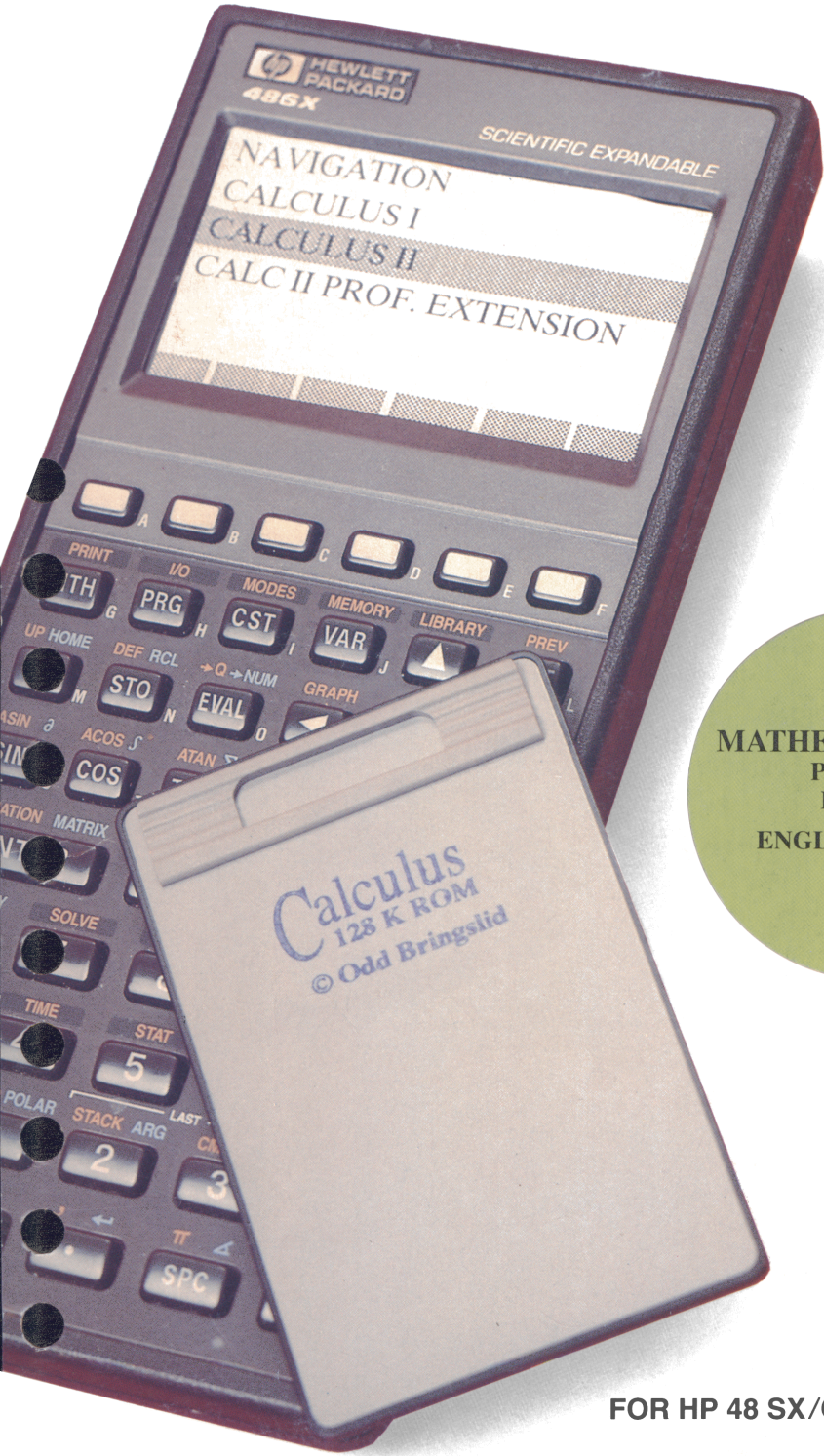
BRINGSLID®
APPLICATION SOFTWARE



MATHEMATIC Pac. 2.

Professional Extension.

ENGLISH VERSION

FOR HP 48 SX/GX

BAS
Mathematics II

VER 2.0A

A mathematical program for calculators (HP 48 GX)

Odd Bringslid
ISV
Spenningsgate 11 3601 Kongsberg
NORWAY

1st edition August 1992

2nd edition June 1994

IMPORTANT!

The basic use of the HP48 is not covered by this manual.
You should consult the User's Guide for this purpose.

Information to the 48SX user:

The documentation is prepared for the version GX.
The SX user has to be aware of the following:

The menu option EGVT in the MATR menu, needs an extra input:
the eigenvalue at stack level 2 (matrix at level 1).

Contents

1

Generally

This is part II of BAS mathematics. As in part I a pedagogical interface is stressed. BAS mathematics is a pedagogical tool in addition to a package for getting things calculated.

Hardware requirements

BAS Math II runs under the calculator HP 48GX. The program card may be inserted into either of the two ports and Math I could be in the other port.

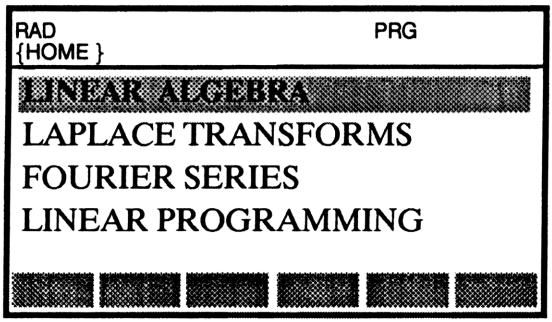
Starting up

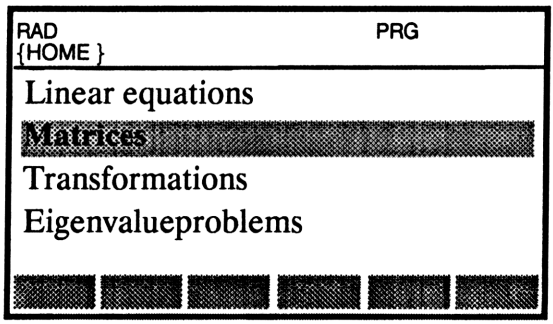
The LIBRARY menu will show up MAII. Pushing the MAII key will lead you into the main menu and then you simply push the START key.

User interface

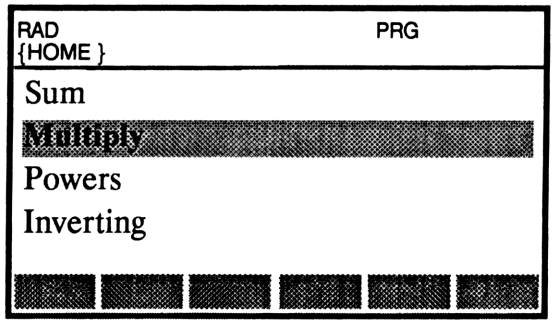
The MAII meny system is easy to use. Using the arrow keys allows you to move the dark bar and select by pushing ENTER.

In the following example you will enter the submenu for LINEAR ALGEBRA and select Matrices/Multiply.





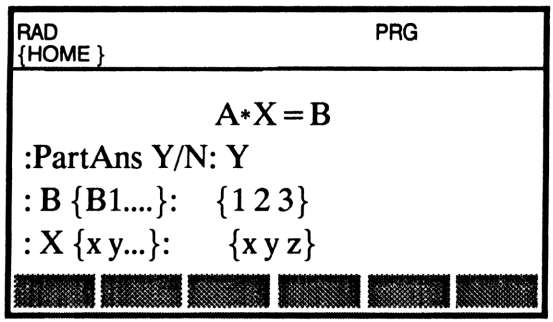
Under Matrices you will choose Multiply and you may multiply two symbolic matrices. The matrices are put into the SYMBOLIC MATRIX WRITER.



In the Matrix Writer you may delete, add, and echo from the stack (see manual for 48GX).

The input Editor

If you select Linear equations under LINEAR ALGEBRA you will enter the editor for input (input screen).



Here the input data can be modified and deleted and you can move around by using the arrow keys.

The cursor is placed right behind :PartAns Y/N: and here you enter Y if you want intermediate results. The arrow keys are used to get right behind :B {B1...}: and here you enter the right side vector of the system.

If you have done a mistake you may alter your input by using the delete keys on the calculator keyboard. You will not be able to continue before the data are correctly put in.

In the input screen there is often information about the problem you are going to solve (formulaes etc). Remember the ' ' in algebraics and separation of several data on the same input line by using blanks (space).

Echoing from the stack

If an expression or a value laying on the stack is going to be used, then the EDIT/↑STK/ECHO/ENTER sequence will load data into the input screen. Be sure to place the cursor correctly

Calculation finished

When a calculation is finished CALCULUS will either show up intermediate results by using the VIEW routine (intrinsic

MAII) or return directly to the menu. In the last case you will need to use the →STK key to see the result laying on the stack.

Moving up and down in the menu

You can move downwards in the menu system by scrolling the dark bar and pressing ENTER. If you need to move upwards the UPDIR key will help. At any time you may HALT MAII and use the calculator independent of MAII by pressing the →STK key. CONT will get you back to the menu system.

STAT and MATR menues

On the menu line at the bottom the choices STAT and MATR are possible. Here you will have access to some routines regardless of your current menu position.

STAT:

- NORM — Normal distribution
- INVN — Inverse normal distribution
- UΣDAT — Sample mean, st. dev., median
- KΣDAT — Class table
- ΣDAT — Discrete table, two columns

MATR:

- ADD — Add symbolic matrices

• BASE?	Check if a set of vectors are linear independent
• DET	Determinant
• DIAG	Diagonalize a matrix
• EGV	Eigenvalues of a matrix
• EGVT	Eigenvectors of a matrix
• INV	Invert
• MULT	Multiply
• ORTH	Orthogonalize a basis
• ORT?	Check if a matrix is orthogonal
• RANK	Rank of matrix
• SYM→	Converts matrix {{}} to [[]]
• TRN	Transpose

The input is a matrix (or two) on stack-level 1 (or 1 and 2). The matrix is stored for later use as MATR in your current VAR menu. The matrix is also echoed to the stack.

Leaving MAII

Pushing the EXIT key will leave MAII.

Intermediate results

In the input screen you may choose PartAns Y/N. Choosing

N the result will be laying on the stack and you have to use →STK to see the answer. Choosing Y, different pages of intermediate results will show up or more than one result is laying on the stack.

The degree of details in the partial answers is somewhat different, but some of the results covers "the whole answer". In every case this will give the user a good help.

Different parts of an answer may be found on different pages and the page number can bee seen (use arrow up/down).

When PartAns Y(es) is chosen all numbers will show up with two figures behind comma. If a more accurate answer is necessary, you will have to look on the stack and perhaps use the N FIX option.

Flag status and CST menu

The flag status and CST menu you had before going into MAII will be restored when you leave by pushing EXIT.

2

Linear algebra

The subject linear algebra covers linear equations with solution also for singular systems, matrix manipulation (symbolic), eigenvalue problems included systems of linear differential equations, linear transformations in two and three dimensions and vector spaces.

Linear equations (Gauss method)

Linear equations with symbolic parameters are handled. The equations have to be ordered to recognize the coefficient matrix and the right side. The equations are given in the form:

$$\{\{\mathbf{A}\}\}*\{\{\mathbf{X}\}\}=\{\{\mathbf{B}\}\}$$

A is the coefficient matrix , **X** a column vector for the unknowns and **B** the right side column vector. Symbolic coefficients are possible.

If Det(**A**) = 0 (determinant) the system will be singular (self contradictory or indefinite). This is stated as "Self contradictory" or the solution will be given in terms of one ore more of the unknowns (indefinite). Example:

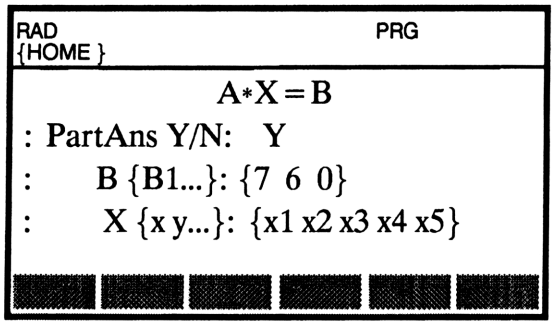
$$\{x,y,z\} = \{x,2*x-1,x-4\}$$

The value of x is arbitrary so there is an infinite number of solutions.

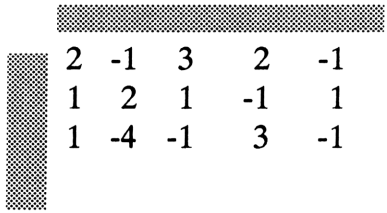
If the system is underdetermined (too few equations), the solution will be given in the indefinite form. If the system is overdetermined (too many equations) the solution will be given in the indefinite form if the equations are lineary dependent or as "Self contradictory" if they are lineary independent.

The solution algorithm is the Gauss elimination. If PartAns Y(es) is selected, the different stages in the process will be given as matrices on the stack which may be viewed by using the MATW option (LIBRARY). The coefficient matrix and the right side vector are assembled in one matrix (**B** is the rightmost column).

Interface:



The symbolic matrix writer will now appear. The following matrix is put into it:



The example solves the system:

$$2x_1 - x_2 + 3x_3 + 2x_4 - x_5 = 7$$
$$x_1 + 2x_2 + x_3 - x_4 + x_5 = 6$$
$$x_1 - 4x_2 - x_3 + 3x_4 - x_5 = 0$$

The system is indefinite (too few equations)) and the solution is given in terms of x_5 and x_4.

The system is indefinite (too few eqautions)) and the solution is given in terms of x5 and x4.

Matrix calculations

Some operations on symbolic matrices are done (not covered by the HP48 intrinsic functions). The matrices are put into the SYMBOLIC MATRIX WRITER and the matrix is put on the stack by pushing ENTER.

Addition
Both matices are put into the matrix writer and added. An error message is given for wrong dimension.

Multiplication
Both matrices are put into the matrix writer and multiplied. An error message is given for wrong dimensions. The first matrix has to have the same number of columns as the second has rows. Be aware of the order of the matrices.

Inverting
The matix is put into the matrix writer and inverted. An error message is given if its not quadratic.

Determinant
The determinant of a symbolic matrix is calculated. The ma-

Rank of a matrix

The rank of a marix is calculated. This routine may be used for testing linear independency of rowvectors. The routine makes the matrix upper traingular and PartAns Y gives the different stages of the process.

Trace

The trace of a square matrix is calculated. An error message is given if the matrix is not quadratic.

Orthogonal matrix

This routine is testing whether the matrix is orthogonal i.e. the inverse is equal to the transpose. An error message is given for wrong dimension (must be quadratic). The answer is logic 0 or 1. May be used to investigate if rowvectors are orthogonal i.e. is an orthogonal basis of a vector space.

Transpose matrix

The transpose of a symbolic matrix is calculated.

Symmetric

Investigates whether a matrix is symmetric or not. Logic 0 or 1.

Linear transformations

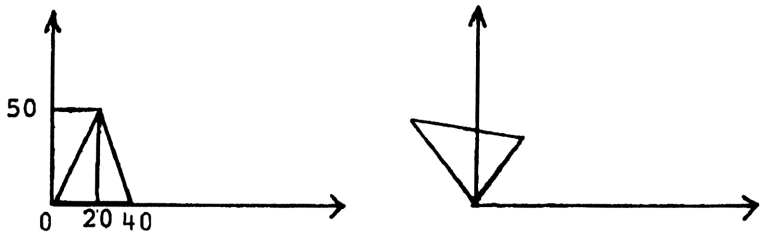
Linear transformations include coordinate transformations in the plane and in the three dimensional space. The transformations are rotation, translation and scaling. The point to be transformed is given relative a rectangular coordinate system.

Mixed transformations (concatinating) is possible. The order of the transformations is important if rotation is one of them.

2 D transformations (two dimensions)

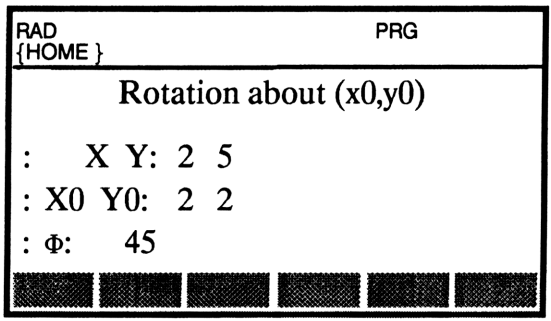
Rotation

The rotation angle must be given in degrees and the transformed point is given as components of a list (to allow symbols). The rotation is counterclockwise for positive angles about an arbitrary point.



2.1 Rotation of a triangle about origo

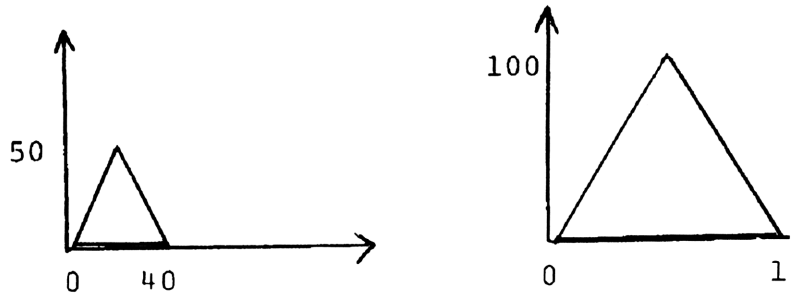
Interface:



The point (2,5) is rotated about (2,2) an angle 45^{o}. To rotate a triangle all three points have to be transformed.

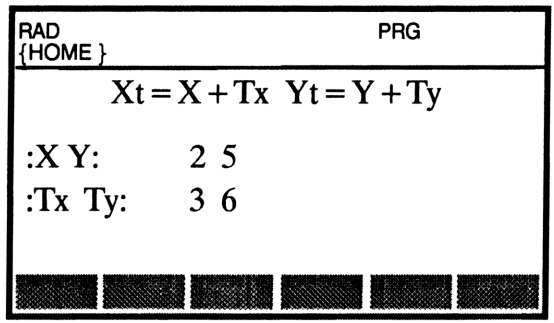
Translation

This is a pure translation of a point, the coordinates are given an addition.



2.2 Translation of a triangle

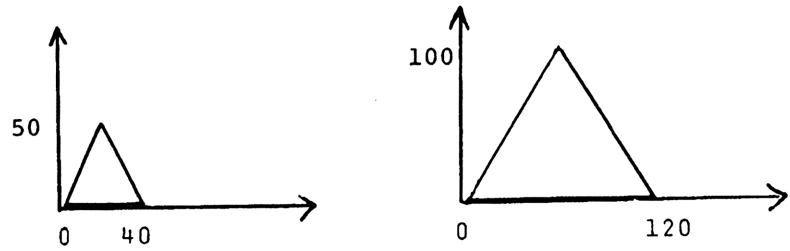
Interface:



The example moves the point (2,5) to (2,5) + (3,6) = (5,11)

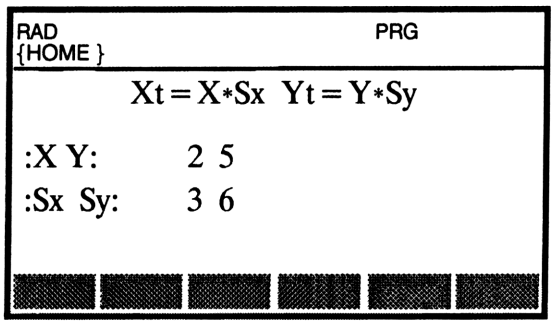
Scaling

The coordinates are multiplied by a factor. For a geometric figure where the points are scaled, this will give a smaller or bigger figure. If the X and Y coordinates are scaled differently this will alter the shape of the geometric figure.



2.3 Enlarging of a triangle

Interface:



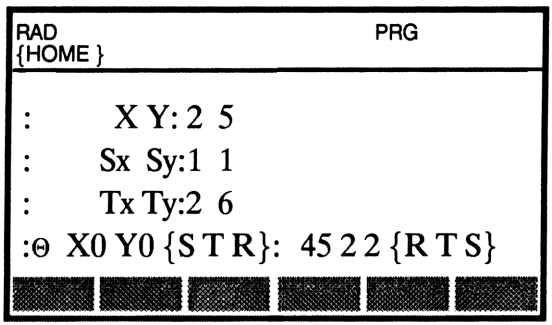
The example multiplies 2 with 3 and 5 with 6 and the point (2,5) is moved.

Concatinating (mixed transformations)

Concatinating means a mixture of several transformations.

The order of the transformations is important, given in a list as {R S T} (rotation, scaling and translation).

Interface:



In the example the point (2,5) is rotated clockwise 45^{o} about the point (2,2) first, then a translation of 2 in the x-direction and 6 in the y-direction. There is no scaling, indicated by 1 1 for the scaling factors.

Rem. No translation gives Tx = Ty = 0 and no rotation gives Θ = 0.

3 D transformations (three dimensions)

Translation

The interface is the same as 2D translation, with one extra coordinate and one extra translation.

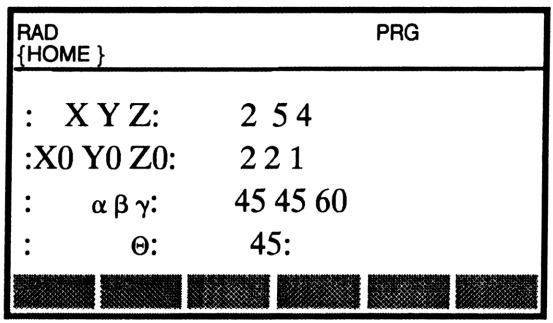
Scaling

The interface is the same as 2D scaling, with one extra coordinate and one extra scaling.

Rotation

3D rotation is somewhat more complicated than in two dimensions. The rotation axes has to be specified, i.e angles relative the coordinate axes and a point.

Interface:

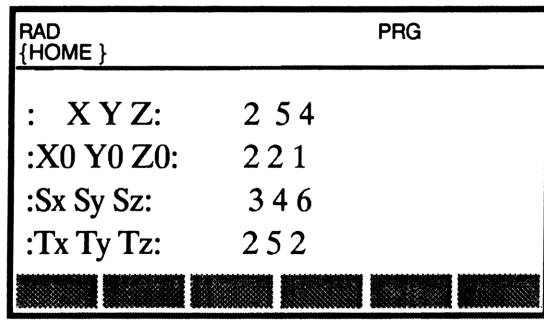


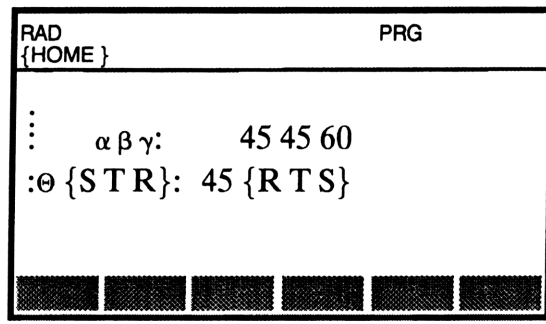
The example rotates the point (2,5,4) about an axes through the point (2,2,1) and with angles relative the x-, y-, and z-axes equal to 45^{o}, 45^{o} and 60^{o}.

Concatinating (mixed)

The same interface as in the 2D case, but the rotation axes now has to be specified.

Interface:





In the example the point (2,5,4) is rotated about the given axes and then the specified translation and scaling is carried out.

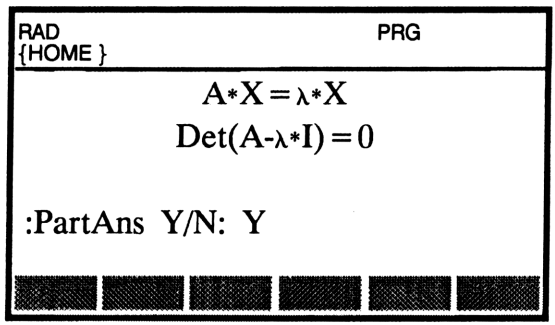
Eigenvalueproblems

Here you may find the eigenvalues and the eigenvectors of a matrix, diagonalize a matrix and solve a system of linear differential equations.

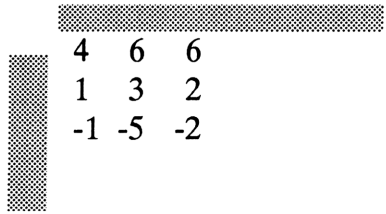
Eigenvalues

The eigenvalues of a matrix are determined by the equation $\mathbf{A}*\mathbf{X}=\lambda*\mathbf{X}$, where **A** is the matrix and **X** a column vector (eigenvector). λ is called the eigenvalue.

Interface:



Now the matrix has to be specified, and you may put it into the symbolic matrix writer:



The example finds the eigenvalues of the matrix:

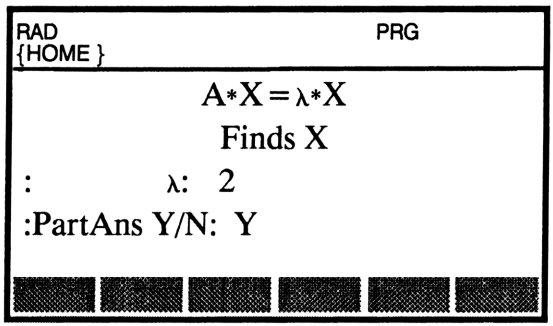
$$\begin{bmatrix} 4 & 6 & 6 \\ 1 & 3 & 2 \\ -1 & -5 & -2 \end{bmatrix}$$

The matrix has an eigenvalue with multiplicity 2 ($\lambda = 2$).

Eigenvectors

A matrix has infinite many eigenvectors because the system of equations that determines the vectors is indefinite. The eigenvectors are given in terms of arbitrary parameters. A set of eigenvectors will normaly be linear independent even if the eigenvalues have multiplicity greater than 1. But this is not always the case.

Interface:



The matrix now has to be put into the matrix writer. We use the same matrix as in the determination of eigenvalues. By choosing PartAns Y the indefinite system of equations that determines the eigenvectors will be given. We see that only two of the eigenvectors are lineary independent (only one arbitrary parameter c).

Diagonalization

For a matrix **A** we can write:

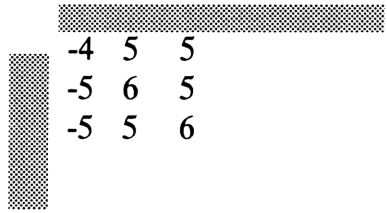
$$\mathbf{D} = \mathbf{K}^{-1} * \mathbf{A} * \mathbf{K}$$

Here **K** is a matrix composed of the eigenvectors of **A** which have to be lineary independent. **D** is a diagonal matrix with the eigenvalues on the diagonal. If the eigenvectors are

lineary dependent (as in the example of eigenvectors), then the matrix cannot be diagonalized (not diagonalizable).

Interface:

The matrix has to be put into the matrix writer:



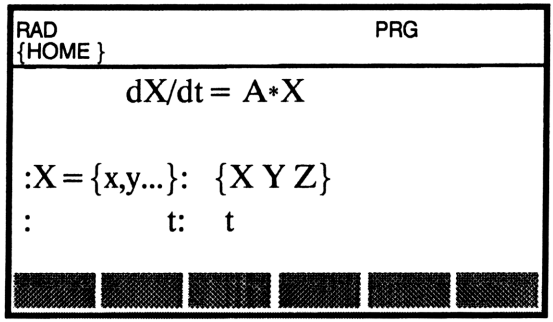
The output is the matrices **K** and **D**. The matrix $\mathbf{K}^{-1}$ may be found by inverting **K**.

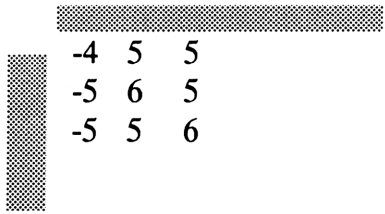
> *Rem. If intermediate results are wanted, you may look at the problems of finding eigenvalues and eigenvectors separately.*

System of differential equations

Here a set of linear, homogenous differential equations with constant coefficients are solved by using the method of diagonalization.

Interface:





The following system is solved:

$$dx/dt = -4x + 5y + 5z$$
$$dy/dt = -5x + 6y + 5z$$
$$dz/dt = -5x + 5y + 6z$$

The output contains the constants C1, C2 and C3 and the independent variable is t.

Rem. If intermediate results are wanted, you may look at the problems of finding eigenvalues and eigenvectors separately.

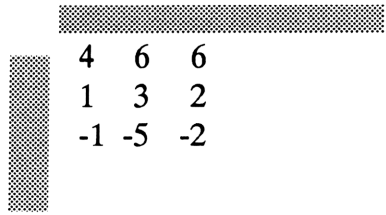
Under Info some information about the solving strategy is given.

Vector spaces

A vector space is a collection of vectors relative a basis where certain operations on them are defined. A basis is a set of linear independent vectors from the space. In an orthogonal basis the vectors are mutually orthogonal (inner product equals zero).

Basis?
This routine examines whether a set of vectors in the space is linear independent. The vectors are put into the matrix writer as rows and the output is logic 0 or 1.

Interface:



The example examines whether the vectors in $\mathbf{R}_3$ {4 6 6}, {1 3 2} and {-1 -5 -2} are lineary independent and then form a basis in $\mathbf{R}_3$ (three dimensional vector space).

Norm

Here the length or absolute value is calculated. The input vector is $\{v_1\ v_2\ v_3....\}$ and the output is a number or an expression if the vector is symbolic.

Norming

A vector is transformed into an unit vector
$e = V/NORM(V).V = \{v_1\ v_2....\}$.

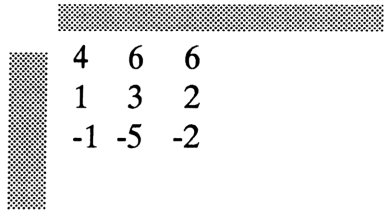
Scalar product (inner product)

The scalar product of two vectors is calculated. Symbolic vectors are possible.

Orthogonalization

An orthogonal basis is calculated with an arbitrary basis as a starting point using the Gram-Schmidt process.

Interface:



The basis b1 = [4 6 6], b2 = [1 3 2] and b3 = [-1 -5 -2] is given in $\mathbf{R}_3$. The basis is not orthogonal, but the routine makes it orthogonal.

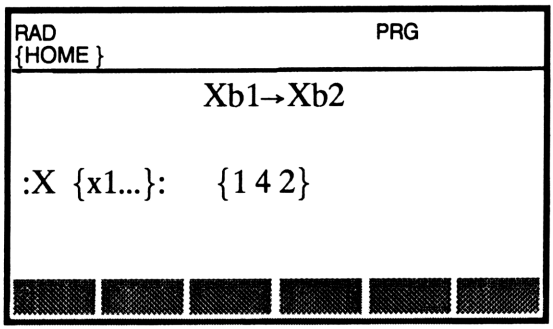
Rem. Symbolic vectors are not possible

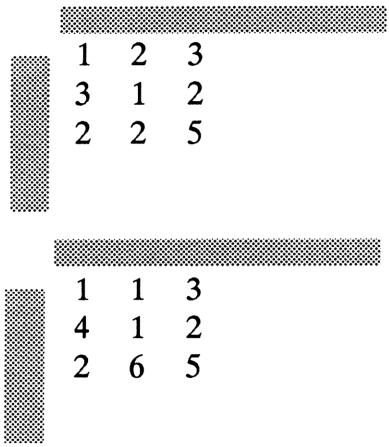
Orthogonal?
The routine examines whether a matrix is orthogonal. If the row vectors building up the matrix form an orthogonal basis, then the matrix is orthogonal.

Orthonorming
The routine is norming an orthogonal basis.

Vector in new basis
Given a vector V_{B1}, i.e. relative a basis B1. A new vector relative a basis B2 is calculated.

Interface:



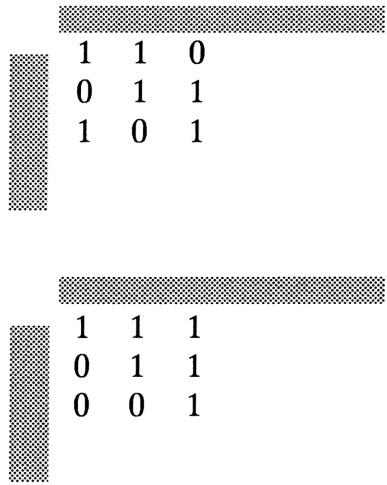


The vector {1 4 2} relative the basis
{ [1 2 3], [3 1 2], [2 2 5]} is transformed to the new basis {[113],[4 1 2],[2 6 5]}.

Transformation matrix in new basis

A matrix defines a linear transformation in a vector space relative the "natural" basis. This routine calculates a new transformation matrix relative a new basis. The "natural" basis is {[1 0 0],[0 1 0],[0 0 1]} in $\mathbf{R}_3$.

Interface:



The transformation matrix {{1 1 0}{0 1 1}{1 0 1}} in natural basis defines the transformation:

$$L(x_1,x_2,x_3) = (x_1 + x_2, x_2 + x_3, x_3 + x_1)$$

The example calculates the new transformation matrix relative basis {[1 1 1],[0 1 1],[0 0 1]}. Symbolic elements are possible in the matrices.

3

Laplace transforms

Laplace transforms are used for solving differential equations and can, contrary to other methods, deal with functions f(t) which are discontinous in the equation

$$a*y'' + b*y' + c*y = f(t)$$

Discontinous f(t) may be composed by using the Unit Step function u(t-a) defined as:

u(t-a): IF t a < THEN 0 ELSE IF t a > then 1 END END

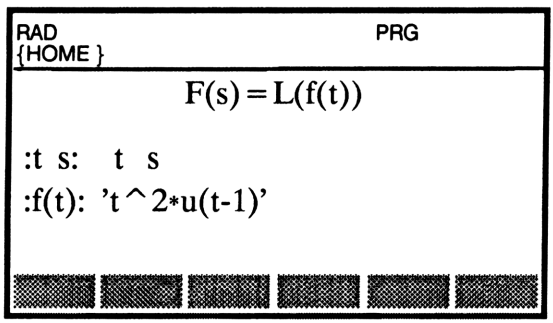
This function is not implemented in CALCULUS in other ways than as a symbol, and the user has to make a program to define it for evaluation.

Laplace transform

The Laplace transform of the following functions may be found:

- $f(t) = t \hat{} n$, $n > -1$
- $f(t) = \mathrm{Sin}(a*t)$, a arbitrary
- $f(t) = \mathrm{Cos}(a*t)$, a arbitrary
- $g(t) = f(t)*e^{at}$, a arbitrary
- $g(t) = f(t)*u(t-a)$, $a \geq 0$
- $g(t) = f(t)*u(t-a)*e^{bt}$, $a \geq 0$ b arbitrary
- $h(t) = g(t)*t$
- Linear combinations of theese functions

Interface:

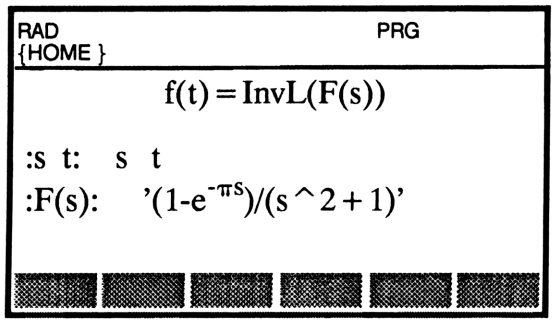


The example calculates the Laplace transform of $f(t) = t \hat{} 2*u(t-1)$.

Rem. If CALCULUS cannot find the Laplace transform an error message is given (the transform does not exist or its not implemented)

Inverse Laplace transform:
The inverse transform is calculated. The types of functions which can be inverted are the transforms of the functions listed on page 37.

Interface:



The example calculates the inverse transform of
$F(s) = (1\text{-}e^{-\pi s})/(s^2 + 1)$.

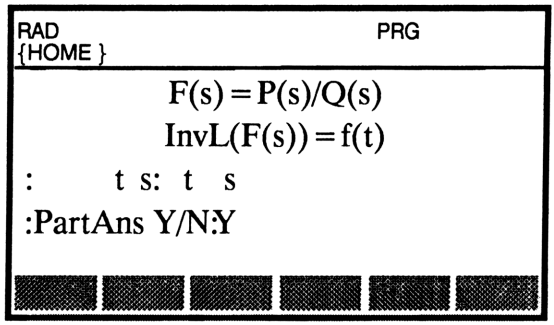
Inverse L Partial fractions

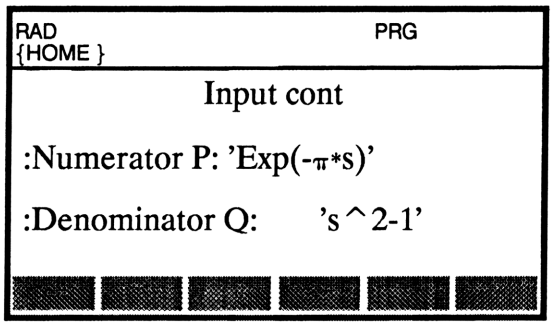
If the denominator of F(s) is of second degree and may be factorized in first degree factors or of a higher degree than 2, the denominator has to be split into partial fractions.

