSP

The **A...Z** key is an "alpha-lock" which is can be toggled on or off (when it is on, you will see a white square lit up on the menu label and the α symbol lit at the top of the screen). If you press the colored shift key before toggling on **A...Z**, it changes the key to **a...z**, a lower-case "alpha-lock".

### Typing in your program

The flashing cursor arrow is an insertion marker. While typing a program you can use the directional arrows for moving from line to line or from character to character.

**SPACE** is on the menu for typing convenience

**BKSP** (backspace) is on the menu for typing convenience

**STO>** is the one command provided on the menu (performs the same action as the **STO>** key in HOME).

**DEL** deletes the character under the insertion arrow (or the last character if you are at the very end)

**ENTER** is a line feed

Numbers, letters, and other characters can be typed from the keyboard as usual. You will find yourself making frequent use of the **CHARS** menu (for retrieving relational symbols like $<$, $>$, $^2$, $^3$ and the ' or " symbols). To use **CHARS**, use the directional arrows to highlight the character you want, and then press **OK**. If there are several characters in a row that you want to retrieve from the **CHARS** menu, press **ECHO** first and then each of the characters you want (press **OK** when you are done). Functions and variable names can either be typed in character-by-character or retrieved from the **VAR** or the **MATH** menus.

All other programming commands (other than **STO>**) are found in the **MATH** menu when **CMDS** is toggled on. An overview and the syntax for all these commands are found in PART 2 of this reference.

### When you are finished...

Once you are finished typing in a program, there is no special action necessary to "EXIT". Simply *go somewhere else!* For example, you could press **HOME** to return to the **HOME** screen, or you could press **PROGRAM** to create a new program or **RUN** the program you have just written.

If you highlight the name of a program in the PROGRAM CATALOG, you can **EDIT** it (this returns you to the program editor where you can make changes). **SEND** and **RECV** are used to send and receive programs through infra-red communications with another HP38G, or by serial wire connection to a disk drive.

### Cutting and pasting the contents of one program into another

Sometimes you may find that you have several lines of program code that you want to use in another program (perhaps with some slight editing). Rather than type the code in again "from scratch", there is a neat way to cut and paste one program into another:

1. Position the flashing cursor at the location in the current program where you want to paste in the contents of another program
2. Press **VAR**
3. highlight the Program category and press **OK**
4. highlight the name of the program whose contents you want to paste
5. toggle on the **VALUE** menu key (instead of **NAME**)
6. press **OK**

## Program structures

Programs that are simply sequences of commands can be very useful, but the real power of programs becomes evident when they allow for *conditional* execution of commands (branching) or *repeated* execution of several commands over and over again (looping). You will see that the HP38G's programming language does not use line numbers or labels and GO TO statements to accomplish branching and looping.

### HP38G Branching structures

You can either type these commands in character-by-character or retrieve them from the Branch category of the **CMDS** menu in **MATH**.

IF ... THEN ... ELSE ... END

The syntax for this structure is as follows:

| | |
|---|---|
| IF test clause | Test clause is some statement that is true (1) or false (0) |
| THEN | |
| do some thing: | |
| do some other thing: | These statements are executed if the test clause is true |
| and so on: | |
| ELSE | (The ELSE part is optional) |
| do some thing: | |
| do some other thing: | These statements are executed if the test clause is false |
| and so on: | |
| END: | Marks the end of the IF THEN ELSE structure |

NOTES:

1. In the statements that follow either THEN or ELSE, it's possible to have loops or additional branch structures, including other IF THEN ELSE END structures.

2. The test clause can actually be any expression, and could have a value other than 1 or 0. The THEN part is executed if the test clause has a nonzero value, while the ELSE part is executed only if the test clause has the *value zero.*

***Example:***

```
IF  A < 0
THEN                   If A is negative, then the sign of A is reversed.
-A STO> A:             (Note that this piece of code has no ELSE part.)
END:
```

***Example:*** *This program records the sign of A in S*

```
IF  A < 0
THEN
-1 STO> S:             S = -1 if A is negative
ELSE
IF A = 0
THEN
0 STO> S:              S = 0 if A = 0
ELSE
1 STO> S:              S = 1 if A is positive
END:
END:
```

## CASE

This structure is handy if you have several clauses to test in order, and you want to branch when you encounter the *first* true clause. The syntax for CASE is:

```
CASE
IF first test clause
THEN
do some thing:
do some other thing:
and so on:
END
IF second test clause
THEN
do some thing:
do some other thing:
and so on:
END
.
.
.
END:
```

Here are two examples to illustrate the distinction between the use of CASE and simply a sequence of several IF THEN statements.

