

---

# Working with the Macro Calculator

---

The macro calculator is a powerful tool that provides the ability to write programs for analyzing and transforming data in the data window. Instead of performing each step of a task in KaleidaGraph yourself, you can often write a macro to do it for you automatically. For example, you can create a macro that clears the contents of any data cell containing masked data.

This chapter explains:

- How to use the Macro Calculator when in Calculator mode.
- The use of all calculator functions. A complete description of each function and some examples are also included.
- The process of programming the Macro Calculator from start to finish. Using sample macros, you learn how to plan, enter, run, document, and save the macro. You also learn how to add and edit existing macros in the Macros menu.
- Each of the available macro commands.

## 1.1 Using the Macro Calculator

---

The Macro Calculator, shown in Figure 1-1, is displayed by choosing **Macro Calculator** from the **Windows** menu. The calculator combines the RPN (Reverse Polish Notation) programming language found in HP calculators with 1000 program steps to give you a number of ways to manipulate and transform data.

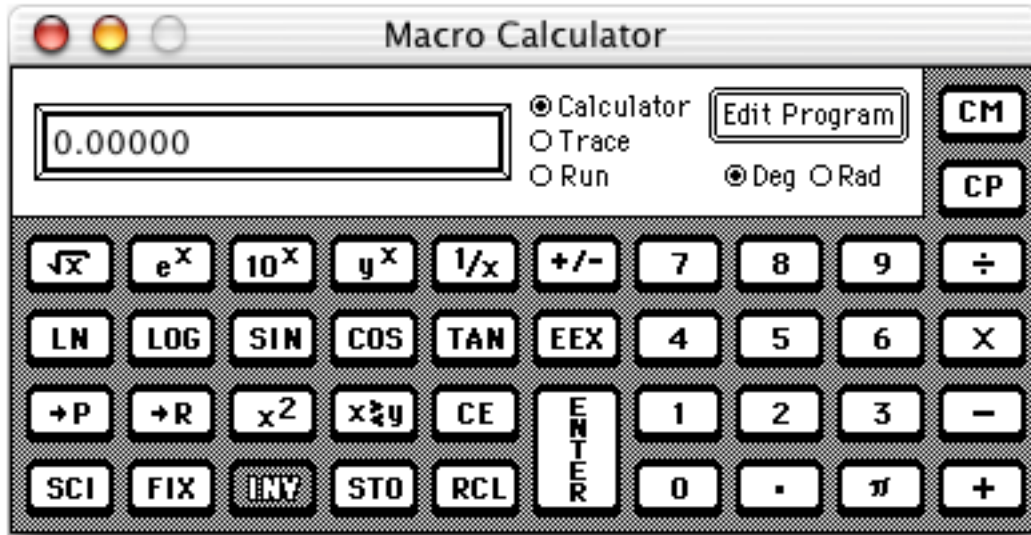


Figure 1-1 Macro Calculator

### 1.1.1 Single Number Functions

RPN calculators use an operating logic that is different from most other calculators. In the RPN calculator, answers appear immediately after a numerical function key is pressed. Therefore, all operands must be entered before a function is executed.

Single number functions operate on the value in the display. Examples of single number functions include: sine, square root, and logarithm.

To use a single number function:

1. Key in the number (if the number is not already in the display).
2. Click a function button.

For example:

1. Key in **4.5**. This can be done with the mouse or the keyboard.
2. Click **COS**. The calculator displays 0.99692 (degrees) or -0.2108 (radians).

### 1.1.2 Two Number Functions

A two number function requires two values to be entered into the calculator before the result can be calculated. Some examples of two number functions are addition, subtraction, multiplication, and division.

To use any two number function:

1. Enter the first number.
2. Click **Enter** to separate the first number from the second.
3. Enter the second number.
4. Click a function button.

## Examples

To Calculate:	Press:	Display:
$8 + 6$	8 ENTER 6 +	14.000
$8 - 6$	8 ENTER 6 -	2.000
$8 * 6$	8 ENTER 6 ×	48.000
$8 / 6$	8 ENTER 6 ÷	1.333

In each of the above examples, the first number is typed followed by the **Enter** button. The second number gets typed and a function button is clicked. Here is what is happening:

- Both numbers are in the calculator before clicking a function button.
- The **Enter** button separates two numbers typed one after the other. If the number is already in the calculator display from a previous function, it is not necessary to click **Enter**.
- Pressing a function button causes the calculator to execute immediately and display the result.

Some more examples are shown below:

To Calculate:	Press:	Display:
$12 * (10 - 6)$	12 ENTER 10 ENTER 6 - ×	48.000
$5 * (8 - 2) / 3$	8 ENTER 2 - 5 × 3 /	10.000
$(9 - 3) * (3 / 9)$	9 ENTER 3 - 3 ENTER 9 ÷ ×	2.000
$625 / (20 + 5)$	625 ENTER 20 ENTER 5 + ÷	25.000

### 1.1.3 The Automatic Stack

The automatic stack is where the calculator retains intermediate results. The stack consists of 12 storage locations, called registers. The bottom value of the stack is always displayed. All values that are entered or are the result of a function are placed in the bottom register.

The bottom register in the stack is the **X register**. The next register up is the **Y register**. The **x $\leftrightarrow$ y** key exchanges the numbers in the X and Y registers without affecting the rest of the stack. The stack can be viewed at any time by clicking the X register display, as shown in Figure 1-2.

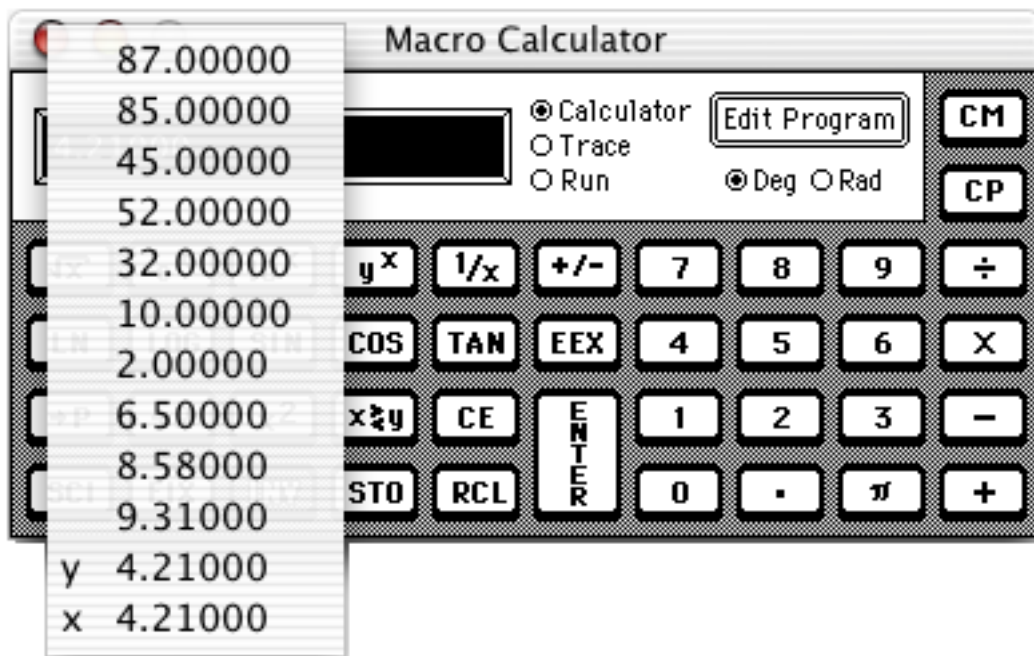


Figure 1-2 Viewing the stack

Single number functions do not cause the stack to drop. The result of the function is placed in the X register, and replaces the value operated on by the function. Two number functions cause the stack to drop whenever they are executed. The result of the function is placed in the X register and the two numbers that were used in the calculation are removed from the stack, causing it to drop.

### 1.1.4 Storing and Recalling Numbers

The Macro Calculator has 100 memory registers numbered 00–99. These registers are accessed through the **STO** (store) and **RCL** (recall) commands. These commands operate using the X register of the calculator.

**Note:** The RCL command causes the stack to be lifted; the STO command does not.

To store a value into a memory register:

1. Enter the value to be stored into the calculator.
2. Click **STO**.
3. Enter the memory location (00–99).
4. The value is stored in the specified memory location.

To recall a value from a memory register:

1. Click **RCL**.
2. Enter the memory location (00–99).
3. The value in the specified memory location is placed in the X register.




For example:

STO 05	Stores the X register's value in memory location 05.
STO 29	Stores the X register's value in memory location 29.
RCL 15	Places the value in memory location 15 into the X register.
RCL 34	Places the value in memory location 34 into the X register.




## 1.2 Calculator Commands

---





### 1.2.1 Clearing the Calculator






-  • **Clear Entry** - This button clears the display (X register) without affecting previous entries.
-  • **Clear Memory** - This button resets the contents of the memory registers to 0.
-  • **Clear Program** - This button erases the entire program in the program text editor.

### 1.2.2 Changing the Display




-  • **Fixed Decimal Notation** - In this notation, the calculator displays numbers rounded to the specified number of decimal places. To change the number of decimal places displayed, click **FIX** and enter the number of decimal places (0–9).
-  • **Scientific Notation** - In this notation, the calculator displays numbers with one digit to the left and a specified number of digits to the right of the decimal point. To change the number of decimal places displayed, click **SCI** and enter the number of decimal places (0–9).
-  • **Exponent Notation** - This button allows numbers to be entered in exponent form ( $A \times 10^B$ ). Enter the decimal (A) portion of the number, click **EEX**, and type the exponent (B).

### 1.2.3 Single Number Functions




-  • **Change Sign** - This button changes the sign of the X register.
-  • **Pi** - This button places an approximate value of Pi (3.14159...) in the X register.
-  • **Reciprocal** - This button computes the reciprocal of the X register.
-  • **Square** - This button computes the square of the X register.