Intermediate results (Partial Answers) is possible to show the splitting into partial fractions.

> *Rem. If the transformation does not exist the error message "does not exist" is given. If the expression is too complicated the message "not rational" may appear. The expression may then be split up.*

Interface:





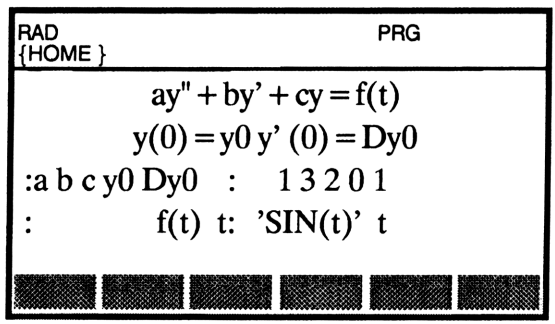
In the example $F(s) = e^{-\pi s}/(s^2-1)$ is split into partial fractions and then transformed. The shift $e^{-\pi s}$ will be taken care of before the splitting into partial fractions.

Differential equations (initial value problem)

Laplace transforms are suitable for solving initial value problems, in particular when the "right hand side function" is discontinous.

The answer is given in the form Y(s) = P(s)/Q(s)/R(s) which has to be transformed into P(s)/((Q(s)*R(s))) before the routine for partial fractions is used to solve the problem.

Interface:



The equation y" + 3y' + 2y = Sin(t) with initial conditions y(0) = 0 and y' (0) = 1 is transformed.

4

Probability

In this chapter of probability theory we will look at unlike discrete probability distributions in addition to the normal distribution which is continous. For the discrete distributions both the cummulative probability and the point probability may be calculated.

For the discrete distributions and in connection with pure combinatorial calculations, we have distinguished between with and without replacement.

> *Rem. Probabilities must be less then or equal to 1 and greater than or equal to 0.*

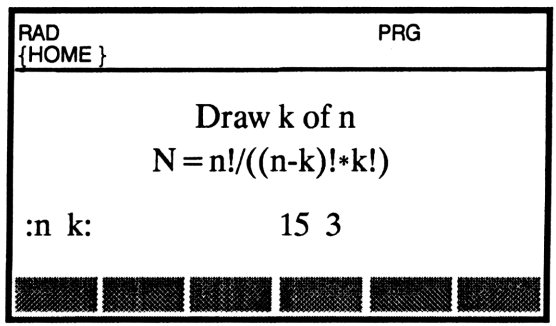
Without replacement

Without replacemnet means that we dont put the drawn element back again.

Combinations, not ordered

This routine calculates the number of possibilities to draw k elements of total n without replacement and without regard to order.

Interface:

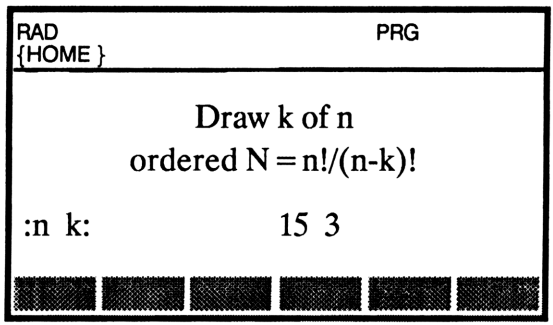


The example calculates the number of possibilities to draw 3 elements from total 15 elements without regard to order.

Combinations, ordered

If the order is important you will have to use this routine. The same elements in different orders will then be separate events.

Interface:

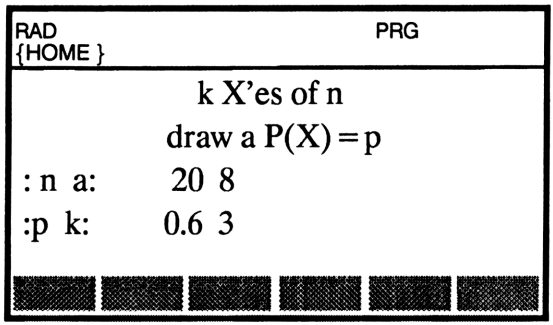


The example calculates the number of combinations when drawing 3 elements from 15 with regard to order.

Hypergeometric distribution

Here the probability of drawing exactly k X'es from a population of n when a elements are drawn at a time without replacement is calculated. The probability for the X to be drawn is p.

Interface:



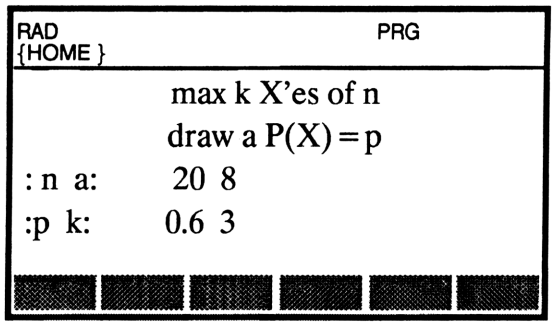
The probability that 3 elements have the mark X when drawing 8 elements of total 20 is calculated. The probability of X to occur is 0.6. If k > a or p > 1 the probability is 0.

The example may be "drawing" individuals from a population of 20 where 12 is women (p = 12/20 = 0.6). The probability that of 8 "drawn" individuals 3 is women is calculated.

Hypergeometric distribution function

The cummulative probability is calculated, i.e. the probability that maximum k elements are drawn. This is the sum of the probabilities of k = 0,k = 1,k = 2 and k = 3.

Interface:



The example calculates the probability that 3 elements is drawn with the mark X (p = 0.6) from a total of 20 by drawing 4 at a time or 1 by 1 without replacement.

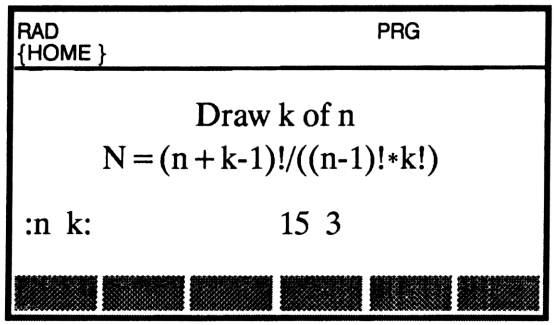
With replacement

Here the elements are replaced by drawing so that the probability is the same every time an element is drawn (unconditional drawing).

Combinations, unordered

This routine calculates the number of combinations of drawing k elements from n without replacement, without regard to order.

Interface:

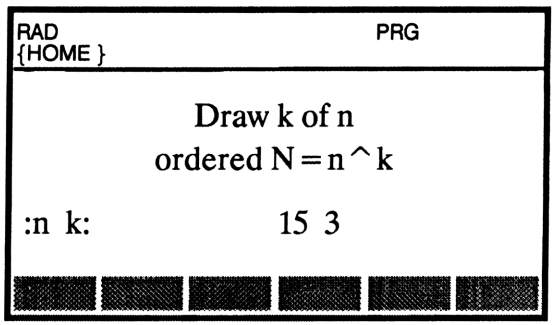


Here the probability of drawing 3 elements of total 15 is calculated. Order is indifferent.

Combinations, ordered

If the order is critical, this routine has to be used. The same elements in different orders are separate events.

Interface:

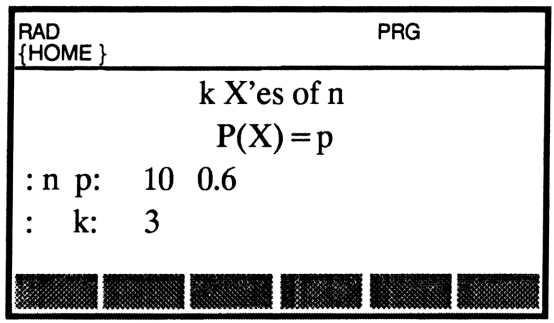


The example calculates the number of possibilities with the same figures as in the previous example, but now with regard to order.

Binomial distribution

The routine calculates the probability of drawing exactly k elements with the mark X of total n, where the probability of X itself is p. Independent trials (with replacement).

Interface:



The probability of drawing 3 elements with the mark X when X has the probability of 0.6 is calculated. The number of independent trials is 10. p > 1 gives an error message.

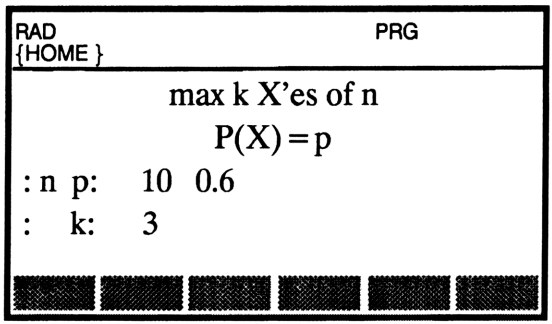
The example may be the production of glasses where the probability of first assortment is 0.6. If 20 glasses are produ-

ced, the example calculates the probability that 3 glasses are first assortment.

Binomial distribution function

The cummulative probability is calculated, i.e. the sum of the probabilities for k = 0, k = 1, k = 2 and k = 3 if k = 3.

Interface:

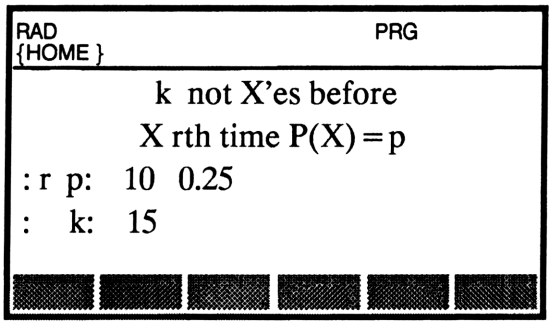


The example calculates the probability that maximum 3 elements have the mark X (p = 0.6) in 10 independent trials.

Negative binomial distribution

This distribution gives the probability of k failures before the r'th success in a series of independent trials each of which the probability of success is p.

Interface:



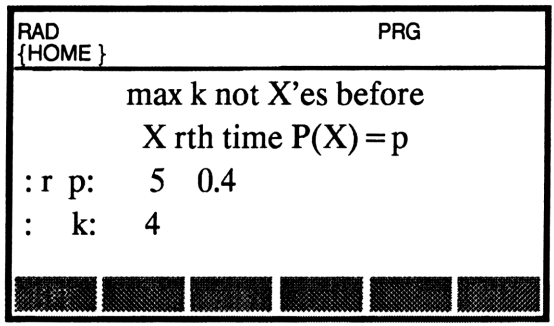
The probability of 15 failures before the 10th success when the probability of success is 0.25 is calculated.

The example may be the drawing of cards and the calculation of the probability of drawing 15 cards that are not clubs before the 10th club.

Negative binomial distribution function

This routine calculates probability of maximum k failures before the r'th success.

Interface:



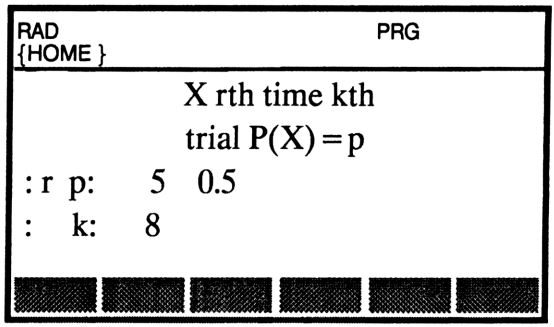
The probability of maximum 4 failures before the 5th success is calculated. Probability of success is 0.4.

The example may be the drawing of balls from a hat that contains 40% white balls. The probability of finding 5 not white balls before drawing maximum 4 white balls is calculated.

Pascal distribution

The probability of the r'th success in k'th trial in a series of independent trials is calculated.

Interface:



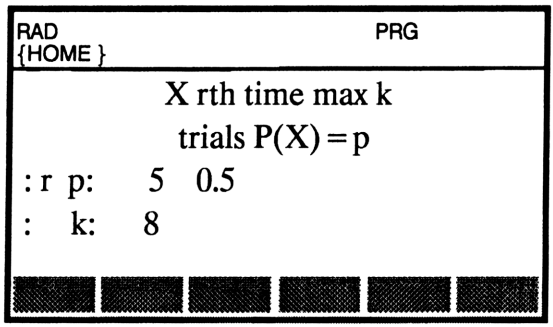
The probability of finding the mark X 5th time in the 8th trial is calculated.

> *Rem. The geometric distribution is a special case with r = 1.*

Pascal distribution function

The probability of the r'th success in maximum k trials is calculated. The probability of success is p.

Interface:

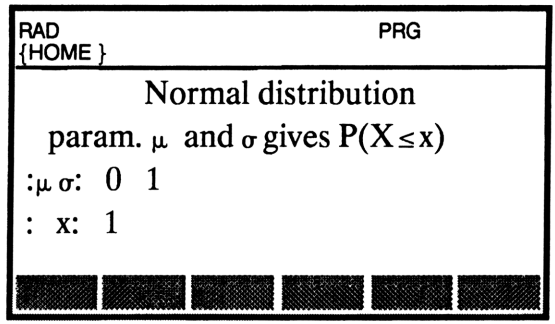


The example calculates the probability of finding the mark X the 5th time in maximum 8 trials (Tossing a fair coin we find the probability of finding the 5th head in maximum 8 trials).

Normal distribution

This is a continous distribution and only cummulative probabilities are calculated.

Interface:



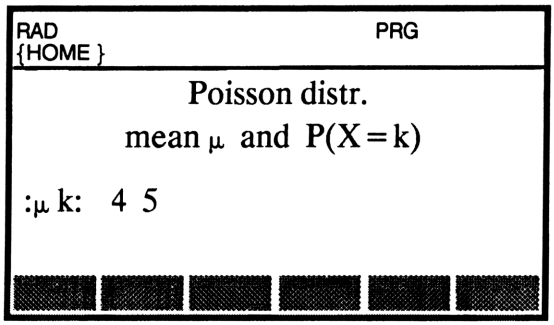
The probability that a random variable is less than or equal to 1 is calculated. The mean and the standard deviation is 0 and 1.

Rem. $P(a \leq x \leq b) = P(x \leq b) - P(x \leq a)$ *and* $P(x > a) = 1 - P(x \leq a)$

Poisson distribution

The Poisson distribution is used as a model when we are interested in events within intervals of time or other variables.

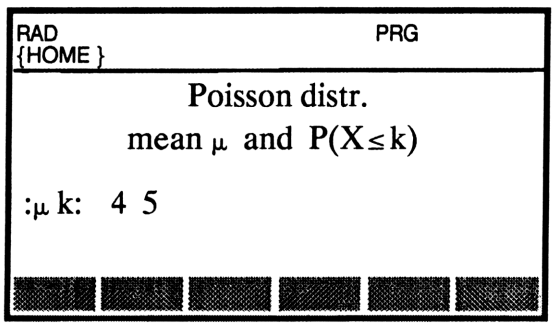
Interface:



The example calculates the probability that a random variable X is exactly 5 when the mean is 4.

Poisson distribution function

Interface:



The probability that X is less than or equal to 5 is calculated, the mean is 4.

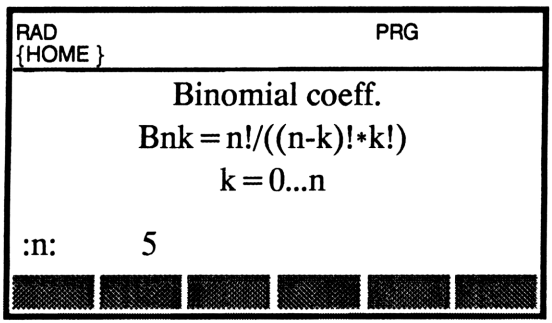
Info

Here information about probability and some distributions is given.

Binomial coefficients

Binomial coefficients Bnk = n!/((n-k)!*k!) are calculated from k = 0 to k = n and put in a list.

Interface:



The example calculates {Bn0 Bn1 Bn2 Bn3 Bn4 Bn5}.

5

Statistics

We will focus on some statistical methods and description of samples. Within description of samples we will use discrete tables and class tables (discrete and class statistics). You can convert from class statistics to discrete statistics by using the mean value of the intervals as the discrete value.

Statistical methods are represented by confidence intervals and hypothesis testing for distributions. The "best" fit for the normal distribution uses the method of least squares.

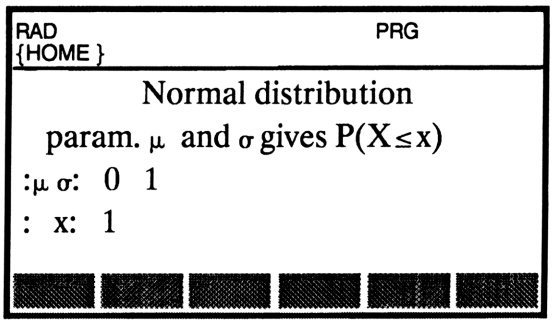
The normal distribution, kji-square distribution and student-t distribution are included and its possible to find both the probability and the value of the random variable for given probability.

Distributions

Normal distribution

The normal distribution gives $p(X \leq x)$ for given x-value. The mean μ and standard deviation σ have to be known.

Interface:

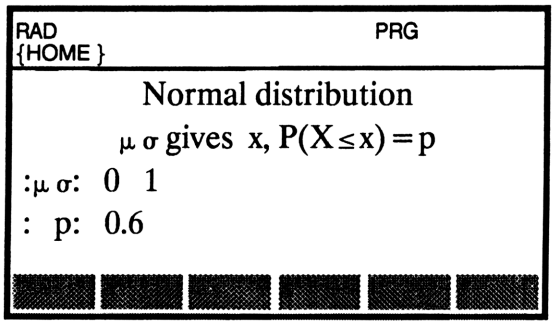


$P(X \leq 1)$ for $\mu = 0$ and $\sigma = 1$ is calculated.

Inverse normal distribution

The routine finds the value of the random variable x with given probability p, μ and σ are known.

Interface:

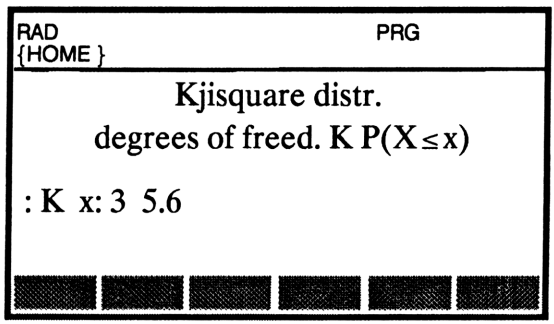


The value of x with $P(X \leq x) = 0.6$, $\mu = 0$ and $\sigma = 1$ is calculated.

Kji-square distribution

The kji-square distribution is used to find confidence intervals and in connection with fitting a distribution to a sample.

Interface:

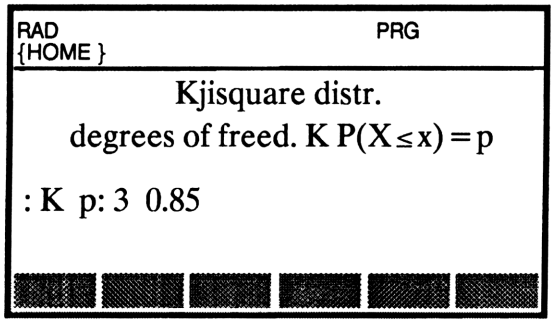


The example calculates $P(X \leq 5.6)$ where X is kji-square distributed with 3 degrees of freedom.

Inverse kji-square

The value of x for given probability is calculated.

Interface:

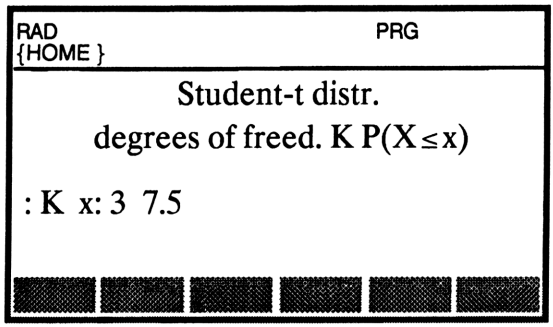


The example calculates the value of X so that $P(X \leq x) = 0.85$ with 3 degrees of freedom.

Studen-t distribution

This distribution is used to find confidence intervals in BAS MAII.

Interface:

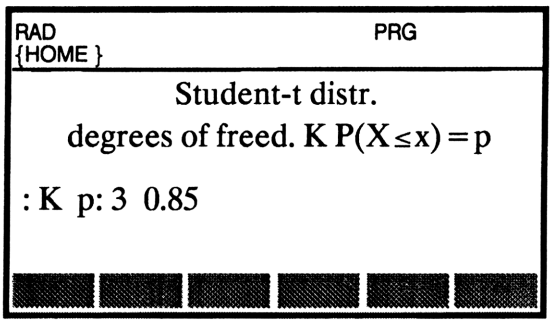


$P(X \leq 7.5)$ with 3 degrees of freedom is calculated.

Inverse student-t

This routine calculates the value of the random variable x.

Interface:



The value of x is calculated so that $P(X \leq x) = 0.85$ with 3 degrees of freedom.

Confidence intervals

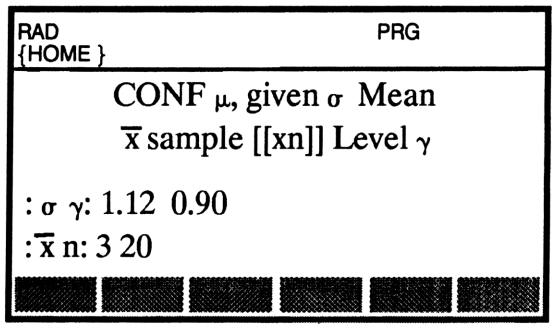
Confidence intervals in connection with the normal distribution are calculated for the mean μ and variance σ^2.

> *Rem. Mean value and standard deviation for a sample may be calculated under this menu. These values may be used as point estimates for the parameters in the distribution function.*

Confidence interval for the mean μ, given value of σ.
For known σ we may use the normal dsitribution to find the value c so that $F(c) = P(x \leq c) = 1/2(\gamma + 1)$ with confidence level γ. The interval is given in the form $[a \leq \mu \leq b]$.

The interval is calculated from a sample [[xi]] and the mean value has to calculated in advance by using the menu option: Mean value.

Interface:



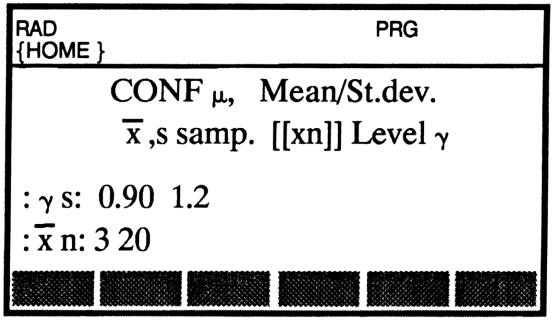
The example calculates the confidence interval for the mean in the normal distribution, based on a sample [[xn]] with mean 3 and confidence level 90%. The number of values in the sample is 20 and the normal distribution has the standard deviation 1.12.

Confidence interval for the mean μ, unknown σ

If σ is not known the estimate s for standard deviation from the sample is used. To find the value of c so that $F(c) = 1/2(\gamma + 1)$, the student-t distribution is used..

The interval is calculated from a sample [[xi]] with n values and the mean and the standard deviation of the sample has to be calculated in advance.

Interface:

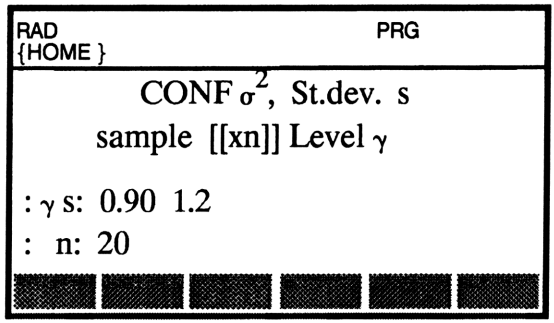


The confidence interval for the mean is calculated based on a sample with 20 values, standard deviation 1.2, mean 3 and with confidence level 90%.

Confidence interval for variance, μ is unknown

The standard deviation of the sample has to be calculated first (separate menu option). The calculation is based on the kji-square distribution.

Interface:

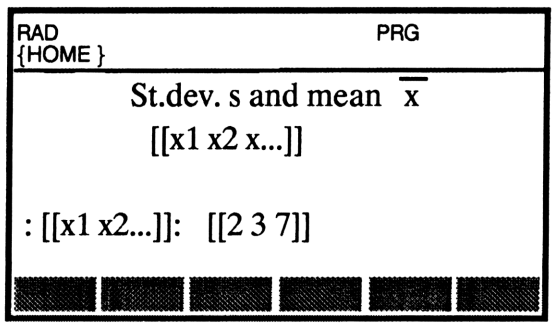


The example calculates the confidence interval for σ^2 based on a sample with standard deviation 1.2, confidence level 90% and 20 values in the sample.

Sample $\bar{x}$, s, n and median

The mean value, standard deviation, number of values and the median are calculated. The table is stored as UΣDAT.

Interface:



The mean value, standard deviation, number of values and the median for the sample [[2 8 7]] are calculated.

Fitting

By using a sample of values, the "best" fit for the normal distribution is calculated, i.e. estimates for the mean μ and standard deviation σ is calculated. Hypothesis testing for assumed distribution is done (kji-square goodness of Fit).

The sample has to be given as a class statistic with given frequences (se description of samples).

In order to calculate estimates for the distribution parameters a discrete statistic has to be stored as ΣDAT (see separate menu option).

Normal distribution
A "best" fit based on the least squares is calculated. The sample has to be stored as a class statistic (KΣDAT) in advance.

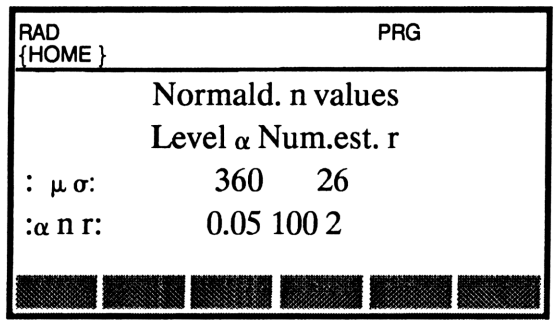
When KΣDAT is stored by using a separate menu option, there is no more input data necessary. Estimates for μ and σ will be calculated.

Hypothesis normal distribution

The class table is stored by using the separate menu option. The separation in different classes is done with minimum 5 values in each class. Upper and lower limit has to be ∞ and $-\infty$ and you can achieve this by using big numbers as the lower and upper limit for the class intervals.

For the calculation of mean and standard deviation as estimates for the parameters the discrete statistic has to be stored with known frequencies (ΣDAT).

Interface:



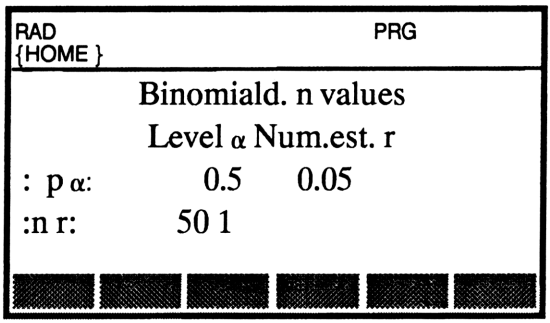
The example is testing whether the sample KΣDAT may be fitted to a normal distribution with significance level 5%. μ and σ are estimates and r, number of estimates, is 2. Number of values in the sample is 100.

The Kji-square distribution is used and the figure $kij0^2$ is tested against the theoretical value c, $P(X \leq c) = 1-\alpha$, where α is the significance level. If $kji0^2 \leq c$, the hypothesis is not rejected.

Hypothesis binomial distribution

The class statistic is stored and the parameter p is, if necessary, estimated as $p = \mu/n$.

Interface:

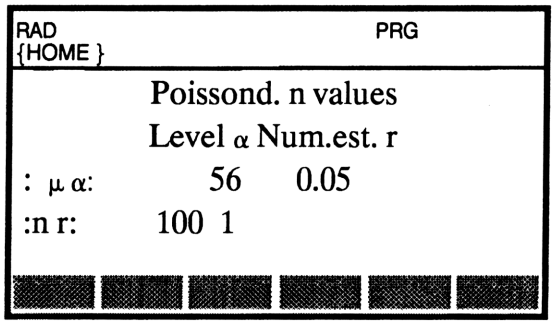


The example is testing whether the sample may be fitted to a binomial distribution with significance level 5%. p is estimated (0.5) and the value of r is 1. Number of values is 50.

Hypothesis Poisson distribution

The class statistic is stored and the parameter μ (mean) is, if necessary, estimated.

Interface:

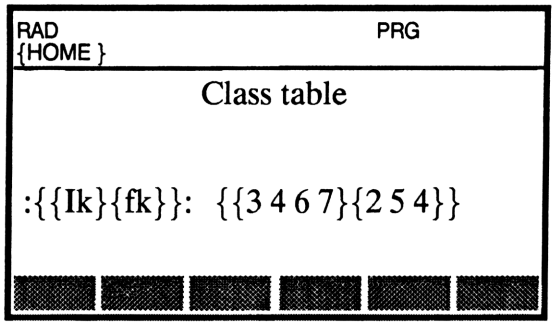


The example is testing whether the sample may be fitted to a Poisson distribution with significance level 5%. μ is estimated and r = 1. Number of values in the sample is 100.

Class tabel (class statistic)

A class table is a double list {{Ik}{fk}} where the first list contains the bounderies of the class intervals and the other list contains the number of values in the separate intervals.

Interface:



Interval limits are 3,4,6 og 7 and number of values in the intervals are 2,3 and 4.

Mean value and standard deviation based on frequency table

If the data is stored as a frequency table, we cannot use the ordinary sample routine to find the mean etc. The data is stored in ΣDAT. Use F for frequency table.

Storing discrete table

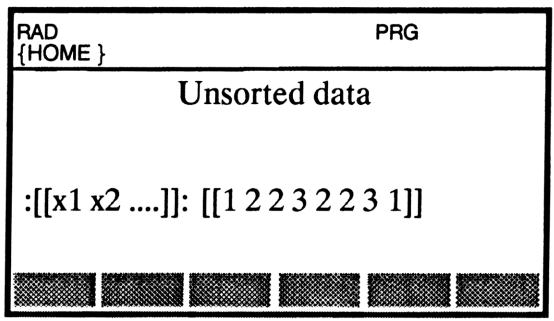
The sample is stored as ΣDAT. The data is put directly into the matrix writer where the first column contains the values and the second column the frequencies.

Description of samples

In this menu some calculations on discrete data and class tables from samples are done. This includes mean values, standard deviation, relative frequencies, histograms and frequency polygons.

Unsorted data

The values in a matrix are counted and sorted and a frequency table is stored as ΣDAT.



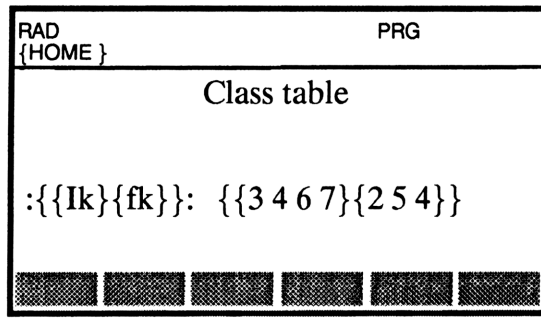
Discrete table ΣDAT

Here the values are stored in the first column and the frequencies in the second. In two variable statistics the second column will be the data for the second variable.

Classes KΣDAT

A class table is a double list {{Ik}fk}} where the first list is the limits of the intervals and the other the numbers in each interval.

Interface:



The limits of the intervals are 3,4,6 and 7 and the number of values is 2,3 and 4. The table is stored as KΣDAT for further uses.

Cummulative table.

The routine is calculating a table from a discrete table (ΣDAT). The new table will include relative frequencies, (column 4) cummulative frequencies (column 3) and cummulative relative frequencies (column 5).

KΣDAT→ΣDAT

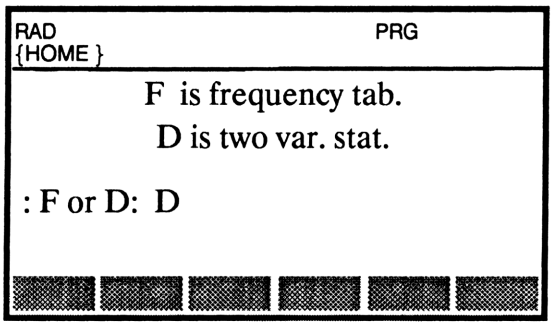
This routine transforms a class statistics into a discrete statistics by using the mean value of each interval as the repre-

sentative value. The table is stored as ΣDAT. Input data is KΣDAT which is stored in advance.

ΣDAT x og s

The mean value and the standard deviation is calculated based on a frequency table (F) or a two variable statistic table (D).

Interface:



The sample has to be stored in ΣDAT and the choice D will give the mean and standard deviation for each variable in a two variable statistics.

Rem. Under menu confidence intervals and STAT menu the mean and standard deviation for simple samples are calculated (one dimensional tables).

Histogram KΣDAT

Creates a histogram based on the class table KΣDAT which has to be stored in advance.

Frequency polygon KΣDAT

Creates a cummulative frequency polygon from KΣDAT.

Linear regression and correlation

A straight line is fitted by the use of least squares from the table ΣDAT. The line will be given as $Y = aX + b$, where X is the data in the second column in ΣDAT.

The correlation coefficient is a measure of the goodness of the fit and a value between -0.7 and 0.7 is a good fit.

Rem. ΣDAT is now a two variable statistics table and not a simple statistics frequency table.

6

Fourier series

Under Fourier series in symbolic form one can find the Fourier series of polynomials up to 2nd degree and a couple of other possibilites. The numeric series for a given number of terms may also be found. The input function may be bifurcated with different expressions in two different intervals.

The Fourier series are generally given as:

$$f(x) = a0 + \sum_{n=1}^{\infty} an*Cos(2*\pi*x*n/T) + \sum_{n=1}^{\infty} b_n*Sin(2*\pi*x*n/T)$$

T is the period and the coefficients are given:

$$a0 = 1/T \int_{-T/2}^{T/2} f(x)dx$$

$$an = 2/T \int_{-T/2}^{T/2} f(x)*Cos(2*\pi*x*n/T)dx$$

$$bn = 2/T \int_{-T/2}^{T/2} f(x)*Sin(2*\pi*x*n/T)dx$$

BAS MAII is able to find the series of the following types of functions:

- $f(x) = kx + b$
- $f(x) = kx^2$
- $f(x) = kSin(ax)$
- $f(x) = kCos(ax)$
- $f(x) = ke^{ax}$

Fourier series, symbolic form

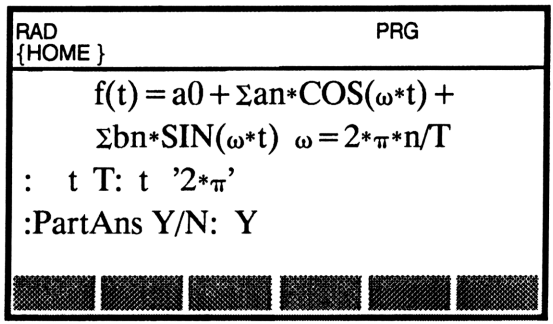
The expressions for a0, an and bn are found for given f(x). The series itself must be set up by the user. The function f(x) may be given as two expressions in two intervals:

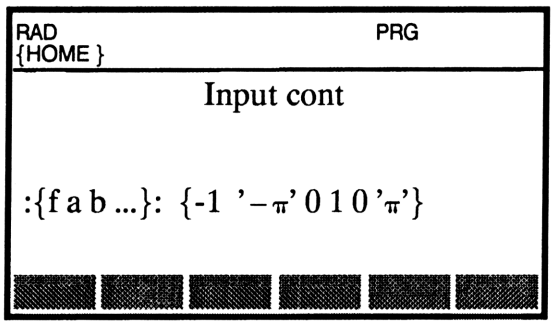
$$f(x) = \begin{cases} f1, a < x < b \\ f2, c < x < d \end{cases}$$

The functions are given as {f a b} where a and b are defining the interval or bifurcated as {f1 a b f2 c d}.

