
User's Guide For RPN: An RPN Interface for the TI89 and TI92+

RPN Version 2.02

Program by Lars Frederiksen

User's guide by Doug Burkett
Created 1 May 2000
Revised 23 September 2000

RPN program © 2000 Lars Frederiksen
User's Guide © 2000 Doug Burkett

Contents

Introduction	2
Revisions	3
Installing and uninstalling RPN	5
Running and exiting RPN	7
Basic operations in RPN mode	9
The command line and stack operations	14
Saving, recalling and deleting variables	21
Using built-in 89/92+ applications	23
Executing functions, programs and commands ...	24
Using delayed evaluation and constraints	28
Algebraic mode operation	29
Menu operation	30
MODE Menu	32
MATH Menu	34
RCL Menu	53
TOOLS Menu	55
Custom Menus	63
Custom Key Assignments	65
Constants, units and unit conversion	68
Special characters and the CHAR menu	74
Supported key functions	75
Alphabetic list of functions	78

Introduction

RPN is an interface program which lets you operate a TI89 or TI92+ calculator using an RPN interface, instead of the normal algebraic interface. RPN is a stack-based interface, which is often faster to use than algebraic entry, especially for complicated expressions. RPN is an acronym for *reverse Polish notation*, which describes the method used to enter operators and operands.

RPN has been written for TI calculator users who prefer an RPN interface, or for those who want to learn to use RPN.

RPN *is not* a 'shell' or a 'kernel'. RPN should not be run with shells or kernels, or you will probably crash your calculator and lose all the memory contents.

This manual uses square brackets to indicate keys on the calculator. For example, [ENTER] means the ENTER key. [DIAMOND] means the green diamond key. [LEFT], [RIGHT], [UP] and [DOWN] refer to the cursor movement keys, which are on the blue cursor control on the 92+. [BACKSPACE] means the backspace key. N: means the contents of stack element N; for example, 2: means the contents of the second stack location. [SHIFT] is the alpha keyboard 'shift' key; that is, the key that results in *A* instead of *a*.

I assume that you are familiar with normal operation of your calculator. For example, I refer to shifted key functions such as [DIAMOND] [UNITS] as just [UNITS].

Since RPN is both the name of the program and the name of an operation method, some confusion may result. The meaning is usually clear from the context.

I sometimes refer to 'normal' or 'normal-mode' operation of the 89/92+. This means the operating mode when RPN is *not* running, and you are using the built-in algebraic interface.

If you have questions, comments or bug reports about RPN, you may email Lars at

lrf@post8.tele.dk

Please do not ask for more programs.

Lars and I also thank the beta testers: Art Belmonte, Rafael Humberto Padilla Velazquez, and TiArc Jason.

This manual is distributed as an Acrobat Reader PDF file. The file is bookmarked extensively, in more detail than the table of contents. If the bookmarks are not automatically shown in a window at the left in Acrobat Reader, use the icon in the menu bar to display the bookmarks.

{Manual author's note: I would like to thank Lars for letting me try out RPN, as well as write this manual. I would also like to thank him for considering and implementing my suggestions.}

Important!

RPN 2.02 does not run reliably on HW2 calculators with AMS 2.05. See the *Crashes* section below, in *Installing and Uninstalling RPN*.

Revisions

Functional changes to RPN program version 2.02 from version 1.01:

- RPN will crash on HW2 calculators with AMS 2.05. This is caused by TI's 24K ASM program limit. There is no known fix.
- [COPY] can be used on marked expressions in the stack.
- These keys operate on command-line expressions: [COPY], [CUT], [PASTE], [SHIFT][LEFT], [SHIFT][RIGHT], [SHIFT][UP], [SHIFT][DOWN].
- You can use [COPY] and [PASTE] to copy expressions between normal-mode and RPN.
- [SHIFT][UP] and [SHIFT][DOWN] can be used to scroll large stack expressions.
- [ON], [OFF] and [DIAMOND][ON] can be used to turn the calculator off.
- The [APPS] key opens the normal-mode applications menu. All built-in applications can be used from RPN. [2nd] [APPS] displays the previous active application. [DIAMOND] [APPS] displays the Flash applications menu.
- On the TI92+, these keys can be used to start the normal-mode applications from RPN: [MEM], [CATALOG], [Y=], [WINDOW], [GRAPH], [TblSet], [TABLE], [VAR-LINK]. These keys can also be used on the TI89 unless certain line menus are shown.
- RPN assumes the role of the Home screen while running. The [HOME] key switches between RPN and the last open application.
- There is a new type of command that is identified by '...',n', which takes a variable number of arguments from the stack, and a number-of-arguments argument 'n' from the command line, or from stack level 1:.
- The old Var-Link menu has been moved to the [RCL] menu. The [RCL] menu can now include programs.
- The [MODE] menu has changed. The Other menu item starts the normal-mode Mode settings menu. The Customize menu item includes new options: 'Enable undo', and 'Enable system MATH'. The 'Enable system MATH' option lets you use the normal-mode Math menu, but only in algebraic mode.
- RPN now has an UNDO feature that restores the stack contents prior to the last operation. Up to 10 operations can be undone. Execute UNDO by pressing [INS], or from the Math Stack menu.
- The double-quote key ["] now automatically switches from RPN to algebraic mode. RPN mode is enabled when there are an even number of double-quotes in the command line.
- The Custom menu can now include programs.
- The Math menu has changed. The Matrix operations are organized in submenus. The LU and QR functions have been added. New menus for List, Stat and Prob functions are added.
- There are keyboard shortcuts for the Math menus: [DIAMOND][1] - [DIAMOND] [9] and [2nd][7].
- There are new functions in the Math Stack menu: PICK 2, PICK n, ROLL 2, ROLL n, DEL, DEL ...,n.
- You can now run Tlbasic and assembly programs from RPN. Assembly programs must be no-stub and include a 'number of arguments' test. Most assembly programs are *not* compatible with RPN. Running an incompatible assembly program will probably crash your calculator, hard.

- You can debug TIbasic programs from RPN.
- A new Tools menu is shown by pressing [DIAMOND][(-)]. This menu includes these submenus: Graphics, Var, Folder, NewData ...,n, and NewProb. The functions in these menus are identical to the normal-mode functions of the same names. There are *lots* of new functions and commands here.
- The [DEL] key now opens a menu to delete variables. These functions act on variable names in the stack.
- The [ANS] key now enters 'ans(1)' in the command line. Push [ENTER] to copy the expression in the normal-mode history display, at level 1:, to RPN stack level 1:. You can also edit the '1' to some other number to retrieve expressions from different levels in the history display.
- The Custom menu and [RCL] menu now display all variable types. If a variable type cannot be evaluated (for example, a PIC variable), then the variable name is pushed on the stack.
- Built-in commands can be run from the command line. TIbasic programs and functions can also be executed from the command line. The programs and functions may be archived.
- Expressions can be graphed from the stack. Use [DIAMOND][(-)] to open the Tools menu, select the Graph menu, then use the appropriate menu item.
- You can now make custom key assignments. Assignments can be functions, programs or expressions. You can also assign a command-line menu to a key. Programs or functions named as key-code names are executed by assigned keys.
- RPN does not run under any assembly shells. Trying to do so will probably result in a crash.
- RPN will not run if it is not archived.
- Auto mode (as compared to Exact and Approx) is now completely functional.
- The calculator will automatically turn off if the cursor is in stack display.

Changes to manual:

- The manual has been substantially rewritten to reflect the changes to RPN 2.02. No errors have been corrected from the 1.01 version.

Installing and uninstalling RPN

Installing RPN

To install RPN, use `newfold()` to create the folder in which you want to install RPN. For example,

```
newfold(rpnc)
```

creates the folder `rpnc` and makes it the current folder. Use GraphLink to send these files to the folder you created:

If you have an 89, send: `rpn.89z`
 `rpn_202.89z`

If you have a 92+, send: `rpn.9xz`
 `rpn_202.9xz`

Because the screen sizes and keyboards are different on the 89 and 92+, you must install the correct version of RPN. These programs `rpn()` and `rpn_202()` are shown as type ASM files in Var-Link. For the examples in this manual, I will assume that the program is installed in folder `\rpnc`.

`rpn()` can be put in any folder. `rpn_202()` must be put in the `\main` folder, or a folder with the name `\rpnc`.

RPN requires about 39,200 bytes of free RAM for installation. Both `rpn` and `rpn_202` must be archived. RPN will not start if it is not archived.