-  • **Square Root** - This button calculates the square root of the X register.
-  • **Common Logarithm** - This button computes the logarithm (base 10) of the X register.
-  • **Natural Logarithm** - This button computes the logarithm (base e) of the X register.
-  • **Common Anti-log** - This button calculates 10 to the power of the X register.
-  • **Natural Anti-log** - This button calculates e (2.7182...) to the power of the X register.

### 1.2.4 Two Number Functions

-  • **Power** - This button raises the number in the Y register to the power specified in the X register. Negative values of Y are only allowed if X is an integer.
-  • **Polar-To-Rectangular** - This button converts the numbers in the X and Y registers (R and  $\emptyset$ ) to rectangular coordinates (X and Y).  $\emptyset$  can be expressed in either degrees or radians, depending on the calculator setting.
-  • **Rectangular-To-Polar** - This button converts the numbers in the X and Y registers (X and Y) to polar coordinates (R and  $\emptyset$ ).  $\emptyset$  can be expressed in either degrees or radians, depending on the calculator setting.

### 1.2.5 Trigonometric Functions

-  • **Sine, Cosine, Tangent and Inverse** - The **SIN**, **COS**, and **TAN** buttons calculate the appropriate trigonometric function of the value in the X register. The  **Deg** and  **Rad** buttons determine whether the values are displayed in degrees or radians. Make sure the correct mode is selected before computing any of these functions. Clicking **INV** before selecting any of the these functions computes the inverse of the specified trigonometric function.







	Function	Inverse
Sine	sin	INV sin = sin <sup>-1</sup>
Cosine	cos	INV cos = cos <sup>-1</sup>
Tangent	tan	INV tan = tan <sup>-1</sup>

## 1.3 Programming the Calculator

This section takes you through the process of writing a macro from start to finish. A sample macro is used to help demonstrate some of the steps involved in writing a macro. A more advanced macro is shown at the end of the section. The commands are covered in complete detail in Section 1.4.

### 1.3.1 Program Text Editor

Clicking the **Edit Program** button of the Macro Calculator displays a program editor. The program text editor is shown in Figure 1-3.

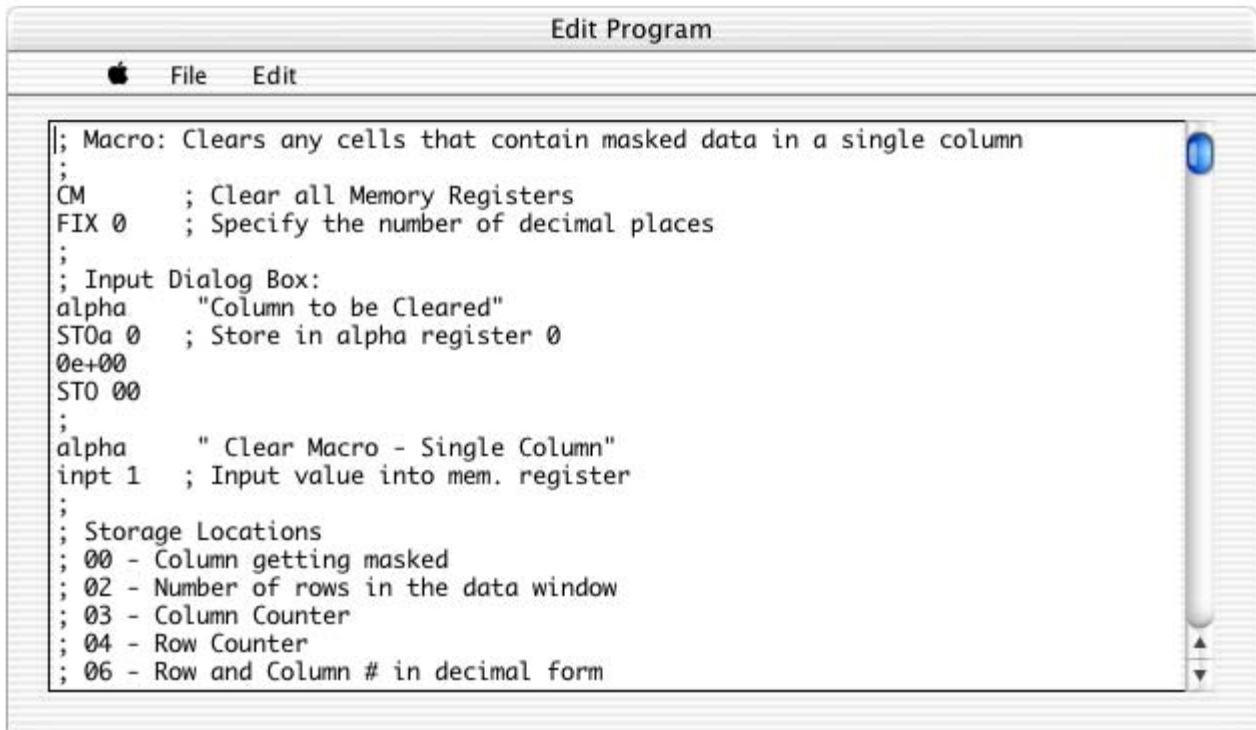


Figure 1-3 Program Text Editor

This editor is used to create new macros and edit existing macros. You can view 20 rows of program text at a time in the editor. Cut, Copy, Paste, and Find commands are supported to aid in moving and locating text. When you are finished editing, choosing **Close** causes the calculator to check the macro for syntax errors. If an error is found, the editor reappears and the error is highlighted. Use the **Find** command to search for other occurrences of the error.

## 1.3.2 Summary of Creating a Macro

### Planning the Macro

The first step in creating a macro is to determine exactly what the macro will do. It is a good idea to write down some of the commands that will be used and to think about how the data window will be accessed (using either indirect or vector commands). As a general rule, if the task can be performed using Formula Entry, vector addressing will normally work for the macro. If not, use indirect addressing.

In our first example, the goal is to write a macro that calculates the factorial of a number. You can compute the factorial of a number by successively subtracting 1 from the number and multiplying the result by the product of its previous values. The factorial of 5, for example, is given by  $5*4*3*2*1$ .

### Entering the Macro

To enter a program, you need to be in the program editor. Click **Edit Program** to display the program editor. If the editor contains any text, choose **New** from the **File** menu to erase the contents of the editor. You are now ready to enter a macro. A sample macro follows.

**Factorial Macro** - Our factorial program is shown below. Enter each of the commands into the editor followed by a carriage return. The comments on the right are shown to identify what each command does.

```

"Enter Number"                ; Copy "Enter Number" to the alpha register.
STOa 1                          ; Store contents of alpha register in alpha memory 1.
"Factorial"                    ; Use this string as the title of the dialog.
prmt 1                          ; Display a dialog and prompt for the factorial number
                                ; and store the number in mem. reg. 01.
1                                ; Place 1.0 in the X register.
STO 02                          ; Store 1.0 in memory register 02.
LBL 00
RCL 01                          ; Recall the contents of mem. reg. 01 to the X register.
MUL 02                          ; Multiply the contents of the X register and register 02.
1                                ; Place 1.0 in the X register.
SUB 01                          ; Subtract 1 from memory register 01.
RCL 01                          ; Recall the contents of register 01 to the X register.
0.0                              ; Place 0.0 in the X register.
x < y                          ; Test if 0.0 < (contents of memory register 01).
GTO 00                          ; Yes: Go to Label 00.
RCL 02                          ; No: Recall the contents of mem. reg. 02 to the X register.
STOP                            ; Halt program execution.

```

### Running the Macro

Once the program is entered into the Macro Calculator, it is ready for execution. If you are in the program editor, choose **Close** to return to the calculator. Click **Run** to execute the program. The program runs until a **STOP** command is reached. If a **STOP** command is not encountered, KaleidaGraph searches the entire 1000 step program memory for another command before aborting the macro.

**Note:** The **STOP** command does not have to be the last program step in a macro. It can be located anywhere that is logical within the program.

When the factorial program is executed, the dialog in Figure 1-4 is displayed.

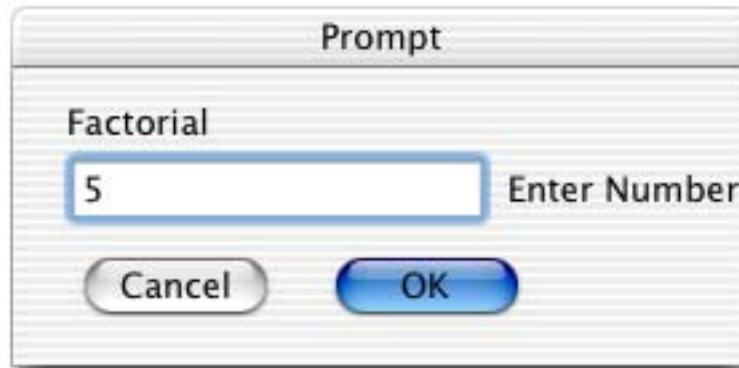


Figure 1-4 Input dialog

Enter the value for which you want the factorial calculated and click **OK**. When the program is finished, the result of the macro is placed in the calculator display.

### Debugging a Macro

There may be times when you run a macro and the results are not what you expect. When this occurs, it is useful to go through the program step by step to determine where the problem lies. There are two buttons on the Macro Calculator (**SST** and **R/S**) that aid in debugging a program.

- SST** • **Single-Step** - This button executes the program one step at a time. You can advance to the next step by clicking **SST** (or by pressing the **Spacebar**, **Return**, or **Enter**). At each step of the program, the current value of the X register is displayed. If any key besides the **Spacebar**, **Return**, or **Enter** is typed, the program switches to **Run** mode and executes the rest of the program.
- R/S** • **Run/Stop** - This button is useful for debugging long loops. When the R/S command is hit during program execution, the calculator switches from **Run** mode to **Single-Step** mode. You can then proceed as you normally would in single-step mode. If any key besides the **Spacebar**, **Return**, or **Enter** is typed, the program returns to Run mode and continues to execute the rest of the program.