> *Rem. If the function is split in more than two intervals, BAS MAII may be used on two and two (or two plus one) intervals.*

Interface:





The example calculates the Fourier coefficients of

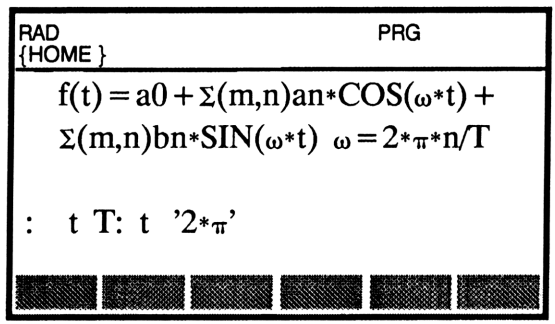
$$f(t) = \begin{cases} -1 & -\pi < t < 0 \\ 0 & 0 < t < \pi \end{cases} \qquad T = 2\pi$$

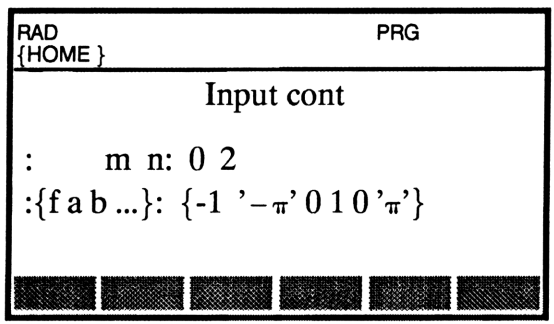
The answer is given with intermediate results (indefinite integrals are given).

Fourier series numeric results

A specified number of terms are calculated, but in terms of the independent variable. The input is the same as in symbolic form, but the number of terms has to be included and also the start and stop of the summation index. The integration is numeric and intermediate results are not given.

Interface:





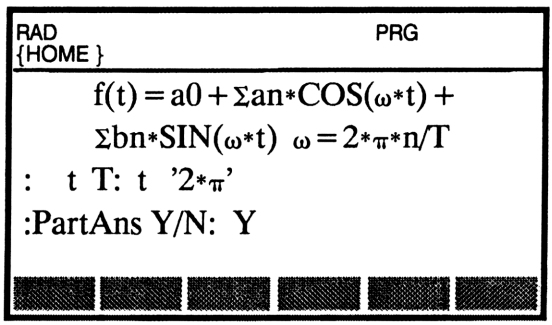
The first three terms are calculated. The terms n = 0 and n = 2 are 0.

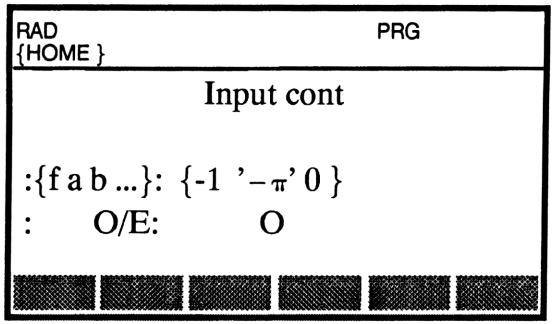
> *Rem. The accuracy of the integration is dependent of the choice for N FIX on the calculator*

Half range expansions

In many situations there is a practical need to use Fourier series in connection with functions that are given merely on some definite interval. They may be done periodic by an extension with the period as the double interval. The extension may be even or odd by choice (E/O).

Interface:





Here f(t) = -1, $-\pi < t < 0$ is given. An odd extension is marked by O.

7

Linear programming

The maximum or minimum of a linear function in several varaibles are calculated. The constraints are given as inequalities ("less then"). The routine does not handle degeneracy or solutions constrained to be natural numbers.

> *Rem. If such constraints are given and a decimal number is the answer, one cannot simply round off to nearest natural number. This will not always give the optimal solution.*

The algorithm used is the simplex method and it finds only the minimum of an object function. Any problem can be written as a minimum problem. If a maximum value of f(x) is going to be found, one may simply change the sign and find the minimum of -f(x).

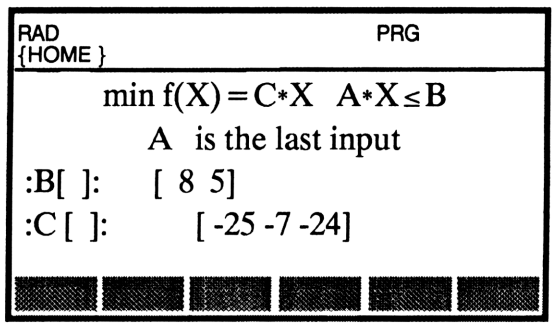
The constraints are defining the constraint matrix A. They are assumed to be of the "less then" type, but if you have got them as "greater then" type you may simply multiply both sides with -1 and change the inequality sign.

If we are going to maximize $f(x) = x1 + 2x2$ and one of the constraints are $x1-2x2 > 2$, then minimize $-x1-2x2$ with the constraint $-x1 + 2x2 < -2$.

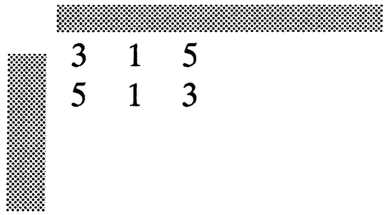
All independent variables are assumed to be positive or zero.

> *Rem. The independent variables have the symbol xi regardless of the symbols used in a given problem.*

Interface:



A:



The example solves the problem of finding the maximum of

$$f(x1,x2,x3) = 25x1 + 7x2 + 24x3 \text{ under the constraints}$$

$$3x1 + x2 + 5x3 \leq 8$$
$$5x1 + x2 + 3x3 \leq 5$$
$$0 \leq x1,x2,x3$$

C vector is the object function f, B vector is the right side of the constraints and A is the constraint matrix (left side). The solution is given as -39.5 which means that the maximum is 39.5.

8

Numerical methods

This chapter is emphasized on numerical methods which are common in the education of mathematics. They form a basis of topics usual in the numerical curriculum, but not all of them are used in pratical work. The version GX /G of HP 48 includes some routines for solving differential equations with a high degree of accuracy and may be used in practical applications.

Zeros

Equations of 1 unknown, 2 unknowns and 3 unknowns are solved numerically. Numerical solutions of equations are used where other methods are impossible. Different kinds of methods may be used and intermediate answers are given in the solution vector [x_0 x_1.....x_n] where x_n is the final answer. The vector may be viewed in the matrix writer (arrow down).

TOL

The input data requires the number TOL, tolerance. TOL has to do with the accuracy of the final solution. The calculation stops when the absolute value of the difference of two successive solutions divided with the solution is less than TOL.

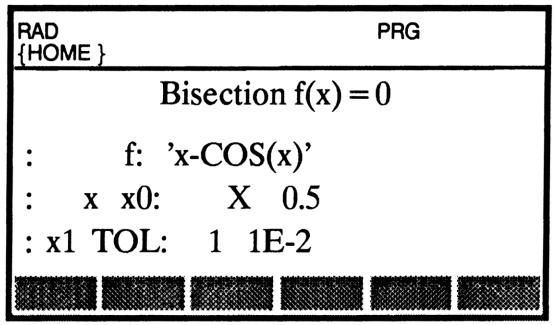
$$\text{STOP: } ABS(x_n - x_{n+1})/ABS(x_{n+1}) < TOL$$

If this cannot be achieved the method is said to diverge with this tolerance and we may choose another tolerance or adjust the starting point x_0. The programs stop after a specific number of iterations in all cases and the notion "diverge" has to be understood in this context.

Bisection method

We are going to solve the equation $f(x) = 0$, where f is a given function. The input is an interval which contains the solution (use f.x. a graphical plot to find the interval). By successive bisections of the interval in such a way that the solution lies in the interval chosen, we will always find a solution. The error will maximum be the width of the last interval. The advantage of the method is that it will always converge, but it is a slow method (many calculations).

Interface:

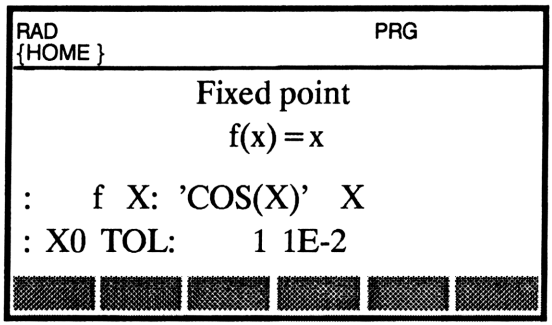


Equation	:	x-Cos(x) = 0
Starting interval	:	[0.5 1]
Tolerance	:	0.01
The result is a vector with 8 elements and 8 calcualtions are necessary		
Final solution	:	Last element of vector $x_9 = 0.74$

Fixed point iteration

Fixed point iteration is somewhat special because there is an infinite number of ways to write the equation, but only one is the best. The equation is written in the form g(x) = x. The equation x-Cos(x) = 0 may obviously be written Cos(x) = x, but this is necessarily not the best way (fewest iterations).

Interface:



Equation	:	x-Cos(x) = 0
Solution	:	$x_{12} = 0.74$
Starting point :		1
Tolerance	:	0.01

Secant method

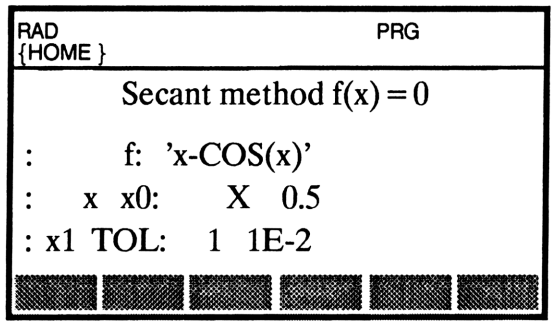
This method is faster than the two earlier methods. New x-values are calculated according to the formula:

$$X_{n+1} = X_n - \frac{X_n - X_{n-1}}{f(X_n) - f(X_n - 1)}$$

To assure that the solution lies in the interval $[X_{n+1}, X_n]$ and in this way be able to control the error, we redefine $X_n = X_{n-1}$ if $f(X_{n+1}) * f(X_n) > 0$. The method make use of the two

latest x-values, but only one new calculation of the function value is necessary for each step.

Interface:

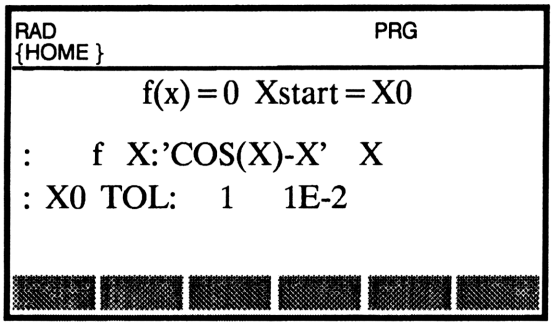


Equation	: x-Cos(x) = 0
Solution	: $x_3 = 0.74$
Starting point :	$x_0 = 0.5$ $x_1 = 1$
Tolerance	: 0.01.

Newton's method

This method usually needs the fewest iterations for a given tolerance, but compared to the secant method two calculations of function values are necessary for each step (f and f '). The method is however sensitive for the choice of starting point.

Interface:



Equation	:	Cos(x)-x = 0
Solution	:	x3 = 0.74
Starting point	:	1
Tolerance	:	0.01

$$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)}$$

Newton's method, 2 unknowns

Here 2 equations with 2 unknowns are solved. The method is an extension of the problem with one unknown and solves:

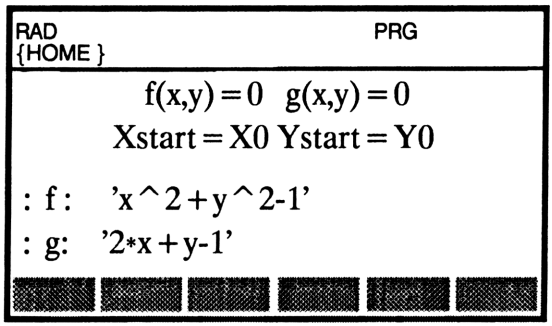
$$F(x,y) = 0,\ G(x,y) = 0, \text{ starting point } [x_0,y_0].$$

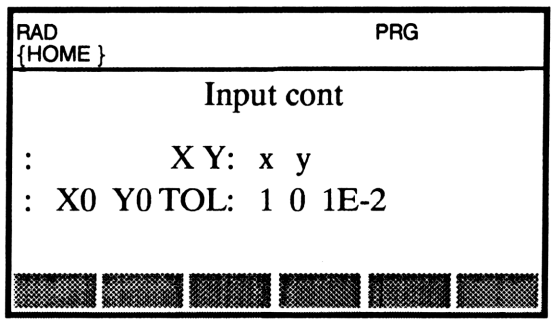
The iteration formula is

$$[X_{n+1},Y_{n+1}] = [X_n,Y_n] - \mathbf{J}^{-1} * [F(X_n,Y_n),G(X_n,Y_n)]$$

J is the Jacobian matrix of the system.

Interface:



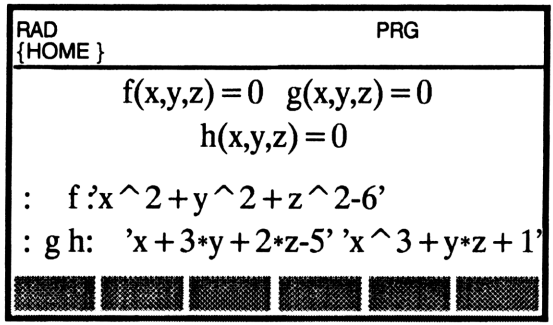


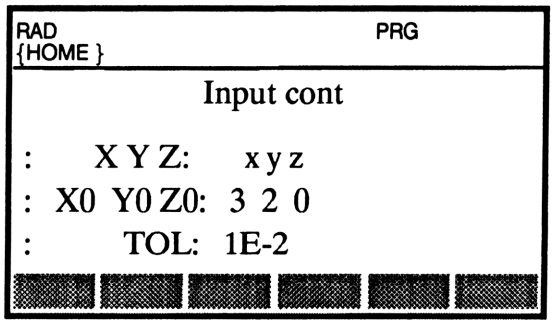
System of equations	:	$x^2+y^2=1$
		$2x+y=1$
Starting point	:	$x_0=1$
		$y_0=0$
Tolerance	:	0.01

Newton's method, 3 unknowns

Here 3 equations with 3 unknowns are given.

Interface:





System of equations	:	$x^2+y^2+z^2=1$ $x+3y+2z=5$ $x^3+yz=-1$
Starting point	:	$x_0=3\ y_0=2\ z_0=0$
Tolerance	:	0.01

Differential equations

Some of the methods chosen for the numerical solution of differential equations are of little practical importance (too inaccurate), but are however important in relation to the education of the stuff because they show some basic and important principals. This will be the case for specially Eulers method (RK1). Both first order and second order equations are handled. First and second order equations are written, respectively:

$$Y' = F(X,Y) \text{ and } Y'' = F(X,Y,Y')$$

The answer is given as a matrix on the stack:

1st column	=	X
2nd column	=	Y
3rd column	=	Y'

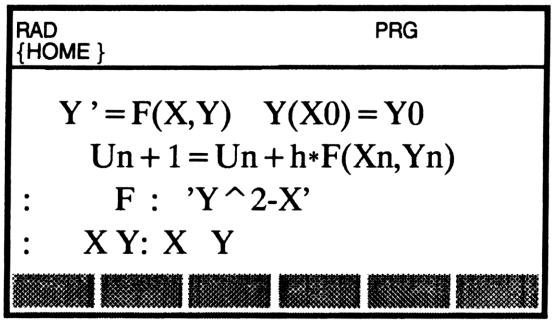
Approximations to Y-values are called U, U_1,U_2....U_n. Starting point is (X_0,Y_0), $U_0 = Y_0$ (1st order) and (X_0,Y_0,Y'_0), $U_0 = Y_0$ (second order). U_1, U_2,.... are calculated for X-values with the step length h, $X_1 = X_0 + h$, $X_2 = X_1 + h$.... $X_N = X_{N-1} + h$.

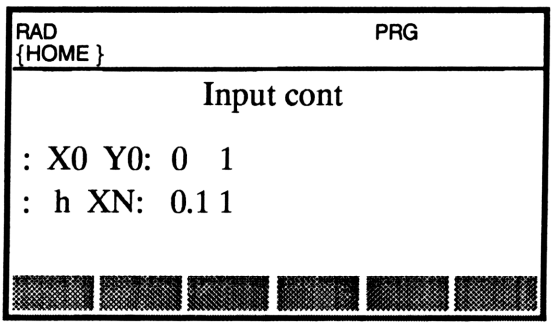
1st order equation, Euler's method

New U-values are calculated according to the formula:

$$U_{n+1} = U_n + h*F(X_n,U_n)$$

Interface:





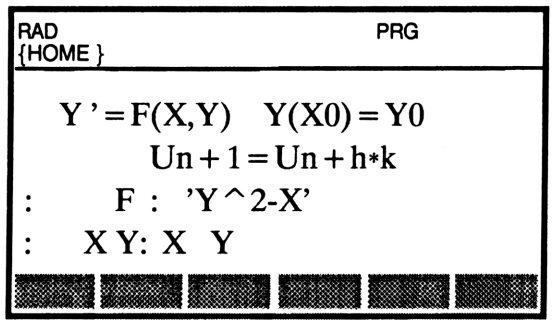
Equation	:	$Y' = Y^2-X$
Starting piont	:	$X_0 = 0, Y_0 = 1$
Ending point	:	$X = 1$
Step length	:	$h = 0.1$

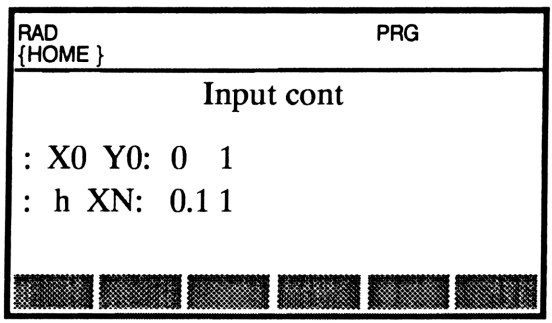
1st order RK2

RK generally means Runge-Kutta methods and RK2 is a development of Euler's method, the so called Heun's method. New U values are calculated according to the formula:

$$U_{n+1} = U_n + (k_1 + k_2)/2*h$$
$$k_1 = F(X_n, U_n), \quad k_2 = F(X_n + h, U_n + h*k_1)$$

Interface:





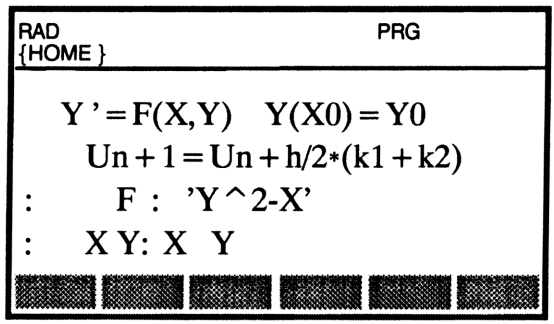
Equation	:	Y' = Y2-X
Starting point	:	X0 = 0, Y0 = 1
Ending point	:	X = 1
Step length	:	h = 0.1

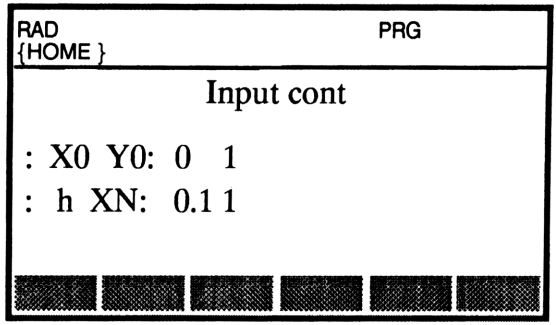
1st order RK4

This is a more accurate method than the two past methods and new U values are calculated according to the formula:

$$U_{n+1} = Un + h*(k_1 + 2*k_2 + 2*k_3 + k_4)/6 = U_n + h*k$$
$$k_1 = F(X_n, U_n),\ k_2 = F(X_n + h/2, U_n + h/2*k_1)$$
$$k_3 = F(X_n + h/2, U_n + h/2*k_2),$$
$$k_4 = F(X_n + h, U_n + h*k_3)$$

Interface:





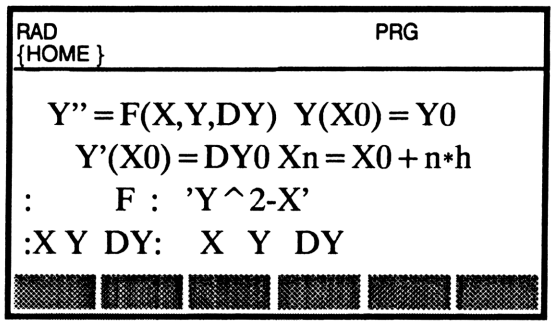
Equation	: $Y' = Y^2 - X$
Starting point :	$X_0 = 0, Y_0 = 1$
Ending point	: $X = 1$
Step length	: $h = 0.1$

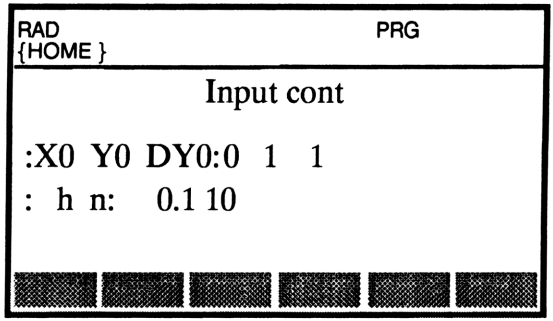
2nd order RK4

A 2nd order differential equation is solved by using a method similar to that of 1st order equations and is written Y'' = F(X,Y,Y'). Here both Y and Y' are approximated incrementally by U and DU:

$$U_{n+1} = U_n + h*(DU_n + (k_1 + k_2 + k_3)/6)$$
$$DU_{n+1} = DU_n + (k_1 + 2k_2 + 2k_3 + k_4)/6$$
$$k_1 = h*F(X_n, U_n, DU_n)$$
$$k_2 =$$
$$h*F(X_n + h/2, U_n + h/2*DU_n + h/8*k_1, DU_n + k_1/2)$$
$$k_3 =$$
$$h*F(X_n + h/2, U_n + h/2*DU_n + h/8*k_2, DU_n + k_2/2)$$
$$k_4 = h*F(X_n + h, U_n + h*DU_n + h/2*k_3, DU_n + k_3)$$

Interface:





Equation	:	$Y'' = Y^2 - X$
Starting point	:	$X_0 = 0,\ Y_0 = 1,\ DY_0 = 1$
Ending point	:	$X = 1\ (n = 10)$
Step length	:	$h = 0.1$

Plot

This routine plots data given in a matrix with X-values (column 1) and Y-values (column 2). A graph of the solution of a differential equation may be plotted.

Phase plot

Y ' values are plotted against Y-values as independent variable . Input data is a matrix where Y values are column 2 and Y ' values are column 3.

Interpolation

In this chapter subjects important in education are stressed. Subjects like divided differences and Newton polynomials are then important to deal with.

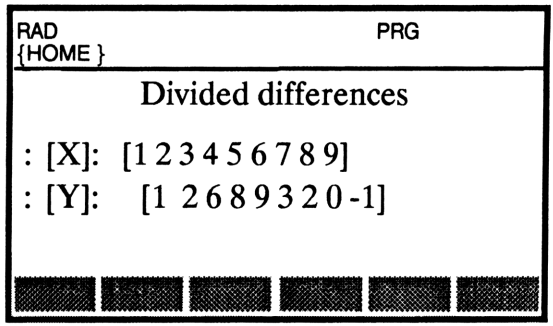
Interpolation means to estimate values between a set of given values. This means to find a function fitting the given points exactly, but giving approximated values at points between. This is an actual matter even if the expression determining the points is known, because in some situations (e.g integration) we need a more simple expression than the given function.

Divided differences

Divided differences are calculated on the basis of tabulated values for X and Y. If a known function is given, the Y-values may be found by using the menu option Function table.

$$
\begin{array}{llllll}
X_0 & Y_0 & & & & \\
 & & D_{22} & & & \\
X_1 & Y_1 & & D_{33} & & \\
 & & D_{32} & & D_{44} & \\
X_2 & Y_2 & & D_{43} & & \\
 & & D_{42} & & & \\
X_3 & Y_3 & & & &
\end{array}
$$

Interface:



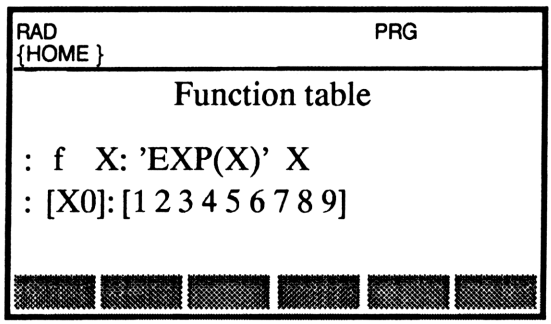
The output data is a matrix with 0-elements in open places in the difference table. Divided differences are defined by the formula:

$$D_{nn} = (D_{nn-1} - D_{n-1n-1})/(X_{n-1} - X_0),\ D_{11} = Y_0,\ D_{21} = Y_1$$

Function table

The function table for a given function may be found.

Interface:



Values of e^x for x = 1,2,3.....9 are calculated. The function values and the X-values are given in separate vectors to match the input form in other routines.

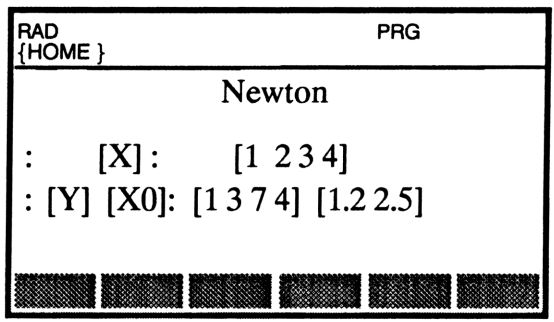
Newton's interpolation polynomial

The polynomials are defined by fitting a polynomial of degree n to a set of n + 1 points. The polynomial is well known as first degree polynomial (linear interpolation and trapezoidal rule in numerical integration) and as 2nd degree polynomial (Simpson's rule, numeric integration).

$$P_n(x) = Y_0 + (x-x_0)*D_{22} + (x-x_0)*(x-x_1)*D_{33} + ...$$

The routine calculates the polynomial for given points $[X_0]$ in the interval $[x_0\ x_n]$.

Interface:



Values for $x = 1.2$ and $x = 2.5$ by using a third degree polynomial through the given points [X,Y] are calculated.

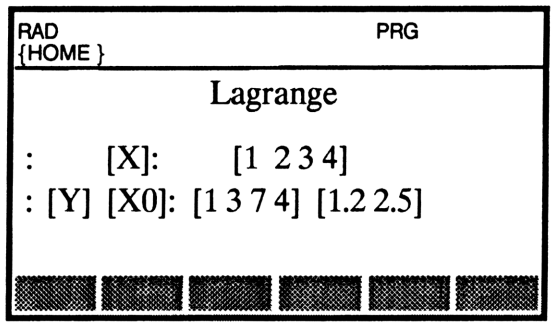
Lagrange polynomial

Lagrange polynomials are polynomials of degree n through $n + 1$ given points $[x_0...x_n]$. The formula for the polynomials:

$$L_n(x) = \sum_{k=1}^{n} l_k(x)/l_k(x_k) * Y_k$$

$$l_k(x) = (x-x_0)*...(x-x_{k-1})*(x-x_{k+1})*...(x-x_n)$$

Interface:



Values for $x = 1.2$ and $x = 2.5$ by using a third degree polynomial through the given points [X,Y] are calculated.

Cubic spline

Newton and Lagrange polynomials have the disadvantage that we may get big amplitude oscillations between the given

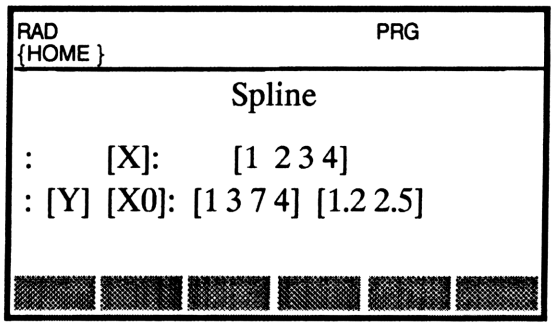
points. To get a better solution we may use third degree polynomials through two and two points (n different polynomials for n + 1 points). The polynomials have continuous derivative and second derivative at the given points to get a smooth curve. The polynomials are given as

$$S_k(x) = Y_k + a_k(x-x_k) + b_k*(x-x_k)^2 + c_k*(x-x_k)^3 \quad [x_k,x_{k+1}]$$

The constants a_k, b_k and c_k in the interval $[x_k, x_{k+1}]$ may be found by using the conditions:

$$Y_k = S_k(x_k) = S_{k-1}(x_k),\ S_{k-1}'(x_k) = S_k'(x_k)$$
$$S_{k-1}''(x_k) = S_k''(x_k),\ k = 1...n-1$$

Interface:



Values for $x = 1.2$ and $x = 2.5$ by using third degree polynomials through the given points [X,Y] are calculated. The polynomials for the intervals [1 2] and [2 3] are used respectively.

Richardson extrapolation

Richardson extrapolation is used to improve a given approximation. If we are looking at the problem of finding the derivative by using a 2nd degree Newton polynomial with two different steplengths, we find:

$$f'(x) \approx (f(x+h)-f(x-h))/2h = D(h,0)$$
$$f'(x) \approx (f(x+2h)-f(x-2h))/4h = D(2h,0)$$

By combining the two formulas we find
$f'(x) \approx (4D(h,0)-D(2h,0))/3$. This formula will give a better approximaton to the derivative and the process may be repeated. The general formula and tabular description will be:

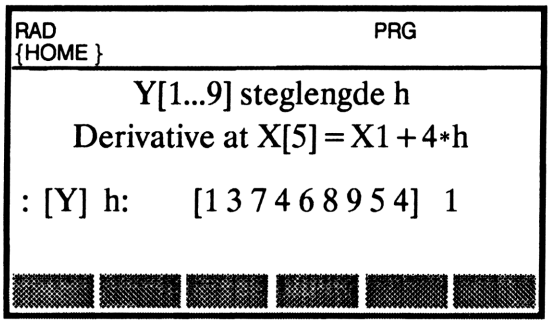
$$D(h,k) = \frac{4^k D(h,k-1)-D(2h,k-1)}{4^k-1}$$

D(h,0)			
	D(h/2,1)		
D(h/2,0)		D(h/4,2)	
	D(h/4,1)		D(h/8,3)
D(h/4,0)		D(h/8,2)	
	D(h/8,1)		
D(h/8,0)			

1st derivative

The basis of the extrapolation will be the formula indicated above. The extrapolation is repeated 3 times each time with the half steplength. We then need 9 points to start with.

Interface:



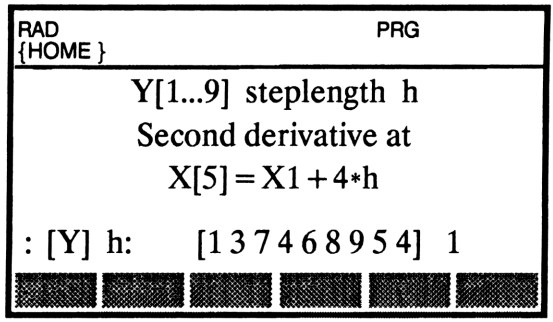
The output is given as three vectors which are the columns 2, 3, and 4 in the tabular description of D(h,k). The last value is the best approximation to f ' (X_5) where $Y_5 = 6$.

2nd derivative

The basis of the extrapolation is

$$f\ ''(x) \approx D2(h,0) = (f(x+2h) - 2f(x) + f(x-2h))/4h^2.$$

Interface:



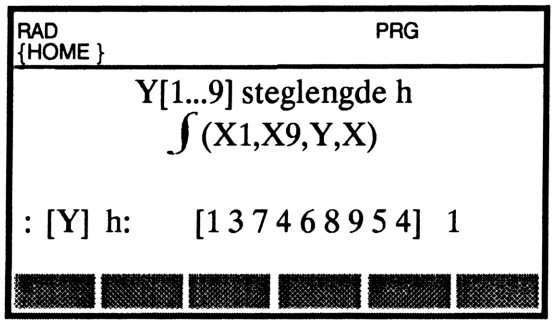
The output is given as three vectors which are the columns 2, 3, and 4 in the tabular description of D(h,k). The last value is the best approximation to f '' (X_5) where $Y_5 = 6$.

Romberg integration

We may use extrapolation on the trapezoidal rule in numeric integration as a basis, and this will give Romberg integration.

1st column	:	Trapezoidale rule
2nd column	:	Simpsons method
3rd column	:	Bools rule (2nd order Romberg)
4th column	:	3rd order Romberg

Interface:



We integrate over the points $Y_1 = 1$ to $Y_9 = 4$ and the output is the different columns in the tabular description for D(h,k), first vector is the trapezoidal rule, second vector is Simpsons formula a.s.o. The last value is 3rd order Romberg and this is the most accurate answer.

9

Solved problems

Linear algebra

In our first problem we will look at finding the determinant, inverse, diagonalization, eigenvalues and eigenvectors of the same matrix without typing the matrix several times. We then have to use the submenu MATR (and not the main menu). The matrix will be

$$\mathbf{M} = \begin{bmatrix} 2 & 4 & 7 \\ 4 & 1 & 1 \\ 6 & 1 & 9 \end{bmatrix}$$

Example 9.1
Find the determinant of matrix **M**. We type the matrix on stack level 1 by going right ahead writing [[2 4 7][4 1 1][6 1 9]] or by using the MatrixWriter (symbolic or numeric). We use the submenu MATR/DET and the answer is -118. The matrix will automaticaly be stored as MATR and also duplica-

ted on the stack. If you loose stack level 1 (the matrix MATR) for some reason, you may simply type MATR and push ENTER to put it back on the stack again.

Example 9.2
Find the inverse of the matrix **M** now laying on the stack. By pushing INV we find the inverse matrix $\mathbf{M}^{-1}$. The form is symbolic {{}} in spite of the matrix beeing numeric. To get a numeric matrix from a symbolic form, just push SYM→.

Example 9.3
Diagonalize the matrix **M**. Push DIAG (NEXT). The diagonalized matrix **D** is given on stack level 2 and the matrix **K** on stack level 1. $\mathbf{D} = \mathbf{K}^{-1} * \mathbf{M} * \mathbf{K}$.

Example 9.4
Find the eigenvalues of the matrix **M**. This problem has in fact been solved in the last example, but we will now use the menu option EGV. Pick up the matrix and push EGV. The answer will be {-2.65,14.40, 3.25}.

Example 9.5
In the last example we will calculate the eigenvectors. We proceed in the same manner as in the last examples and the

answer will be the eignevalues and a matrix where the corresponding eigenvectors are given (one of indefinitely many sets of eigenvectors).

> *Rem: Intermediate answers are not possible in the submenu MATR.*

Example 9.6
Symbolic matrices. In BAS MAII the SymbolicMatrix writer is implemented. Symbolic matrices may then be handled. We may calculate the same problems as in examples 9.1-9.5, exept the calulation of eigenvectors and diagonalization where the matrix has to be numeric. We will manipulate the matrix

$$\mathbf{A} = \begin{bmatrix} x & 0 & 2 \\ 3 & 0 & x-1 \\ 1 & 1 & 0 \end{bmatrix}$$

A symbolic matrix is written {{x 0 2}{3 0 'x-1'}{1 1 0}} and is put on stack level 1. The use of the programs is as in the last examples. The determinant is given as 6-(-1+x)*x. This expression may be simplified by using SYMBOLIC/EXPA/COLCT on the calculator. The answer is then given as $6-x^2+x$.

Example 9.7

The program Linear diffequations under the menu Eigenvalue problems is able to solve a set of linear, homogenous diffequations. If the eqautions are not homogenous, they may be made so by a proper substitution of new variables. Let us look at the system

$$dx/dt = 5x + 8y + 1$$
$$dy/dt = -6x-9y + 2$$

First we solve the stationary state (constant in time) by using $dx/dt = 0$, $dy/dt = 0$. This gives a system of ordinary equations.

$$5x + 8y + 1 = 0$$
$$-6x-9y + 2 = 0$$

The equations must be written

$$5x + 8y = -1$$
$$-6x-9y = -2$$

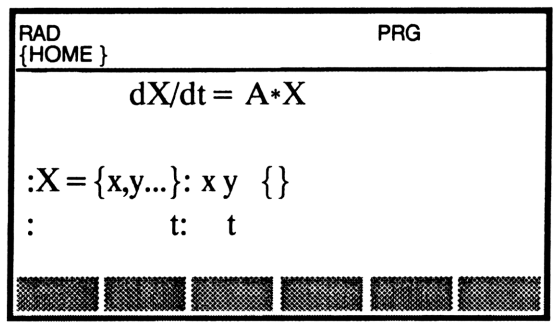
This may be solved by using the menu option Linear equations (UPDIR/UPDIR/SHIFT RIGHT ↑ /ENTER). But in a simple case like this we will use the intrinsic HP 48 matrix calculations directly.