***Example 1:***

```
CASE
IF I > 5
THEN I-5 STO> S:
END
IF I ≥ 5
THEN 0 STO> S:
END
IF I < 5
THEN 5 – I STO> S:
END
END:
```

***Example 2:***

```
IF I > 5
THEN I-5 STO> S:
END:
IF I ≥ 5
THEN 0 STO> S:
END:
IF I < 5
THEN 5 – I STO> S:
END:
```

Suppose we store 7 in I (7 STO> I on the HOME screen), and then run the first program involving CASE. The result will be that 2 is stored in S, because the clause in the second IF THEN statement won't even be tested.

In contrast, if the second program is run, the final result will be that 0 is stored in S, for *all* three of the clauses in the IF THEN statements will be tested.

*NOTE the punctuation differences between these two examples, particularly the use of colons.*

### IFERR ... THEN ... ELSE

This structure is one designed for "error trapping." Some program statements may result in an error (thus causing the program to stop) depending on the values of certain variables at the time of execution. IFERR will check any number of statements for such errors and give you the chance to gracefully branch.

The syntax for IFERR is:

```
IFERR
first command:
second command:
et cetera:
THEN                    If an error resulted from any of these commands
do some thing:            then these statements are executed
do some other thing:
and so on:
ELSE                    Optional—if no error resulted
do some thing else:       then these statements are executed instead
do some other thing
else:
and so on:
```

***Example:***

```
IFERR
ABS(COT(X)) STO> C:                 If this statement results in an error
THEN                                  (for example, when X=0)
1 STO> A:                             then store 1 in A
MAXREAL STO> C:                       and store the largest real number in C
ELSE                                If no error resulted,
0 STO> A:                             then store 0 in A instead
END:
```

For example, if X had the value 0 at execution, the attempt to store ABS(COT(X)) as the value of C would result in an error.

Rather than stopping, this program branches to store the largest real number that can be represented on the HP38G as the value of C, and 1 is stored in A (as a marker that an error occurred, perhaps).

If the X has a value that does not result in an error, then ABS(COT(X)) is stored in C and 0 is stored in A.

## RUN

You can also branch to other programs using the RUN command. The syntax is:

RUN name: where name is the name of the program.

### HP38G Looping Structures

You can either type these commands in character-by-character or retrieve them from the Loop category of the **CMDS** menu in **MATH**.

## FOR = TO ... STEP ... END

The syntax for this basic loop structure is:

```
FOR <index variable> = <start value> TO <end value> STEP <increment value>;
do some thing:
do some other thing:
and so on:
END:
```

***Example:***

```
0 STO> S:
1 STO> P:
FOR I = 1 TO 10 STEP 1;
S+I STO> S:
P*I STO> P:
END:
```

The result of this program is that the sum of the integers 1 through 10 is stored in S, and their product is stored in P.

## DO ... UNTIL ... END

This loop structure will repeatedly execute a body of statements until some test clause is true. The syntax is:

```
DO
some thing:
some other thing:
and so on:
UNTIL <test clause>
END:
```

***Example:***

```
0 STO> S:
1 STO> P:
1 STO> I:
DO
S+I STO> S:
P*I STO> P:
I+1 STO> I:
UNTIL I > 10
END:
```

The result of this program is the same as the previous one: the sum of the integers 1 through 10 is stored in S, and their product is stored in P.

## WHILE ... REPEAT ... END

This structure repeats the execution of some body of statements while a test clause remains true. The syntax is:

```
WHILE <test clause>
REPEAT
do some thing:
do some other thing:
and so on:
END:
```

***Example:***

```
0 STO> S:
1 STO> P:
1 STO> I:
WHILE I ≤ 10
REPEAT
S+I STO> S:
P*I STO> P:
I+1 STO> I:
END:
```

The result of this program is the same as the previous two: the sum of the integers 1 through 10 is stored in S, and their product is stored in P.