Try running the factorial program again, but this time click **Trace** and use the **SST** button to step through the program. The current step is highlighted in the window on the calculator and the result of the step is shown in the display.

### Adding Comment Lines

After a macro runs correctly, comments can be added to the program. Comments are used to make a program more readable, by describing the purpose of the program and the use of each memory register. Comments that are placed with a program step should explain exactly what the step is doing.

Comments can be added anywhere in the macro with the use of a semicolon. Anything after the semicolon to the first occurrence of a carriage return is treated as a comment.

Try adding some of the comments that were to the right of the original program. The comments can be on the same line as the command, or they can be on a separate line. An example is shown below:

“Enter Number” ; Copy the string to the alpha register

## Saving the Macro

Once the macro is complete, it should be saved for future use. Choose **Save As** in the program editor to display the Save dialog. Give the program a name and click **Save** to complete the process of creating a macro.

### 1.3.3 Sample Macro

This section introduces you to a much more advanced macro than the factorial example. This macro shows the use of looping, subroutines, and indirect addressing. Most of the program steps have comments with them to describe what that command is doing.

**The Clear Macro - (Single Column)** - The purpose of this macro is to step down a specified data column and clear any cells that contain masked data. Immediately following the program is a description of how the program works.

```

; Macro: Clears any cells that contain masked data in a single column.
;
;
1  CM                                ; Clear all memory registers.
   FIX 0                             ; Specify the number of decimal places.
;
; Input Dialog Box:
alpha "Column to be Cleared"
STOa 0                                ; Store in alpha register 0.
0.000000000e+00                       ; Specify default value.
STO 00
;
alpha "Clear Macro - Single Column"
inpt 1                                ; Input value, store in mem. reg. 0.
;
; Storage Locations
; 00 - Column getting cleared
; 02 - Number of rows in the data window
; 03 - Column Counter
; 04 - Row Counter
; 06 - Row and Column # in decimal form
;
;
2  FIX 6                             ; Reset the number of decimal places.
   ibase                             ; Allow indirect addressing to access all 1000 columns.
   size                               ; Returns the size of the data window.
STO 02                                ; Stores the number of rows that are in the data window.
RCL 00                                ; Recall column number.
STO 03                                ; Set column counter.
XEQ 50                                ; Store address in decimal form, set row counter.
;
3  LBL 20
   RCLi 06                            ; Recall data from cell specified in mem. reg. 06.
   test 2                             ; Test for masked data cell.
   CLRi 06                             ; Clear the cell specified in 06.
0.000000000e+00
DSE 04                                ; Decrement counter by 1, are contents of 04 > 0.0.
GTO 40                                ; Yes - Increase row number by 1.
STOP                                  ; No - Stop program.
;
4  LBL 40
   XEQ 70                             ; Increase current address.
   GTO 20                             ; Go to Label 20.
;
   LBL 50
   RCL 03                             ; Recall current column.

```

```

1.000000000e+03
/
; Divide column by 1000.
STO 06
; Store column in decimal form.
XEQ 60
; Reset row counter.
GTO 20
;
LBL 60
RCL 02
STO 04
; Reset row counter.
RTN
; Return to next step after XEQ.
;
LBL 70
1.000000000e+00
ADD 06
; Increase address by 1.
RTN
; Return to next step after XEQ.

```

1. The first part of the program clears the memory registers and generates the input dialog shown in Figure 1-5. The **alpha** and **STOa** commands are used to place a description next to the value to be input. The second alpha command is used for the title of the dialog. The **inpt 1** command tells the program that the value entered is stored in the first memory register (00).

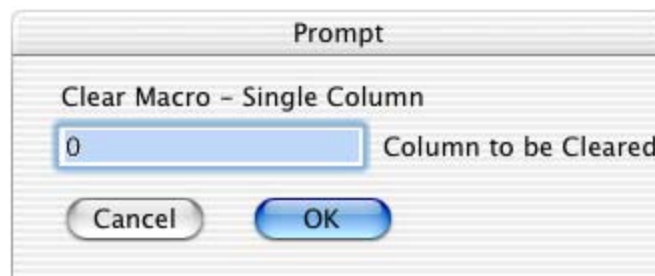


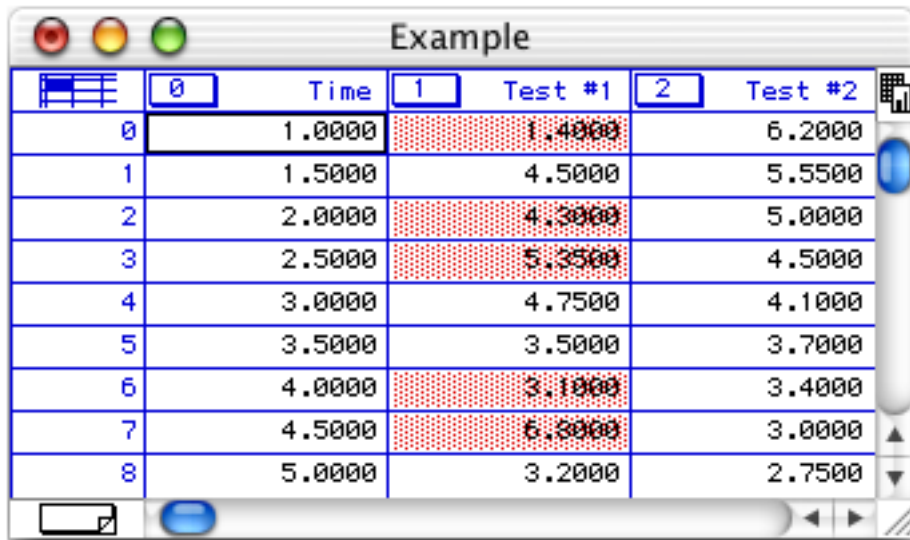
Figure 1-5 Input dialog

2. The next part of the program determines the size of the data window and stores the number of rows in memory location 02. The **ibase** command allows indirect addressing to access all 1000 columns. The **XEQ 50** command causes the column number to be recalled, divided by 1000, and stored in memory location 06. **LBL 60** is executed, which results in the row counter being set equal to the number of rows in the data window.
3. The **LBL 20** portion is the main part of the program. This section recalls the data from the cell specified in memory location 06. The **test 2** command checks if the data is masked. If the data is masked, the contents of the cell are cleared. If the data is not masked, the program moves on to the next row. This section is also used to decrement the counter and check if the last row in the data window has been reached. If the last row is reached, the program stops. Otherwise the program continues to operate.

The **DSE 04** decreases the counter by 1 each time. If the counter is greater than 0, the program jumps to **LBL 40**, which increases the address by 1 so that the next row is read. If the counter is equal to 0, the last row in the data window has been reached. When the counter equals 0, the program skips the next step (**GTO 40**) and halts execution because the **STOP** command has been reached.

4. Notice how all of the subroutines have been placed after the main body of the program. This format is commonly used for organizational purposes.

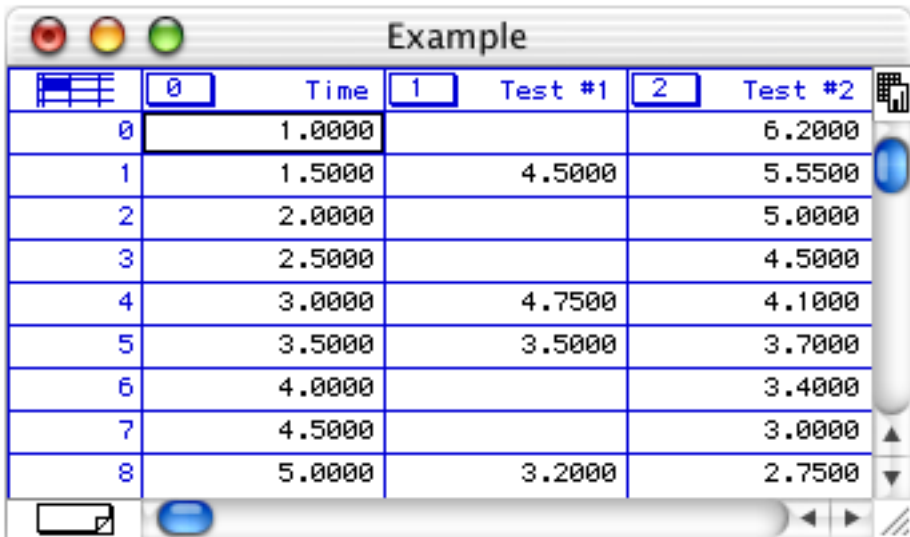
Figure 1-6 shows a sample data set before the Clear macro is executed.



	0	Time	1	Test #1	2	Test #2
0		1.0000		1.4000		6.2000
1		1.5000		4.5000		5.5500
2		2.0000		4.3000		5.0000
3		2.5000		5.3500		4.5000
4		3.0000		4.7500		4.1000
5		3.5000		3.5000		3.7000
6		4.0000		3.1000		3.4000
7		4.5000		6.3000		3.0000
8		5.0000		3.2000		2.7500

Figure 1-6 Initial data set

Figure 1-7 shows the data set after the Clear macro is executed.