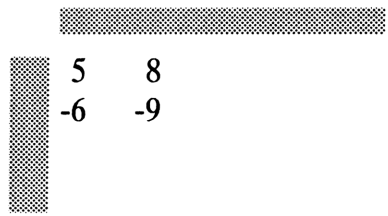
When using →STK in the submenu the calculator HALTs and may be used independently of BAS MAII. The right side [-1 -2] is put on stack level 2 and the matrix [[5 8][-6 -9]] is put on stack level 1. Then use division and x and y is found to be x(stationary) = 8.33, y(stationary) = -5.33. If we now use the new variables X = x-8.33 and Y = y + 5.33 we transform the inhomogenous system into

$$dX/dt = 5X + 8Y$$
$$dY/dt = -6X-9Y$$

This is a homogenous system solved directly by using MAII.

Interface:





The solution is {'C1e^{-t}-C2e^{-3t}, '-0.75C1e^{-t} + C2e-^{3t},} which means that X = C1e^{-t}-C2e^{-3t} and Y = -0.75C1e^{-t} + C2e^{-3t}. The full solution is now found by transforming back to the old variables which gives x = X + 8.33 and y = Y-5.33.

Example 9.8

In this example we will test whether a set of vectors in 3D are linearly independent, if they are orthogonal and if not, use the Gram-Schmidt process to make them so. The new vectors will then make an orthogonal basis in the vector space 3D. Vectors are linearly independent if none of them can be expressed as a linear combination of the others and orthogonal if they are linearly independent and mutualy orthogonal (90^{o} angles in 3D). If they are linearly independent they may form a basis.

We will look at the set of vectors {[2 4 5] [1 3 2] [1 6 3]}. First we test for linear independency and we may then test the rank of the matrix

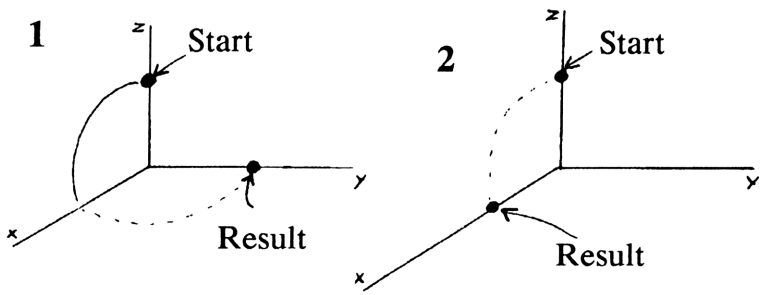
$$\begin{bmatrix} 2 & 4 & 5 \\ 1 & 3 & 2 \\ 1 & 6 & 3 \end{bmatrix}$$

We use the menu option MATR/RANK (or BASE?) putting the matrix

[[2 4 5][1 3 2][1 6 3]] on stack level 1. The rank is 3 which means that the vectors are linearly independent.

The matrix is now on stack level 1 and by pushing ORT? you will find whether it's orthogonal or not. 0 means not. It's not and then, since the set of vectors is linearly independent, they may be orthogonalized by the Gram-Schmidt process. The matrix is still on level 1 and you simply push ORTH. Finally you push ORTN to get a normed basis (lenght of vectors equal to 1). The result of the Gram-Schmidt process and norming will be {{0.30 0.60 0.75}{-0.06 0.79 -0.61}{-0.95 0.14 0.27}} which is transformed into an ordinary numeric matrix by pushing SYM→.

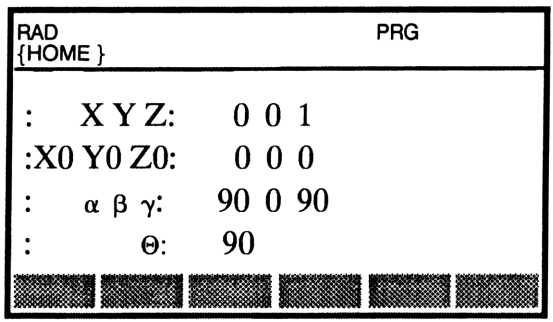
Example 9.9
In this example we will look at the rotation of points in 3D.



If we first (1) rotate (in 3D) 90^o about the y-axis, and then 90^o about the z-axis this is not the same as (2) rotateing 90^o about the z-axis and then 90^o about the y-axis. The point is (X Y Z) = (0 0 1).

The y-axis is specified as (x0 y0 z0) = (0 0 0) with angles $\alpha = 90^o$ (relative x-axis) $\beta = 0^o$ (relative y-axis) and $\gamma = 90^o$ (relative z-axis). For the z-axis the the angles are $\alpha = 90^o$, $\beta = 90^o$ and $\gamma = 0^o$.

Interface rotation about y-axis:

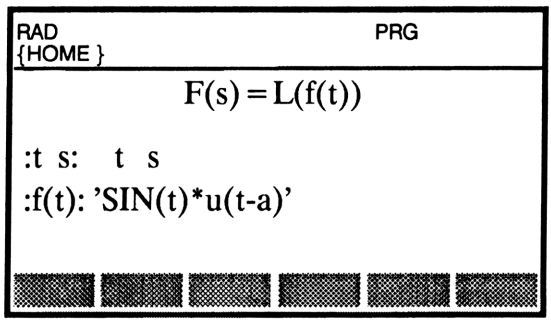


The point is now transformed into (1 0 0) and the second rotation about the z-axis then gives (0 1 0). The (2) rotations give first the same point (0 0 1) and then (1 0 0), a different result.

Laplace transformes

Example 9.10
Find the Laplace transform of f(t) = Sint*u(t-a), a is constant.
We use the menu option LAPLACE TRANSFORM/Laplace transform.

Interface:



The result is put on the stack and by pushing STK→ the answer is given as

$$F(s) = (1/(s^2 + 1)*COS(a) + s/(s^2 + 1)*SIN(a))*e^{-as}$$

Example 9.11
Find the Laplace tramsform of $f(t) = a + bt + ct^2$. We proceed as in the last example and the answer will be

$$F(s) = 2c/s^3 + b/s^2 + a/s$$

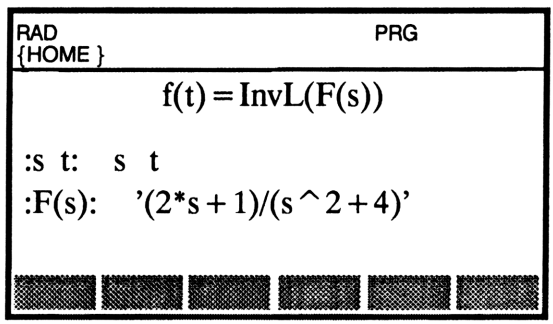
Example 9.12
Find the Laplace transform of $f(t) = COS(\omega t + \Theta)$. Answer:

$$F(s) = COS(\Theta)*(s/(\omega^2 + s^2) - SIN(\Theta)*\omega/(\omega^2 + s^2)$$

Example 9.13
Find the Laplace transform of $f(t) = SIN(t)*e^t + t*COS(t)$.

$$F(s) = -1/(1 + s^2) - 2s^2/(1 + s^2)^2 + 1/(1 + (s-1)^2)$$

Example 9.14
Find the Inverse Laplace transform of $F(s) = (2s + 1)/(s^2 + 4)$
Interface:



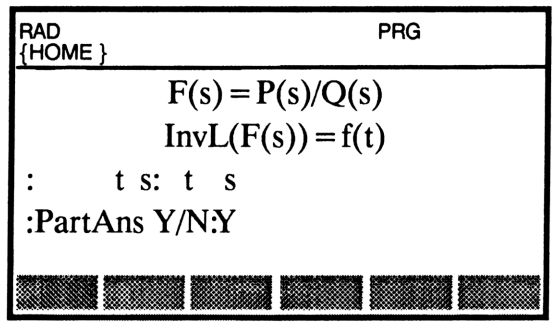
We use the menu option Inverse L-transform. The answer will be

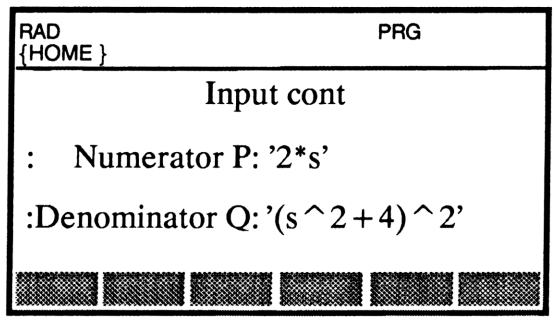
$$f(t) = 0.50*SIN(2t) + 2*COS(2t)$$

Example 9.15

Find the Inverse Laplace transform of $F(s) = 2s/(s^2+4)^2$. If we try Inverse L-transform we will be suggested to try partial fractions and we try.

Interface:





Since we have chosen PartAns Y we will get the splitting up of the function into partial fractions:

Splitting gives:

$$2s/(s^2+4)^2 =$$
$$As/(s^2+4)^2 + B/(s^2+4)^2 +$$
$$Cs/(s^2+4) + D/(s^2+4) \qquad \text{(Next page)}$$

Coefficients:

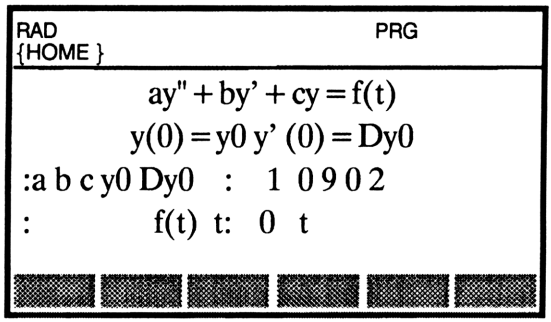
$$A = 2$$
$$B = 0$$
$$C = 0$$
$$D = 0 \qquad \text{(Next page)}$$

Solution:

$$1/2*SIN(2t)*t$$

Example 9.16
Solve the initial value problem $y'' + 9y = 0\ y(0) = 0,\ y'(0) = 2$.

Interface:



In the first place we find $Y(s) = 2/(s^2 + 9)$. To transform this back to the time domain we use Inverse Laplace and import the function Y(s) from the stack by echoing (EDIT/↑STK/ECHO/ENTER) Be sure that the cursor is on the right place on the input screen. The answer will be

$$y(t) = 0.67*SIN(3t)$$

Example 9.17
When solving more complicated initial value problems we cannot import Y(s) directly from the stack, we have to make the expression a litle bit "nicer".

Solve the problem $y'' + 9y = u(t)$, $y(0) = 0$, $y'(0) = 2$. This gives $Y(s) = (1/s + 2)/(s^2 + 9)$ which has to be written $(1 + 2s)/((s^2 + 9)s)$ (broken fractions are not possible). You may then put numerator $(1 + 2s)$ and denominator $(s(s^2 + 9))$ of this expression on the stack and import then as in the last example into the routine Inverse L p.fractions. The solution will be

$$y(t) = -1/9\text{COS}(3t) + 2/3\text{SIN}(3t) + 1/9$$

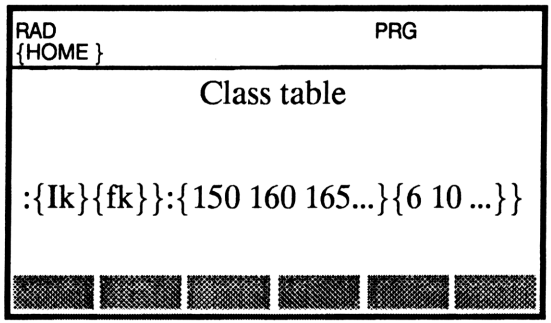
Statistics

Example 9.18

This example will show how to fit data (class table) to the normal distribution. From a population of 100 individuals the following data are available:

Height	Population
[150 160]	6
[160 165]	10
[165 170]	18
[170 175]	29
[175 180]	21
[180 185]	10
[185 190]	5
[190 195]	1

The data have to be stored as a class table using the menu option Class table (STATISTICS/Fitting) or the routine KΣDAT under the menu STAT.

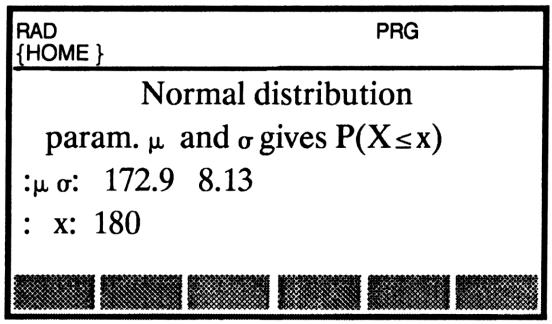


The input looks like:
{{150 160 165 170 175 180 185 190 195}{6 10 18 29 21 10 5 2}}

No more input is necessary and the parameters μ and σ are calculated (mean and standard deviation). $\mu = 172.9$ $\sigma = 8.13$.

Example 9.19
We may now find the probability that a member of the population in the last example has a height less then or equal to 180. Under the menu STAT you will find NORM.

Interface:



The probability to find someone with height less than or equal to 180 is 81% (0.81).

Example 9.20
We will now look at a histogram and a cummulative frequency polygon of the data in the last examples. These menu options you can find under the description of samples. No further input data are necessary since the class table already is stored as KΣDAT. Push the ATTN (ON) key to quit the graphics.

Example 9.21
Transforming a class table into a discrete frequency table. The representative value is the mean in each interval. You have to use the menu option KΣDAT→ΣDAT under descrip-

tion of samples. Since a class table already is stored (last examples) no further input is necessary.

Example 9.22
From the discrete table calculated in the last example we may now calculate the mean and the standard deviation. These values then have to match the parameters of the fitted normal distribution properly. The answers are 172.8 and 8.20, which is about the same values as the parameters. Use F for frequency table.

Example 9.23
We will now calculate a cummulative table giving relative frequencies, cummulative frequencies and cummulative relative frequencies. The routine is based on the table ΣDAT stored from example 9.21. The actual values are the first column, the frequencies the second column, cummulative frequencies (3rd column), relative frequencies (4th column) and cummulative relative frequencies (5th column). Pushing ENTER will put the matrix on the stack.

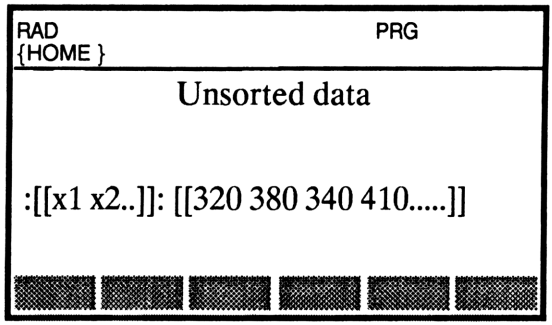
Example 9.24
In practice we often meet the situation that we have a big amount of unsorted data. The menu Description of samples/ Unsorted data will give us the possibility to put in an unsorted amount of data and then get a frequency table ΣDAT. The

routine counts the number of occurences of the several values.. Suppose we have the following amount of data, 100 values of the splitting tensile strength of concrete sylinders ($lb/in.^2$).

320	380	340	410	380	340	360	350	320	370
350	340	350	360	370	350	380	370	300	420
370	390	390	440	330	390	330	360	400	370
320	350	360	340	340	350	350	390	380	340
400	360	350	390	400	350	360	340	370	420
420	400	350	370	330	320	390	380	400	370
390	330	360	380	350	330	360	300	360	360
360	390	350	370	370	350	390	370	370	340
370	400	360	350	380	380	360	340	330	370
340	360	390	400	370	410	360	400	340	360

If we measure in practice this is what we get. We now want to sort the data and count the occurence of the different values (frequency table). This is done by the menu option Description of samples/Unsorted data and the result is stored as ΣDAT.

Interface:



The values are in column 1 and the absolute frequencies in column 2.

Example 9.25
We may now use MAII to calculate the mean and standard deviation. We use the menu Description of samples/ΣDAT x and s. Use F for frequency table. The mean is 364.7 and the st.dev. 26.83.

Example 9.26
Goodness of fit. We will now use the kji-square distribution test to find out the goodness of fit to the normal distribution. First we have to construct a class table based on the frequency table which looks like (example 9.24):

300	2
320	4
330	6
340	11
350	14
360	16
370	15
380	8
390	10
400	8
410	2
420	3
440	1

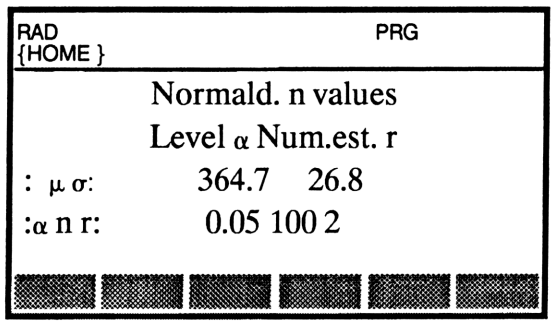
We have to decide the limits of the classes and we have to use f.ex. -10000 as the lower limit and 10000 as the upper. We want minimum 5 values in each class and try a class width of 10.

[-10000 325]	6
[325 335]	6
[335 345]	11
[345 355]	14
[355 365]	16
[365 375]	15
[375 385]	8
[385 395]	10

[395 405]	8
[405 10000]	6

We now use the menu option class table to construct KΣDAT. To test the hypothesis for the normal distribution we use the menu Fitting/Hypoth. normald. We use the estimated values for mean and st.dev. from example 9.25, 364.7 and 26.8.

Interface:



A significance level of 5% is selected and the hypothesis of a normal distribution is not rejected ("good margin", $2.83 < 14.07$).

Example 9.27

We may now fit a normal distribution to the data for concrete sylinders by using Fitting/Normal distribution. The KΣDAT is stored and no other input is necessary. We find

$\sigma = 26.1$ and $\mu = 364.8$, in good agreement with the estimated mean and standard deviation.

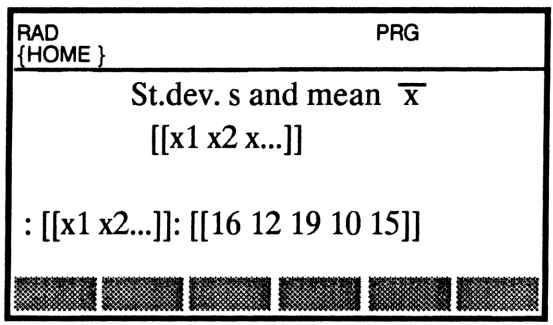
Example 9.28

We may find the probability that a cylinder has a strength between 350 and 360, P(360)-P(350). We calculate the two probabilities once at a time and get the answer 0.43-0.29 = 0.14 or 14%.

Example 9.29

Find a 95% confidence interval for the mean using the samle [[16 12 19 10 15]] when standard deviation is given as 2.5. We first find the mean value of the sample using the menu option Conf.int normald./Sample x,s and n.

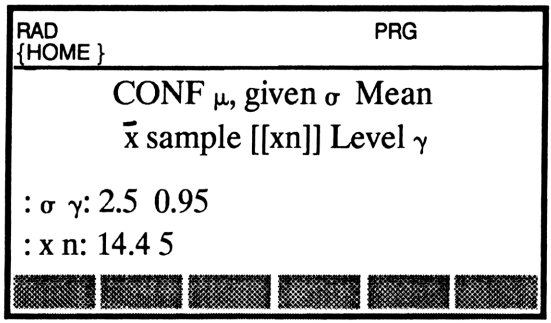
Interface:



We find the mean value x = 14.40.

The value will be input in the routine for determing the confidence interval. The number of values in the sample is 5 and the st.dev. is given as 2.5.

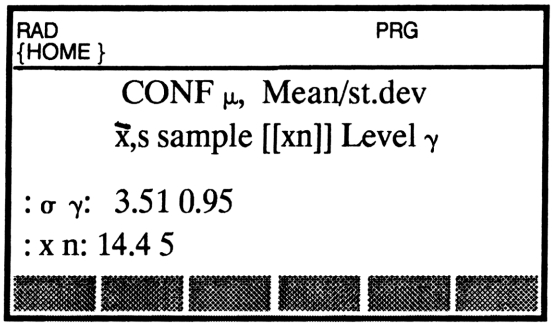
Interface:



The interval is [12.21 16.59] which means that the probability of finding a value within this interval is 95%.

Example 9.30
Find a 95% confidence interval for the mean using the samle [[16 12 19 10 15]]. Here the st.dev. is unknown but we use the mean (14.4) and st.dev (3.51) calculated from the sample.

Interface:

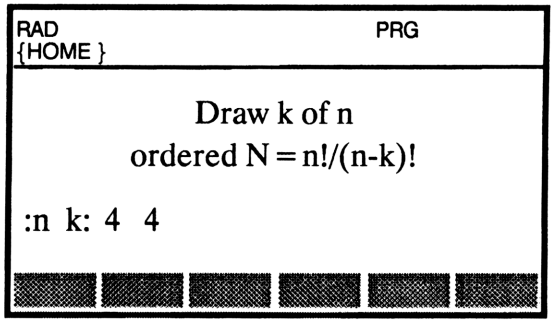


The interval is now [10.04 18.76], wider because of the greater standard deviation.

Probability

Example 9.31
Find all combinations of the numbers 1,2,3,4 taken all at a time. This will be ordered combinations without replacement, and we choose 4 elements at a time among 4. This is the same as the number of permutations of 4 elements (4!).

Interface:

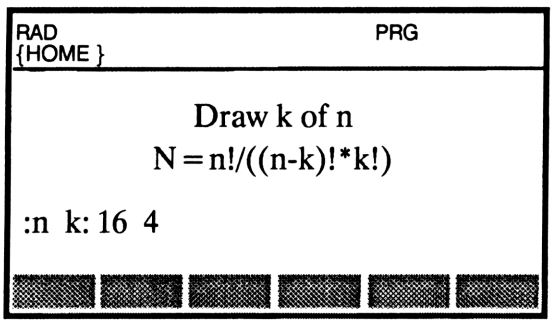


The answer is 24.

Example 9.32

In how many ways can a commitee of four be chosen from 16 persons? This will be non-ordered combinations without replacement. (A, B, C, D) is the same commitee as (B,C,D,A) and one particular person cannot be chosen more than once (without replacement).

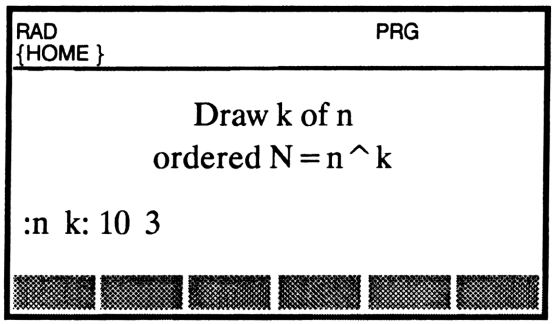
Interface:



The answer will be 1820.

Example 9.33
How many different license plates showing 2 letters and 3 digits are possible (alphabet 26 letters)? This will be the product of the possible combinations of 3 numbers out of 10 and 2 letters out of 26. Since the same letter or the same digit may be used several times and the order is important this will be ordered combinations with replacement.

Interface:

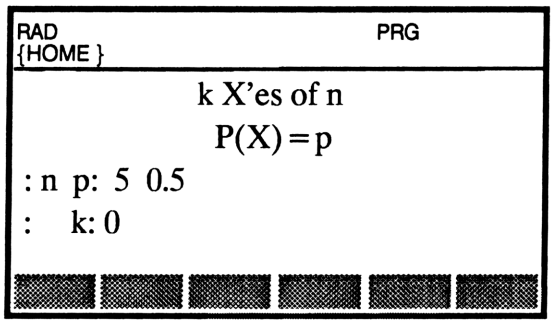


The answer will be $10^3 * 26^2 = 676000$.

Example 9.34
Determine the number of bridge hands (13 out of 52 cards). This will be without replacement and non-ordered.The answer will be 635,013,559,600.

Example 9.35
5 fair coins are tossed simultaneously and this may be looked at as independent trials one by one. What is the probability that none of them is a head? The binomial distribution may be used.

Interface:



The probability of head is 0.5 and the number of heads is 0. The answer will be 0.031 (3.1%).

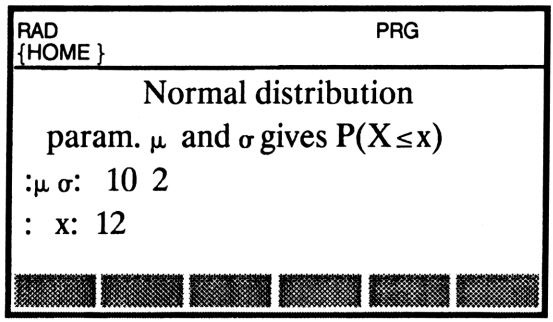
Example 9.36
The probability of hitting a target is 25%, what is the probability of hitting the target at least once in 4 trials? The trials are independent and we use the binomial distribution. The

answer is 1-P(0), and we find the P(0) to be 0.316.
1-P(0) = 0.684 (68.4%).

Example 9.37
A variable X is normal with mean 10 and variance 4. Find $P(X > 12)$, $P(9 \leq X \leq 13)$. The routine for normal distribution needs the st.dev. which is the square root of the variance. The st.dev. then is 2.

We find $P(X \leq 12)$ and calculate $P(X > 12)$ by $1\text{-}P(X \leq 12)$.

Interface:

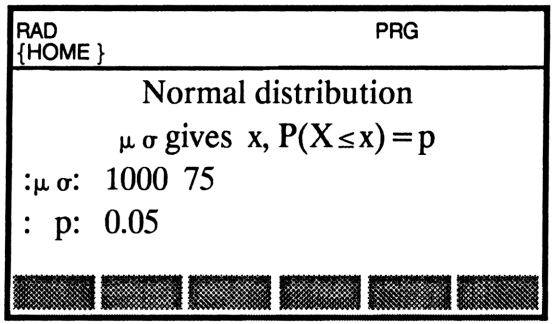


$P(X \leq 12) = 0.841$. $1\text{-}P(X \leq 12) = P(X > 12) = 0.159$. We find $P(9 < X < 13) = P(X \leq 13)\text{-}P(X \leq 9) = 0.93\text{-}0.31 = 0.62$.

Example 9.38
A manufacturer produces airmail envelopes whose weight is normal with st.dev.0.05 and mean 1.95. The envelopes are sold in lots of 1000. How many envelopes in a lot will be heavier than 2.0? We calculate $P(X>2)$ by $1\text{-}P(X\leq 2)=0.159$. The number will then be $0.159*1000=159$ (about 160).

Example 9.39
The breaking strength X of a certain type of plastic block is normaly distributed with mean 1000 and st.dev. 75. What is the maximum load such that we can expect no more than 5% of the blocks to break? We use the inverse normal distribution with $p=0.05$. The inverse normal distribution may be found under STATISTICS or in the submenu STAT.

Interface:



The answer is 876.

Linear programming

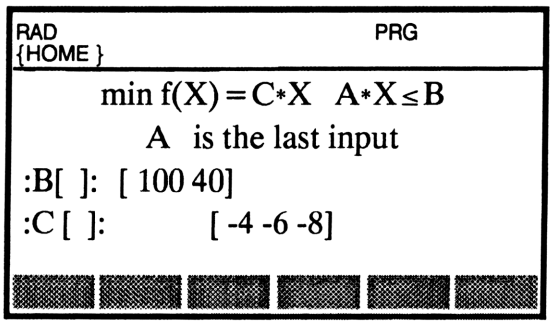
Example 9.40

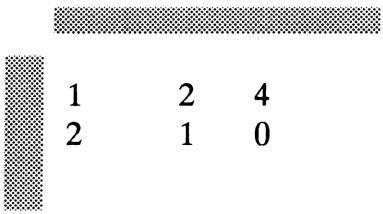
A factory produces 3 kinds gaskets, X1, X2 and X3 with net profits \$4,6,8 respectively. Maximize the totale daily profit $P(x_1,x_2,x_3) = 4x_1 + 6x_2 + 8x_3$ under the constraints

$$x_1 + 2x_2 + x_3 \leq 100 \text{ and } 2x_1 + x_2 \leq 40$$

We minimize the function $-P(x_1,x_2,x_3)$.

Interface:





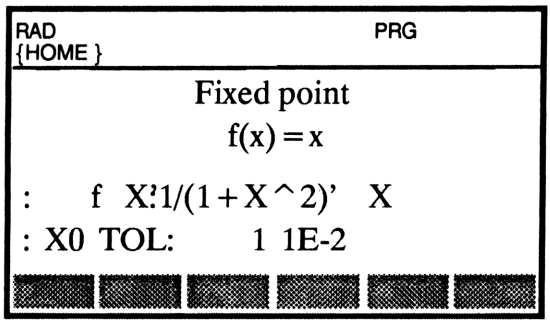
The answer is -280, $x_1 = 0$, $x_2 = 40$ and $x_3 = 5$. The maximum then is 280.

Numerical methods

Example 9.41

Solve the equation $x^3 + x - 1 = 0$ by fixed point iteration. We write the equation in the form $x = g(x) = 1/(1 + x^2)$. We have in mind that abs(g'(x)) < 1 in the neighbourhood of the solution. To find a proper starting point x0 you may graph the function. Use menu NUMERICAL METHODS/Zeros.

Interface:

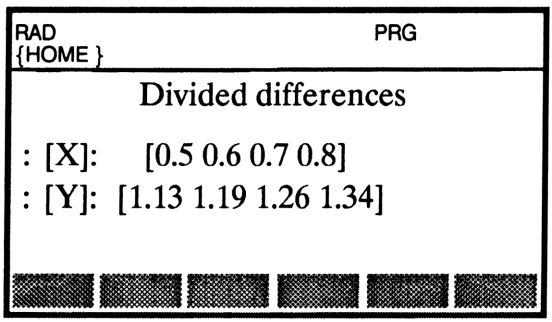


Equation	:	$x - 1/(1 + x^2) = 0$
Solution	:	$x_{12} = 0.68$
Starting point :		1
Tolerance	:	0.01

Example 9.42
Calculate a divided difference table for the function $Cosh(x) = (e^{x} + e^{-x})/2$ for x = [0.5 0.6 0.7 0.8]. Use the menu Interpolation/Divided diff.table. First you have to find the function values using function table in the same menu.

The function values are laying on the stack together with the x-values and both may be imported to the routine diff.table by echoing.

Interface:

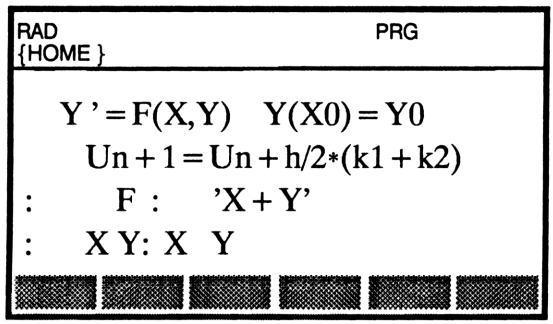


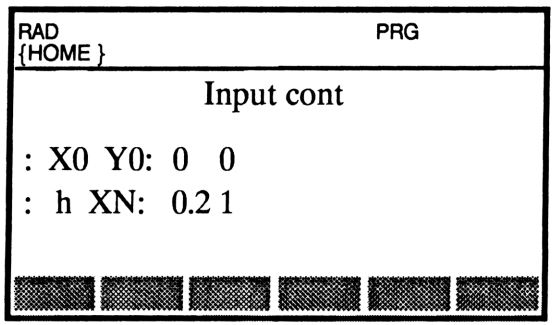
The ouput is a matrix filled with blanks to "simulate" the typical triangular form of the table.

Example 9.43

Solve the initial value problem $y' = x + y$ $y(0) = 0$ by RK4. This problem has the exact solution e^x-x-1 so the error is known. We choose step length $h = 0.2$.

Interface:





The answer is given in a matrix where the first column are the x-values, the second column the y-values and the third column the values of y'.

0	0	0
0.2	0.02	0.22
0.4	0.09	0.49
0.6	0.22	0.82
0.8	0.43	1.23
1.0	0.72	1.72

The matrix may be seen in the matrix writer by pushing arrow key down (VIEW-key).

Example 9.44
We want to make a plot and a phase plot. We duplicate the matrix on the stack (ENTER) and no further input is necessary.

HP 48 SX / GX SmartROM™

User's Guide

INTRODUCTION

FOREWORD

Thank you for the purchase of the SmartROM. You bought a quality software that will greatily help in your work, hobby or study!

If you are new to the HP48's world, we suggest to read carefully the manual before diving into the deep waters of the SmartROM.

The well tempered HP48

The SmartROM is the most comprehensive library of commands available for the HP48. Its goal is to minimize programmer's endeavours by providing an extensive set of powerful utilities which significantly shorten program development.

For those who need an in-depth knowledge of the SmartROM, an Hidden Commands Reference manual is provided on the supplied disc.

The SmartROM has been conceived with the intent of setting some periods in the big ocean of software. This means that sometimes you could find somewhere a specific program doing the same things slighlty faster or simply better. The power of the SmartROM lies in its high integration, consistency and support. The SmartROM is not just a collection of RPL commands. It is the ideal platform for any powerful program. Just think that the SmartROM contains about 180 hidden commands fully documented in an additional manual, called *Hidden Commands Reference*. You can find this manual on the supplied disc under the name \DOCS\TECHREF.TXT.

If you wish to be kept informed on future releases of the SmartROM, we suggest you either to fill the enclosed form and send it by mail to ForCALC or contact us directly via E-mail at the address given in Appendix A.

All the information contained herein, when a specific source is not specified, come from our own experiments on the calculator and are proprietary. Appendix C, containing Saturn Assembly Language description, is based on the description given in the HP-71 IDS Vol. I-II by Hewlett-Packard Company.

OVERVIEW

Chapter 1: SmartROM Commands Reference

The reference section of the manual covering the entire set of commands listed in alphabetical order. For each command a stack diagram is given and remarkable information as well. Examples are collected at the end of each paragraph.

Chapter 2: Omnibus user's guide

The chapter describing every aspect of the spreadsheet environment. It includes a tutorial section, programming guidelines, dozens of practical examples and pictures. If the package is not furnished with the spreadsheet library, this chapter is missing.

Appendix A: Warranty, Service and Support

Refer to this section if you encounter problems using the SmartROM.

Appendix B: Objects

HP-48 objects structure is explained in detail.

Appendix C: The Saturn Microprocessor

A description of the CPU and its machine language instruction set is given.

Appendix E: Error Messages

Contains the complete list of the errors generated by the SmartROM along with possible causes.

Appendix H: Hidden Commands

Contains the exhaustive list of SmartROM hidden commands of version 1:C, subdivided in logic categories.

LIST OF SMARTROM PROGRAMMABLE COMMANDS

This is the complete list of RPL commands subdivided in logic categories. Some commands appear in several categories being polymorphic functions.

Stack Manipulation

AAB	BAA
BAB	BBA
BCAC	BCDA
C2M	CAB
CBA	MARK
NIP	RDOWN
RDROP	RDUP
RUP	SHIFT
XLVLS	

Argument Checking

CHL?	CHSET?
CHST?	

List Manipulation

FIND	L2M
LOP1	LOPN
LVOP	NULL
REPLACE	REV
RPT	SPLIT
SRDIFF	SRGE
SRGT	SRLE
SRLT	

String Manipulation

→Char	FIND
LINES→	→LINES
LOC	MEMBER
NULL	PARSE
REPLACE	ROWCOL
REV	RPT
SPAN	SPLIT

Meta Object Manipulation

COPY	DELETE
LINES→	→LINES
META	METOP
MOVE	MREV
NDUPN	PKMETA
PRG→	→PRG
SRT	SRTD

Type Management

→B	→Char
EXT→	→EXT
FALSE	MSBIT
PRG→	→PRG
→R	→SYS
→TorF	→Xlib
TRUE	

Symbolic Matrix Handling

ADD	ADDCON
CMPL	CONFORM?
CONST	DELCOL
DELROW	DETERM
DIMS	EQUAL?
FACTOR	IDNT
MATWRT	MGET

MPUT	MSYMB?
MULT	SQUARE?
SUBT	SYMBMAT→
→SYMBMAT	TRNSP

Program Editing

FIND	PRG→
→PRG	REPLACE

I/O

KEIWAIT

Utility

CSTMENU	RPT

ROM Revision

ROMV	VER$

LIST OF APPLICATIONS

Applications listed below are stored on the diskette along with their documentation.