***Example:***

This example assumes that a positive integer has been stored in M. Then it sets N=M and calculates 3N+1 if N is odd or N/2 if N is even. This becomes the new value of N and the process is repeated until N=1. The value of S at the end of the program's execution is the number of different values of N. (S=0 if M is not a positive integer). NOTE: It is an open question whether S is finite for all positive integers M.

```
IF (M ≤ 0) OR (M ≠ INT(M))
THEN 0 STO> S:
ELSE
1 STO> S:
M STO> N:
WHILE N ≠ 1
REPEAT
IF N MOD 2
THEN
3*N+1 STO> N:
S+1 STO> S:
ELSE
N/2 STO> N:
S+1 STO> S:
END:
END:
```

## EXAMPLE PROGRAMS

Probably the best way to learn how to program the HP38G is to study some interesting working programs that actually use many of the basic programming constructions. We provide a few examples here along with line-by-line documentation.

Simply entering these programs will give you some practice in finding commands and familiarize you with the most important elements of program syntax. Some suggestions are made for ways that you could adapt some of the programs to perform other tasks.

### HP38G EXAMPLE 1.

This program looks for perfect numbers (positive integers *N* with the property that *N* = *S,* the sum of all positive integer factors < *N*). It tests all the positive integers from 2 to 1000, displaying each along with the sum of its divisors. When *N* = *S* , the program displays that value with the label "PERFECT". Hence, you will see the last perfect number found by the program as it runs.

The program illustrates the use of WHILE REPEAT loops as well as IF THEN statements. It also illustrates the use of the DISP command for displaying text and values on specified lines of the display screen.

***Program PERFECTN***

```
ERASE:
2 STO> N:
WHILE N ≤ 1000
REPEAT
2 STO> F:
1 STO> S:
WHILE F ≤ √N
REPEAT
IF N MOD F == 0 THEN
S+F+(N/F) STO> S:
END:
F+1 STO> F:
END:
IF N==S THEN
DISP 3; "PERFECT: "N:
END:
DISP 1;"N= ";N:
DISP 2;"S= ";S:
N+1 STO> N:
```

*NOTE: Here are some possible adaptations you could make to this program:*

*a) Save the list of the perfect numbers found in LO.*

*b) Allow the user to input the value of N (the largest integer checked).*

*c) Search for prime numbers (or other integers with special properties).*

## HP38G EXAMPLE 2.

This program illustrates the use of PIXON to light specific pixels on the display screen, as well as the use of nested WHILE REPEAT loops. It assumes that your viewing window is the default window [-6.5, 6.5] by [-3.1,3.2]. The program checks the coordinates of each pixel in the third quadrant and uses the integer part of the value of X^2+Y^2 MOD 2 (i.e., the remainder after division by 2) to decide whether to light up the pixels or not (1 for yes, 0 for no). It also lights up the other three symmetrically located pixels in the other three quadrants. It takes a while to run, but the result is a striking picture exhibiting some interesting diffraction patterns. (The picture is stored in the SKETCH view of the current ApLet.)

NOTE: Try replacing the expression X^2 + Y^2 with another expression symmetric in X and Y to produce a more exotic picture.

### *Program RIPPLES*

```
ERASE:                          Clear the display screen
(Xmax-Xmin)/130 STO> H:         Calculate pixel width H
(Ymax-Ymin)/63 STO> K:          Calculate pixel height  K
Xmin STO> X:                    Initialize X to left edge of screen
WHILE X ≤ 0                     OuterWHILE loop begins
REPEAT
Ymin STO> Y:                    Initialize Y to bottom of the screen
WHILE Y ≤ 0                     InnerWHILE loop begins
REPEAT
IF INT (X^2+Y^2) MOD 2          If the integer part of
THEN                              X^2+Y^2 MOD 2 is 1, then
PIXON X;Y:                      light up that pixel
PIXON X;Ymax-(Y-Ymin):            and the symmetrically located
PIXON Xmax-(X-Xmin);Y:            pixels in the other  3 quadrants
PIXON Xmax-(X-Xmin);
Ymax-(Y-Ymin):
END:
Y+ K STO> Y:                    Increment pixel  y-coordinate
END:                            Inner WHILE loop ends
X+H STO> X:                     Increment pixel x-coordinate
END:                            Outer WHILE loop ends
DISPLAY→  Page                  Save finished picture in  SKETCH
```

## HP38G EXAMPLE 3.

This program "animates" a numerical limit by displaying a simultaneous readout of X and F(X) values. (In this case, the particular function used is SIN(X)/X.) Pay particular attention to the INPUT statements, for they provide a very simple way to give a professional look to your programs, complete with a title bar, a highlighted input form (with a default value supplied if you wish), and a help message to the user to prompt for the necessary input.