	0	Time	1	Test #1	2	Test #2
0		1.0000				6.2000
1		1.5000		4.5000		5.5500
2		2.0000				5.0000
3		2.5000				4.5000
4		3.0000		4.7500		4.1000
5		3.5000		3.5000		3.7000
6		4.0000				3.4000
7		4.5000				3.0000
8		5.0000		3.2000		2.7500

Figure 1-7 Result of running the macro

### 1.3.4 Adding a Program to the Macros Menu

Once you have written a program, you can save it in the **Macros** menu and use it again later. A maximum of 100 macros can be stored in the menu at once. If the macro is in the Calculator, the steps required to add it are:

1. Choose **Show Macros** from the **Macros** menu. The dialog in Figure 1-8 appears.

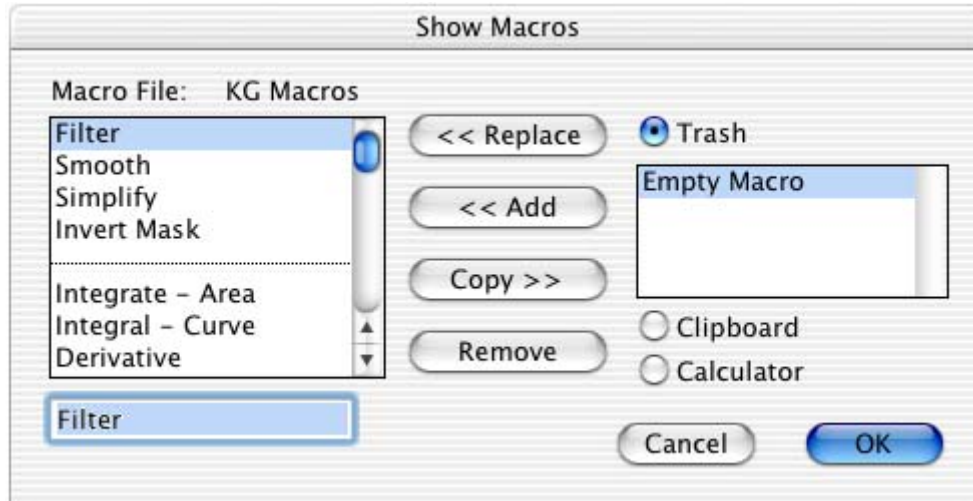


Figure 1-8 Show Macros dialog before clicking Add

2. Click **Calculator** to specify the current location of the macro you wish to add.
3. Select one of the macros from the macro list on the left side of the dialog. The macro you are adding is inserted after this macro.
4. Click **Add**. The name **Calculator** is inserted into the macros list. This is the default name assigned to the macro. The dialog should look similar to Figure 1-9 after completing this step.

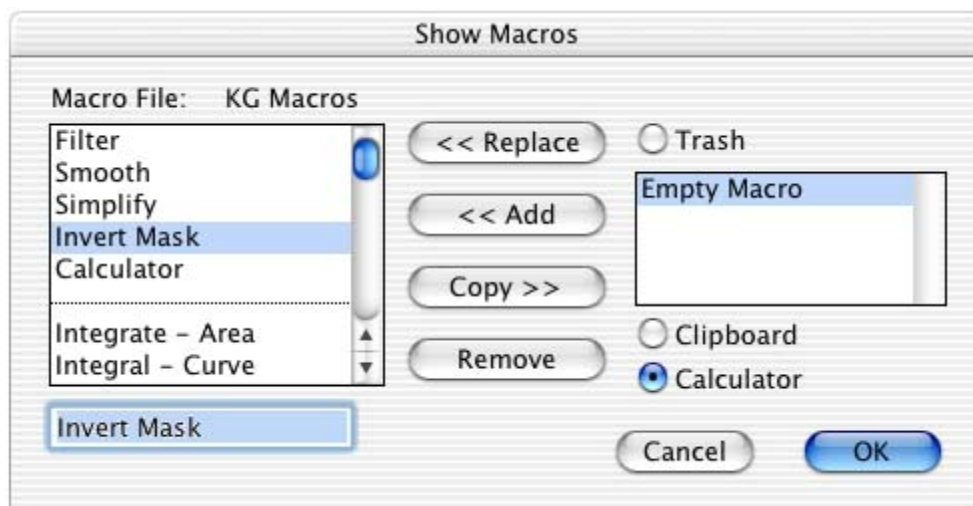


Figure 1-9 Show Macros dialog after clicking Add

5. Select the default name of the macro. The name can now be edited in the text field below the macros list.
6. The name change takes effect when a different macro is selected or you click **OK**.

### 1.3.5 Retrieving a Macro for Editing

Once a program is in the **Macros** menu, it is possible to retrieve it for more editing. If the macro was not added to the Macros menu, it can be opened from within the program editor. In the case of the Clear macro, it would be nice to modify the program so that it can operate on a range of columns.

The steps required to edit the macro are:

1. Choose **Edit Macros** from the **Macros** menu. The dialog in Figure 1-10 appears.

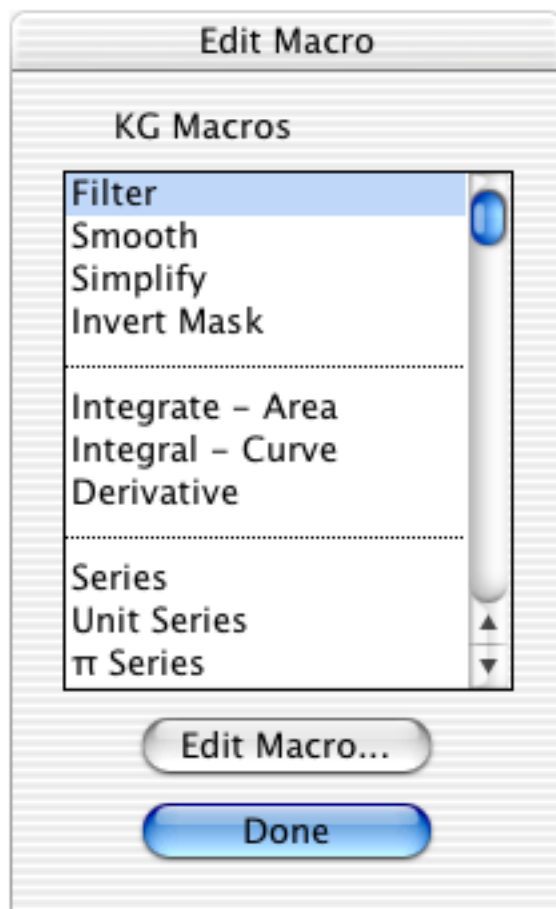


Figure 1-10 Edit Macros dialog

2. Either select the name of the macro you want to edit and click **Edit Macro** or double-click on its name. The program's source code appears in a text editor. Once the original macro has been modified, save the changes and choose **Close** to leave the editor.
3. Click **Done** to exit the Edit Macros dialog.

The modified program follows. Most of the original program remains intact. Anything that has been added or changed is displayed in bold type and has a number in front of it. A description of the changes follows the program listing.

**Note:** This example macro can easily be modified to perform a number of operations besides clearing column cells. The basic commands are already present for using indirect addressing to perform operations on a range of columns.

## The Clear Macro - (Multiple Column)

```

; Macro: Clears any cell that contains masked data in a range of columns.
;
CM                               ; Clear all memory registers.
FIX 0                             ; Specify the number of decimal places.
;
; Input Dialog Box:
alpha  "Initial Column to be Cleared"
STOa 0
0.000000000e+00
STO 00
alpha  "Last Column to be Cleared"
STOa 1
1.000000000e+00
STO 01
alpha  "Clear Macro - Multiple Columns"
inpt 2
;
; Storage Locations
2 ; 00 - First column getting cleared
; 01 - Last column getting cleared
; 02 - Number of rows in data window
; 03 - Column Counter
; 04 - Row Counter
; 06 - Row and Col. # in decimal form
;
FIX 6                               ; Reset the number of decimal places.
ibase                               ; Allow indirect addressing to access all 1000 columns.
size                               ; Returns the size of the data window.
STO 02                             ; Stores the number of rows that are in the data window.
RCL 00                             ; Recall column number.
STO 03                             ; Set column counter.
XEQ 50                             ; Store column in decimal form.
;
3 ; LBL 10
; XEQ 20
; RCL 01                               ; Recall final column value.
; ISG 03                               ; Increment counter by 1, are contents of 03 <= 01.
; GTO 50                               ; Yes - Increase column.
; STOP                                ; No - Stop program.
;
; LBL 20
; RCLi 06                             ; Recall data from cell specified in mem. reg. 06.
; test 2                               ; Test for masked data cell.
; CLRi 06                             ; Clear the cell specified in 06.
; 0.000000000e+00
; DSE 04                               ; Decrement counter by 1, are contents of 04 > 0.0.
; GTO 40                               ; Yes - Increase row number by 1.
4 ; RTN
;
; LBL 40
; XEQ 70                               ; Increase current address.
; GTO 20
;

```

5

```

LBL 50
RCL 03 ; Recall current column.
1.000000000e+03
/ ; Divide column by 1000.
STO 06 ; Store column in decimal format.
XEQ 60 ; Reset row counter.
GTO 10
;
LBL 60
RCL 02
STO 04 ; Reset row counter.
RTN
;
LBL 70
1.000000000e+00
ADD 06 ; Increase address by 1.
RTN

```

### Explanation of Changes

1. The first alpha command was altered and another alpha command was added to get the column values. A **STOa** command was added to store the extra alpha command. The **inpt** command was changed to store the extra value that is entered in the input dialog. Figure 1-11 shows the dialog that appears when the modified macro is run.