CALENDAR	FFT	LWC
UPC	MATMENU	PRSYMB
PROBJ	PRTHREAD	L→TH
TH→L	SHRINK	UPTRIM
PIE	BARPLOT2	INVRT
→FONT	*alfa*ORDER	POPDIR
CGINDIR	REPLP	CGXLIB
XREF		

ODD BRINGSLID:

SMARTROM

TYPESETTING CONVENTIONS

DISPLAY

Used to represent text as it appear on the display or anything you must type.

Italics

Italics are used to introduce a new term or emphasize words or concepts.

[KEY]

Shift keys are represented by **[Blue]**, **[Gold]** and **[Alpha]**.

LABEL

This typeface represents a menu label.

TYPOGRAPHIC CONVENTIONS

The special set of characters implemented in the 48 requires a special treatment in order to avoid misunderstandings.

#b	Binary integer (also called hex string) #12345h, #01101b, etc.
External	System Address
<h>	System Binary
n, d, i, j, k, l,	Real numbers
TRUE	System boolean (External)
FALSE	System boolean (External)
Complex	Complex number
'NAME'	A global name
'name'	A local name
obj	Any object
Symb	Symbolic Expressions, real and complex numbers, units, gloabal or local names
prg	RPL program
"..."	User RPL Program

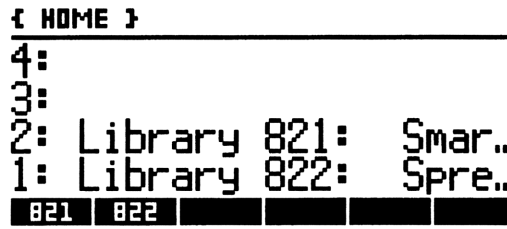
str	String of characters
Char	Character object
[...]	Array (Vector)
[[...]...[...]]	Array (Matrix).
{...} or List	List.
{{$Symb_{11}$ ... $Symb_{1n}$} **{$Symb_{m1}$ ... $Symb_{mn}$}}**	Symbolic Matrix.
obj_1...obj_n n	Any Meta-object.
str_1...$strn$ n	String Meta-object.
XLIB LID Num	External Library name.

AUTOMATIC INSTALLATION OF A ROM CARD

The installation of the SmartROM is straightforward. Turn the HP48 off and carefully follow the instructions written in chapter 34 of the HP48 Owner's manual volume II. If the HP48 shows INVALID CARD DATA, the card requires service.

AUTOMATIC INSTALLATION OF A RAM BASED PACKAGE

1. Installing a RAM based version of the SmartROM requires a few minutes.Be sure to have at least 32K bytes of free RAM.You need a PC and a transfer program running the Kermit protocol.
2. Start the Kermit program on the PC and change the current drive to A: or B: whichever applies to your case. Ensure you have the same baud rate (usually 9600) on both sides (PC and HP48).
3. Put the HP48 in server mode by pressing **[blue][PRG].**
4. On the PC type: TAKE INSTALL **[Enter]**. The transfer should begin.
5. Recall the contents of variables smart and spread to the stack and purge both variables, by pressing appropriate menu keys. You should see the following items on the first two levels of the stack:



6. Store the libraries in a port where you have enough free space (32K). To accomplish the store operation, verify that both libraries are on the first two levels of the stack and press twice **[n] [STO]**, where n is 0,1 or 2 depending on the port where you want to put the libraries into.
7. Turn the HP48 off and on again.

Note that the configuration software automatically attaches the library (either Smart or Spread) to the library chain of the HOME directory. If you want to manually attach the library in a specified subdirectory, you must follow the procedure described in Chapter 34 of the HP48 Owner's Manual. However, in the case of a warm start, (i.e. pressing **[ON] [C]** the library will be attached to the HOME directory again.

8. The SmartROM is ready for use.

VERIFYING THE INSTALLATION

You can quickly verify the installation of the SmartROM by following the procedure described later on.

This procedure applies to version 1:C only.

1. Manually type the command ROMV SmartROM 's logo should appear in the display. If the logo is displayed on a black background or with sharper lines, don't worry about it, it is just a random drawing of the SmartROM logo

2. If the logo does not appear, the library has not been installed because you forgot to turn the HP48 off and on again, or you did not install the library in a port.

COMMANDS REFERENCE

This section explains in detail each command of the library, giving its stack representation, remarkable information, examples and names of applications containing the command. Topics, like CPU structure, hidden commands and other highly specific arguments have been collected in the Appendices.

All the commands of the library are accessible through SmartROM's custom menu, which can be installed executing the command CSTMENU.

All the commands contained in this section belong to Library 821 (Smart) except MATWRT which belongs to Library 822 (Symatw).

Named commands have their XLIB number in range 0-95. Remaining functions, called 'hidden functions', have no associated text. To learn more about hidden functions, please refer to the manual supplied on the disk under the name \DOCS\TECHREF.TXT. SmartROM hidden functions are fully supported and their entry/exit conditions are preserved across subsequent releases of the library.

Hidden commands contained in the Symbolic Matrix Writer library are not supported and therefore they cannot be used.

THE BALLADE OF THE PIRATE

The SmartROM represents a challenge for the HP handheld market. Thanks to its exclusive power-tools it saves you time and money but, given its low cost, its success relies entirely on the seriousness of any HP48 user.

By copying it, you are effectively contributing to the extinction of a rare species: the HP calculator software developer. If you deem that this risk is well worth running, just pick up a ROM copying software and go on. However, should you buy any other HP calculator, don't complain about
the lack of quality software. Perhaps you contributed actively to his death.

AAB

Category Stack manipulation

Affected by flag none

Input

2: Obj_A

1: Obj_B

Output

3: Obj_A

2: Obj_A

1: Obj_B

Function Duplicates the object on the second level of the stack.

See also BAA, BAB, BBA, BCAC, BCDA, CAB, CBA, NIP

ADD

Category Symbolic matrix manipulation

Affected by flag -3 (Numerical result)

Input

2: $\{\{SymbA_{1,1}\ SymbA_{1,2} \ldots SymbA_{1,n}\}$

$\ldots$

$\{SymbA_{m,1}\ SymbA_{m,2} \ldots\ SymbA_{m,n}\}\}$

1: $\{\{SymbB_{1,1}\ SymbB_{1,2} \ldots SymbB_{1,n}\}$

$\ldots$

$\{SymbB_{m,1}\ SymbB_{m,2} \ldots\ SymbB_{m,n}\}\}$

Output

1: $\{\{SymbA_{1,1}+SymbB_{1,1}\ SymbA_{1,2}+SymbB_{1,2}$
$\ldots\ SymbA_{1,n}+SymbB_{1,n}\}$

$\ldots$

$\{SymbA_{m,1}+SymbB_{m,1}\ SymbA_{m,2}+SymbB_{m,2}$
$\ldots\ SymbA_{m,n}+SymbB_{m,n}\}\}$

Function Returns the sum of two arrays (either symbolic or numerical).

Notes A one-dimensional array must be written as one-row or one-column two-dimensional array. To avoid run-time errors, flag -3 must be clear, alternatively each symbolic expression must evaluate to a numeric value.

You can supply either numerical or symbolic arrays; the result will be numerical if, and only if, both arrays are numerical. The routine performs the sum on all the pairs

of objects accepted by the standard operator + . To restrict object types to symbolics, use MSYMB? beforehand.
ADD, FACTOR and SUBT are instances of a generalized function, designed to apply binary operators to ordered pairs of elements. For more information on this topic, please refer to the Commands Reference manual.

See also

ADDCON, DIMS, EQUAL?, MATWRT, MULT, MSYMB?, SUBT

Applications

MATMENU, PRSYMB

Examples

```
2:      {{ 1 2  3 } {  'X'   'X+1' 'X-3' }}
1:      {{-1 2 -3 } {'-X-1' 'X+1' '3-X' }}
```

ADD **[ENTER]**

```
1:      {{ 0 4 0 } {-1 '2+2*X' 0 }}
```

```
2:      {{ [ 1  2  3]  [ 4 5 6 ] }}
1:      {{ [-1 -2 -3 ] [ 1 2 3 ] }}
```

ADD **[ENTER]**

```
1:      {{ [ 0 0 0 ] [ 5 7 9 ] }}
```

ADDCON

Category Symbolic matrix manipulation

Affected by flag -3 (Numerical result)

Input

2: {{ $Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$ }

{ $Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$ }}

1: Symb

Output

2: {{Symb+$Symb_{1,1}$ Symb+$Symb_{1,2}$ ... Symb+$Symb_{1,n}$}

{Symb+$Symb_{m,1}$ Symb+$Symb_{m,2}$ ... Symb+$Symb_{m,n}$}}

Function Adds a constant value to all the elements of a list-matrix.

Notes You can supply either a numerical or symbolic array; the result will be numerical if, and only if, the array and the value are numerical (no matter if the value is real or complex).

ADDCON and FACTOR are instances of a generalized function, designed to apply unary operators to each element of the matrix. In this case the operator is 'const +'. For more information on this topic, please refer to the Hidden Commands Reference manual

See also ADD, FACTOR, MATWRT, MSYMB?

Applications MATMENU, PRSYMB

Examples

```
2       {{ 1     2      3}
          {'X' 'X+1' 'X-3'}}
1       '-X-1'
```

ADDCON **[ENTER]**

```
1:      {{'-X' '1-X' '2-X'}
        { -1   0    -4  }}
```

→B

Category Type conversion

Affected by flag -5 to -10 (Binary Wordsize) and -11 to -12 (Binary Integer Base)

Input 1: h 1: Char 1: <<h>>

Output 1: #h

Function Converts a numeric value into a binary integer.

Notes →B accepts binary integers as well; dummy conversions prevent errors at no expense.

Pay attention to the current wordsize that may cause a truncation in the result. Extended precision numbers are not allowed. Although binary integers can be greater or smaller, in size, than 16 nibbles, →B always returns an integer of 16 nibbles. Moreover the conversion is performed on a numerical basis, hence the source object must evaluate to a number.

See also EXT→, →EXT, →R, →SYS

Example

5: 30
4: #20h
3: <13h>>
2: 1.5
1: 4

{→B } METOP **[ENTER]**
5: #1Eh
4: #20h
3: #13h
2: #2h
1: 4

BAA

Category Stack manipulation

Affected by flag none

Input

2: Obj_A

1: Obj_B

Output

3: Obj_B

2: Obj_A

1: Obj_A

Function Swaps the objects and duplicates the first level.

See also AAB, BAB, BBA, BCAC, BCDA, CAB, CBA, NIP

BAB

Category Stack manipulation

Affected by flag none

Input

2: Ob_A

1: Obj_B

Output

3: Obj_B

2: Obj_A

1: Obj_B

Function Duplicates the object on the first level and inserts it above the second level.

See also AAB, BAA, BBA, BCAC, BCDA, CAB, CBA, NIP

BBA

Category Stack manipulation

Affected by flag none

Input

2: Obj_A

1: Obj_B

Output

3: Obj_B

2: Obj_B

1: ObjA

Function Duplicates the object on the first level and rotates the first three levels.

See also AAB, BAA, BAB, BCAC, BCDA, CBA, NIP

BCAC

Category Stack manipulation

Affected by flag none

Input

3: Ob_A

2: Obj_B

1: Obj_C

Output

4: Obj_B

3: Obj_C

2: Obj_A

1: Obj_C

Function Rotates the first three levels and duplicates the second level.

See also AAB, BAA, BAB, BBA, BCDA, CAB, CBA, NIP

BCDA

Category Stack manipulation

Affected by flag none

Input

4: Obj_A
3: Obj_B
2: Obj_C
1: Obj_D

Output

4: Obj_B
3: Obj_C
2: Obj_D
1: Obj_A

Function Rotates the first four levels of the stack.

See also AAB, BAA, BAB, BBA, BCAC, CAB, CBA, NIP

C2M

Category Stack manipulation and meta-object manipulation

Affected by flag none

Input

N+1	'MARK'
N:	Obj_1
...:	...
2:	$Objn_{-1}$
1:	Obj_n

Output

N+2:	'MARK'
N+1:	Obj_1
...:	...
2:	Obj_n
1:	n

Function Counts the elements between the marker and the top of thestack. The marker is not removed from the stack after the evaluation.

Notes

This function is the RPL counterpart of Postscript's counttomark. Its primary function is to provide a mean for separating groups of objects lying on the stack. We may call this kind of weird entities unbound s, because they can shrink or expand without painful counter updates. Unbound meta-object management is usually slower than using counted meta-objects, because all the operations (except popping or pushing) require the random search of the marker. Nevertheless, there are circumstances where unbound meta-objects are still more convenient to use. Morevoer, with the introduction of release 1:C of the library, C2M has been translated in machine language with a speed increment of 1000% over the former version.

This is a summary of the situations where stack markers are preferred to counted meta-objects:

* when you cannot predict how many objects will be pushed on the stack by the user;
* when you cannot predict how many objects will be returned by an operation;
* when updating the counter requires an expensive algorithm;
* when you need to delimit stack areas;
* When the marker is missing the error MISSING STACK MARKER is issued. This feature was introduced in the third release (1:C) of the SmartROM. In order to reproduce the behavior of the former version, that is returning DEPTH when the marker is missing, you should replace all the occurrences of C2M with the following contruct:

```
IFERR C2M THEN DEPTH END
```

'MARK is a private symbol. Hidden functions make use of an alternate marker to avoid collision with user-defined programs.

See also

L2M, MARK, META

Examples

3: "MARK'
2: 10
1: 20

C2M **[ENTER]**

4: "MARK'
3: 10
2: 20
1: 2

CAB

Category Stack manipulation

Affected by flag none

Input

3: Obj_A

2: Obj_B

1: Obj_C

Output

3: Obj_C

2: Obj_A

1: Obj_B

Function Rotates backwards the first three levels.

See also AAB, BAA, BAB, BBA, BCAC, BCDA, CBA, NIP

CBA

Category Stack manipulation

Affected by flag none

Input

3: Obj_A

2: Obj_B

1: Obj_C

Output

3: Obj_C

2: Obj_B

1: Obj_A

Function Reverses the order of the first three levels.

See also AAB, BAA, BAB, BBA, BCAC, BCDA, CAB, NIP

CHL?

Category Argument checking

Affected by flag none

Input

2: { obj_1 obj_2 ...-obj_p }

1: { t_1 t_2 ... t_n }

Output

3: { obj_1 obj_2 ... obj_p }

2: { obj_1 obj_2 ... obj_p } 2: { err_1 err_2 ... err }

1: 0 1: 1

Function

Tests whether the elements of a list comply with the types specified. Each element of the list on the first level can be a real value (rounded to an integer) or a list of reals. Argument checking is strictly positional in CHL?. If you need non-positional argument checking, use CHSET?.

Type numbers follow the classification given by the command TYPE.

When a sublist of reals is specified, a multiple choice is allowed on that position of the input list.

The command return a boolean flag indicating success when the flag is 0 and a fault when the flag is 1 plus a list of mismatching positions.

The underlying meta-object structure of the result unifies the boolean convention with the meta-object convention. The output is especially suitable for IFT or IFTE input.

See also CHSET?, CHST?

Applications MATMENU

Examples

```
2:      { 1 2 "" }
1:      { 0 0 2 }
```

CHL? **[ENTER]**

```
2:      { 1 2 "" }
1:      0
```

```
2:      { 1 2 "" }
1:      { 0 0 { 0 1 } }
```

CHL? **[ENTER]**

```
3:      { 1 2 "" }
2:      { 3 }
1:      1
```

CHSET?

Category Argument checking

Affected by flag none

Input

2: { obj_1 obj_2 ... obj_p }

1: { t_1 t_2 ... t_n }

Output

3: { obj_1 obj_2 ... obj_p }

2: { obj_1 obj_2 ... obj_p } 2:{ err_1 err_2 ... err_k }

1: 0 1: 1

Function

Tests whether the objects contained in the list are of the type specified, regardless of their position. Each element of the list of types must be a real number (rounded to an integer). Type numbers follow the classification given by the command TYPE. The command return a boolean flag, indicating success when the flag is 0 and a fault when the flag is 1 plus a list of mismatching positions.

The underlying meta-object structure of the result unifies the boolean convention with the meta-object convention. The output is especially suitable for IFT or IFTE input.

CHSET? is useful for checking whether a list contains homogeneus data with minimal memory requirements.

See also CHL?, CHST?

Examples

```
2:      { 1 2 "" }
1:      { 0 2 }

CHSET? [ENTER]

2:      { 1 2 "" }
1:      0

2:      { 1 2 "" }
1:      { 0 }

CHSET? [ENTER]

3:      { 1 2 "" }
2:      { 3 }
1:      1
```

CHST?

Category Argument checking

Affected by flag none

Input

N+1:	obj_1	
...:	...	
2:	obj_n	
1:	$\{ t_1\ t_2 \ldots t_n \}$	

Output

N+2:	obj_1		
...:	...	N+1:	obj_1
K :	obj_k	...:	...
...:	...	K :	obj_k
3:	obj_n	...:	...
2:	$\{ \ldots err_k \ldots \}$	2:	obj_n
1:	1	1:	0

Function

Tests whether the elements of the stack comply with the types specified in the list. Each element of the control liston the first level can be a real value (rounded to an integer) or a list of reals. Argument checking is strictly positional in CHST?. The first element of the control list specifies the type or the set of types allowed for the object lying on the correspondent level of the stack. In effect objects on the stack are mapped as if they were collected in a list, with the element on the top that comes last.

Type numbers follow the classification given by the command TYPE.

When a sublist of reals is specified, a multiple choice is allowed on the corresponding level of the stack. The command return a boolean flag indicating success when the flag is 0 and a fault when the flag is 1 plus a list of mismatching positions.

The underlying meta-object structure of the result unifies the boolean convention with the meta-object convention. The output is especially suitable for IFT or IFTE input.

See also

CHL?, CHSET?

Applications

MATMENU

Examples

See also the example under SYS.

```
4:      1
3:      2
2:      “”
1:      { 0 0 2 }
```

CHST? **[ENTER]**

```
4:      1
3:      2
2:      “”
1:      0

4:      1
3:      2
2:      “”
1:      { 0 0 { 0 1 } }
```

CHST? **[ENTER]**

```
5:      1
4:      2
3:      “”
2:      { 3 }
1:      1
```

→*Char*

Category	Type conversion
Affected by flag	none
Input	1: n 1: <n>> 1: #n or 1: Str 1: Character
Output	1: Character
Function	Converts the input object into a character object.
Notes	Character objects are used internally to minimize storage overhead (in spite of strings). →CHARaccepts character objects as well;dummy conversions prevent errors at no expense.
See also	→B, EXT→, →EXT, →R, →SYS

CMPL

Category Symbolic matrix manipulation

Affected by flag none

Input 2: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
...
$\{Symb_{n,1}\ Symb_{n,2} \ldots Symb_{n,n}\}\}$

1: { i j }

Output 1: $\{\{Symb_{1,1} \ldots Symb_{1,i-1}\ Symb_{1,i+1} \ldots Symb_{1,n}\}$
...
$\{Symb_{j-1,1} \ldots Symbj_{-1,i-1}\ Symbj_{-1,i+1} \ldots Symb_{n,n}\}$
$\{Symb_{j+1,1} \ldots Symb_{j+1,i-1}\ Symb_{j+1,i+1} \ldots Symb_{n,n}\}$
...
$\{Symb_{n,1} \ldots Symb_{n,i-1}\ Symb_{n,i+1} \ldots Symb_{n,n}\}\}$

Function Returns the complement of a square list-matrix, given element's subscripts. The resulting list is of n-1th order, thus the input list can't be of first order or null.

Notes

To get the algebraic complement of an element, the following procedure can be used.

-3 CF	Enables symbolic results;
BAB	saves element subscripts,
CMPL DETERM	takes the determinant of the minor;
SWAP OBJ→ DROP	recalls the subscripts
+ -1 SWAP ^ *	adjusts the sign

See also

ADD, ADDCON, CONSTMAT, DETERM, FACTOR, MATWRT, MULT, SQUARE?

Applications

MATMENU

Examples

```
2:      {{1 0 0 } {'-X'  '1-X' '2-X'} {-1  0 -4}}
1:      { 1 1 }
```

CMPL **[ENTER]**

```
1:      {{ '1-X' '2-X'} { 0 -4}}
```

CONFORM?

Category Symbolic matrix manipulation

Affected by flag none

Input

2: {{SymbA1,1 ... SymbA1,n}
...
{SymbAm,1... SymbAm,n}}

1: {{SymbB1,1... SymbB1,p}
...
{SymbBn,1 ... SymbBn,p}}

2: {{SymbA1,1 ... SymbA1,n}}
...
{SymbAm,1 ... SymbAm,n}}

{{SymbB1,1 ... SymbB1,q}
...
{SymbBp,1 ... SymbBp,q}}

Output

4: {{SymbA1,1 ... SymbA1,n}
{SymbAm,1 ... SymbAm,n}}

3: {{SymbB1,1 ... SymbB1,p}
...
{SymbBn,1 ... SymbBn,p}}

2: { m n n p }

1: 1

...

3: {{SymbA1,1 ... SymbA1,n}
...
{SymbAm,1 ... SymbAm,n}}

2: {{SymbB1,1 ... SymbB1,q}
...
{SymbBp,1 ... SymbBp,q}}

1: 0

Function

Tests whether dimensions of the two input lists are suitable for row-by-column product. The result of the test is a meta-object whose counter may be also interpreted like a flag (1=true, 0=false).

Notes

The condition is called conformability. This check is automatically performed by MULT which returns an error when the condition is not met. By using CONFORM? beforehand, you can take an appropriate action without resorting to trapping routines and CASE constructs.

All the routines implementing symbolic matrix operations check list dimensions, but do not check list contents. This design strategy implies several interesting side effects:

- faster execution with shorter code
- easier implementation of future extensions
- the possibility of performing a run-time check exploiting built-in dispatching capabilities of operators and functions.

Whenever you need to verify list contents, use the command MSYMB? that explicitly checks each element.

See also

DIMS, MATWRT, MSYMB?, MULT, SQUARE?

Applications

MATMENU, PRSYMB

Examples

```
2:      { { 1 2 } { 'X' 'Y-2' } }
1:      { { 0 1 '-Z' }      { -2 'Y' 3 } }
```

CONFORM? **[ENTER]**

```
4:      { { 1 2 } { 'X' 'Y-2' } }
3:      { { 0 1 '-Z'}       { -2 'Y' 3 } }
2:      { 2 2 2 3}
1:      1
2:      { { 1 2 3 4 }     { 5 6 7 8 } { 9 10 11 12 } }
1:      { {  'X' 0 } }
```

CONFORM? **[ENTER]**

```
3:      { { 1 2 3 4 }     { 5 6 7 8 } { 9 10 11 12 } }
2:      { {  'X' 0 } }
1:      0
```

CONSTMAT

Category Symbolic matrix manipulation

Affected by flag none

Input

2:	Any	2:	Any
1:	{ m n }	1:	n

Output

1: {{Any1,1 Any1,2 ... Any1,n}
1: {{Any1,1 Any1,2 ... Any1,n}
... ...
{Anym,1 Anym,2 ... Anym,n}}
{Anyn,1 Anyn,2 ... Anyn,n}}

Function Returns a constant symbolic matrix of specified dimension.

Notes The command is suitable to build templates for the user who shall replace the elements with values, making obsolete verbose messages like input a 3x5 matrix and press **[ENTER].**

The usage of CONSTMAT described above is a special case of inizialization which is the primary purpose of the ommand.

Of course you can initialize a matrix with any object provided that you set proper flags before calling the Matrix Writer. placeholder '?', often used in the examples, is especially suitable because it gives you the idea of something missing but does not require special treatment being a global name.

See also DIMS, CONFORM?, MATWRT, MSYMB?

Applications

MATMENU, PRSYMB

Examples

A few coding suggestions:
creating a matrix with the same dimensions as that given:

DIMS '?' SWAP CONSTMAT

creating a matrix with the same number of columns but n rows:

DIMS '?' SWAP 1 n PUT CONSTMAT

creating a matrix with transposed dimensions as that given

DIMS REV '?' SWAP CONSTMAT

See also the examples under MULT and EQUAL?.

```
2:              'X-1'
1:              { 2 3 }
```

CONSTMAT **[ENTER]**

```
1:              {{ 'X-1' 'X-1' 'X-1' }
                 { 'X-1' 'X-1' 'X-1' }}

2:              “”
1:              3
```

CONSTMAT **[ENTER]**

```
1:              {{“” “” “”}
                 { “” “” “” }
                 { “” “” “” }}
```

COPY

Category Meta-object manipulation

Affected by flag none

Input

...:	obj_1
...:	...
...:	obj_{from}
...:	...
...:	obj_{to}
...:	...
...:	obj_{above}
...:	...
5:	obj_n
4:	n
3:	from
2:	to
1:	above

Output

...:	obj_1
...:	...
...:	obj_{from}
...:	...
...:	obj_{to}
...:	...
...:	$obj_{above-1}$
...:	obj_{from}
...:	...
...:	objto
...:	obj_{above}
...:	...
...:	obj_n
1:	n+abs(f-t+1)

Function

Copies the section of the meta-object delimited by from and to above level above and updates the counter accordingly.

Notes

The first element of the meta-object (that is the lement indexed with 1) lies on level n+1, as if it were the first element of a list.

Input parameters from and to can be given in any order. If you specify for above a value greater than n, the section will be appended to the tail of the meta-object.

Note that you can copy a block within itself specifying for above a value between from and to.

See also

DELETE, MOVE

Examples

```
6:    "TOP
5:    "these lines"
4:    "get copied"
3:    "after the bottom"
2:    "BOTTOM"
1:    5

2 4 6 COPY [ENTER]

9:    "TOP
8:    "these lines"
7:    "get copied"
6:    "after the bottom"
5:    "BOTTOM"
4:    "these lines"
3:    "get copied"
2:    "after the bottom"
1:    8

5:    "HEAD"
4:    "SUBHEAD"
3:    "GO OVER THE TOP"
2:    "ME TOO"
1:    4

3 5 1 COPY [ENTER]

7:    "GO OVER THE TOP"
6:    "ME TOO"
5:    "HEAD"
4:    "SUBHEAD"
3:    "GO OVER THE TOP"
2:    "ME TOO"
1:    6

5:    "HEAD"
4:    "EXPAND MYSELF #1"
3:    "EXPAND MYSELF #2"
2:    "BOTTOM"
1:    4
```

2 3 3 COPY **[ENTER]**

7: "HEAD"
6: "EXPAND MYSELF #1"
5: "EXPAND MYSELF #1"
4: "EXPAND MYSELF #2"
3: "EXPAND MYSELF #2"
2: "BOTTOM"
1: 6

CSTMENU

Category Utility

Affected by flag none

Input

Output

Function Replaces System variable;Variables >CST contents with SmartROM's custom menu.

Notes CSTMENU saves the contents of CST for recovery by LASTARG. This means that you can restore the previous contents of CST immediately after issuing CSTMENU but not later. This feature was introduced with
the release 1:C of the library.
SmartROM's custom menu subdivides the command set in six main areas for Commands; quicker access:

STACK $&L META SYMB TYPES MISC

It also adds the following six features:

CHR A keytrap for entering
a character given its ASCII code. The trap waits indefinitely key presses and does not allow meaningless keystrokes. Character 0 cannot be entered nor can characters above 255. To disable the beeper set flag -56. The message "Enter three digits" remains in the status area until you press

the last key. The **ON** key is disabled along with other meaningless keys. When you enter **[2] [5]**, the keyboard is mapped to respond only to keys **0 - 5**. Alarms coming due while typing keys are automatically queued.

VISIT Enhanced Visit Function:

Input

Absolute position
2: 0 (replace) or 1 (insert)
1: 'name'

3: { row col }
2: 0 (replace) or 1 (insert)
1: 'name'

3: "string"
2: occurrence

1: 'name'

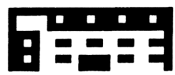
This function lets you edit the contents of a variable starting from a specified position within the text.

The position can be specified in three different ways as shown in the stack diagram. When you specify a search string and supply a 0 as occurrence, the cursor will be placed on the last occurrence of the string. If the string is not contained
in the text, the cursor will be placed at the end of the last line.

SYSEVAL Syseval typing aid

$<>G Toggles between a string object and a global name. Identifiers cannot exceed 255 characters. The length is truncated modulo 256, thus an identifier of length 257 is treated as if it were of length 1. Note that assigning to an identifier a name already used by an internal command, you can ovverride it. This feature can be useful to extend system commands (system functions are more difficult beasts). See on page <$R[P#,tolocal]148> to learn how to get a local name.

Interactive stack trap. This feature lets you access the interactive stack from anywhere. While you are in the editor, you can enter the stack environment, even if you are in the middle of a program, and copy stack items as text to the current cursor position.



The spreadsheet icon labels the Symbolic Matrix Writer (Omnibus) trap. By pressing the rightmost key, you enter an empty worksheet.

[Gold] This keystroke lets you edit a list-matrix lying on the stack.

DELCOL

Category Symbolic matrix manipulation

Affected by flag none

Input

2: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$ }
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$ }}

1: j

Output

1: {{$Symb_{1,1}$ ... $Symb_{1,j-1}$ $Symb_{1,j+1}$ ... $Symb_{1,n}$ }
...
{$Symb_{m,1}$ ... $Symb_{m,j-1}$ $Symb_{m,j+1}$ ... $Symb_{m,n}$ }}

Function Deletes the specified column from the list.

Notes No restrictions on the shape of the list. If an invalid subscript is specified, the corresponding error is issued. The list-matrix can contain any object type. If you need to delete a column and a row, CMPL lets you do it in one shot.

See also CMPL, CONSTMAT, DELROW

Examples

2: {{1 0 0 } {'-X' '1-X' '2-X'} {-1 0 -4}}
1: 1

DELCOL **[ENTER]**

1: {{ 0 0 }
{ '1-X' '2-X'}
{ 0 -4}}

DELETE

Category Meta-object manipulation

Affected by flag none

Input

...:	obj_1
...:	...
...:	obj_{from}
...:	...
...:	obj_{to}
...:	...
4:	obj_n
3:	n
2:	from
1:	to

Output

...:	obj_1
...:	...
...:	obj_{from-1}
...:	obj_{to+1}
...:	...
2:	obj_n
1:	n-ABS(from-to+1)

Function Deletes a block from the meta-object, updating the counter.

Notes

from and to can be given
in any order. Values out of
range will generate an error message. When you delete
all the objects, on the stack remains only the counter 0.

See also

DELETE, MOVE

Examples

6: "delete after this line"
5: "garbage"
4: "garbage"
3: "garbage"
2: "last"
1: 5

2 4 DELETE **[ENTER]**

3: "delete after this line"
2: "last"
1: 2

DELROW

Category Symbolic matrix manipulation

Affected by flag none

Input

2: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

1: i

Output

1: $\{\{Symb_{1,1}\ \ Symb_{1,2}\ \ \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{i-1,1}\ Symb_{i-1,2} \ldots Symb_{i-1,n}\}$
$\{Symb_{i+1,1}\ Symb_{i+1,2} \ldots Symb_{i+1,n}\}$
$\ldots$
$\{Symb_{m,1}\ \ Symb_{m,2}\ \ \ldots Symb_{m,n}\}\}$

Function Deletes the specified row from the list.

Notes No restrictions on the shape of the list. If an invalid subscript is specified, the corresponding error is issued. The list-matrix can contain any object type. If you need to delete at the same time a column and a row, CMPL lets you do it in one shot.

See also CMPL, CONSTMAT, DELCOL

Examples

```
2:      {{1 0 0 } { '-X' '1-X' '2-X'} { -1 0 -4}}
1:      1
```

DELROW **[ENTER]**

```
1:      {{ '-X' '1-X' '2-X'}
         {  -1   0     -4}}
```

DETERM

Category Symbolic matrix manipulation

Affected by flag -3 (Numerical result)

Input 1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$

$\ldots$

$\{Symb_{n,1}\ Symb_{n,2} \ldots Symb_{n,n}\}\}$

Output 1: Symbolic

Function Returns the determinant of a square matrix.

Notes The result is not simplified. You can use EXPAND and COLCT or the program EXCO described in Chapter 31 of the HP-48 User's Manual. The algorithm used is described in the Hidden manual>Commands Reference.

The algorithm is optimized for sparse matrices. The precision of a numeric result varies from case to case. Sometimes is more precise than DET, while in other cases is worse.

See also SQUARE?

Applications INVRT, MATMENU, PRSYMB

Examples

```
1:  {{ 'X'    4    -1  '(X-2)/Y'}
    {  0      1   '-X'   '2*Y' }
    { 'Y/X' '1/X'  -3   'X-Y' }
    {  0      1    Y    'Y-2*X'}}
```

DETERM **[ENTER]** EXCO **[ENTER]**

'-2-1/X*Y^2-8/X*Y^3-2/X*Y+X*Y^2+4*X^2-
X^3+12*X*Y-2*Y^2+X-Y'

DIMS

Category Symbolic matrix manipulation

Affected by flag none

Input

1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$

...

$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

Output

2: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$

...

$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

1: { r c}

Function Returns the dimensions of the list-matrix, checking matrix consistency. If a dimensional error is detected, an error messagge is issued.

Notes No check is made on object types. If you want to check each element, use MSYMB?.

When a row contains an invalid number of objects, the error "Invalid Sub-list" is issued.

See also CONFORM?, MATWRT, MSYMB?, SQUARE?

Applications INVRT, MATMENU, PRSYMB

EQUAL?

Category Symbolic matrix manipulation

Affected by flag none

Input

```
2:  {{SymbA1,1 ... SymbA1,n}        2:  {{SymbA1,1... symbA1,n}
                ...                                 ...
    {SymbAm,1 ... SymbAm,n}}            {SymbAm,1 ... SymbAm,n}}
1:  {{SymbB1,1 ... SymbB1,n}        1:  {{SymbB1,1 ... SymbB1,q}
                ...                                 ...
    {SymbBm,n ... SymbBm,n}}            {SymbBp,1 ... SymbBp,q}}
```

Output

```
4:  {{SymbA1,1 ... SymbA1,n}
    ...
    {SymbAm,1 ... SymbAm,n}}
3:  {{SymbB1,1 ... SymbB1,n}        3:  {{SymbA1,1 ... SymbA1,n}
                ...                                 ...
    {SymbBm,1 ... SymbBm,n}}            {SymbAm,1 ... SymbAm,n}}
2:  { m n }                         2:  {{SymbB1,1 ... SymbB1,q}
                                                    ...
                                        {SymbBp,1 ... SymbBp,q}}
1:  1                               1:  0
```

Function Checks if matrices have the same dimensions. If a dimension mismatch is found, a 0 is returned.

Notes The underlying meta-object structure of the result unifies the boolean convention with the meta-object convention. The output is especially suitable for IFT or IFTE input.

See also CONFORM?, DIMS, MSYMB?, SQUARE?

Applications MATMENU

Example Let's create a program taking two matrices, using the Matrix Writer, that performs a sum only if the two arrays have the same dimensions, otherwise asks the user to modify either of the twos. Then we will examine an alternate method which imposes more restrictive rules to the user.

```
«       → mat «                 First of all, let's
        DO                      define a subprogram.
                mat             Its function is

        UNTIL                   to iterate the request
                MATWRT          until the user satisfies it.

        END »
»

'ASK' [STO]                     Name it ASK.

«       33 CF                   And now the program.
                                Enables symbolic mode. See under MATWRT.

        1 2 START               Asks the user for two matrices.
                { { } }         The request is repeated until both
                ASK             errays are entered
        NEXT
```

```
WHILE                          Tests whether the
          EQUAL? NOT           arrays are
REPEAT                         compatible, if not
      " Can't do the sum !     redisplays each
        modify either of the   array for editing
        two arrays."           until the user returns
      3 DISP                   a valid pair
      SWAP ASK                 of arrays.
      SWAP ASK
END

      DROP ADD                 Returns the sum
»                              of the two arrays.
```

The program explained above lets you change freely matrix dimensions, thanks to the Matrix Writer auto-redimensioning features. The keyword EQUAL? is responsible for accepting or not the input.
The following program instead forces the user to enter the second matrix with the same dimensions as the first one without a second pass (and without using EQUAL?).

```
«     33 CF                    Turns on
      {{}} ASK                 Symbolic mode.
      34 SF 35 SF              Locks matrix
                               dimensions; see
                               under
                               MATWRT.

      DIMS '?' SWAP            Creates a
      CONSTMAT                 template for the
                               second matrix;
```

```
ASK ADD
34 CF 35 CF
»
```

enters the second matrix, returns the sum and unlocks matrix dimensions.

EXT→

Category Type conversion

Affected by flag none

Input 1: obj

Output 1: addr

Function Returns the memory address where the object is stored. When the address is less than <70000h> the object is stored in ROM.