After installing RPN, use these steps to archive the program:

1. Push [2nd] [VAR-LINK] to display the Var-Link screen.
2. Push [F2] (View) to display the Var-Link View dialog box. Select the folder in which you installed RPN, and press [ENTER] to close the dialog box.
3. Push [F5] (All), then push [1] (Select All) to select both `rpn()` and `rpn_202()`.
4. Push [F1] (Manage), then push [8] (Archive Variable). RPN is now archived.

You can check which version is installed when RPN is running by pressing [MODE] [4]. An "about" screen is displayed which shows the version number, calculator version and the author's email address.

Uninstalling RPN

To uninstall RPN, delete all the variables in the folder you created when you installed RPN. Delete the folder. Delete the list variable `rpnsk` in the Main folder. RPN is now uninstalled.

Crashes

There are three known conditions under which RPN will 'crash' the calculator, or cause it to 'freeze up' or 'lock up'.

The first condition is *not* caused by a bug in RPN, but instead by corrupt or invalid system settings, which result in a crash when you try to run various assembly language programs, and not just RPN. One cause for this condition is restoring a hardware version 1 backup file to a hardware version 2 calculator. The only known solution is to reset all memory and restore your programs and variables individually, without using the GraphLink *Send Backup* command.

The second condition applies only to a HW2 TI92+. A crash may occur if *rpn()* and *rpn_201()* are *not* archived. The solution is to archive RPN before running it for the first time.

RPN was not designed or intended to be run under assembly shells, and it probably will not run. RPN is known not to run under DoorsOS. There are no plans to make RPN compatible with any assembly shells. If you try to run RPN from such a shell, you will probably crash your calculator and lose all the memory contents.

The third condition applies to HW2 calculators and AMS 2.05. RPN will crash on a TI89 or TI92+ with HW2 with AMS 2.05. The crash appears to be caused by the 24K ASM program size limit incorporated on AMS 2.05. The crash will occur when there is less than about 148K of free RAM.

One possible fix is to use the HW2PATCH v2.20 program written by Julien Muchembled. You can get this patch from www.ticalc.org. Martin Daveluy reports that installing the patch apparently fixes the crash problem. Some users prefer not to install patches like this because of possible reliability problems. While there are no known problems with HW2PATCH, patches like these are written without formal documentation from TI. These comments are not intended to criticize Mr. Muchembled or his work, but instead to alert you to possible problems.

Neither Lars nor I can recommend that you use the HW2PATCH to run RPN. It is truly unfortunate that RPN will not run on the latest TI hardware and software. Future developments may solve this problem.

It *seems* that RPN will run reliably if more than 148K of free RAM is available. This has not been proven. If you are willing to risk a crash, you can try this launcher:

```
kbdprgm2( )
Prgm
©Launcher for RPN w/RAM check
©21sep00/dburkett@infinet.com
if getConfig()[20]>155000 then
  rpnc\rpn()
else
  dialog
    text "Can't start RPN,"
    text "not enough free RAM"
  enddlog
endif
EndPrgm
```

This program is executed by pressing [DIAMOND] [2]. It will start RPN if there is more than about 148K of RAM available. Otherwise, it will display the error message. This launcher *does not* guarantee that RPN will run reliably. This launcher is not necessary with HW1 calculators.

To reduce the amount of RAM that is used, archive everything that can be archived. Don't create large variables, and delete unused variables as soon as you are done with them.

Running and exiting RPN

Running RPN

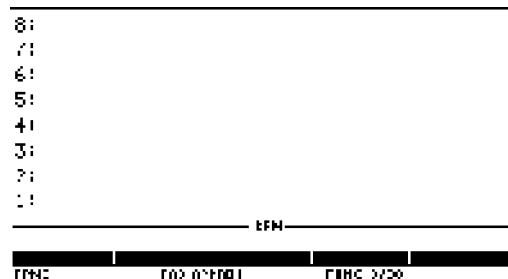
To run RPN, make the installation folder the current folder. Use the *Current Folder* item in the MODE menu, or use

```
setfold(rpnc)
```

then enter this at the command line:

```
rpn()
```

RPN will start, and you will see the stack display screen:



```
8:
7:
6:
5:
4:
3:
2:
1:
----- RPN -----
RPNC  RPNC  RPNC  RPNC
```

Note that the status indicators are shown in the status line, as in normal 89/92+ operation. In the example above, the current folder is RPNC, the angle mode is RAD, and the calculation mode is APPROX.