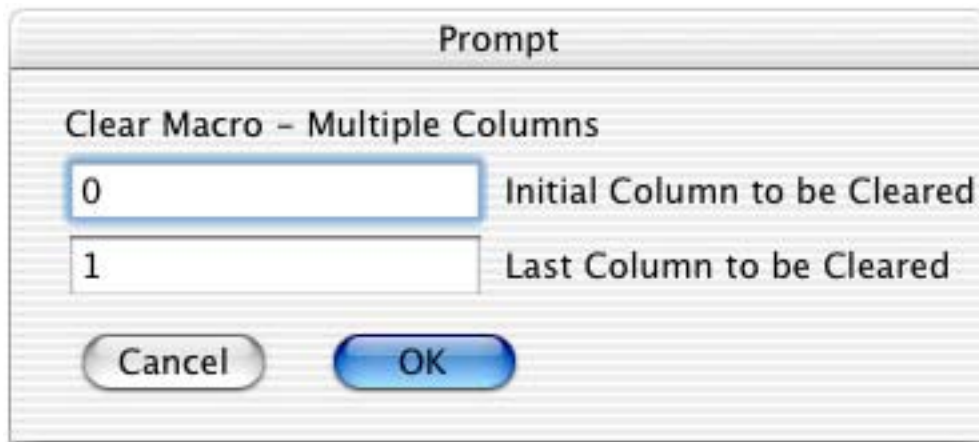


Figure 1-11 Input dialog for modified macro

2. Storage location 00 is now used to store the value of the initial column to be cleared. Location 01 was added to store the value of the last column to be cleared.
3. The commands in **LBL 10** are used to increment the column counter. The program jumps to **LBL 20** and continues until the end of the column is reached. The program returns inside **LBL 10** to recall the value of the last column and test it against the column counter. If the counter is less than or equal to the final column value, the program jumps to **LBL 50** to store the new address of the next column. If the counter is greater than the final column, the program stops.
4. The **STOP** command was changed to a **RTN** (return) statement. This causes the program to return inside **LBL 10** when the last row in a column is reached.
5. The target of the jump was changed from **LBL 20** to **LBL 10**.

Figure 1-12 shows a sample data set before the modified Clear macro is executed.

	0	Time	1	Test #1	2	Test #2	3	Test #3
0		1.0000		1.4000		6.2000		7.7000
1		1.5000		4.5000		5.5500		7.0500
2		2.0000		4.3000		5.0000		6.5000
3		2.5000		5.3500		4.5000		6.0000
4		3.0000		4.7500		4.1000		5.6000
5		3.5000		3.5000		3.7000		5.2000
6		4.0000		3.1000		3.4000		4.9000
7		4.5000		6.3000		3.0000		4.5000
8		5.0000		3.2000		2.7500		4.2500

Figure 1-12 Initial data set

Figure 1-13 shows the data set after the modified Clear macro is executed, with column 0 as the initial column and column 3 as the last column to be cleared.

	0	Time	1	Test #1	2	Test #2	3	Test #3
0						6.2000		
1						5.5500		7.0500
2		2.0000		4.3000		5.0000		6.5000
3								
4		3.0000		4.7500		4.1000		5.6000
5		3.5000		3.5000				
6		4.0000		3.1000		3.4000		4.9000
7								
8		5.0000		3.2000		2.7500		4.2500

Figure 1-13 Resulting data set

## 1.3.6 Making Changes Permanent

### Saving Macros

The macros that appear in the **Macros** menu are taken from the default Macros file. If any changes are made to these macros or if any macros are added or deleted, you need to save the changes to the Macros file using one of the following ways.

- Choose **Preferences** from the **File** menu and select **Prompt** from the **Macros** pop-up menu. When exiting KaleidaGraph, a dialog asks whether or not to overwrite the Macros file that was opened at launch.
- Choose **Preferences** from the **File** menu and select **Always** from the **Macros** pop-up menu. When exiting KaleidaGraph, the Macros file that was opened at launch is automatically overwritten.
- Choose **File > Export > Macros** and overwrite the default Macros file.

### Protecting Macros

Another way of making the changes permanent is to protect a macro. A protected macro cannot be viewed or edited, EVER. Protecting a macro is useful if you want to keep others from viewing your macro source code or from editing specific macros.

A macro is protected by selecting it in the **Edit Macro** dialog and typing **Command+Shift+P** (Macintosh) or **Ctrl+Shift+P** (Windows). If you want to make a copy of the source code, you must create that copy before protecting the macro. It is not possible to unprotect a macro that has been protected.

## 1.4 Program Commands

---

### 1.4.1 Constraints

Most of the commands have either **n**, **nn** or **nnn** after them to specify the constraints associated with each command. If the command is followed by **n**, the value of **n** can range from 0 to 9, except for the **inpt** command which can range from 1 to 6. If a command is followed by **nn**, the value of **nn** can range from 0 to 99. If **nnn** follows a command, the value of **nnn** can range from 0 to 999.

#### Indirect Addressing Constraints

Indirect addressing allows you to specify the row and column address of a number to be used in a calculation. The address of the number is stored in a memory register. The integer portion of the number is the row address and the fractional portion of the number is the column address. For example, if a memory register has 9.03 stored, an indirect function operates on the data cell at row 9, column 3.

If the **ibase** command appears anywhere in the macro, the column address can range from .000 to .999. If the **ibase** command is not in the macro, the column address can only range from .00 to .99. For more information on the **ibase** command, see Section 1.4.6.

**Note:** All address calculations are referenced to the start of a data selection.

## 1.4.2 Constants

Constants can be used within a macro by simply typing the number. When a constant is encountered during program execution, the number is placed in the X register. Constants are commonly used to set the default values in dialogs.

Constants are double precision by default. If you want to have a single precision number, precede the number with **const**.

For example:

```
35.958145799          ;This constant is in double precision.
const 35.9581         ;This constant is in single precision.
```

## 1.4.3 Program Control and Looping

### Program Labels

- **LBL nn** - There are 100 program labels numbered 00–99. Labels are used as markers at the beginning of a series of program steps and as targets for either forward or backward program jumps.

For example:

```
LBL 40                ;This label marks the beginning of a group of steps.
```

### Jumps

- **GTO nn** - This command is used to branch to another portion of the program. When the GoTo command (**GTO nn**) is encountered, the program jumps to **LBL nn** and continues execution from there.
- **GTOi nn** - This command is another form of the GoTo command. The indirect jump (**GTOi nn**) reads the value of the specified memory register and uses it as the target label for the jump. This command allows the target of a jump to be modified during the execution of the program.

For example:

```
GTO 20                ;Causes the program to jump to LBL 20 and continue running.
GTOi 15               ;Jumps to the label whose value is in register 15 and keeps running.
```

### Calling Subroutines

- **XEQ nn** - The execute command (**XEQ nn**) is similar to the GoTo statement. As with the GoTo command, the XEQ command branches to another part of the program and continues running. However, when a return command (**RTN**) is found, the program returns to the program step immediately following the execute command it came from originally.
- **XEQi nn** - This command is another form of the XEQ statement. The difference between them is that **XEQi** reads the value of the specified memory register and uses that value as the target label for the subroutine. The program continues to run from that point until a return command (**RTN**) is encountered. The program then returns to the program step immediately following the XEQi command it came from originally. This instruction allows the target subroutine to be modified during the execution of the program.

For example:

XEQ 70                   ;Branches to LBL 70 and continues until a RTN is found.  
XEQi 20                 ;Branches to the label whose value is specified in memory register 20.

**Note:** A maximum of 20 XEQ/XEQi statements can be nested before finding a RTN.

## Looping

- **ISG nn** - The increment and skip if greater command (**ISG nn**) is useful for looping in programs. This function increments the value in the specified memory register (**nn**) by 1.0 and tests it against the value in the X register. If the value of the memory register is less than or equal to the X register, the next program step is executed. If its value is greater than the X register, the next program step is skipped.
- **DSE nn** - The decrement and skip if equal command (**DSE nn**) is useful for looping in programs. This function decrements the value in the specified memory register (**nn**) by 1.0 and tests it against the value in the X register. If the value of the memory register is greater than the X register, the next program step is executed. If its value is less than or equal to the X register, the next program step is skipped.

The following example shows the factorial program from Section 1.3.2. It has been modified to include the ISG and DSE commands. The original program and the two new programs are shown below.

<b>Original</b>	<b>ISG Command</b>	<b>DSE Command</b>
“Enter Number”	“Enter Number”	“Enter Number”
STOa 1	STOa 1	STOa 1
“Factorial”	“Factorial”	“Factorial”
prmt 1	prmt 1	prmt 1
1	1	1
STO 02	STO 02	STO 02
LBL 00	STO 03	LBL 00
RCL 01	LBL 00	RCL 01
MUL 02	RCL 02	MUL 02
1	MUL 03	0.0
SUB 01	RCL 01	DSE 01
RCL 01	ISG 02	GTO 00
0.0	GTO 00	RCL 02
x < y	RCL 03	STOP
GTO 00	STOP	
RCL 02		
STOP		

## Entry and Exit Subroutines

The **entry** and **exit** functions are useful for initializing macros or for clean-up after a macro is executed. These commands are used together with the **end** command to define special subroutines that are executed only at the beginning or end of a macro. These functions must be placed after the **STOP** command at the end of the macro.

The factorial example below shows the use of the entry and exit commands.

### Factorial Example

```

LBL 00
RCL 01           ; Recall the contents of memory register 01.
MUL 00          ; Multiply the contents of 00 by the X register.
0.0
DSE 01         ; Test to see if the contents of 01 > 0.0.
GTO 00         ; Yes: Go to label 00.
STOP          ; No: Stop the program.

entry          ; This part is executed before anything else.
"Enter Number" ; Place this string in the alpha register.
STOa 1        ; Store the string in alpha memory 1.
"Factorial"   ; Use this string as the title of the dialog.
prmt 1       ; Prompt the user to input a number.
1
STO 00       ; Store 1.0 in memory register 00.
end          ; End the entry function.

exit          ; This part is executed after the STOP command is reached.
RCL 00       ; Recall the results of the macro.
"Answer ="   ; Use this string as the title of the output dialog.
view 1      ; View the results of the macro (memory register 00).
end          ; End the exit function.