Notes EXT→is useful to decipher system RPL programs. External is the raw representation of a system address or External object. Externals are explained in Appendix B. System addresses are listed in a special library supplied with the ESA Assembler. Please refer to the program on page <$R[P#,Xlib]160> for printing the list of the system addresses.

See also →EXT, SYS→, →SYS.

Examples

1 EXT→ **[ENTER]**

1: #2A2C9 SYSEVAL **[ENTER]**

1: 1

TRUE **[ENTER]**

1: External

EXT→ **[ENTER]**

1: <03A81h>

→EXT

Category Type conversion

Affected by flag -5 to -10 (Binary Wordsize) and -11 to -12 (Binary Integer Base) only when the input number is a binary integer.

Input

1: addr

or

1: #n

Output

1: Obj

Function How to...;get the object at address #hhhhh. Returns the address of an object.

Notes

If the address specified represents the beginning of a machine language routine, the display will show External on the first level of the stack. Unfortunately the HP48 represents with 'External' meaningless address too, therefore pay attention before evaluating a socalled External.

Beware of binary wordsize and current base.

We need to point out that there is no real distinction between a system RPL program and a user RPL program from the execution's viewpoint.

The main difference is represented by a faster execution of the system RPL due to the higher specialization of the code and to the depletion of stack checks.

If a system RPL program has got an associated label (name) and number (Xlib number), we are referring to an RPL command (or function). Of course we may have a program retaining an associated number (Xlib number) but no associated name, in this case we had rather to speak of a hidden command. Appendix H contains a list of SmartROM hidden commands and the Hidden CommandsReference manual describes each entry.

See also appendix B for further details.

See also EXT→, →PRG.

Applications L→TH, SHRINK, TH→L.

Examples

1: #30794h

→EXT **[ENTER]**

1: External

[EVAL]

1: "HPHP48-x" x = revision letter (A,B,C,D,E or J)

1: #2B0F2h

→EXT **[ENTER]**

1: Long Real

FACTOR

Category Symbolic matrix manipulation

Affected by flag -3 (Numerical result)

Input

2: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

1: Symb

Output

1: {{Symb*Symb1,1 Symb*Symb1,2 ... Symb*Symb1,n }
...
{Symb*Symbm,1 Symb*Symbm,2 ... Symb*Symbm,n }}

Input

2: $\{\{Symb_{A1,1}\ Symb_{A1,2} \ldots Symb_{A1,n}\}$
$\ldots$
$\{Symb_{Am,1}\ Symb_{Am,2} \ldots Symb_{Am,n}\}\}$

1: $\{\{Symb_{B1,1}\ Symb_{B1,2} \ldots Symb_{B1,n}\}$
$\ldots$
$\{Symb_{Bm1}\ Symb_{Bm,2} \ldots Symb_{Bm,n}\}\}$

Output 1: $\{\{Symb_{A1,1}*Symb_{B1,1}\ Symb_{A1,2}*Symb_{B1,2}\ \ldots\ Symb_{A1,n}*Symb_{B1,n}\}$

$\ldots$

$\{Symb_{Am,1}*Symb_{Bm,1}\ Symb_{Am,2}*Symb_{Bm,2}\ \ldots\ Symb_{Am,n}*Symb_{Bm,n}\}\}$

Function Multiplies all the elements of the list by the Symbolic value Symb or returns the in-place product of two arrays.

Notes A one-dimensional array must be written as one-row or one-column two-dimensional array. To avoid run-time errors, flag -3 must be clear, alternatively each symbolic expression must evaluate to a numeric value.

In the case of two arrays, the result is a matrix where each element is the product of the correspondent pair of elements of the input matrices. Allowed pairs of objects are those accepted by the standard operator *. To restrict object types to symbolics, use MSYMB? beforehand, or, alternatively, force the user to enter the matrices within the Symbolic Matrix Writer environment; an example can be found under EQUAL?.

FACTOR is an instance of a generalized function, designed to apply binary operators to all the pairs of elements. For more information on this topic, please refer to the Hidden Commands Reference manual.

See also ADD, DETERM, MULT, SQUARE?, SUBT

Applications MATMENU

Example

```
1:      {{ 2 'X' 4 }
        { -1  0 'X/2' }
        { 0  3 'X+1'}}
```

2 FACTOR **[ENTER]**

```
1:      {{ 4   'X*2'   8    }
        { -2     0  'X/2*2' }
        { 0      6  '(X+1)*2'}}
```

```
1:      [ 1 2 3 ]
```

'X' FACTOR **[ENTER]**

```
1:      {{ 'X'  '2*X'  '3*X' }}
```

```
2:      [ 1 2 3 ]
1:      [ -1 3 5 ]
```

FACTOR **[ENTER]**

```
1:      [ [ 1 4  9 ] ]
```

FALSE

Category Type conversion

Affected by flag none

Input

Output 1: External (FALSE)

Function Returns the system boolean FALSE.

Notes System booleans are machine language routine addresses which merely return themselves when evaluated. The command →TorF turns a system boolean into a real boolean.

FALSE resides at address #03AC0.

See also EXT→, TRUE, →TorF

Examples Create a program to convert a real valued boolean (0=FALSE 1=TRUE) into a system boolean:

« 0 > TRUE FALSE IFTE »

Store it as '→BOOL' for later use.

FIND

Category String, List and program editing function.

Affected by flag none

Input

2:	str1	2:	prg	2:	list	2:	hex1
1:	str2	1:	obj	1:	obj	1:	hex2

Output

2: input_object
1: Found: occurences

Function Returns the total number of occurences of an object or (hex) substring within a composite object or (hex)string respectively.

See also REPLACE

Examples

2: { "" 2 4 { 2 "" } }
1: ""

FIND **[ENTER]**

2: { "" 2 4 { 2 "" } }
1: Found: 2

2: "ABCDABCDEFABC"
1: "ABC"

FIND **[ENTER]**

2: "ABCDABCDEFABC"
1: Found: 3

1: **<< n << n ROLL DROP >>**

{ ROLL } OBJ→ DROP FIND **[ENTER]**

2: **<< → n << n ROLL DROP >>**
1: Found: 1

See also under REPLACE for an interesting application involving hex strings.

IDNT

Category Symbolic matrix manipulation

Affected by flag none

Input 1: n

Output 1:
```
{{1 0 ... 0}
 {0 1 ... 0}
    ...
 {0 0 ... 1}}
```

Function Returns the identity matrix of order n.

See also CONSTMAT

Applications MATMENU

Examples 1: 3

IDNT **[ENTER]**

1:
```
{{ 1 0 0 }
 { 0 1 0 }
 { 0 0 1 }}
```

KEYWAIT

Category Input/Output

Affected by flag none

Input

Output

2: <<KEYh>>
1: <<SHIFTh>>

Function

Waits indefinitely for a keypress. The **[ON]** key is trapped like any other key.

Notes

The keyboard is numbered, starting from the upper left corner down to the lower right corner in row major order. The key associated to A is numbered <<1h>> and the last is <<2Fh>> (49d). KEYWAIT detects shifted keys so that [ALPHA], **[Gold]**and **[Blue]**cannot be trapped singularly. The shift status is so encoded:

<1h>	no SHIFT
<2h>	**[Gold]**
<3h>	**[Blue]**
<4h>	[alfa]
<5h>	[alfa] **[Gold]**
<6h>	[alfa] **[Blue]**

This encoding system requires fewer processing than that required to dispatch a key trapped with 0 WAIT. Moreover the **[ON]** key is trapped like any other key.

The KEYWAIT internal routine is the most simple application of what HP calls Parameterized Outer Loop, also known as ParOuterLoop. This routine is the core of any interactive built-in application thanks to its flexibility. The GRAPHics editor uses the same basic routine as KEYWAIT ! ParOuterLoop is well explained in the documentation provided by HP called HP48 development tools.

JOINR

Category Graphics

Affected by flag none

Input 2: $Grob_{m \times h}$

1: $Grob_{p \times h}$

Output 1: $Grob_{(p+m) \times h}$

Function Appends the grob on the first level to the right side of the grob on the second level.

Notes Grobs must have the same height. The result is placed in

a new grob.In the Appendix H, you will find a hidden function called gop performing logical operations between grobs. More details about it can be found in the Hidden Commands Reference manual.

See also JOINUP, RPT

Applications →FONT, PROBJ

Example See also on the next page.

,

JOINUP

Category Graphics

Affected by flag none

Input 2: $Grob_{w \times h}$

1: $Grob_{w \times l}$

Output 1: $Grob_{w \times (h+l)}$

Function Puts the grob on the first level onto the top of the grob lying on the second level.

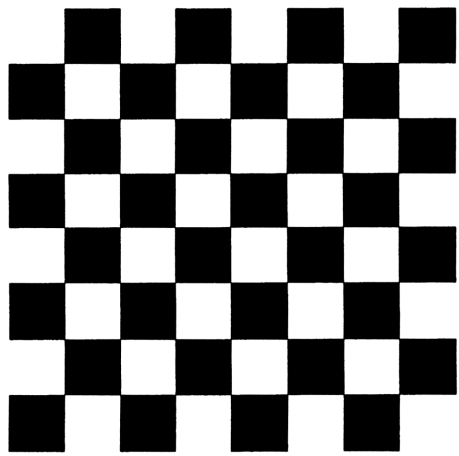
Notes Grobs must have the same width.

See also JOINR

Example Let's suppose you want to create a chessboard. By using JOINUP, JOINR and RPT, you can build a grob in a few steps without loops. A technique alike can be used also for creating grids or shadings.

```
#8 #8 BLANK     First of all let's
                create a square grob;

DUP NEG         then duplicate it
                and invert pixel color;
```

```
DUP2 JOINR      make a copy of
                both objects and
                join them horizontally;

31 CF 4 RPT     build the first line

CBA             take apart the first line
                and invert the order
                of the grobs

JOINR 4 RPT JOINUP

                join them horizontally and vertically

31 SF 4 RPT 31 CF

                Ready !
```



L2M

Category List and stack manipulation

Affected by flag none

Input

N+1:	"MARK'	
N:		obj1
...:		...
2:		objn-1
1:		objn

Output

1:	{obj1 obj2 ... objn}

Function Collects all the objects between the marker and the TOS. The marker is always removed from the stack.

Notes If no marker is on the stack, L2M issues the corresponding error.

'MARK is a private marker. Hidden commands make use of an alternate marker to avoid collision with user defined programs.

See also L2M, MARK, META

Examples

```
3:      'MARK
2:      "Enter as many numbers as you want"
1:      { V }

<< IFERR INPUT
  THEN
    KILL
  END
  OBJ→
  L2M OBJ→
  1 -
  << MAX >>
  RPT
  "The greatest is "
  SWAP +
  7 DISP 7 FREEZE
>>
```

→*LINES*

Category String and Meta-object manipulation

Affected by flag none

Input

1:	str

Output

N+1:	str1
...:	...
2:	str_n
1:	n

Function Splits the string into several lines breaking it at linefeeds.

Notes Linefeeds are used as newline characters in the editor. Moreover they are translated to the sequence CR LF during the transmission to a printer when the translation parameter in the global variable IOPAR has a value greater than 0. →LINES removes the linefeeds, and pushes on the stack resulting strings. However the command ignores linefeeds embedded in double quotes. This feature leaves message strings intact and lets you replace linefeeds, which are rendered with a black box, with some more descriptive sequence of characters.

See also LINES→

Example See also the example under SPLIT.

Suppose you want to enhance the readability of your printed output by replacing embedded linefeeds with the following sequence of characters: "<<LF>>"

```
→LINES
{ 10 CHR "<LF>" REPLACE DROP } METOP
LINES→
```

LINES→

Category String and Meta-object manipulation

Affected by flag none

Input

N :	str_1	obj_1
...:	...	...
2:	str_n	obj_n
1:	n	n

Output

1:	str

Function Joins objects by means of a linefeed.

Notes It is the inverse of →LINES.

See also →LINES, SPLIT

Applications PROBJ

Example See also the examples under SPLIT.

LOC

Category String function

Affected by flag none

Input

3:	str_1		3:	hex_1
2:	str_2		2:	hex_2
1:	pos		1:	pos

Output 1: pos

Function Seeks str_2 in str_1 starting from position pos.

If no match is found returns 0, otherwise the absolute location of the match.

Notes LOC is an extension of POS.

See also MEMBER, SPAN

Examples

3: "abcdabcdabcd"
2: "abc"
1: 2

LOC **[ENTER]**

1: 5

Searching for a sequence of objects within a program or list:

Set hex mode and word size to 64 before starting.

Retrieve the program HSUB, and the program HSIZE,

```
«
  { { 5 8 9} 5 } }          2 arguments;
  IF CHST? NOT              tests whether the object
                            on the third level is a
                            program or list and the first
                            level contains a list.
  THEN DUP2                 Ok, proceed.
  →Xlib                     translates into hex format the search
                            key
  EVAL 6 OVER
  HSIZE                     returns the length of the hex string
  5 -                       returns length-5
  HSUB                      gets the substring [6;length-5]
  SWAP                      prepares the main object
  821 275 →Xlib             translates it into hex format
  EVAL
  SWAP                      reorders
  1 LOC                     seeks the sequence of objects starting
                            from the beginning
  1 0 IFTE                  returns 1 if found, 0
                            otherwise
  ELSE DROP
  "Stack parameters:        2: Prog, List, Alg
                            1: List
```

```
                                        "

    3 DISP          When parameters mismatch, the stack
                    diagram is shown and the program
                        aborted

    3 FREEZE

    END »
```

Save the program. The example is stored on the disk in the directory EXAMPLES.1C under the name BFIND.

```
    ELSE DROP

    "Stack parameters:      2: Prog, List, Alg
                            1: List
```

LOP1

Category List manipulation

Affected by flag none

Input

2: $\{obj_1\ obj_2 \ldots obj_n\}$

1: $\{cmd_1\ cmd_2 \ldots cmd_k\}$

Output

1: $\{obj_1\ obj_2 \ldots obj_n\}$

Function Given a list of objects in level 2 and a list of commands in level 1, applies the commands to each element of the operand-list and puts the result in a new list.

Notes The operator list must return one, and only one, result at any time. If the operator returns more than one object as result, use LOPN instead. This command has got common aspects with the induction postulate:

1) define a procedure working on a single object
2) proof if it works on the first element.
3) apply to all elements.

See also LOPN, LVOP, METOP

Examples

2: { 2 3 4 }
1: { INV →Q }

LOP1 **[ENTER]**

1: { '1/2' '1/3' '1/4' }

VARS **[ENTER]**

1: { PROG1 PROG2 PROG3 }

{ DUP BYTES NIP SWAP →TAG } LOP1 **[ENTER]**

1: { :PROG1:307 :PROG2:8604.5 :PROG3:1233
}

LOPN

Category List manipulation

Affected by flag none

Input

2: $\{obj_1\ obj_2 \ldots obj_n\}$ Argument list

1: $\{ cmd_1\ cmd_2 \ldots cmd_k\}$ Command list

Output

1: $\{\{obj_{1,1} \ldots obj_{1,p}\}$

$\ldots$

$\{ obj_{n,1} \ldots obj_{n,q}\}$

Function Applies a sequence of commands to a list of objects. Each operation may return an arbitrary number of objects which are collected into a list.

Notes Commands (or functions) following the first command in the commands list must accept the arguments in form of a list.

See also LOP1, LVOP, METOP

Examples VARS **[ENTER]**

1: { PROG1 PROG2 PROG3 }

{ BYTES } LOPN **[ENTER]**

1: {{ #1AE1 307 }
{#11D1 8718 }
{#113D 214.5}}

LVOP

Category List manipulation

Affected by flag none

Input

3: { obj_1 obj_2 ... obj_n }

2: {obj_1 obj_2 ... obj_p}

1: { cmd_1 cmd_2 ... cmd_k}

Output

1: {obj_1 obj_2 ... $obj_{min(n,p)}$}

Function Applies each operator to the pairs of elements taken from the operand-list in levels 2 and 3.

Notes When the operand-lists have different size, exceeding objects of the bigger list are discarded. If you need to perform a calculation based on the current value of the counter, use the identifier 'idx' as counter. It will be replaced by its current value.

See also LOP1, LOPN, METOP

Examples

3: { 1 2 3 }
2: { 10 20 }
1: { + }

LVOP **[ENTER]**

1: { 11 22 }

3: { 3 5 }
2: { 1 1 }

1: { SWAP / }

LVOP **[ENTER]**

1: { 0.333333333333 0.2 }

3: { 1 "" (2,0) 30 5}
2: { 0 "" X 30 7}
1: { SAME { idx } IFT }

LVOP **[ENTER]**

1: { 2 4 }

MARK

Category Meta-object manipulation and stack manipulation

Affected by flag none

Input

Output 1: "MARK'

Function Puts the private marker on the stack.

Notes 'MARK is an unresolved global name. Do not store any object using such name.

The marker delimits an unbound meta-object. See the section 'Notes'
on page <$R[P#,C2M]13> for further details.

Internal routines make use of an alternate marker to avoid collisions between user programs.

See also C2M, L2M

Example See the example under SPLIT.

MATWRT

Category Spreadsheet and MatrixWriter

Affected by flag -15 through -18 and -45 through -50, -5.
User flags from 32 to 40

Input

1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

Output

2: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,q}\}$
$\ldots$
$\{Symb_{p,1}\ Symb_{p,2} \ldots Symb_{p,q}\}\}$

1: 1

1: 0

Function Allows interactive editing of a list-matrix and returns 1 if **[ENTER]** was pressed or 0 if **[ON]** was pressed.

Notes MATWRT is a command of library 822 (Omnibus).

See Chapter 2 for further information or read the Omnibus programming guide.

See also CSTMENU

Applications CALENDAR

Examples See the examples under EQUAL?, MULT and SQUARE?.

MEMBER

Category String function

Affected by flag none

Input

3: str_1

2: str_2

1: pos

Output

1: pos

Function

Returns the absolute position of the first character in Str_1 comprised in Str_2, starting from position pos. If no match is found, returns 0.

Notes

MEMBER is useful to skip text given a particular set of characters, typically punctuation characters or delimiters.

Frequently used string-constants have been stored in ROM to save user memory:

DIGIT$ = "0123456789"
alfaLOW$ = "ABCDEFGH...XYZ"
alfaUPP$ = "abcdefgh...xyz"

They are accessible in the fifth page of menu $&L in CST.

Try also 821 250 →Xlib **[ENTER] [EVAL]** and 821 251 →Xlib **[ENTER] [EVAL]**

MEMBER and SPAN are useful to check the input from the user, for the presence or absence of certain characters. Typically they are used to implement parser routines in conjunction with SPLIT, SUB and other string manipulation functions. Appendix H lists hidden string functions.

See also

SPAN, SPLIT

Examples

3: "123456789A12DEF"
2: "ABCDE"
1: 11

MEMBER **[ENTER]**

1: 13

META

Category Meta-object manipulation

Affected by flag none

Input

1:	n

Output

N+1:	1
...:	...
3:	n-1
2:	n
1:	n

Function Creates a numbered meta-object in increasing order.

Notes META is useful to create index arrays in conjunction with →ARRYor for numbering rows or columns in the Matrix Writer.

See also METOP, NDUPN, SRT, STRD

Applications CALENDAR

METOP

Category Meta-object manipulation

Affected by flag depends on the operators passed for evaluation

Input

N+1:	obj_1	
...:		...
3:		obj_n
2:		n
1:		list

Output

N+1:	obj_1	
...:		...
2:		obj_n
1:		n

Function Applies a sequence of commands evaluating to a single result to each object of the meta-object. The final result is a meta-object of the same size.

Notes The list in level 1 must contain a sequence of commands whose result is a single object. This convention, in practice, does not restrict the usage of METOP. When you use up an object doing some operation, you can refill the empty with a boolean or a dummy object.

See also LOP1, LOPN, LVOP

Applications L→TH, TH→L

Examples

```
7:  #45h
6:  37
5:  <22h>
4:  12
3:  #11h
2:  5
1:  { →SYS }
```

METOP **[ENTER]**

```
6:  <45h>
5:  <25h>
4:  <22h>
3:  <Ch>
2:  <11h>
1:  5
```

MGET

Category Symbolic matrix manipulation

Affected by flag none

Input

3: $\{\{Symb_{1,1} \ldots Symb_{1,k} \ldots Symb_{1,n}\}$
$\ldots$
$\{Symb_{m,1} \ldots Symb_{i,k} \ldots Symb_{m,n}\}$
$\ldots$
$\{Symb_{m,1} \ldots Symb_{m,k} \ldots Symb_{m,n}\}\}$

2: i

1: k

Output

1: $Symb_{i,k}$

Function Extracts an element from a list-matrix.

Notes No check is made on the type of the objects contained in the list, nor on the consistency of the structure of the matrix. This feature lets you extract elements from two-dimensional lists of arbitrary structure. If the pointee is missing an error is issued.

See also MPUT

Applications MATMENU, PRSYMB

MOVE

Category Meta-object manipulation

Affected by flag none

Input

N+4:	obj_1	
...:		...
...:		obj_{from}
...:		...
...:		obj_{to}
...:		...
...:		obj_{above}
...:		...
5:		obj_n
4:		n
3:		from
2:		to
1:		above

Output

N+1:	obj_1	
...:		...
...:		obj_{from-1}
...:		...
...:		obj_{to+1}
...:		...
...:		$obj_{above-1}$
...:		obj_{from}
...:		...
...:		obj_{to}
...:		obj_{above}
...:		...
...:		obj_n
1:		n

Function

Shifts the block of objects delimited by from and to above object above.

Notes

Top of stack contains the counter while level 2 contains he last object of the meta-object (as if it were the last element of a list). The first element of the meta-object lies on level n+1(after the execution of the command).

Input parameters from and to can be given in any order. If you specify for above a value greater than n, the section will shift after the tail of the meta-object. Despite of COPY, MOVE does not allow a value for destination between from and to.

See also

DELETE, COPY

Examples

6: "I stay here"
5: "I get moved"
4: "me too"
3: "I'll go up"
2: "End"
1: 5

2 3 5 MOVE **[ENTER]**

6: "I stay here"
5: "I'll go up"
4: "I get moved"
3: "me too"
2: "End"
1: 5

6: "I'll stay here"
5: "I get moved"
4: "me too"
3: "I'll go up"
2: "End"
1: 5

2 3 6 MOVE **[ENTER]**

6: "I stay here"
5: "I'll go up"
4: "End"
3: "I get moved"
2: "me too"
1: 5

MPUT

Category List manipulation and symbolic matrix manipulation

Affected by flag none

Input

4: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$
...
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,n}\}\}$

3: i

2: k

1: Any

Output

1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,k} \ldots Symb_{1,n}\}$
...
$\{Symb_{i,1}\ Symb_{i,2} \ldots$ Any $\ldots Symb_{i,n}\}$
...
$\{Symb_{m,1}\ Symb_{m,2} \ldots Symb_{m,k} \ldots Symb_{m,n}\}\}$

Function Replaces the value contained at location (i,k) with Any.

Notes Any means any object type.

See also MGET, MSYMB?

Applications MATMENU, PRSYMB

MREV

Category Meta-object manipulation

Affected by flag none

Input

N+1:	obj_1	
...:		...
2:		obj_n
1:		n

Output

N+1:	obj_n	
...:		...
2:		obj_1
1:		n

Function Reverses the order of the objects in the meta-object.

See also SRTD

Examples

1: [1 2 3 4 5 6]

OBJ→ OBJ→ DROP MREV →ARRY **[ENTER]**

1: [6 5 4 3 2 1]

MSBIT

Category Type conversion

Affected by flag -5 through -10 for binary integers only

Input 1: n 1: <n> 1: #n

Output 1: msbit

Function Returns the position of the most significant bit in the mantissa of the input number.

Notes Real numbers are automatically rounded to integers before the operation.

MSBIT returns a value between 0 and 20. 0 means no bit set.

The value returned complies with the following definition:

MSBIT=INT(LOG2(n))+1 for n#0.

MSBIT=0 for n=0.

Examples 1: h

MSBIT **[ENTER]**

1: 3

MSYMB

Category Symbolic matrix manipulation

Affected by flag none

Input

1: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$}
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$ }}

Output

2: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$ }
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$ }}
1: 1

or

m*n+2: $Symb_{1,1}$
m*n+1: $Symb_{1,2}$
... : ...
... : Syntax: Obj
... : ...
3: $Symb_{m,n}$
2: { 1 3 }
1: 0

Function Checks the contents of the list-matrix. Any object whose type is not Real, Complex, Unit, Symbolic or Global is tagged with the string "Syntax".

Notes SYMB?does not check dimensions. To check dimensions use DIMS.

See also CONFORM?, DIMS, EQUAL?, SQUARE?

MULT

Category Symbolic matrix manipulation

Affected by flag -3 (Numerical result)

Input

2: $\{\{SymbA_{1,1}\ SymbA_{1,2} \ldots SymbA_{1,n}\}$

...

$\{SymbA_{m,1}\ SymbA_{m,2} \ldots SymbA_{m,n}\}\}$

1: $\{\{SymbB_{1,1}\ SymbB_{1,2} \ldots SymbB_{1,p}\}$

...

$\{SymbB_{n,1}\ SymbB_{n,2} \ldots SymbB_{n,p}\}\}$

Output

1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,p}\}$

...

$\{Symb_{m,p}\ Symb_{m,p} \ldots Symb_{m,p}\}\}$

Function Returns the row by column product of two arrays (either symbolic or numerical).

Notes Arrays must have compatible dimensions, that is, if the first matrix is (m,n), the second must be (n,p). We called this special condition "conformability" When the aforementioned condition is violated, a special error is issued. When flag -3 is set (Numerical result), errors may happen if symbolic expressions cannot be resolved to a numeric value. In-place multiplication is performed by FACTOR. The last

example below shows you the difference between row-by-column and in-place multiplication.

You can supply either a numerical or symbolic array; the result will be numerical if, and only if, the array and the value are numerical (no matter if the value is real or complex).

See also CONFORM?, FACTOR

Applications MATMENU, PRSYMB

Examples

```
2:      {{ 1 2 }{ 'X' 'X-1' }}
1:      {{ '-X' -1 }}
```

MULT **[ENTER]**

```
1:      {{ '-X-2' }
         { 'X*-X-(-1+X)' }}
```

```
2:      {{ 1 2 }{ 6 7 }}        {{ 1 2 }{ 6 7 }}
1:      {{ 2 1 }{ 3 4 }}        {{ 2 1 }{ 3 4 }}
```

MULT **[ENTER]** FACTOR **[ENTER]**

```
1:      {{ 8 9 } { 33 34 }}     {{ 2 2 }{ 18 28 }}
```

Let's create a program asking the user for two arrays but forcing the user to enter the second array according with the dimensions of the first one. We will use the program ASK, explained under EQUAL? and the capabilities of MATWRT.

```
«
        33 CF
        {{}} ASK        Turns on Symbolic mode.

DIMS REV '?' SWAP
        2  1 PUT
        CONSTMAT
        34 SF 35 CF     Creates a template
                        for the second matrix; locks
                        matrix rows; see under
                        MATWRT.

        ASK MULT        enters the second matrix,
        34 CF           returns the product
»                       and unlocksmatrix rows.
```

The program explained above is stored under the name ASK&MULT in the directory EXAMPLES.1C.

NDUPN

Category Meta-object manipulation

Affected by flag none

Input

```
2:          Obj
1:          n
```

Output

```
N+1:  Obj
...:        ...
2:          Obj
1:          n
```

Function Creates a meta-object by duplicating n times a given object.

Notes If n=0 then NDUPN creates a null meta-object.

See also META

Examples

```
1:      { hello }

4 NDUPN [ENTER]

5:      { hello }
4:      { hello }
3:      { hello }
2:      { hello }
1:      4

1:      { hello }
0 NDUPN
1:      0
```

NIP

Category Stack manipulation

Affected by flag none

Input

2: Obj_A

1: Obj_B

Output

1: Obj_B

Function Removes the object on the second level from the stack.

See also AAB, BAA, BAB, BBA, BCAC, BCDA, CAB, CBA

NULL

Category Type conversion

Affected by flag user flag 31

Input 1: obj

Output 1: obj (null)

Function Replaces the input object with the null object of the same type respect to the addition operation.

Only the objects listed below have a correspondent null object.

Type	Null element
0	0
1	(0,0)
2	""
3	[0...0] or [[0...0]...[0...0]]
4	[(0,0)...(0,0)] or [[(0,0)...(0,0)]...[(0,0)...(0,0)]]
5	{}
6	0
7	0
9	0
10	#0h
11	Blank 0x h if flag 31 is clear. Blank wx0 if flag 31 is set
12	inherits from the ancestor if defined.
13	0_unit
20	<0h>

Notes

A Tagged object inherits ancestor's type.

Polymorphism is a property of RPL language (at user level). It allows you to design object-independent algorithms. Of course some operations make sense only with certain objects, but setting the algorithm free from object slavery, you will save time later, when you need to recycle the routine for doing some other task.

The NULL command lets you design recursive algorithms or loop structures independently from the input object type. Typically such algorithms need some initialization code in order to start a chain calculation. The + (plus) Command is the most flexible operator built in the HP48. The NULL command lets you initialize every routine based on concatenation or addition without knowing in advance the object type.

See also

RPT

Examples

:tag:{ 1 2 3 } NULL **[ENTER]**

```
1:      { }
```

```
1:      [[ 0 1  5]
         [ 4 3 -2]]
```

NULL **[ENTER]**

```
1:      [[ 0 0 0 ]
         [ 0 0 0 ]]
```

PARSE

Category String function

Affected by flag -5 through -10 (wordsize), -15 and -16 (coordinate system), -17 and -18 (angle mode)

Input

1: str

Output

		4:	str
3	<last>		
2:	prg "characters"	2:	
1:	External (TRUE)	1:	External (FALSE)

Function Performs the parsing of the input string. If no syntax error is detected an object containing the executable code is returned along with the system boolean TRUE. Otherwise the original string, the absolute position of the last character scanned and the text containing the syntax error are returned along with the system boolean FALSE.

Notes By extending the system parser, it is possible to handle unsupported object types like system binaries. The example given below shows a possible technique, you could enhance at your will.

on-the-fly parser handling system binaries:

« 0 → current initializes replacement counter

« {} '$SUBST' STO initializes temporary storage area

DO PARSE →TorF parses input string

IF NOT THEN	if error then
NIP	take apart the string
SPLIT OVER DUP	
'$SUBST' STO+	and save it.
CAB + SWAP	take the rest of the string
"$sub" 'current' INCR +	give a name to the substring
REPLACE DROP	replace all the occurences of the string with the identifier
+	reconstruct the string for parsing.
0	prepare to parse again
ELSE 1	exit parsing
END	
UNTIL	
END	loopback if error during parse
→PRG	make it a program
IF current THEN	if any replacement
$SUBST	
OBJ→ DUP 2 +	retrieve strings and prepare for loop

```
ROLL 1 ROT
FOR i
"$sub" i + OBJ→          build indentifier
ROT 1 SPLIT              parse unrecognized text
ROT  DROP NIP "#"
SWAP + EndOfString SPLIT
DROP DUP
IF"" SAME
THEN DROP
ELSE +
END
OBJ→ →SYS
REPLACE DROP             convert into a sysbin and
                         replace all the occurrences.
EXT→
END » '$SUBST'


PURGE EVAL »
'Parse' STO              Try it on this string
"« (0,0) 3 0 360 60 <1h> <2h> 821 247 →Xlib
»"
```

```
1:      « (0,0) 3 0 360
          60 <1h> <2h>
          821 247 →Xlib »
```

EVAL [GRAPH]

PKMETA

Category Meta-object manipulation

Affected by flag -56 Beep

Input

N+5:	obj_1
...:	...
6:	obj_n
5:	n
4:	begin
3:	current
2:	lines
1:	row

Output

[ENTER]			**[ON]**	
N+5:		obj_1		
...:		...	N+4:	
	obj_1			
6 :		obj_n	...:	...
5:		n	5:	
	obj_n			
4:		begin	4:	n
3:		current	3:	
	begin			
2:		objcurrent+begin	2:	current
1:		1	1:	0

Function

Shows a catalog of a specified number of lines beginning on a specified display line. The selection of the object is interactive and retains the functionalities of the built-in catalog.

Notes

Command parameters are so defined:

obj1
...
objn
n

input meta-object;

begin

index of the first object on which beginning the page minus one (from 0 to n-lines);

current

current element within the page (from 1 to lines);

lines

height of the page in lines (from 1 to 8-row); row starting row of the display (from 1 to 7). When using a value less than 2, FREEZE is required.

Values exceeding the limits are rejected with an error.

Selecting an object means moving the pointer to the line containing the object and press **[ENTER]**. Once you press **[ENTER]**the catalog is exited and the selected object is duplicated as shown in the stack diagram. The selection may be aborted by pressing **[ON]**. In this case only the last pointer position is returned. Arrow keys perform pointer movements in the same way the built-in catalog allows.

Notes

Command parameters are so defined:

obj1
...
objn
n

input meta-object;

begin

index of the first object on which beginning the page minus one (from 0 to n-lines);

current

current element within the page (from 1 to lines);

lines

height of the page in lines (from 1 to 8-row); row starting row of the display (from 1 to 7). When using a value less than 2, FREEZE is required.

Values exceeding the limits are rejected with an error.

Selecting an object means moving the pointer to the line containing the object and press **[ENTER]**. Once you press **[ENTER]**the catalog is exited and the selected object is duplicated as shown in the stack diagram. The selection may be aborted by pressing **[ON]**. In this case only the last pointer position is returned. Arrow keys perform pointer movements in the same way the built-in catalog allows.

PKMETA is very flexible because each action associated to a key may be redefined. Each defined key must be associated to a program stored in user memory. the following table summarizes the PKMETA auxiliary programs and data structures and the default keys along with their actions:

Global Name	Keycode	Action
KEYS	n/a	Keeps the list of defined keys. Each element of the list is a list containing two system binaries. The first number represents the absolute key number 0h to 2Dh, the second represents the shift plane 1h to 6h. This encoding system matches KEYWAIT format.
ACTIONS	n/a	Holds the list of the actions associated to the keys. The list must always have at least one element. The first element is a name of the program that must be called when an undefined key is pressed. By default this name is BADKEY. The second element corresponds to the first keycode stored in KEYS and so on. ACTIONS must always contain a number of actions equal to SIZE(KEYS)+1. Otherwise a special error code will be issued.
BADKEY	n/a	The action taken when an undefined key has been pressed.
ATTN	<2Dh> <1h>	The action associated to the pressing of **[ON]**. By default it aborts the selection.
DOWNARR	<11h> <1h>	The action associated to the pressing of **[↓]**. It moves the pointer downwards, eventually scrolling up the page by one line at a time.

UPARR	<0Bh>	The action associated to the pressing of [↑].
	<1h>	It moves the pointer upwards, eventually scrolling down the page by one line at a time.
ENTER	<19h>	
	<1h>	The action associated to the pressing of **[ENTER]**. By default it selects the current item and exits the catalog.
PGUP	<0Bh>	
	<2h>	The action associated to the pressing of **[Gold][↑]**. It displays the previous page.
PGDOWN	<11h>	
	<2h>	The action associated with the pressing of **[Gold] [↓]**. It displays the next page.
TOP	<0Bh >	
	<3h>	The action associated to the pressing of **[Blue] [↑]**. Moves the pointer to the top of catalog.
BOTTOM<11h>		
	<3h>	The action associated to the pressing of **[Blue] [↓]**. Moves the pointer to the bottom of the catalog.
ENHANCE	n/a	The routine that highlights the current line.
NOR	n/a	The routine cancelling the highlight from the current line.

All the programs described above can be modified at your will. There are 5 parameters stored in temporary variables which contain the information needed to perform an action. They hold the current value of the input parameters passed on the stack upon entry. Please note that these values are stored as system binaries and not as real numbers.

n	counter	Retains the total number of elements of the meta-object (the counter)
o	begin page	This is the offset to the first line (element) where the page begins.
c	current element	This is the current element within the page.
r	row	This is the row of the display whereon the page is anchored.
h	lines	Total number of lines per page.

These variables can be accessed by name or by their order in the temporary variables chain. If you know the entry points to recall and store temporary variables by creation order, you can use them freely. The variables have been created in the order shown in the table above, that is n is the fifth of the chain and h is the first. To recall h use entry point #613B6h. However the safest way is to recall and store them by name.

When you perform an action such adding or deleting objects from the stack, remember to update the values stored in the local variables. While PKMETA is running, the stack contains the body of the meta-object depleted of the counter.

You can add or modify or change name to actions, by simply modifying the ACTIONS list and updating, wherever necessary the KEYS list. All the customization of the command is up to you.

PRG →

Category Program and type conversion

Affected by flag none

Input

1:	prg	1:	list	1:	Algebraic

Output

N+1:	obj_1		
...:		...	
2:		obj_n	
1:		n	

Function Splits a program in meta-object format.