NOTE: This program can be adapted to investigate the limit of an arbitrary function by adding an input statement that allows the user to select a function in the SYMB menu. This program can also be used to investigate limits "at infinity" by using MAXREAL or -MAXREAL as the TARGET value A, and setting a suitably large increment H.

***Program LIMITS***

```
INPUT X;                          Begin INPUT statement for X
"LIMIT ANIMATION";                Title bar
"INITIAL X";                      Label for input blank for X
"ENTER STARTING X";               Help message to prompt user
1:                                Default value for X
INPUT A;                          Begin INPUT statement for A
"LIMIT ANIMATION";                Title bar (same as before)
"TARGET";                         Label for input blank  for A
"ENTER DESTINATION";              Help message  to prompt user
0:                                Default value for A
INPUT H;                          Begin INPUT statement for H
"LIMIT ANIMATION";                Title bar (same as before)
"DELTA X";                        Label for input blank for H
"ENTER INCREMENT";                Help message to prompt user
.1:                               Default value for H
ERASE:                            Clears the display screen
IF (X-A)H>0                       If  H has the same sign as X-A,
THEN -H STO> H:                     then reverse the sign of H
END:
WHILE 1                           This WHILE loop will run
REPEAT                              until the user presses  ON
DISP  1;"X         "X:            Display the value of X on line 1
IFERR SIN(X)/X STO> Z             If the evaluation of SIN(X)/X results
THEN 0 STO> Z:                      in an error, set the value to 0
END:
DISP 3;                           Display the value of SIN(X)/X on line 3
"SIN(X)/X "Z:
```

```
X STO> O:                 Store current value of X in O
X+H STO> X:               Increment X by H
IF (A-X)(A-O) ≤ 0         If the new value of X is equal to A or
THEN                        on the opposite side as the previous
ERASE:                      value of X, then clear the display,
O STO> X:                   restore the old value of X, and
H/10 STO> H:                reduce  increment by a factor of 10
END:
END:
```

## HP38G EXAMPLE 4.

This is actually a "suite" of four programs (three are used as subroutines of the main program, called ARCHIMEDES). They provide an exploration of the Archimedean method for approximating the area of a circle. This program shows how the SETVIEWS command operates. Look at PLOT to see the graph of a regular polygon that can be inscribed in the unit circle. Look at VIEWS after running the program ARCHIMEDES for special interactive options to change the number of sides and to see the computed area of the regular polygon.

### *Program ARCHIMEDES*

```
SELECT Polar:               Activate Polar ApLet
1 STO> Angle:               Set ApLet mode to degrees
1 STO> HAngle:              Set Home mode to degrees
1 STO> R1(θ):               Set unit circle polar equation in R1
0 STO> θmin:                Set starting angle to 0
361 STO> θmax:              Set ending angle to 361
CHECK 1:                    Check R1 in SYMB
3 STO> N:                   Initialize number of sides to N=3
120 STO> θstep:             Initialize angle step to 360/3=120
SETVIEWS                    Set up special views
"DOUBLE # SIDES";           First special view allows user
ARCHI.1;1;                    to double number of sides
"INPUT # SIDES";            Second special view allows user
ARCHI.2;1;                    to set number of views
"AREA A=?";                 Third special view will display
ARCHI.3;1                     area of the inscribed N-gon
```

### *Program ARCHI.1*

```
2*N STO> N:                 Double size of N, the number of sides
360/N STO> θstep:           Set the corresponding new angle step
```

### *Program ARCHI.2*

```
INPUT N;                    INPUT for N, number of sides
"ARCHIMEDES";               Title bar
"# SIDES";                  Label for input blank
"ENTER NUMBER OF SIDES";N:  Help message to prompt user
360/N STO> θstep:           Set the corresponding new angle step
```

### *Program ARCHI.3*

```
ERASE:                      Erase the display
DISP 1;"A= ".5*N*SIN(360/N): Compute the area of the polygon
FREEZE:                     Freeze the display screen
```