```

### 1.4.4 Conditional Tests

A conditional test allows you to make decisions during the execution of a program. The basic format is:

```
If a conditional test is TRUE
    then execute the next statement
else
    skip the next step and jump to the following statement
```

The conditional test is used to determine whether or not the next statement should be executed or skipped. This statement can be any valid command. A complete list of the conditional tests follows:

- **x > y** True if X register > Y register.
- **x < y** True if X register < Y register.
- **x <= y** True if X register <= Y register.
- **x >= y** True if X register >= Y register.
- **x != y** True if X register is not equal to the Y register.
- **x = y** True if X register = Y register.

For example:

```
RCL 05 ;Recall the contents of memory register 05.
RCL 06 ;Recall the contents of memory register 06.
x > y ;Test to see if value in 06 is greater than value in 05.
GTO 20 ;Yes - Perform this statement, No - Move on to the next step.
GTO 80
```

**Note:** To use these commands in a macro, they must be typed correctly into the program editor. A space must be entered after the **x** and before the **y** for KaleidaGraph to recognize the command.

## 1.4.5 Boolean Functions

### Boolean Tests

Boolean tests are similar to conditional tests in that the values in the X and Y registers are being compared. The difference is that the Boolean places the result of the comparison (0=false, non-zero = true) in the X register and drops the original X and Y values from the stack. A complete list of the Boolean tests follows:

- **bool 0**            y > x            True if Y register > X register.
- **bool 1**            y >= x           True if Y register >= X register.
- **bool 2**            y < x            True if Y register < X register.
- **bool 3**            y <= x           True if Y register <= X register.
- **bool 4**            y = x            True if Y register = X register.
- **bool 5**            y != x           True if Y register is not equal to the X register.
- **bool 6**            y || x            True if Y register **or** the X register is true.
- **bool 7**            y && x           True if Y register **and** the X register are true.

### Other Boolean Commands

- **BMv nnn** (Boolean mask vector) - If the Boolean value in the X register is true (non-zero), the current row in column **nnn** is masked.
- **BUv nnn** (Boolean unmask vector) - If the Boolean value in the X register is true (non-zero), the current row in column **nnn** is unmasked.
- **ifelse** - This command uses the bottom three registers in the stack: Boolean, Y, and X. If the Boolean is true (non-zero), the value in the Y register is placed in the X register. If the Boolean is false (0), the value in the X register remains unchanged. All other registers are dropped from the stack.

For example:

```
RCL 05            ;Recall the contents of memory register 05.
RCL 06            ;Recall the contents of memory register 06.
bool 4            ;Test to see if value in 06 is equal to value in 05.
BMv 109           ; Mask the current row in column 109 if the Boolean test above is true.
```

## 1.4.6 Addressing the Data Window

### Indirect Addressing

- **ibase** - The **ibase** command must be used to address columns outside of the 100 column default range. When this command is used, the column portion of the indirect address can range from .0 to .999. This command can appear anywhere in the macro and only needs to be used once.

### Recalling Data

- **RCL nn** - Recalls the value from memory register **nn** and places the value in the X register.
- **rclr nn** - The relative recall (**rclr nn**) reads the value from the target memory register specified in memory register **nn**. The value is placed in the X register.
- **RCLv nnn** - This function is used to recall data from a data selection on a row-by-row basis. The program is evaluated once for each row in the selection. This command skips over any masked data cells.
- **RCLi nn** - This command recalls the data value from the address specified in memory register **nn**. The integer portion of the number is used as the row address and the fractional part of the number is the column number. For example, if a memory register has 9.03 stored, a **RCLi** command returns the data value in row 9, column 3.

For example:

```
RCL 08           ;Recalls the value in memory register 08.  
rclr 20         ;Recalls the value from the memory register specified in 20.  
RCLv 000       ;Recalls the first column of a selection on a row by row basis.  
RCLi 05        ;Recalls the data from the address specified in 05.
```

- **getcell** - Gets the value of a cell at a specific address and places the value in the X register. The address is specified by the bottom two registers in the stack (X and Y). The column address value must be less than 1000 and is located at the bottom of the stack (X). The row address value is in the stack above it (Y). When the command is executed, the addresses are removed from the stack and the value that is retrieved from the cell is placed in the X register.

For example:

```
45              ;The first value is placed on the stack. This is the row address (Y).  
110            ;The next value is placed on the stack This is the column address (X).  
getcell        ;Puts the value of the cell at row 45, column 110 in the X register.
```

**Note:** If a macro tries to recall a number in a row or column that does not exist, it stops with a message that you specified a position that has not been created.

## Storing Data

- **STO nn** - Stores the value of the X register in memory register **nn**.
- **stor nn** - The relative store (**stor nn**) places the value in the X register into the target memory register specified in memory register **nn**.
- **STOv nnn** - This command is used to store the results of a vector calculation. The results are stored on a row-by-row basis in column **nnn**. As a special case, if an alpha command is placed immediately before a **STOv** command, the text string is written into the output column, provided the column is formatted as Text.
- **STOi nn** - This command stores the value of the X register in the address specified in memory register **nn**. The integer portion of the number is used as the row address and the fractional part of the number is the column number. As a special case, if an alpha command is placed immediately before a **STOi** command, the text string is written into the output column, provided the column is formatted as Text.

For example:

```
STO 12           ;Stores the value in the X register into memory register 12.
stor 22          ;Stores the X register in the memory register specified in 22.
STOv 101        ;Stores the results in column 101 of the data window.
STOi 15         ;Stores the X register in the address specified in memory register 15.
"Pass"          ;This string is placed in the alpha register.
STOv 10         ;Places the text string "Pass" in column 10, provided this column is formatted
                as a Text column.
```

- **setcell** - Stores the value of the X register in a cell at a specific address. The value to be stored is in the bottom register of the stack (X). The address is specified by the next two registers in the stack (Y and Z). The column address value must be less than 1000 and is located in the Y register. The row address value is in the Z register. When the command is executed, the addresses are removed from the stack and the X register is left undisturbed.

For example:

```
35              ;The first value is placed on the stack. This is the row address (Z).
112             ;The next value is placed on the stack. This is the column address (Y).
13.98          ;The value to be stored is placed on the stack in the X register.
setcell        ;Places 13.98 in the cell at row 35, column 112.
```

**Note:** If a program tries to store a number in a row or column that has not been created, KaleidaGraph expands the data window. This is limited by the maximum number of columns in a data window (1000) and the amount of available memory.

## Error Conditions

There are three commands that are used to test for errors. The first test is used to detect any errors during the execution of the program. The other two tests are used to detect recall indirect command (**RCLi**) errors.

- **test 0** - This test is true if any type of error occurs during the execution of the program. Once this flag is set, it cannot be reset until the program stops.
- **test 1** - This test is used to determine if a data cell contains a number before using the value in a calculation. The test is true if the last **RCLi** command attempted to read data from an empty cell. The flag is reset when the next **RCLi** instruction is executed.
- **test 2** - This test is used to determine if a data cell is masked before trying to use the value in a calculation. The test is true if the last **RCLi** command attempted to read data from a masked cell. The flag is reset when the next **RCLi** instruction is executed.

For example:

```
RCLi 06      ;Recall data from cell specified in memory register 06.  
test 2      ;Determine if the data is masked.  
GTO 30      ;If the data is masked, go to label 30.  
XEQ 50      ;If the data is not masked, execute label 50.
```

## Clearing Data Cells

- **CLRv nnn** - This command clears the current cell during a vector calculation. A macro using this command is repeatedly executed on a row-by-row basis in the data selection.
- **CLRi nn** - This command clears the cell specified by the address in memory register **nn**. The integer portion of the number is used as the row address and the fractional part of the number is the column number.

For example:

```
CLRv 202    ;Clears the contents of the current cell in column 202.  
CLRi 08     ;Clears the cell specified by the address in memory register 08.
```

## Masking/Unmasking Data

- **Mv nnn** - This command masks the current cell during a vector calculation. A macro using this command is repeatedly executed on a row-by-row basis in the data selection.
- **UMv nnn** - This command unmaskes the current cell during a vector calculation. A macro using this command is repeatedly executed on a row-by-row basis.
- **Mi nn** - This command masks the cell specified by the address in memory register **nn**. The integer portion of the number is used as the row address and the fractional part of the number is the column number.
- **UMi nn** - This command unmaskes the cell specified by the address in memory register **nn**. The integer portion of the number is used as the row address and the fractional part of the number is the column number.

For example:

Mv 109	;Masks the current cell in column 109.
UMv 200	;Unmasks the current cell in column 200.
Mi 18	;Masks the cell specified by the address in memory register 18.
UMi 12	;Unmasks the cell specified by the address in memory register 12.

## Other Commands

- **index** - The **index** command is used in macros that use vector addressing. It enables the index number of the current row to be referenced in calculations. The index number is referenced to the start of a data selection. The index command returns a value of 0 for the first row in a selection.
- **size** - The **size** command determines the number of rows and columns in the current data selection. If no selection exists, the size of the entire data window is determined (including blank rows and columns). The number of rows is placed in the X register and the number of columns is placed in the Y register.

### 1.4.7 Using Strings

The Macro Calculator allows up to 100 **strings** in a single program. Strings can be used to name columns in a data window, to prompt the user for input, or to label an output display.

Strings can be up to 31 characters in length. The first and last characters of the string should be the double-quote (") character.

For example:

```
"Enter the initial column to be cleared" ;sample string
```

The **alpha register** is an important factor in the use of strings. The alpha register is a temporary storage place for strings. Strings are placed in the alpha register when they are encountered in the execution of a program.

**Note:** The alpha register and its memory locations are independent of the normal data memory locations and calculator stack. It is possible to read and write strings from the data window. Text that is recalled from a data window is stored in the alpha register; text that is placed in a data window is written from the alpha register.

- **STOa n** - The **STOa** command stores a string once it has been placed in the alpha register. The string is stored in any one of ten (0–9) alpha memory locations.
- **RCLa n** - The **RCLa** command recalls the contents of a specific alpha memory location and places the string in the alpha register.
- **Ncol nnn** - This command is used to name data columns in the active data window. Specify a column number (0–999) and the selected column is renamed with the contents of the alpha register.