Notes A program is a collection of objects and pointers (threads) like a list or an algebraic expression. The main difference between programs and other composite object lies in its direct execution capability opposite to the indirect execution capabilities of lists and algebraics. Direct execution means that once the prolog of the program is executed, it starts executing objects within the program, while lists and algebraics merely push themselves on the stack. Once they are on the stack these objects can be executed via EVAL. Using PRG→ you will be able to modify in whatever manner you want a compiled program, deleting, moving, changing the objects it contains. The possiblity to modify compiled programs directly on the stack makes much faster editing session of large programs, typically when you need to swap objects or make little changes in the

source. Nevertheless PRG→ opens a wide range of applications dealing with program editing and in fact the SmartROM uses heavily this kind of commands. We suggest you try to edit a program via Interactive Stack.

May be you ramain quite surprised after expanding a User RPL program on the stack. In fact the built-in parser often adds hidden threads to perform safely dangerous operations like pushing an identifier on the stack. Pay attention not to delete these hidden threads ! Moreover, as you will see, programming structures collect the commands between delimiters in a program object. To expand this kind of program, you need to move the object on the first level of the stack and call PRG→ once more, then after editing it, you must recontruct the program with →PRG and move the program back to the original position.

See also EXT→, →EXT, →PRG

Applications L→TH, SHRINK, TH→L, UPTRIM

Examples

```
« IF 0 >>
  THEN DROP SWAP
  END »
```

PRG→ **[ENTER]**

```
9:    «
8:    IF
7:    0
6:    »
5:    THEN
4:    DROP SWAP
3:    END
2:    »
1:    8
```

As you may see , when you split a program in the stack format, several interesting aspects come out.

The french arrows are stand-alone objects. What is their role ? In brief we can summarize it as follows:

they are effectively used like delimiters by the parser to recognize user programs.

upon evaluation of the program they check the state of the **[ON]** key. If the user pressed this key, the process is immediately stopped and the execution is transferred to the calling programas if an error condition occurred. This means that the IFERR
construct traps the **[ON]**key press too.

Therefore by depleting a program of the delimiters you can prevent the user from interrupting the program and contemporarily you make a program virtually uneditable (in the sense that you cannot reparse it due to the lack of delimiters). This is exactly what the application UPTRIM does.

The second aspect worth noting is represented by the sequence DROP SWAP.

Why aren't separated these two commands ? Because the parser collects them in a single object for the sake of efficiency.

→PRG

Category Meta-object manipulation, program editing and type conversion

Affected by flag none

Input

N+1:	ext_1
...:	...
2:	ext_n
1:	n

Output

1:	prg

Function Builds up a program object from a meta-object.

Notes See under PRG→.

See also EXT→, →EXT, PRG→

Applications →TH, SHRINK, TH→L, UPTRIM

→R

Category Type conversion

Affected by flag -5 through -10 when the input is a binary integer

Input 1: <<nh>> 1: #n 1: Char

Output 1: n

Function Converts the input number into a real.

Notes →R accepts also real numbers as input. This feature sets you free to use →R also when the object type should not require any conversion.

See also →B, →Char, EXT→, →EXT, →SYS

Examples Let's create a program that returns the length of an hex string as a real number. We will use an internal routine for doing the dirty job. Be sure to set the wordsize to 20 bit (or higher) and hex mode before typing the program.

```
«
DUP TYPE            Binary string required
IF 10 ==            any mismatch ?
THEN        No, proceed
```

```
#05616h
SYSEVAL         returns the length as system
                binary.

→R              translates into a real number

ELSE            Error

#202h DOERR     Bad Argument type

END


»
```

done.

Save the program as HSIZE, we will use it later.

RDOWN

Category Stack manipulation

Affected by flag none

Input

N+2:	obj_1
...:	...
T+2:	obj_t
...:	...
3:	obj_n
2:	n
1:	t

Output

N:	obj_t
...:	...
N-T+1:	obj_n
N-T:	obj_1
...:	...
1:	obj_{t-1}

Function Rolls down n objects t times.

Notes The command is smart enough to choose the best roll direction (upwards or downwards). Rolling down 100 objects 99 times is a good exercise of aerobyc dance for your 48, but it is not that kind of exercise we really need to do. We had better to roll up 100 objects one time !

See also RUP,XLVLS

RDROP

Category Stack manipulation

Affected by flag none

Input

...:	...
U+2:	liv_u
...:	...
P+2:	liv_p
...:	...
3:	liv_1
2:	p
1:	u

Output

.. :	...
U :	liv_{u+1}
P-1:	liv_{p-1}
...:	...
1:	liv_1

Function Deletes the segment of the stack delimited by levels p and u.

Notes p and u can be given in any order.

See also RDUP, SHIFT, XLVLS

Examples

5: "first"
4: "to get rid"
3: "to get rid"
2: "second-last"
1: "last"

4 3 RDROP **[ENTER]**

3: "first"
2: "second-last"
1: "last"

RDUP

Category Stack manipulation

Affected by flag none

Input

....:	...
P+3:	liv_p
....:	...
U+3:	liv_u
....:	...
D+3:	liv_d
....:	...
4:	liv_1
3:	p
2:	u
1:	d

Output

....:	...
....:	liv_p
....:	...
....:	liv_u
....:	...
....:	liv_{d-1}
....:	liv_p
....:	...
D+1:	liv_u
D :	livd
....:	...
1:	liv_1

Function

Copies the segment of the stack delimited by p and u above level d.

See also

RDROP, SHIFT, XLVLS

Examples

5: "first"
4: "get copied"
3: "get copied"
2: "second-last"
1: "last"

4 3 1 RDUP **[ENTER]**

7: "first"
6: "get copied"
5: "get copied"
4: "second-last"
3: "get copied"
2: "get copied"
1: "last"

REPLACE

Category String function, list manipulation and program editing.

Affected by flag none

Input

3:	str_1	3:	hex_1	3:	Prg	3:	List
2:	str_2	2:	hex_2	2:	obj_1	2:	obj_1
1:	str_3	1:	hex_3	1:	obj_2	1:	obj_2

Output

2:	str	2:	hex	2:	Prg	2:	List
1:	Replaced:n						

Function Substitutes all the occurences of the search-key with the object or (hex) string given and returns the total number of replacements.

Notes The replacements are limited to objects stored in user memory. Because some internal routines are recursive, the search level is limited to threads stored in RAM. This preserves from endless loops. Internal routines of the SmartROM are able to perform selected substitutions at arbitrary depth within threads. Refer to the Hidden Commands Reference for more information on this topic.

See also FIND, LOC, SPLIT

Examples

Let's create a program for replacing
a sequence of objects within a program.

Set Hex Mode and Word size to 64 before starting.

Retrieve the program HSUB

```
«

{{5 8 9}
 {5 9}
 {5 9}}          3 arguments;

IF CHST? NOT     tests whether the object on
                 the third level is a
                 program or list and the first
                 two levels contain lists or
                 algebraics.

THEN             Ok, proceed.

→Xlib            Translates into hex format

EVAL 6 OVER

HSIZE            returns the length of the hex string

5 -              returns length-5

HSUB             gets the substring [6;length-5]

SWAP             prepares second object

821 275 →Xlib    translates into hex format

EVAL 6 OVER
```

HSIZE	returns the length of the hex string
5 -	
HSUB	gets the substring [6,length-5]
ROT	
821 275 →Xlib	translates into hex format
EVAL CBA	reconstructs original order
REPLACE	executes the replacement
SWAP	
→Xlib	translates back into RPL object
EVAL SWAP	Done.
ELSE DROP	
"Stack parameters:	3: Prog, List, Alg 2: List, Alg 1: List, Alg"

3 DISP diagram — When parameters mismatch, the stack is shown and the program aborted

3 FREEZE

END » — Save the program.

You should get a program whose checksum is #3A6Bh and whose length is 357.5 bytes. The program is stored on the disk in the subdirectory MANUAL of the file EXAMPLES.1C under the name BREPLACE.

```
3:  « 4 ROLL SWAP DROP
    { OVER 4 ROLL } »
2:  ROLL
1:  ROLLD
```

REPLACE **[ENTER]**

```
2:  « 4 ROLLD SWAP DROP
    { OVER 4 ROLLD} »
1:  REPLACED: 2
```

```
3:  "ABC ABC ABC"
2:  "ABC"
1:  "HELLO"
```

REPLACE **[ENTER]**

```
2:  "HELLO HELLO HELLO"
1:  REPLACED: 3
```

REV

Category String function

Affected by flag -5 to -10 (binary integers only)

Input 1: str 1: {obj_1 obj_2 ... objn} 1: #abcde

Output 1: str 1: {objn obj_{n-1} ... obj_1} 1: #edcba

Function Reverses the order of the characters for strings, the order of the objects for lists and the order of the digits for binary integers.

Notes Binary integers are reversed according with their actual size. User binary integers may be no longer than 16 nibbles (in hex mode). However the 48 can handle binary integers of arbitrary size. For example, when you apply BYTES to an object, the binary checksum you get is always 4 nibbles long, no matter the current wordsize is. Of course the display shows it according to the wordsize, but its size remains 4 nibbles.

See also MREV

Examples

See also the examples under CONSTMAT.

1: "123456789A12DEF"

REV **[ENTER]**

1: "FED21A987654321"

1: { A B C }

REV **[ENTER]**

1: { C B A }

Supposing current wordsize of 64 bit.

1: #123456h

REV **[ENTER]**

1: #6543210000000000h

Create a command to find the last occurence of a substring in a given string:

```
«
  OVER SIZE
  → s l
  « REV s REV
    POS DUP


    IF 0 >> THEN
       l SWAP - 1 +
    END
  »
»
```

takes the length of the string and stores it as lreverses the string and the substring and seeks it; saves the value on the stackif found thenconverts leftwards position into a rigthwards absolute position otherwise returns zero.

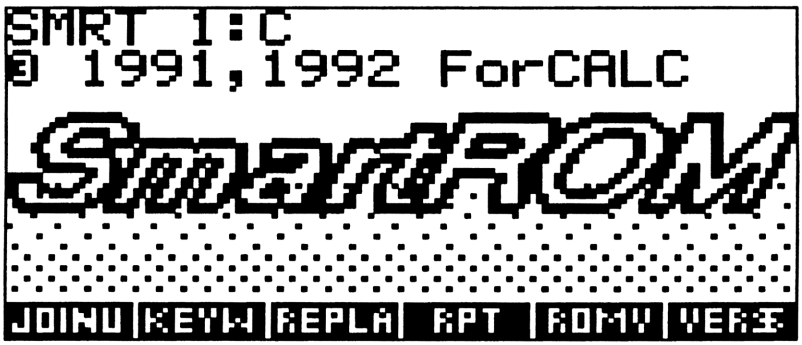
ROMV

Category ROM Version

Affected by flag none

Input none

Output



Function Shows information about the SmartROM.

Notes The routine uses random values to set the background color and the thickness of the outlines. Characters are defined by vectors stored in a systembinary array and are drawn by a special routine. This routine is quite generic and can be used to draw any shape. More information in the Hidden Commands Reference manual.

Example ROMV **[ENTER]**

ROWCOL

Category String function

Affected by flag none

Input

2: str
1: pos

Output

3: str
2: row
1: col

Function Computes the coordinate of the absolute position of a character in terms of rows and columns, by counting the linefeeds contained in the input string.

Notes The coordinates returned by ROWCOL can be used as parameters in the input list of command INPUT to place the cursor at a certain point within the editor. Typically this method is used to place the cursor on a particular occurence of a substring, previously found with LOC or POS.

Examples

"HELLO BOYS, THIS IS THE THIRD LINE OF TEXT"

[ENTER]

DUP SIZE ROWCOL **[ENTER]**

3: "HELLO BOYS, THIS I..."
2: 3
1: 18

RPT

Category String function, list manipulation, utility, graphics

Affected by flag User flag 31

Input

2:	str		2:	n				
1:	n		1:	str				
or								
2:	{ Obj_1... Objn}		2:	n				
1:	n		1:	{ Obj_1... Obj_n}				
or								
2:	#b	2:	n	2:	Grobw x h	2:	n	
1:	n	1:	#b	1:	n		1:	Grobw x h
or								
2:	prg	2:	n	2:	Global	2:	n	
1:	n	1:	prg	1:	n	1:	Global	

Output

1:	str...str	1:	{ Obj_1... Obj_n... Obj_1... Obj_n}
1:	#bbbb..b	1:	$Grob_{(w*n)\ x\ h}$ if flag 31 is clear
		1:	$Grob_{wx(h*n)}$ if flag 31 is set

or

N:	Obj
...:	...
1:	Obj

Function

Chains data objects n times or executes n times a given procedure or identifier.

Notes

RPT is affected by flag 31 when chaining grobs. When the flag is clear, RPT chains the grobs horizontally, otherwise chains the grobs vertically. The example given under JOINUP explains in detail how it works.

RPT is one of the most flexible commands of the SmartROM. Thanks to its fast loop generator, it can link string and list quicker than any other command seen to date. Try with a string of ten characters repeated 100 times (with at least 2K free).

RPT is useful at most when you need to perform iterated operations without referencing counters. RPT does not preserve stack contents, so that you can push or drop objects from the stack freely.If you have a DEMO program and you want to iterate it almost indefinitely, try this:

'DEMO' 1000000 RPT.

The loop normally cannot be interrupted. If you need to interrupt
it press **[ON]** [C].

In the examples given below procedures are standard programs. Nevertheless you can push on the stack individual commands by doing so:

{ MAX } OBJ→ DROP

See also

METOP

Examples

See also the example given under JOINUP.

2: { 1 2 3 }
1: "ABC"

2 { 3 RPT } METOP **[ENTER]**

3: { 1 2 3 1 2 3 1 2 3 }
2: "ABCABCABC"
1: 2

« RAND 10 * IP » 10 RPT

10: 3
9: 7
8: 5
7: 2
6: 7
5: 0
4: 8
3: 3
2: 6
1: 0

« MAX » 10 RPT **[ENTER]**

1: 8

Suppose you want to move a hundred variables from user memory to PORT
1 where you have a 128 K RAM:

1: {Name1 Name2 ... Name100}

OBJ→ « DUP RCL BAA PURGE 1 →TAG STO » RPT **[ENTER]**

RUP

Category Stack manipulation

Affected by flag none

Input

N+2:	obj_1	
...:		...
T+2:	obj_t	
...:		...
3:		obj_n
2:		n
1:		t

Output

N:		obj_{t+1}
...:		...
T+1:	obj_n	
T :		obj_1
...:		...
1:		obj_t

Function Rolls up n objects t times.

Notes The command is smart enough to choose the best roll direction (upwards or downwards). Rolling up 100 objects 99 times is silly. It is better roll down 1 time !

See also RDOWN, XLVLS

Example

```
«
 4 PICK SIZE
 →k t m n

 « k SPLIT + m SPLIT
   4 ROLLD<N>+ OBJ→

                DUP 'm' STO
   t DUP ABS SWAP 0
   IF >> THEN RUP
   ELSE RDOWN
   END
   m →++
 »
»
```

Write a command for rotating m objects of t places starting from object k; if t is greater than 0 the rotation takes place leftwards else rightwards:

{ 1 2 3 4 5 6 7 8 9 }

3 3 5 ROTATE **[ENTER]**

{ 1 2 6 7 3 4 5 8 9 }

3 -3 5 ROTATE **[ENTER]**

{ 1 2 3 4 5 6 7 8 9 }

1 2 9 ROTATE **[ENTER]**

{ 3 4 5 6 7 8 9 1 2 }

SHIFT

Category Stack manipulation

Affected by flag none

Input

....:	...
P+3:	liv_p
....:	...
U+3:	liv_u
....:	...
D+3:	liv_d
....:	...
4:	liv_1
3:	p
2:	u
1:	d

Output

....:	...
P :	liv_{p-1}
P-1:	liv_{u+1}
....:	...
....:	liv_{d-1}
....:	liv_p
....:	...
D+1:	liv_u
D :	liv_d

..: ...

1: liv_1

Function

Moves the segment of the stack between level p and level u above level d.

Notes

Stack-oriented commands require one parameter less than their meta-object-oriented counterparts. On the other hand, meta-objects let you count objects.

See also

RDROP, RDUP, XLVLS

Examples

5: "I stay here"
4: "I get moved"
3: "me too"
2: "I'll go up"
1: "End"

4 3 1 SHIFT **[ENTER]**

5: "I stay here"
4: "I'll go up"
3: "I get moved"
2: "me too"
1: "End"

SPAN

Category String function

Affected by flag none

Input

3: str_1

2: str_2

1: pos

Output 1: pos

Function Returns the absolute position of the first character in Str1 not comprised in Str2, starting from position pos. If no match is found, returns 0.

Notes A typical usage of SPAN occurs when checking for the presence of extraneous characters, especially when the input string comes from the user. Suppose the user must enter a numeric value without decimal point and Exponent. To check the string you can do so:

3: "758833"	This is the string given by the user;
2: "0123456789"	this is the string containing allowed characters;
1: 1	beginning position.

SPAN **[ENTER]**

1: 0

Another frequent usage is when you need to skip blanks between words. In this case the test string must contain only a blank : " ". the position returned (if any) is that of next non-blank character. You could also ignore periods or any other punctuation by appending them to the test string.

" .;," lets you skip blanks, periods, commas and semicolons.

Frequently used string-constants has been stored in the library to
save memory:

DIGIT$ = "0123456789"
alfaLOW$ = "ABCDEFGH...XYZ"
alfaUPP$ = "abcdefgh...xyz"

They are accessible in the fifth page of menu $&L in CST.

Try also 821 250 ÄXlib **[ENTER]** **[EVAL]** and 821 251 ÄXlib **[ENTER]** **[EVAL]**

See also MEMBER

Examples

3: "123456789A12DEF"
2: "1234567890"
1: 3

SPAN **[ENTER]**
1: 10

SPLIT

Category String function and list manipulation

Affected by flag none

Input

2:	str	2:	{ obj_1 obj_2 ... obj_p ... obj_n }
1:	p	1:	p

or

2:	p	2:	p
1:	str	1:	{ obj_1 obj_2 ... obj_p ... obj_n }

or

2:	str	2:	{ obj_1 obj_2 ... obj ... obj_n }
1:	strsub	1:	obj

or

2:	hex
1:	hexsub

Output

3:	str_1	3:	hex_1	3:	$\{obj_1 \ldots obj_{p-1}\}$
2:	str_{sub}	2:	hex_{sub}	2:	obj_p
1:	str_3	1:	hex_1	1:	$\{obj_{p+1} \ldots obj_n\}$

Function

Splits the string or the list in three chunks.

Notes

Empty string or lists are valid input objects.

If p is greater than the total size of the object, it is considered as SIZE(obj). If p is equal to 0 or the search-key is missing an error is issued.

Please note that the original object can be recontructed with two consecutive + (addition) operations.

Examples

Let's create now a program to format a string so that each line won't exceed n characters, unless a word is longer than n. The program is stored in APP.1C under the name 'FORMAT$'.

```
« → n
  « → LINES

     { MARK SWAP
     WHILE
     DUP SIZE n >

     REPEAT
       n SPLIT CAB +

<N>       REV DUP
        " ;,:" 1 MEMBER

        IF  DUP THEN
           SPLIT + REV
           10 CHR +
           CAB
           REV SWAP +
```

```
    ELSE
      DROP REV
      10 CHR +
      SWAP
    END
  END
  C2M 1 -
  { + } OBJ→ DROP
  RPT
  SWAP DROP
  }<N>METOP

  LINES→
 »
»
```

This stores maximum line length and splits the string in several lines. Defines the procedure that gets repeated on each line. While the length is greater than the maximum length splits at n and saves the rest on the stack , then checks if we are in the middle of a word and seeks the best break position.
If found then:

- breaks the line,
- adds a linefeed
- ,reconstructs the string
- and prepares for next iteration
- otherwise if a line break cannot be found breaks at n
- and adds a linefeed
- then prepares for next iteration
- counts new lines
- and makes them a single one
- Deletes stack marker
- Applies the procedure to every line
- Reconstructs the string

Try now FORMAT$ on a list of numbers with several values:

{ 1 2 3 4 5 6 7 8 9 }

"{ 1
2 3
4 5
6 7
8 9 }"

5 FORMAT$

"{ 1 2 3
4 5 6 7
8 9 }"

7 FORMAT$

2: "123456789A12DEF"
1: 5

SPLIT **[ENTER]**

3: "1234"
2: "5"
1: "6789A12DEF"

2: "123456790"
1: "456"

SPLIT **[ENTER]**

3: "123"
2: "456"
1: "790"

2: { 1 "abc" 'x/y'}
1: "abc"

SPLIT **[ENTER]**

3: { 1 }
2: "abc"
1: { 'x/y' }

SQUARE?

Category Symbolic matrix manipulation

Affected by flag none

Input 1: {{ $Symb_{1,1}$ ... $Symb_{1,n}$ }
...
{ $Symb_{n,1}$ ... $Symb_{n,n}$ }}

Output 3: {{ $Symb_{1,1}$ ... $Symb_{1,n}$ }
...
{ $Symb_{n,1}$ ... $Symb_{n,n}$ }}
2: { n n }
1: 1

2: {{ $Symb_{1,1}$ ... $Symb_{1,n}$ }
...
{ $Symb_{m,1}$ ... $Symb_{m,n}$ }}
1: 0

Function Tests whether a list-matrix is square. If yes it returns 1 along with matrix dimensions, otherwise returns 0. The result can be interpreted as a meta-object.

See also DETERM, EQUAL?, CONFORM?

Applications INVRT, MATMENU, PRSYMB

Examples

Let's create a program that ask the user to modify a matrix until it is square.

```
«       33 CF 34 CF 35 CF


        {{ }}

        DO ASK
        UNTIL
                SQUARE?
        END DROP
»
```

```
1:      {{ X Y Z }
         { 1 3 -2 }}
```

SQUARE? **[ENTER]**

```
2:      {{ X Y Z } { 1 3 -2 }}
1:      0
```

SRDIFF

Category List manipulation

Affected by flag none

Input

2:	List
1:	obj

Output

1:	pos

Function Returns the position of the first object different than Obj. If all objects are the same as Obj returns 0.

Notes The implementation of a routine performing a test on all elements of a list is straightforward in internal RPL.

Pass 1 Create a test procedure taking two objects from the stack and returning a system boolean (See TRUE and FALSE).

Pass 2 Store it in a variable for easier reference.

Pass 3 Push on the stack the list and the object being tested

Pass 4 Push on the stack the name of the variable or directly the test procedure.

Pass 5 #64426 SYSEVAL **[ENTER]**

See also SRGE, SRGT, SRLE, SRLT

Examples

2: { 1 1 1 1 1 2 3 }
1: 1

SRDIFF **[ENTER]**

1: 6

SRGE

Category List manipulation

Affected by flag none

Input

2: $\{ n_1 n_2 \ldots n_p \}$

1: n

Output

1: pos

Function Returns the position of the first real number greater or equal than n, otherwise returns 0.

Notes See under SRDIFF.

See also SRDIFF, SRGT, SRLE, SRLT

Examples

2: { 1 1 1 1 1 3 6 }
1: 2

SRGE **[ENTER]**

1: 6

SRGT

Category List manipulation

Affected by flag none

Input

2: $\{n_1\ n_2 \ldots n_p\}$

1: n

Output

1: pos

Function Returns the position of the first real number greater or equal than n, otherwise returns 0.

Notes See under SRDIFF.

See also SRDIFF, SRGE, SRLE, SRLT

Examples

2: { 1 1 1 1 1 3 6 }
1: 3

SRGT **[ENTER]**

1: 7

SRLE

Category List manipulation

Affected by flag none

Input

2: { n_1 n_2 ... n_p }

1: n

Output

1: pos

Function Returns the position of the first real number greater or equal than n, otherwise returns 0.

Notes See under SRDIFF.

See also SRDIFF, SRGE, SRGT, SRLT

Examples

2: { 3 3 4 2 5 3 6 }
1: 2.9

SRLE **[ENTER]**

1: 4

SRLT

Category List manipulation

Affected by flag none

Input

2: $\{ n_1 n_2 \ldots n_p \}$

1: n

Output

1: pos

Function Returns the position of the first real number greater or equal than n, otherwise returns 0.

Notes See under SRDIFF.

See also SRDIFF, SRGE, SRGT, SRLE

Examples

2: { 3 4 10 2 1 3 6 }
1: 2

SRLT **[ENTER]**

1: 5

SRT

Category Meta-object manipulation

Affected by flag none

Input

N+1:	obj_1
...:	...
2:	obj_n
1:	n

Output

N+1:	obj_1
...:	...
2:	obj_n
1:	n

Function Sorts the data in ascending order.

Notes Objects must be compatible with the << operator (less than). To this category of objects belong:

Global nameshandled as stringsComplex numbersordered by real partReal numbersBinary integersStringsSystem BinariesTagged objects falling in one of the classes listed above except global names and complex numbers.

If you want to sort local names you need first to translate into global names, then use SRT and convert them back to locals. To convert a local name into a global name back and forth, use the following procedure:

#2464F SYSEVAL Local to Global

#2465F SYSEVAL Global to Local

To apply the translation to all the identifiers you can do the following:

```
N+2:   Local
...:             ...
3 :    Local
2:               n
1:               { #2464Fh SYSEVAL }
```

METOP **[ENTER]**

The inverse function needs only #2465Fh instead of #2464Fh.

Symbolic values are not allowed. The hidden code is able to sort any data for which a sort procedure has been defined. This means that you could sort any object given a proper sort criterion. Refer to the Hidden Commands Reference manual for further details.

See also SRTD, MREV

Applications alfaORDER

Examples

```
6:    "STAN"
5:    "FRED"
4:    "paul"
3:    "LUISE"
2:    "HENRY"
1:    5

SRT [ENTER]


6:    "FRED"
5:    "HENRY"
4:    "LUISE"
3:    "STAN"
2:    "paul"
1:    5
```

Problem:

Sort a directory by increasing size of variables.

Solution:

Recall all the variables, take their size, tag each name to the corresponding value, sort them and rebuild a list of names suitable for ORDER.

```
«
VARS
OBJ→ { DUP BYTES NIP SWAP →TAG} METOP
SRT
{ OBJ→ NIP #05B15h SYSEVAL } METOP
→LIST ORDER
»
```

Problem: Sort objects by tag.

Solution: Call the sort independent hidden function and pass the appropriate procedure to it. The program → BOOL has been described under the command FALSE.

```
«
OBJ→ NIP SWAP OBJ→ NIP << → BOOL
»

'SRTFUNC' <B>[STO]

4:      B:      2
3:      C:      "HELLO"
2:      A:      [0 1 2]
1:              3

→ SYS 'SRTFUNC' 821 110 → Xlib EVAL
→ R

4:      A:      [0 1 2]
3:      B:      2
2:      C:      "HELLO"
1:              3
```

SRTD

Category Meta-object manipulation

Affected by flag none

Input

N+1:	obj_1
...:	...
2:	obj_n
1:	n

Output

N+1:	obj_n
...:	...
2:	obj_1
1:	n

Function Sorts data in descending order.

Notes

Objects must be compatible with the >> operator (greater than).

To this category of objects belong:

Global names
Real numbers
Binary integers
Strings
System Binaries
Tagged objects falling in one of the classes listed above except Global
names and Complex numbers.

Further information and practical examples under SRT.

See also

SRT

SUBT

Category Symbolic matrix manipulation

Affected by flag -3 (Numerical result)

Input

2: $\{\{SymbA_{1,1}\ SymbA_{1,2} \ldots SymbA_{1,n}\}$

$\ldots$

$\{SymbA_{m,1}\ SymbA_{m,2} \ldots SymbA_{m,n}\}\}$

1: $\{\{SymbB_{1,1}\ SymbB_{1,2} \ldots SymbB_{1,n}\}$

$\ldots$

$\{SymbB_{m,1}\ SymbB_{m,2} \ldots SymbB_{m,n}\}\}$

Output

1: $\{\{SymbA_{1,1}\text{-}SymbB_{1,1}\ SymbA_{1,2}\text{-}SymbB_{1,2}$ $\ldots\ SymbA_{1,n}\text{-}SymbB_{1,n}\}$

$\ldots$

$\{SymbA_{m,1}\text{-}SymbB_{m,1}\ SymbA_{m,2}\text{-}$ $SymbB_{m,2} \ldots SymbA_{m,n}\text{-}SymbB_{m,n}\}\}$

Function Returns the difference of the two arrays (either symbolic or numerical). The arrays must have the same dimension.

Notes The routine performs the difference on all the pairs of objects accepted by the standard operator -.

ADD, FACTOR and SUBT are instances of a generalized function, designed to apply binary operators to the pairs of elements. For more information on this topic, please refer to the Hidden Commands Reference manual.

See also ADD, ADDCON, DIMS, EQUAL?, MULT, MSYMB?

Applications MATMENU, PRSYMB

SYMBMAT→

Category Symbolic matrix manipulation

Affected by flag none

Input 1: $\{\{Symb_{1,1}\ Symb_{1,2} \ldots Symb_{1,n}\}$ … $\{Symb_{m1}\ Symb_{m2} \ldots Symb_{m,n}\}\}$

1 :$[[N_{1,1}\ N_{1,2} \ldots N_{1,n}]$ … $[N_{m,1}\ N_{m,2} \ldots N_{m,n}]]$

Output

N+1:	$Symb_1$	N+1:	$N_{1,1}$
...:	...	...:	...
2:	$Symb_{m,n}$	2 :	$N_{m,n}$
1:	{m n}	1 :	{m n}

Function Splits the list-matrix or the array on the stack.

Notes SYMBMAT→ is useful for converting numeric arrays into their symbolic counterpart. It handles numeric vectors as if they were arrays 1 x m.

Note that doing:

EVAL * you get a meta-object

See also DIMS, MSYMB?, →SYMBMAT, TRNSP

Applications MATMENU, PRSYMB

Examples

```
1:      {{ 1 2 3 }
        { 'X' 'X-1' -1 }}
```

SYMBMAT→ **[ENTER]**

```
7:      1
6:      2
5:      3
4:      'X'
3:      'X-1'
2:      -1
1:      { 2 3 }
```

```
1:      [ 1 2 3 4 5 ]
```

SYMBMAT→ **[ENTER]**

```
6:      1
5:      2
4:      3
3:      4
2:      5
1:      { 1 5 }
```

→SYMBMAT

Category Symbolic matrix manipulation

Affected by flag none

Input

N+1:	$Symb_1$	
...:	...	
3:	$Symb_{n-1}$	
2:	$Symb_n$	
1:	{r c}	

Output

1: $\{\{Symb_{1,1}\ Symb_{1,2}\ ...\ Symb_{1,c}\}$

...

$\{Symb_{r,1}\ Symb_{r,2}\ ...\ Symb_{r,c}\}\}$

Function Collects the data on the stack in a symbolic matrix.

Notes The command does not check the type of the objects, which can be checked with MSYMB?. This feature lets you build list-matrices for arbitrary purposes.

If too few arguments are there on the stack, the corresponding error is issued.

The inverse function is SYMBMAT→.

See also CONSTMAT, DIMS, MSYMB?, SYMBMAT→, TRNSP

Applications MATMENU, PRSYMB

Examples

```
7:      1
6:      2
5:      3
4:      'X'
3:      'X-1'
2:      -1
1:      { 2 3 }

→SYMBMAT [ENTER]

1:      {{ 1   2   3 }
         { 'X' 'X-1' -1 }}
```

→SYS

Category Type conversion

Affected by flag -5 through -10 bynary integer wordsize (binary integers only)

Input 1: n 1: #n 1: Char

Output 1: <nh>

Function Convertsan input number into a system binary.

Notes →SYS accepts system binaries as well;
dummy conversions prevent errors at no expense.

The structure of the system binary object is explained in appendix C.

See also →B, →Char, →EXT, →R

Examples Let's create a program that extracts a hex substring from '
a hex string. We will use an internal routine for accomplishing most of the work. Be sure to have hex mode set and a wordsize of 20 bits (or higher) before typing the program.

```
«

{ 10 0 0 }          Binary, real, real

 IF CHST?           any mismatch ?

THEN                yes, error

#202h DOERR         Bad argument type

 ELSE               Ok, proceed

→SYS SWAP           end position

→SYS SWAP           beginning position

 #05815h
SYSEVAL             takes the substring

END

 »                  done.
```

Name it HSUB. This program is stored in the directory EXAMPLES.1C.

→TORF

Category	Type conversion			
Affected by flag	none			
Input	1:	External (TRUE)	1:	External (FALSE)
Output	1:	1	1:	0
Function	Converts a system boolean value into a numeric boolean. The input address must necessarily be either #03A81(TRUE) or #03AC0 (FALSE) otherwise "Bad Argument Value" is issued.			
See also	FALSE, TRUE, EXT→, →EXT			

TRNSP

Category Symbolic matrix manipulation<$IArrays;symbolic>

Affected by flag none

Input 1: {{$Symb_{1,1}$ $Symb_{1,2}$ ... $Symb_{1,n}$}
...
{$Symb_{m,1}$ $Symb_{m,2}$ ... $Symb_{m,n}$}}

Output 1: {{$Symb_{1,1}$ $Symb_{2,1}$ ... $Symb_{1,m}$}
...
{$Symb_{1,n}$ $Symb_{2,n}$ ... $Symb_{n,m}$}}

Function Returns the transpose of the matrix.

Notes The matrix is not required to be square.

See also SQUARE?

Applications MATMENU, PRSYMB

Examples

1: {{ 1 2 }
{ X -1 }
{ '-X' 5 }}

TRNSP **[ENTER]**

1: {{ 1 X '-X' }
{ 2 -1 5 }}

TRUE

Category Type conversion

Affected by flag none

Input

Output 1: External (TRUE)

Function Pushes on the stack the system boolean TRUE.

Notes System booleans are machine language routines, residing at a fixed address, which merely return themselves when evaluated. Appendix B explains the structure of such objects.

The address of TRUE is # 03A81h.
FALSE is located at #03AC0.

The command →TorF turns a system boolean into a real boolean (see on page <$R[P#,FALSE]43> for further details).

See also EXT→, →EXT, FALSE, →TorF

VER$

Category Rom version

Affected by flag none

Input

Output 1: "SMRT 1:C"

Function Returns a string encoding the current version of the SmartROM.

Notes VER$ can be useful for creating programs running with different versions of the library.

→XLIB

Category Type conversion

Affected by flag none

Input
2: LID
1: Num

Output 1: XLIB LID Num or 1: Cmd

Function Pushes on the stack the eXternal LIBrary object specified.

Notes When you push on the stack a named XLIB object the display will show it by name. If you push on the stack an hidden function, the display will merely show XLIB LID num, where LID is a real number returning the Library ID number and num is the number of the command.
There are two ways to know if a XLIB object is referenced:

By evaluating it: Very dangerous!

By executing the entry point #07E99h with SYSEVAL

The latter method is safer (unless you type in a wrong address...).

If the XLIB name is actually a pointer, #07E99 SYSEVAL returns the pointee (an object) along with the boolean TRUE, otherwise it merely returns FALSE.

See also →EXT, EXT→

Examples

2: 2
1: 81

Xlib **[ENTER]**

1: SIN

2: 821
1: 45

→Xlib **[ENTER]**

1: MGET

2: 821
1: 246

→Xlib **[ENTER]**

1: XLIB 821 246

To print the complete list of the operating system 'frozen' entry points, that is those system addresses which are mantained in a fixed location in spite of revisions, the contents of the library 1656 provided with the ESA Assembler may be printed. The program below shows how to do it. Be sure to have the library stored in a port.

```
«       0 2157
        FOR n
                1656 n →Xlib
                DUP EVAL SWAP
                →STR →TAG
                PR1 DROP
        NEXT
»
```

XLVLS

Category Stack manipulation

Affected by flag none

Input

...:	...
P+2:	obj_p
...:	...
U+2:	obj_u
...:	...
3:	obj_1
2:	p
1:	u

Output

...:	...
P :	obj_u
..:	...
U :	obj_p
...:	...
1 :	obj_1

Function Swaps levels p and u.

See also RDROP, RDUP, SHIFT

Examples

```
5:  "ABCDE"
4:  1
3:  2
2:  3
1:  "hello"
```

4 5 XLVLS **[ENTER]**

```
5:  1
4:  "ABCDE"
3:  2
2:  3
1:  "hello"
```

APPENDIX B

OBJECTS STRUCTURE

The classification proposed herein follows the order estabilished by function TYPE. For each type of object the internal code used by the dispatching routines is given as well.

Each RPL object is formed by a Prolog, i.e. the header of the object which determines the behavior during evaluation, and a body containing the data. The chapter explains in detail the structure of each object and its features.

Each object is then subdivided in logical fields whose length is specified by a subscript value (in nibbles).