For example:

```
"Month"           ;This string is placed in the alpha register.
STOa 1            ;Stores the string in alpha memory location 1.
RCLa 1           ;Recalls the contents of alpha memory 1 to the alpha register.
Ncol 120         ;Renames column 120 with the contents of the alpha register.
```

- **load** - This command loads a plot script specified by an alpha string or a RCLa command. The script is loaded from your launch directory (program, style, or macro), or from the last directory from which a script was loaded during the current session.
- **srun** - This command executes the current plot script.

For example:

```
"Script5"        ;This string is the name of a plot script.
load             ;Load the plot script.
srun            ;Execute the current plot script.
```

### 1.4.8 Input and Output

- **prmt n** - The **prmt** command is used to input a single value into a macro during execution. This command is useful if a program has information or constants that change each time the program is executed.

When the prompt command is executed, a dialog with a title, a prompt string, and an edit field is displayed. The field displays a default value and allows you to change its value. The default value can be defined by storing a value into the same memory register the prmt command accesses when it is executed.

Figure 1-14 shows the dialog that is displayed when the following portion of the Factorial macro is executed.

```
"Enter Number"           ; Enter a string into the alpha register.
STOa 2                   ; Store the alpha register in alpha memory 2.
5.0                       ; This is the default value that will appear in the dialog.
STO 02                    ; Store this value in memory register 02.
"Factorial"              ; This string is used as the title of the dialog.
prmt 2                   ; Prompt the user to enter a value. Store the value in register 02.
(Rest of the macro)
```

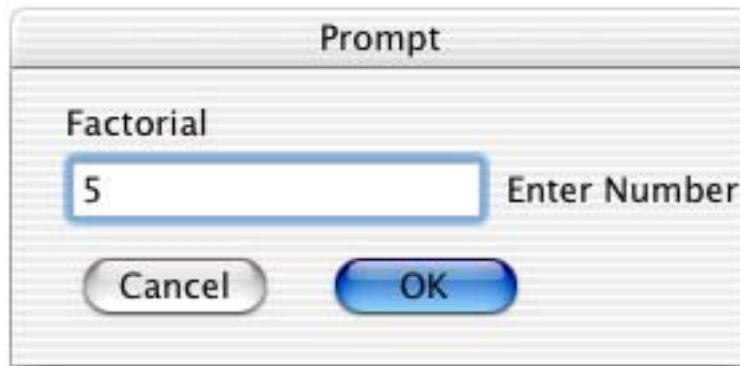


Figure 1-14 Prompt dialog

This command makes use of the contents of the alpha register, an alpha memory location, and a numeric memory location. The contents of the alpha register are used as the title of the dialog and alpha memory **n** is used as the prompt string. If you click **OK**, the value entered in the dialog is placed in data memory location **n**. Thus, the first ten data memory registers (0–9) are matched with the ten alpha memories (0–9).

- **inpt n** - The **inpt** function is also used to supply input into a program. The advantage of this command over the prompt command is that you are able to enter up to six different values at one time.

When the **inpt** command is executed, a dialog is displayed with a title and up to six different strings and numeric fields. Each field displays a default value and allows you to change its value. The default values can be defined by storing a constant in the data memory register which corresponds to the alpha memory register displayed to the right of the field.

Figure 1-15 shows the dialog that appears when the following portion of the Clear macro is run. The entire macro appears in Section 1.3.5.

```

"Initial Column to be Cleared"           ; Enter string into alpha register.
STOa 0                                   ; Store in alpha register 0.
0.0                                       ; Enter default value.
STO 00                                   ; Store default value in register 00.
"Last Column to be Cleared"             ; Enter string into alpha register.
STOa 1                                   ; Store in alpha register 1.
1.0                                       ; Enter default value.
STO 01                                   ; Store default value in register 01.
"Clear Macro - Multiple Columns"        ; Enter title string into alpha register.
inpt 2                                   ; Input 2 values. Store them in registers 00 and 01.
(Rest of the macro)

```

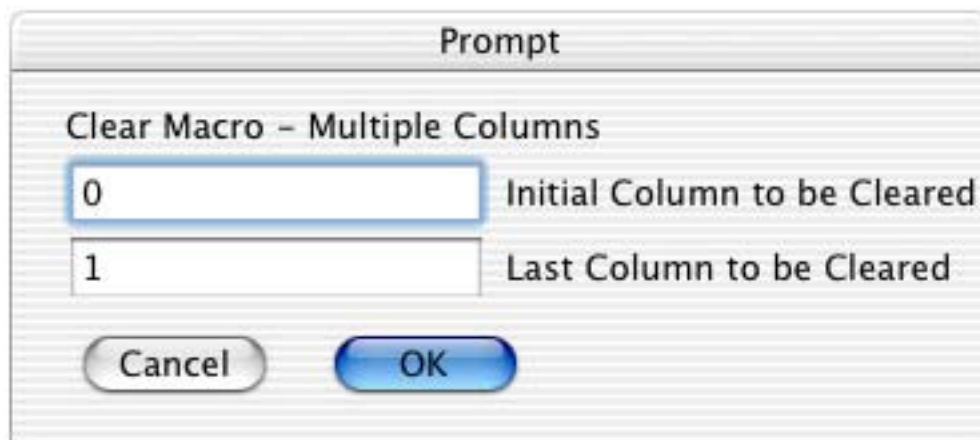


Figure 1-15 Input dialog

This command uses the contents of the alpha register, and up to six alpha and data memory locations (0–5). The title of the dialog is taken from the contents of the alpha register. The contents of the alpha memory locations are displayed to the right of the fields, which contain the default values. If you click **OK**, the values entered in the dialog are placed in the appropriate data memory registers.

**Note:** For the **inpt** command, the value of **n** must be between 1 and 6.

- **view n** - The **view** command is useful for displaying the results of a calculation. Up to nine variables can be displayed at once.

When this command is executed, a dialog appears which contains a title and a maximum of nine different output values. Each of the output variables can also have a string to describe its meaning. The output values are stored in memory registers 00 through 08 and the strings are stored in the corresponding alpha memory locations. The title of the dialog is taken from the alpha register.

Figure 1-16 shows the input dialog and Figure 1-17 shows the output dialog when the Factorial macro below is executed.

```

"Enter Number"                                ; Enter a string into the alpha register.
STOa 1                                         ; Store the alpha register in alpha memory 1.
"Factorial"                                    ; This string is used as the title of the dialog.
prmt 1                                         ; Prompt the user to enter a value. Store the value in register 01.
1
STO 00                                         ; Store 1 in memory register 00.
LBL 00
RCL 01                                         ; Recall the contents of register 01 to the X register.
MUL 00                                         ; Multiply the contents of 00 times the contents of the X register.
0,0
DSE 01                                         ; Decrement register 01 and test it against the X register.
GTO 00                                         ; If register 01 > X register, go to LBL 00.
"Answer ="                                     ; Load string into alpha register.
view 1                                         ; Display 1 output variable (from memory register 00).
STOP
    
```

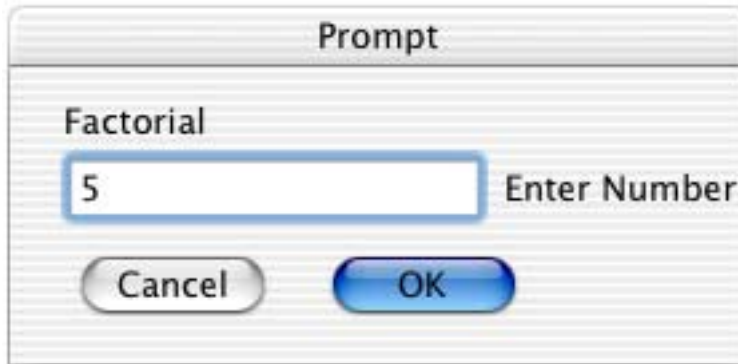


Figure 1-16 Prompt dialog

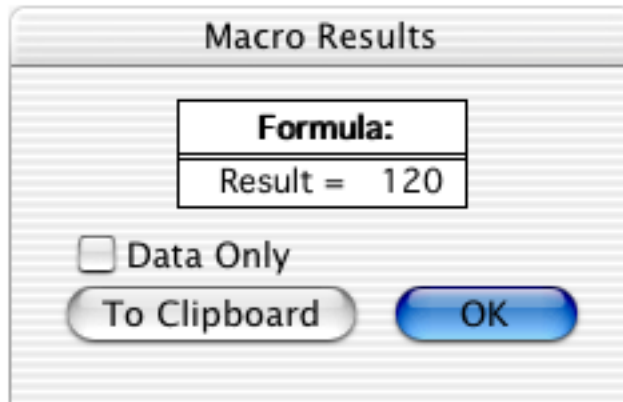


Figure 1-17 View dialog

## 1.4.9 Register Calculations

### Single Number Functions

- **abs** - This function determines the absolute value of the value in the X register.
- **erf** - Calculates the error function of the value in the X register.
- **erfc** - Calculates the complementary error function of the value in the X register.
- **exp** - Calculates the value of e (2.7182...) raised to the power specified in the X register.
- **factorial** - Calculates the factorial of the value in the X register.
- **ln** - Calculates the natural logarithm (base e) of the value in the X register.
- **log** - Calculates the common logarithm (base 10) of the value in the X register.
- **ran#** - Generates a random number between 0 and 1.
- **sqrt** - Calculates the square root of the value in the X register.
- **+/-** - Changes the sign of the value in the X register.
- **pi** - Places an approximate value of  $\pi$  (3.14159...) in the X register.
- **1/x** - Computes the reciprocal of the value in the X register.
- **x^2** - Computes the square of the value in the X register.
- **10^x** - Calculates 10 to the power of the value in the X register.
- **int** - This function determines the integer portion of the X register. If 5.08 is in the X register, the **int** command causes 5.00 to be stored in the X register.
- **frac** - This function determines the fractional portion of the X register. If 5.08 is in the X register, the **frac** command causes 0.08 to be stored in the X register.