Please note that the HP48 arranges the data in memory in reverse order, so that the prolog is written with the least significant nibble that comes first.

REAL NUMBER

Type 0

Internal Type <1h>

Prolog <02933h>

Structure ($Prolog_5$) ($Exponent_0$) ($Mantissa_{12}$) ($Sign_1$)

Dimensions 21

Data ($Exponent_3$)

BCD Exponent in ten's complement (-500 to 500)

($Mantissa_{12}$)

BCD Mantissa

($Sign_1$)

Sign: 0 = positive, 9 = negative

Example

0	is equal to 33920000000000000000
pi	is equal to 33920000953562951 4130
-1	is equal to 33920000000000000019
-11	is equal to 33920100000000000119
-.5	is equal to 33920999000000000059

COMPLEX NUMBER

Type 1

Internal Type <2h>

Prolog <02977h>

Structure ($Prolog_5$) ($Exponent_3$) ($Mantissa_{12}$) ($Sign_1$) ($Exponent_3$) ($Mantissa_{12}$) ($Sign_1$)

Dimensions 37

Data The Real part is composed by the first number while the imaginary part comes next. Number representation is the same as for reals.

STRING

Type 2

Internal Type <3h>

Prolog <02A2Ch>

Structure $(Prolog_5)$ $(offset_5)$ characters.

Dimensions 5 + offset

Data The total number of characters is (offset-5)/2. (offset-5) is generally an even number.

Notes Characters are stored byte reversed.

Example

"CIAO" is equal to:	C2A20D000034E414F4
"" is equal to :	C2A2050000

REAL ARRAY

Type 3

Internal Type <4h>

Prolog <02E48h>

Structure (Prolog$_5$) (offset$_5$) (Real Prolog$_5$) (n-dim$_5$) (dim$_1$) ... (dim$_n$) (R$_{1...1}$) ... (R$_{dim1\ dim_2...dim_n}$)

Dimensions 5 + offset.

Data Matrices are stored in row major order incrementing the rightmost counter faster.

Notes It is possible to create n-dimensional order arrays. However there are no provisions in the system for handling individual elements when n is greater than 2. If you put on the stack a 3-dimensional array, you will get only "Array of reals".

COMPLEX ARRAY

Type 4

Internal Type <4h>

Prolog <02E48h>

Structure (Prolog$_5$) (offset$_5$) (Complex Prolog) (n-dim) (dim$_1$) ... (dim$_n$) (Cdim$_1$ 1 ... 1) ... (Cdim$_1$ dim$_2$ ... dim$_n$)

Dimensions 5 + offset.

Data Matrices are stored in row major order incrementing the rightmost counter faster.

Notes It is possible to create n-dimensional order matrices. However there are no provisions in the system for handling individual elements when n is greater than 2. If you put on the stack a 3-dimensional array, you will get only "Array of complex>".

ARRAY

Type | 4

Internal Type <4h>

Prolog <02E48h>

Structure (Prolog$_5$) (offset$_5$) (Data Prolog) (n-dim) (dim$_1$) ... (dim$_n$) (Data$_1$) ... (Data$_n$)

Dimensions 5 + offset.

Notes You can create non-numeric arrays for storing type-homogeneous data. Unfortunately there are no provisions to handle efficiently this kind of objects. Error Messages in HIDE area are stored in several one-dimensional string arrays. If you want to store some data preserving it from editing, non-numeric arrays are a good place because of their inaccessibility.

LIST

Type 5

Internal Type <5h>

Prolog <02A74h>

Structure $(Prolog_5)\ Obj_1\ Obj_2 \ldots Obj_n\ (End_5)$

Dimensions $5 + Length(Obj_1) + Length(Obj_2) + \ldots + Length(Obj_n) + 5$

Data A list is a composite object whose body is a sequence of objects or pointers to objects terminated by a special pointer <0312Bh>.

Notes The list object is similar to program objects and symbolic expressions. If you want to translate a Symbolic expression into a List, change its prolog to <02A74h>. When you evaluate a List or a Symbolic Expression through EVAL, special code is called to change the prolog of the composite object into the prolog of a program object.

Example

{} is equal to 47A20B2130
{ 1 "" } is equal to 47A209C2A2FD550B2130
or to
47A203392000000000000000010C2A2050000B2130

GLOBAL NAME

Type 6

Internal Type <6h>, <Ah>

Prolog <02E48h>

Structure ($Prolog_5$) ($Length_2$) characters.

Dimensions 7 + (Length) * 2

Notes Maximum lenght of an indentifier is 255 characters with no restrictions on the name. However there are restrictions due to the parser safety rules, which prevents you from creating names conflicting with reserved variables used by the system.

Example 'MARK is equal to 84E205072D41425B4

LOCAL NAME

Type 7

Internal Type <7h>

Prolog <02E6Dh>

Structure ($Prolog_5$) ($Length_2$) characters.

Dimensions 7 + (Length) * 2

Notes See on the previous page.

Example matA is equal to D6E2040D6164714

PROGRAM

Type 8

Internal Type <8h>

Prolog <02D9Dh>

Structure $(Prolog_5)\ Obj_1\ Obj_2 \ldots Obj_n\ (End_5)$

Dimensions $5 + Length(Obj_1) + Length(Obj_2) + \ldots + Length(Obj_n) + 5$

Notes A program is a composite object whose body is a sequence of objects or pointers to objects terminated by a special pointer 8Bh. Main difference with List and Symbolic lies in its direct execution capability.

Example Internal program performing Rot Dup2 without stack checking:

D9D2059230CA130B2130

ALGEBRAIC

Type 9

Internal Type <9h>, <Ah>

Prolog <02AB8h>

Structure $(Prolog_5)\ Obj_1\ Obj_2 \ldots Obj_n\ (End_5)$

Dimensions $5 + Length(Obj_1) + Length(Obj_2) + \ldots + Length(Obj_n) + 5$

Notes

Symbolic objects are similar to program objects and symbolic expressions. If you want to translate a Symbolic expression into a List, change its prolog to A74h. When you evaluate a List or a Symbolic Expression through EVAL, special code is called to change the prolog of the composite object into the prolog of a program object.

BINARY INTEGER

Type 10

Internal Type <Bh>

Prolog <02A4Eh>

Structure ($Prolog_5$) ($offset_5$) nibbles

Dimensions 5 + offset

Data Raw nibbles.

Notes Binary integers may exceed 64 bit width.
However internal arithmetic routines are tailored for handling 64 bit max integers. Other internal routines (like Append or Size) work well independently of integer length. A curious aspect of a Library structure is that the execution, decompile and text tables are stuffed in huge binary integers objects. User defined binary integers generally have a standard length of 16 nibbles independently of the actual wordsize.

Example The shortest representation of #1234 is

E4A20900004321

GRAPHIC OBJECT

Type 11

Internal Type <Ch>

Prolog <02B1Eh>

Structure ($Prolog_5$) ($Offset_5$) ($Rows_5$) ($Columns_5$) nibbles

Dimensions 5 + offset.

Data Graphics objects are stored in row major order using a bit for each pixel on a byte-aligned scheme. Thus, each row of pixel must have an even number of nibbles eventually padding with garbage bits the last byte. The least significant bit of each nibble represents the leftmost pixel of 4 pixel block.

Example Take the character A in the font ROM8x14:

```
E1B20B2000E0000800000000183C66C6CEF6C6C6C
000000

1       ........        00
2       ........        00
3       ....*...        01
4       ...***..        83
5       ..**.**.        C6
6       .**...**        6C
7       .**...**        6C
8       .*******        EF
9       .**...**        6C
A       .**...**        6C
B       .**...**        6C
C       ........        00
D       ........        00
E       ........        00
```

TAGGED OBJECT

Type	12
Internal Type	<Dh>
Prolog	<02AFC>
Structure	($Prolog_5$) ($Length_2$) characters Obj
Dimensions	7 + (Length) * 2 + Length(Obj)
Notes	Tagged objects does not inherit the behavior of the ancestor type unless you make a recursive call to the routine being executed after deleting the tag.
Example	PIGREEK: 3.14159265359 is represented by : CFA2070059474255454B4339200009535629514130

UNIT

Type 13

Internal Type <Eh>

Prolog <02ADAh>

Structure ($Prolog_5$) (value) (string) ... (string) (oper5) ... (oper5) (End_5)

Dimensions 5 + 21 + Length(string)+ ...+ Length(string) + 5 * num(oper) + 5

Example 1.5129_m^2 is equal to:
ADA2033920000000000921510C2A20700006
ED2A227B0168B01B2130

XLIB NAME

Type 14

Internal Type <0Fh>

Prolog <02E92h>

Structure $(Prolog_5)$ (LID_3) (Num_3)

Dimensions 11

Data External Library Names are uniquely identified by a Library Identification Number and a library-local command number. User commands have a double-face. They are referenced by a fixed address pointer which allows faster execution and compact storage. However, under special circumstances, the address is replaced with the Library IDentification code to prevent the routine from being phisically copied.

DIRECTORY

Type 15

Internal Type <2Fh>

Prolog <02A96h>

Structure (Prolog$_5$) (Attach 3) (offset1) (00000) Namen Obj$_n$ (offset n+1) Namen-1 Obj$_n$-1 (offsetn) ... Name1 Obj$_1$ (offset2)

Dimensions 8 + offset1 + 5

Data Object are stored in reverse order. (offset$_1$) points to the field (offset$_2$) at the bottom of the directory where lies the first object. A sequence of backward offsets lets you jump like a frog till the last object at the top of the directory. The last offset field (00000) marks the end of the chain. The Attach field retains the Library Identification number of one library. In the HOME directory this field counts the number of libraries currently attached, because you may have more than one library attached at the same time.

LIBRARY

Type 16

Internal Type <8Fh>

Prolog <02B40h>

Structure (Prolog$_5$) (offset5) (Length 2) Characters (LID$_3$) (Offset

TexTTbl 5) (Offset MsgTbl 5) (Offset LinkTbl 5) (Offset Conf 5) Nibbles (Cksum 4)

Dimensions 5 + offset

Data

(Length $_2$) Characters	Library Title.
(LID$_3$)	Library IDentification number.
(Offset TextTbl$_5$)	Offset to a binary integer containing command names text. The table begins with 16 field of 5 nibbles (80 nibbles total) each one of them pointing to the first command of a given length. The first field points to the first command (in alphabetical order) of length 1.
(Offset MsgTbl$_5$)	Offset to the table of messages. The table is contained in a string-array.
(Offset LinkTbl$_5$)	Offset to a binary integer containing all the executable code of commands.

(Offset Config$_5$)	Offset to the configuration program of the library.
Nibbles	Library contents.
(Cksum 4)	Checksum.

BACKUP

Type	17
Internal Type	<9Fh>
Prolog	<02B62h>
Structure	($Prolog_5$) (offset 5) (Length 2) characters nibbles
Dimensions	5 + offset
Notes	Backup objects normally exist only in memory ports.

SYSTEM FUNCTION

Type 18

Internal Type <8h>

Prolog <02E92h>

Structure ($Prolog_5$) (LID_3) (Num_3)

Dimensions 11

Data A rom-based program is recognized as function when a special code precedes its execution address. Formerly it is a normal program object but it receives special handling by the parser and decompile routines when you enter formulas in algebraic style. The process of recognizing algebraic functions is tricky and goes beyond the scope of this manual. There are several bits specifying analytic properties like differentiability and others specifing if the program is a command or a function and where to place the decompiled text (before, between or after).

Example SIN (system function) is identified by XLIB 2 81.
Its special header is CC0 and precedes the LID program in ROM. If you scan memory a little before location B4ACh, you will see the following sequence :

CC0200150D9D20....

CC0 means integrable, invertible, differentiable, function.

Thanks to this encoding system, adding external function is very, very hard.

SYSTEM COMMAND

Type 19

Internal Type <8h>

Prolog <02E92h>

Structure $(Prolog_5)$ (LID_3) (Num_3)

Dimensions 11

Data A rom-based program is recognized as built-in command when a special code precedes its execution address. Most commands have a single nibble header (added to the six specifying the Library ID and number) whose value is 8. It seems that the most significant bit of this nibble play a key-role in the game. If this bit is 0, the program is some kind of function. The study of program USAG, supplied with the Interface Kit, with the aid of a RPL decompiler, is helpful to decipher such hieroglyphs.

Example STO (system command) is identified by XLIB 2 341.

SYSTEM BINARY

Type 20

Internal Type <1Fh>

Prolog <02911h>

Structure $(Prolog_5)$ $(Nibble_5)$

Dimensions 10

Data System binaries are the most used numeric entities throughout the Operating System. Providing a faster throughput than real numbers and smaller storage requirements, they are the optimum choice for the system programmer.

Example FFFFFh is equal to 11920FFFFF

12345h is equal to 1192054321

LONG REAL

Type 21

Internal Type <3Fh>

Prolog <02955h>

Structure ($Prolog_5$) ($Exponent_5$) ($Mantissa_{15}$>) ($Sign_1$)

Dimensions 26

Data ($Exponent_5$)

Exponent BCD in ten's complement (from -50000 to 50000)

(Mantissa 15)

BCD Mantissa

($Sign_1$)

Sign: 0 = positive, 9 = negative

Example Extended precision PI is equal to

559200000097985356295141 30

LONG COMPLEX

Type	22
Internal Type	<4Fh>
Prolog	<0299Dh>
Structure	$(Prolog_5)$ $(Exponent_5)$ $(Mantissa_{15})$ $(Sign_1)$ $(Exponent_5)$ $(Mantissa_{15})$ $(Sign_1)$
Dimensions	47
Data	Real part comes first. The number encoding system is the same as for Extended Reals.

LINKED ARRAY

Type 23

Internal Type <5Fh>

Prolog <02A0Ah>

Structure $(Prolog_5)$ $(Offset_5)$ (Data $Prolog_5$) (n-dim) (dim_1) ... (dim_n) $(pointer_{1,...,1})$ ... $(pointer_{dim\ 1,...,\ dim\ n})$

Dimensions 5 + Offset.

Data This is one of the most exotic data structures built in the 48. In the 256K of the operating system, there is no evidence of its existence. It seems that this structure is suitable for sparse arrays because a missing element is merely represented by a 00000 pointer. Unfortunately nothing else is given to know up to date.

CHARACTER

Type 24

Internal Type <6Fh>

Prolog <029BFh>

Structure $(Prolog_5)$ $(Byte_2)$

Dimensions 7

Data Contains a single-byte (a character) byte reversed.

Example Letter A is represented by FB92014

CODE

Type 25

Internal Type <7Fh>

Prolog <02DCCh>

Structure $(Prolog_5)$ $(Offset_5)$ Nibbles

Dimensions 5 + Offset

Data The code starts at the first nibble of the body. execution is transferred to this location by the prolog of the object.

Notes See the Appendix C for an explanation of the Saturn Assembly Language. The ESA Assembler greatly helps in writing assembly language programs.

Example The routine DONOTHING:

```
A=DAT0  A
D0=D0+  5
PC=(A)
```

Reads the address of next object, updates the thread pointer and skips to the next thread (address):

CCD20F0000142164808C

LIBRARY DATA

Type 26

Internal Type <AFh>

Prolog <02B88h>

Structure $(Prolog_5)$ $(offset_5)$ (LID_3) (num_3) Obj_1 Obj_2 ... Obj_n (End_5)

Dimensions 5 + Offset

Data Data appear on the stack merely as 'Library Data'.

Useful for storing non-editable objects.

ADDRESS

Type 27

Internal Type 0h

Prolog <hhhhh>

Structure ($Prolog_5$)

Dimensions 5

Data Any object not mentioned before falls in this category. Adding new object types to the HP-48 RPL is not too difficult but may be dangerous because of the removability of ROM cards. Furthermore there are no hooks for displaying an external object in a readable format. Hence, under normal circumstances, this category represents only machine language calls (atomic threads) or meaningless pointers. The atomic thread represents the address where a machine language subroutine begins. Built-in machine language routines (callable as RPL routines) have the following form:

Addr Addr+5

Prolog Nibbles....

Prolog is a 5 nibble address where Saturn codes effectively begins. RPL routines, written in machine language, usually have a prolog pointing to Addr+5, but some routines violate this convention to implement hyper-compact Dup'n go routines. In effect a machine language routine residing at a fixed address either implements an object structure or merely acts as a RPL subroutine.

APPENDIX C

THE SATURN MICROPROCESSOR

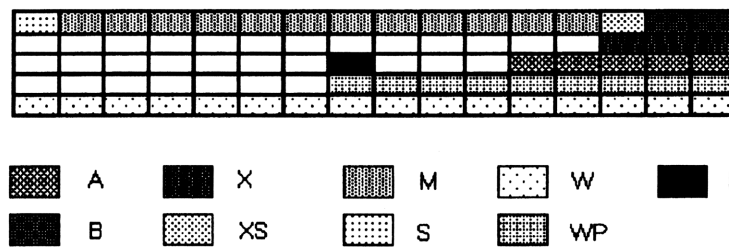
The microprocessor used in the HP-48 family of calculators is the evolution of that appeared in the HP-71 handheld computer and HP-28 pocket calculator. Saturn is a low power consumption CPU, optimized for BCD calculation. Memory Addressing limit is 1 M nibbles, that is 512 Kbytes. With the help of bank switching software techniques the 1Mb nibble barrier can be broken . Tripod Data Systems memory cards concentrate 512K bytes of RAM on a single card, made usable as four independent banks of 128K by custom software drivers.

The Saturn Assembly Language Instruction Set explained in the following pages complies with HP's specifications contained in the HP-71 Hardware IDS, except for a few mnemonics whose name have been modified since the introduction of the ESA Assembler, a ForCALC product. ESA runs under the SmartROM platform and is the first of a series of applications produced by ForCALC.

Microprocessor's registers

The CPU is organized in several registers with different characteristics and power. Main registers are subdivided in fields for easier manipulation.

The following picture shows how each working register is mapped:



Scratch Registers There are five temporary registers (Scratch Registers)
of 64 bit length called R0, R1, R2, R3 and R4. These registers serve as storage area during operations. It is possible to read and write specific fields ranging from 1 nibble up to the whole register.

Arthmetic Registers Four arithmetic registers (A, B, C and D) of 64 bit
of lentgh, denoted as working registers. Each register can be accessed through several fields. Registers A and C can read data from memory directly or indirectly. B and D are used for calculations or temporary storage.

Field Name	Length in nibbles	Length in bits	Meaning
W	16	64	Word
A	5	20	Address
B	2	8	Byte
M	12	48	Mantissa
S	1	4	Sign
X	3	12	Exponent
XS	1	4	Exponent Sign
P	1	4	Pointer
WP	P+1	(P+1)*4	Word Through Pointer

Register C is the most powerful of the four because it allows data exchange with any other register of the CPU.

Control registers

ST Status Register. It is a 16 bit register, where the four most significant bits (from bit 12 to 15) are normally accessed only by the operating system. Can exchange data with C.

PC Program Counter, 20 bit register controlling the execution flow. It is accessible directly or indirectly through register A or C.

RSTK Return Stack. It is a (LIFO) stack with 8 levels of 20 bit where subroutine return addresses are automatically pushed. It can be accessed also through register C.

P Pointer, 4 bit register specifying a nibble within arithmetic or scratch registers. Register P determines also the length of field WP. It can exchange data with C.

D0 Address register. 20 bit register used for addressing data in memory. It can exchange data with A or C.

D1 Same as above.

IN Input Register. 16 bit read-only register used by the system to control the keyboard.

OUT Ouptut Register. 12 bit write-only register

used by the system to enable the keyboard and the beeper.

SB Sticky Bit. 1 bit field of Hardware Status Register
whose value is determined by right shift operations of arithmetic registers.

SR Service Request Bit. 1 bit field of Hardware Status Register whose value is determined by external events.

MP Module Pulled Bit. 1 bit field of Hardware
Status Register whose value is determined by the pulling or pushing of hardware modules.

XM External Module Missing Bit. 1 bit field
of Hardware Status Register whose value is determined by software.

Saturn Assembly Language Mnemonics

The conventions explained herein are used in the following pages to represent parameters and values which may vary in a certain range.

Field Selection Table

Values of the column (a) determine the field of instructions like AaC, that is A2C stands for the instruction A=A-1 XS.<R>
Column (b) applies to opcodes like 9b0oo.

	Field Length	**Nibble value**		
	(a)	(b)		(d)
P	0	8		1
WP	1	9	(P)+1	
XS	2	A	1	
X	3	B	3	
S	4	C		1
M	5	D	12	
B	6	E		2
W	7	F	16	

Table A.1

Opcodes versus Mnemonics

The legenda explains the meaning of symbolic values appearing in the list.

Legenda

a a value between 0 and 7 representing a field as specified in table A.1

b a value between 8 and F representing a field as specified in table A.1

d n+1

h...h a sequence of nibbles

oo a two nibble offset in two's complement

ooo a three nibble offset in two's complement

oooo a four nibble offset in two's complement

aaaaa a five nibble absolute address

n a value in the range 0-15

fs a field (P,WP,B,X,XS,M,W)

Hex	Mnemonic	Field
00	RTNSXM	
01	RTN	
02	RTNSC	
03	RTNCC	
04	SETHEX	
05	SETDEC	
06	RSTK=C	
07	C=RSTK	
08	CLRST	
09	C=ST	
0A	ST=C	
0B	CSTEX	
0C	P=P+1	
0D	P=P-1	
0EF0	A=A&B	A
0EF1	B=B&C	A
0EF2	C=C&A	A
0EF3	D=D&C	A
0EF4	B=B&A	A
0EF5	C=C&B	A
0EF6	A=A&C	A
0EF7	C=C&D	A
0EF8	A=A!B	A
0EF9	B=B!C	A
0EFA	C=C!A	A
0EFB	D=D!C	A
0EFC	B=B!A	A
0EFD	C=C!B	A
0EFE	A=A!C	A
0EFF	C=C!D	A
0Ea0	A=A&B	fs
0Ea1	B=B&C	fs
0Ea2	C=C&A	fs
0Ea3	D=D&C	fs
0Ea4	B=B&A	fs
0Ea5	C=C&B	fs
0Ea6	A=A&C	fs
0Ea7	C=C&D	fs

0Ea8	A=A!B	fs
0Ea9	B=B!C	fs
0EaA	C=C!A	fs
0EaB	D=D!C	fs
0EaC	B=B!A	fs
0EaD	C=C!B	fs
0EaE	A=A!C	fs
0EaF	C=C!D	fs
0F	RTI	
100	R0=A	
101	R1=A	
102	R2=A	
103	R3=A	
104	R4=A	
108	R0=C	
109	R1=C	
10A	R2=C	
10B	R3=C	
10C	R4=C	
110	A=R0	
111	A=R1	
112	A=R2	
113	A=R3	
114	A=R4	
118	C=R0	
119	C=R1	
11A	C=R2	
11B	C=R3	
11C	C=R4	
120	AR0EX	
121	AR1EX	
122	AR2EX	
123	AR3EX	
124	AR4EX	
128	CR0EX	
129	CR1EX	
12A	CR2EX	
12B	CR3EX	
12C	CR4EX	
130	D0=A	

131	D1=A	
132	AD0EX	
133	AD1EX	
134	D0=C	
135	D1=C	
136	CD0EX	
137	CD1EX	
138	D0=AS	
139	D1=AS	
13A	AD0XS	
13B	AD1XS	
13C	D0=CS	
13D	D1=CS	
13E	CD0XS	
13F	CD1XS	
140	DAT0=A	A
141	DAT1=A	A
142	A=DAT0	A
143	A=DAT1	A
144	DAT0=C	A
145	DAT1=C	A
146	C=DAT0	A
147	C=DAT1	A
148	DAT0=A	B
149	DAT1=A	B
14A	A=DAT0	B
14B	A=DAT1	B
14C	DAT0=C	B
14D	DAT1=C	B
14E	C=DAT0	B
14F	C=DAT1	B
150a	DAT0=A	fs
151a	DAT1=A	fs
152a	A=DAT0	fs
153a	A=DAT1	fs
154a	DAT0=C	fs
155a	DAT1=C	fs
156a	C=DAT0	fs
157a	C=DAT1	fs
158n	DAT0=A	d

159n	DAT1=A	d
15An	A=DAT0	d
15Bn	A=DAT1	d
15Cn	DAT0=C	d
15Dn	DAT1=C	d
15En	C=DAT0	d
15Fn	C=DAT1	d
16n	D0=D0+	d
17n	D1=D1+	d
18n	D0=D0-	d
19hh	D0=(2)	hhhhh
1Ahhhh	D0=(4)	hhhhh
1Bhhhhh	D0=(5)	hhhhh
1Cn	D1=D1-	d
1Dhh	D1=(2)	hhhhh
1Ehhhh	D1=(4)	hhhhh
1Fhhhhh	D1=(5)	hhhhh
2n	P=	n
3nh...h	LC(d)	hh...h
400	RTNC	
420	NOP3	
4oo	GOC	
500	RTNNC	
5oo	GONC	
6300	NOP4	
64000	NOP5	
6ooo	GOTO	
7ooo	GOSUB	
800	OUT=CS	
801	OUT=C	
802	A=IN	
803	C=IN	
804	UNCFNG	
805	CONFIG	
806	C=ID	
807	SHUTDN	
8080	INTON	
80810	RSI	
8082nhh...h	LAHEX	hh...h
8083	BUSCB	

8084n	ABIT=0	n	
8085n	ABIT=1	n	
8086noo	?ABIT=0	n	
8087noo	?ABIT=1	n	
8088n	CBIT=0	n	
8089n	CBIT=1	n	
808Anoo	?CBIT=0	n	
808Bnoo	?CBIT=1	n	
808C	PC=(A)		
808D	BUSCD		
808E	PC=(C)		
808F	INTOFF		
809	C+P+1		
80A	RESET		
80B	BUSCC		
80Cn	C=P	n	
80Dn	P=C	n	
80E	SREQ?		
80Fn	CPEX	n	
810	ASLC		
811	BSLC		
812	CSLC		
813	DSLC		
814	ASRC		
815	BSRC		
816	CSRC		
817	DSRC		
818a0n	A=A+d		fs
818a1n	B=B+d		fs
818a2n	C=C+d		fs
818a3n	D=D+d		fs
818F0n	A=A+d		A
818F1n	B=B+d		A
818F2n	C=C+d		A
818F3n	D=D+d		A
818a8n	A=A-d		fs
818a9n	B=B-d		fs
818aAn	C=C-d		fs
818aBn	D=D-d		fs
818F8n	A=A-d		A

818F9n		B=B-d	A
818FAn		C=C-d	A
818FBn		D=D-d	A
819a0	ASRB	fs	
819a1	BSRB	fs	
819a2	CSRB	fs	
819a3	DSRB	fs	
819F0	ASRB	A	
819F1	BSRB	A	
819F2	CSRB	A	
819F3	DSRB	A	
81Aa00		R0=A	fs
81Aa01		R1=A	fs
81Aa02		R2=A	fs
81Aa03		R3=A	fs
81Aa04		R4=A	fs
81Aa08		R0=C	fs
81Aa09		R1=C	fs
81Aa0A		R2=C	fs
81Aa0B		R3=C	fs
81Aa0C		R4=C	fs
81AF00		R0=A	A
81AF01		R1=A	A
81AF02		R2=A	A
81AF03		R3=A	A
81AF04		R4=A	A
81AF08		R0=C	A
81AF09		R1=C	A
81AF0A		R2=C	A
81AF0B		R3=C	A
81AF0C		R4=C	A
81Aa10		A=R0	fs
81Aa11		A=R1	fs
81Aa12		A=R2	fs
81Aa13		A=R3	fs
81Aa14		A=R4	fs
81Aa18		C=R0	fs
81Aa19		C=R1	fs
81Aa1A		C=R2	fs
81Aa1B		C=R3	fs

81Aa1C	C=R4	fs
81AF10	A=R0	A
81AF11	A=R1	A
81AF12	A=R2	A
81AF13	A=R3	A
81AF14	A=R4	A
81AF18	C=R0	A
81AF19	C=R1	A
81AF1A	C=R2	A
81AF1B	C=R3	A
81AF1C	C=R4	A
81Aa20	AR0EX	fs
81Aa21	AR1EX	fs
81Aa22	AR2EX	fs
81Aa23	AR3EX	fs
81Aa24	AR4EX	fs
81Aa28	CR0EX	fs
81Aa29	CR1EX	fs
81Aa2A	CR2EX	fs
81Aa2B	CR3EX	fs
81Aa2C	CR4EX	fs
81AF20	AR0EX	A
81AF21	AR1EX	A
81AF22	AR2EX	A
81AF23	AR3EX	A
81AF24	AR4EX	A
81AF28	CR0EX	A
81AF29	CR1EX	A
81AF2A	CR2EX	A
81AF2B	CR3EX	A
81AF2C	CR4EX	A
81B2	PC=A	
81B3	PC=C	
81B4	A=PC	
81B5	C=PC	
81B6	APCEX	
81B7	CPCEX	
81C	ASRB	
81D	BSRB	
81E	CSRB	

81F	DSRB	
821	XM=0	
822	SB=0	
824	SR=0	
828	MP=0	
82F	CLRHST	
831yy	?XM=0	
832yy	?SB=0	
834yy	?SR=0	
838yy	?MP=0	
84n	ST=0	n
85n	ST=1	n
86nyy	?ST=0	n
87nyy	?ST=1	n
88nyy	?P#	n
89nyy	?P=	n
8A0yy	?A=B	A
8A1yy	?B=C	A
8A2yy	?C=A	A
8A3yy	?D=C	A
8A4yy	?A#B	A
8A5yy	?B#C	A
8A6yy	?C#A	A
8A7yy	?D#C	A
8A8yy	?A=0	A
8A9yy	?B=0	A
8AAyy	?C=0	A
8AByy	?D=0	A
8ACyy	?A#0	A
8ADyy	?B#0	A
8AEyy	?C#0	A
8AFyy	?D#0	A
8B0yy	?A>B	A
8B1yy	?B>C	A
8B2yy	?C>A	A
8B3yy	?D>C	A
8B4yy	?A<B	A
8B5yy	?B<C	A
8B6yy	?C<A	A
8B7yy	?D<C	A

8B8yy	?A→B	A
8B9yy	?B>=C	A
8BAyy	?C>=A	A
8BByy	?D>=C	A
8BCyy	?A<=B	A
8BDyy	?B<=C	A
8BEyy	?C<=A	A
8BFyy	?D<=C	A
8Coooo	GOLONG	
8Daaaaa	GOVLNG	
8Eoooo	GOSUBL	
8Faaaaa	GOSBVL	
9a0yy	?A=B	fs
9a1yy	?B=C	fs
9a2yy	?C=A	fs
9a3yy	?D=C	fs
9a4yy	?A#B	fs
9a5yy	?B#C	fs
9a6yy	?C#A	fs
9a7yy	?D#C	fs
9a8yy	?A=0	fs
9a9yy	?B=0	fs
9aAyy	?C=0	fs
9aByy	?D=0	fs
9aCyy	?A#0	fs
9aDyy	?B#0	fs
9aEyy	?C#0	fs
9aFyy	?D#0	fs
9b0yy	?A>B	fs
9b1yy	?B>C	fs
9b2yy	?C>A	fs
9b3yy	?D>C	fs
9b4yy	?A<B	fs
9b5yy	?B<C	fs
9b6yy	?C<A	fs
9b7yy	?D<C	fs
9b8yy	?A>=B	fs
9b9yy	?B>=C	fs
9bAyy	?C>=A	fs
9bByy	?D>=C	fs

9bCyy	?A<=B	fs
9bDyy	?B<=C	fs
9bEyy	?C<=A	fs
9bFyy	?D<=C	fs
Aa0	A=A+B	fs
Aa1	B=B+C	fs
Aa2	C=C+A	fs
Aa3	D=D+C	fs
Aa4	A=A+A	fs
Aa5	B=B+B	fs
Aa6	C=C+C	fs
Aa7	D=D+D	fs
Aa8	B=B+A	fs
Aa9	C=C+B	fs
AaA	A=A+C	fs
AaB	C=C+D	fs
AaC	A=A-1	fs
AaD	B=B-1	fs
AaE	C=C-1	fs
AaF	D=D-1	fs
Ab0	A=0	fs
Ab1	B=0	fs
Ab2	C=0	fs
Ab3	D=0	fs
Ab4	A=B	fs
Ab5	B=C	fs
Ab6	C=A	fs
Ab7	D=C	fs
Ab8	B=A	fs
Ab9	C=B	fs
AbA	A=C	fs
AbB	C=D	fs
AbC	ABEX	fs
AbD	BCEX	fs
AbE	CAEX	fs
AbF	DCEX	fs
Ba0	A=A-B	fs
Ba1	B=B-C	fs
Ba2	C=C-A	fs
Ba3	D=D-C	fs

CC	A=A-1	A
CD	B=B-1	A
CE	C=C-1	A
CF	D=D-1	A
D0	A=0	A
D1	B=0	A
D2	C=0	A
D3	D=0	A
D4	A=B	A
D5	B=C	A
D6	C=A	A
D7	D=C	A
D8	B=A	A
D9	C=B	A
DA	A=C	A
DB	C=D	A
DC	ABEX	A
DD	BCEX	A
DE	CAEX	A
DF	DCEX	A
E0	A=A-B	A
E1	B=B-C	A
E2	C=C-A	A
E3	D=D-C	A
E4	A=A+1	A
E5	B=B+1	A
E6	C=C+1	A
E7	D=D+1	A
E8	B=B-A	A
E9	C=C-B	A
EA	A=A-C	A
EB	C=C-D	A
EC	A=B-A	A
ED	B=C-B	A
EE	C=A-C	A
EF	D=C-D	A
F0	ASL	A
F1	BSL	A
F2	CSL	A
F3	DSL	A

F4	ASR	A
F5	BSR	A
F6	CSR	A
F7	DSR	A
F8	A=-A	A
F9	B=-B	A
FA	C=-C	A
FB	D=-D	A
FC	A=-A-1	A
FD	B=-B-1	A
FE	C=-C-1	A
FF	D=-D-1	A

APPENDIX E

ERROR MESSAGES

The following error messages are unique to the SmartROM:

Error number	Error Message
33501	Conformability
33502	Type Mismatch
33503	Invalid Sub-List
33504	Argument Out Of Range
33505	KEYS/ACTIONS mismatch

Subscript Out Of Range Issued when a subscript in a list-matrix overflows or underflows its valid range.

KEYS/ACTIONS mismatch Issued when the length of the lists KEYS and ACTIONS are not compatible. Length(ACTIONS) := Length(KEYS)+1

Missing Var Issued when the list KEYS or the list ACTIONS are missing. Issued by the application →FONT when the global name FONT is missing or cannot be found in the search path or does not contain a valid data structure for the operation being performed.

Missing Stack Marker C2M or L2M cannot find the stack marker. (since release 1:C).

Can't edit null array Issued when you invoke MATWRT with either rows or columns locked and a null matrix as input.

Too many objects Issued when you try to enter more object than the matrix can hold. Occurs only when both dimensions are locked.

APPENDIX H

HIDDEN COMMANDS

This appendix contains the complete list of SmartROM hidden commands arranged in logic categories for easier referencing. Each command along with its entry and exit conditions is discussed in detail in the SmartROM Hidden Commands Reference, you can find on the auxiliary disc under the name \DOCS\TECHREF.TXT.

	Original Name	**XLIB number**
String Utilities	capit$	269
	center$	244
	div$	278
	find$	156
	lfpos	186
	lines→	142
	→lines	98
	ltrim$	225
	lwc$	224
	mcentr$	245
	member$	214
	→msg$	220
	norm$	222
	null	125
	pad$	253
	replace$	119
	revs	218
	rowcol	115
	rpt	114
	rtrim$	226
	span$	215
	split$	248
	splith	105
	trim$	221
	upc$	223

BRINGSLID©
APPLICATION SOFTWARE

CAN OFFER YOU, PROBABLY THE BEST APPLICTION SOFTWARE ON THE MARKET FOR YOUR HP 48SX AND HP 48GX.

MATHEMATICS:

* **NEW MATHEMATIC Pac.1.**
  High School and College Mathematics.

* **MATHEMATIC Pac.2.**
  College and University Mathematics.

* **MATHEMATIC Pac.2. Professional Extension.**
  With Extended Possibilities for the programmer.

GEOMETRY:

* **GEOMETRY Pac.1.**
  For Students and Surveyors.

NAVIGATION:

* **NAVIGARE. Navigation Pac. GPS-connection**
  For students and practical navigators.

* **FLIGHT NAVIGATION SYSTEM.**
  GPS-connection

BUSINESS:

* **BUSINESS Pac. 1.**
  More features than other financial calculators.

Distributor:
BB MARKETING ANS.
Ensjøveien 12b. 0655 Oslo Norway
Phone: (+47) 22 67 11 57. Fax. (+47) 22 19 01 61