### Two Number Functions

- **x<>y** - Exchanges the values in the X and Y registers.
- **y^x** - Raises the number in the Y register to the power specified in the X register. Negative values of Y are only allowed if X is an integer.
- **p->r** - Converts the numbers in the X and Y registers (R and  $\emptyset$ ) to rectangular coordinates (X and Y).  $\emptyset$  can be expressed in either degrees or radians, depending on the calculator setting.
- **r->p** - Converts the numbers in the X and Y registers (X and Y) to polar coordinates (R and  $\emptyset$ ).  $\emptyset$  can be expressed in either degrees or radians, depending on the calculator setting.

### Addition

- **+** - Adds the value in the Y register to the value in the X register.
- **Add nn** - Adds the value in the X register to the specified memory register **nn**.
- **Addi nn** - Adds the value in the X register to the memory location specified in memory register **nn**.

- **rsum nnn** - Computes the running sum of column **nnn** and places the result in the X register.

### Subtraction

- **-** - Subtracts the value in the X register from the value in the Y register.
- **Sub nn** - Subtracts the value in the X register from memory register **nn**.
- **Subi nn** - Subtracts the value in the X register from the memory location specified in memory register **nn**.
- **diff nnn** - Calculates the difference between each value and the one that follows it in column **nnn**.

### Multiplication

- **\*** - Multiplies the value in the Y register by the value in the X register.
- **Mul nn** - Multiplies memory register **nn** by the value in the X register.
- **Muli nn** - Multiplies the memory location specified in memory register **nn** by the value in the X register.

### Division

- **/** - Divides the value in the Y register by the value in the X register.
- **Div nn** - Divides memory register **nn** by the value in the X register.
- **Divi nn** - Divides the memory location specified in memory register **nn** by the value in the X register.
- **mod** - The **mod** function computes the integer value of X mod Y and places the result in the X register.

### Distribution

- **norm** - This function calculates the normal distribution of the value in the X register.
- **inorm** - This function calculates the inverse normal distribution of the value in the X register.

### Trigonometric Commands

- **cos** - Determines the cosine of the value in the X register. The value is in radians or degrees, based on the setting in the Macro Calculator.
- **inv-cos** - Determines the inverse cosine of the value in the X register. The value is in radians or degrees, based on the setting in the Macro Calculator.
- **inv-sin** - Determines the inverse sine of the value in the X register. The value is in radians or degrees, based on the setting in the Macro Calculator.
- **inv-tan** - Determines the inverse tangent of the value in the X register. The value is in radians or degrees, based on the setting in the Macro Calculator.
- **sin** - Determines the sine of the value in the X register. The value is in radians or degrees, based on the setting in the Macro Calculator.
- **tan** - Determines the tangent of the value in the X register. The value is in radians or degrees, based on the setting in the Macro Calculator.

- **->deg** - Assuming the X register contains a value in radians, this function computes the corresponding value in degrees.
- **->rad** - Assuming the X register contains a value in degrees, this function computes the corresponding value in radians.

### Other Commands

- **CM** - Resets the contents of the memory registers to 0.
- **FIX n** - Using this command, the calculator displays numbers rounded to the specified number of decimal places.
- **SCI n** - Using this command, the calculator displays numbers with one digit to the left and a specified number of digits to the right of the decimal point.
- **tbl** - This command performs a linear approximation of x, based on the function defined by the data in an x column and a y column. The x and y column numbers can range from 0–999. The result is that for a given value of x, this function determines a linear estimate for y. This command uses the last three registers in the stack (X, Y, and Z):

Z = x value.  
Y = y column number.  
X = x column number.

For example:

Assume c0 is the x column, c1 is the y column and 2.5 is the x value (Z=2.5, Y=1, X=0). Further, c0 is a series that starts at 0 and increments by 1 (0,1,2,3 ...) and that c1 is the square of c0. The following macro linearly interpolates between squared values in c1 to estimate the squared value for 2.5.

```

2.5                                ;This is the x value (Z).
1.0                                ;This is the y column number (Y).
0.0                                ;This is the x column number (X).
tbl                                ;Calculates the linear approximation.
STO 00                             ;Stores the result in register 00.
alpha "Result ="                   ;This string precedes the value in the dialog.
STOa 0                             ;Store the string in alpha memory 0.
alpha "Formula:"                   ;Use this string as the title of the dialog.
view 1                             ;View the results of the linear approximation.
STOP                               ;Halt execution of the program.

```

The output of the macro is displayed in Figure 1-18:

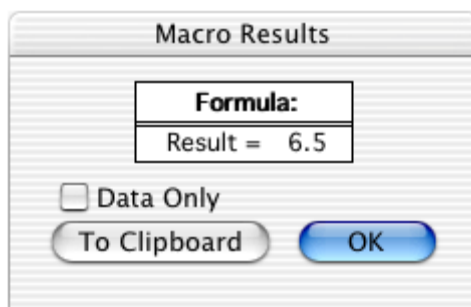


Figure 1-18 Linear approximation for the square of 2.5

## 1.4.10 Statistics

The following commands allow you to perform statistics on a specific set of data. The data to be used is determined by the row and column addresses entered before the command is executed. These commands are able to operate on the entire data window. The addresses for these commands are stored in the bottom four registers of the stack (S0–S3). All of these commands follow the same pattern shown below:

S3 contains the number of the starting row.  
S2 contains the number of the ending row.  
S1 contains the number of the starting column.  
S0 (the X register) contains the number of the ending column.

After the command is executed, the four addresses are removed from the stack and the result of the function is placed in the X register.

- **cmin** - Determines the minimum value within the specified data range.
- **cmax** - Determines the maximum value within the specified data range.
- **csum** - Calculates the sum of all the values in the data range.
- **npts** - Determines the number of points within the specified data range.
- **mean** - Calculates the mean of the data range.
- **median** - Calculates the median of the data range.
- **rms** - Calculates the root mean square of the data range.
- **std** - Calculates the standard deviation of the data range.
- **var** - Calculates the variance of the data range.
- **stderr** - Calculates the standard error of the data range.
- **skew** - Calculates the skewness of the data range.
- **kurtosis** - Calculates the kurtosis of the data range.

For example:

0	;The first value, the starting row address (S3), is placed on the stack.
127	;The second value, the ending row address (S2), is placed on the stack.
100	;The third value, the starting column number (S1), is placed on the stack.
101	;The last value, the ending column number (S0), is placed on the stack.
cmin	;Determines the minimum value within this data range and places the result in the X register.

## 1.4.11 Curve Fit Functions

The following commands are useful if a curve fit has been applied to any of the columns in the data window. Each command finds  $f(x)$  for a specific value of  $x$ . The value in the X register is used for the value of  $x$ . The address of the data column which has the appropriate curve fit is used for the value of **nnn** in the command.

**Note:** The answer is only accurate if  $x$  is within the original range of data. If  $x$  lies outside of the original range, the answer is **linearly** interpolated.

### Non-Linear Curve Fits

- **gen nnn** - This command calculates the value of the General curve fit applied to column **nnn** at the value specified in the X register.
- **genf nnn** - This command is similar to the **gen nnn** command. The difference is that the **genf nnn** command makes use of the alpha register to determine which General curve fit should be used to determine the value of  $x$ . This is useful if more than one general curve fit is applied to a column of data.

For example:

```
"fit1"           ;The name of the general fit is placed in the alpha register.
25.48           ;Place the value of x in the X register.
genf 150        ;Determine a value of y based on the fit1 curve fit applied to column 150.
```

### Least Squares Curve Fits

- **lin nnn** - This command calculates the value of the Linear curve fit applied to column **nnn** at the value specified in the X register.
- **poly nnn** - This command calculates the value of the Polynomial curve fit applied to column **nnn** at the value specified in the X register.
- **expr nnn** - This command calculates the value of the Exponential curve fit applied to column **nnn** at the value specified in the X register.
- **logr nnn** - This command calculates the value of the Logarithmic curve fit applied to column **nnn** at the value specified in the X register.
- **pow nnn** - This command calculates the value of the Power curve fit applied to column **nnn** at the value specified in the X register.

## Smoothing Curve Fits

- **smh nnn** - This command calculates the value of the Smooth curve fit applied to column **nnn** at the value specified in the X register.
- **wgt nnn** - This command calculates the value of the Weighted curve fit applied to column **nnn** at the value specified in the X register.
- **spln nnn** - This command calculates the value of the Cubic Spline curve fit applied to column **nnn** at the value specified in the X register.
- **intp nnn** - This command calculates the value of the Interpolate curve fit applied to column **nnn** at the value specified in the X register.

For example:

```
56.74           ;Place the value of x in the X register.
spln 102        ;Determine a value of y based on the cubic spline curve fit applied to
                column 102.
```

## 1.4.12 Miscellaneous Commands

- **peek nn** - This command displays the value of any specified memory register (**nn**) without changing the contents of the X register. The result of this command is placed in the display region of the Macro Calculator. This can be an invaluable aid in debugging a macro, where the address of an indirect recall or store operation is calculated in the program.
- **abort** - The **abort** command can be used to stop the execution of a macro. This is particularly useful if an error is detected during the execution of the program.
- **version** - This command returns the current KaleidaGraph version number. The version number is multiplied by 100 and placed in the X register.
- **up** - This command moves all of the values in the stack up one position.
- **down** - This command moves all of the values in the stack down one position.

For example:

Stack registers	Original stack	Original stack after the Down command	Original stack after the Up command
S3	5	5	8
S2	8	5	15
S1	15	8	29
S0	29	15	0