The command line in RPN serves the same purpose as in normal-mode operation: you use it to enter the commands and operations you want to perform. In this manual I show the command line below the stack with horizontal lines, like this:

```
3: a
2: b
1: c
-----
d
-----
```

In this example, *a*, *b*, and *c* are on the stack, and *d* is in the command line.

You can also start RPN from any current folder by specifying the installation folder. For example, if RPN is installed in a folder called *rpnc*, then enter this at the command line:

```
rpnc\rpn()
```

Exiting RPN

To exit RPN, press [QUIT], and you will return to the normal 89/92+ environment. The stack and all settings are saved when you exit RPN. You can also exit RPN by pressing [DIAMOND] [OFF], which turns the calculator off. RPN will run the next time you turn the calculator on.

Turning the calculator off from RPN

[ON], [OFF] and [DIAMOND][ON] can all be used to turn the calculator off from RPN. When you next turn the calculator on, RPN resumes running.

RPN exits (stops running) if you have switched to normal mode and turn the calculator off.

A keyboard program for starting RPN

If you often switch back and forth between normal operation and RPN, you may want to use this short program to start RPN:

```
kbdprgm2( )  
Prgm  
⊙RPN launcher  
rpnc\rpn()  
EndPrgm
```

Since this program is one of the special 'keyboard' programs, you can run it from any folder with

```
[DIAMOND] [2]
```

To use another key instead of [2] to start RPN, change the '2' in *kbdprgm2* to some other number from 1 to 9. *kbdprgm2*() must be in the *main* folder. See page 314 of the *TI89/TI92+ Guidebook* for more details on keyboard programs.

Basic operations in RPN mode

RPN may be operated in one of two modes: RPN, or algebraic. This section discusses RPN mode. The current mode is always shown in the status line just under the first stack level display. In RPN mode, the RPN is shown; in algebraic mode, ALG is shown.

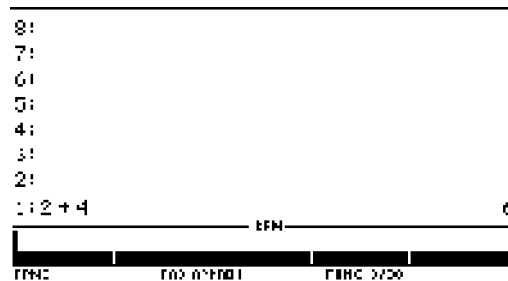
This section is a very basic description of RPN operation. If you are familiar with RPN calculators, you can probably skip this section: RPN operates like they all do.

Most keys perform the same function in RPN as they do in normal-mode operation. Function names are the same, as well as the arguments to the functions.

As an example, press

2 [ENTER] 4 +

The screen now shows



and you have just added 2 and 4, with a result of 6. This example shows all the basic principles of RPN:

1. Operations are performed on expressions on the stack or in the command line.
2. You push expressions onto the stack with the [ENTER] key
3. It is not always necessary to press [ENTER] to finish an operation.

Considering our example, keystroke by keystroke:

2	enters 2 on the command line
[ENTER]	pushes 2 onto stack level 1:
4	enters 4 on the command line
+	adds the contents of stack levels 1: and 2: which are 2 and 4

A stack is a data structure in which expressions are 'pushed on' from the bottom, and 'dropped off' from the bottom. The [ENTER] key is the means for pushing expressions onto the stack. For example, this keystroke sequence

1 [ENTER] 2 [ENTER] 3 [ENTER] 4 [ENTER]

results in this stack:

```
4: 1
3: 2
2: 3
1: 4
```

The numbers to the left of the colons in the stack display are only labels to identify the stack levels. You don't really use these numbers for anything.

Math functions require one or more arguments. In RPN, the arguments are taken from the stack. If a function requires two arguments, they are taken from stack levels 1: and 2: . If a function requires one argument, it is taken from stack level 1:

Here are some examples of how you would enter some simple problems involving two arguments:

```
2 + 4      2 [ENTER] 4 +
2 - 4      2 [ENTER] 4 -
2 * 4      2 [ENTER] 4 *
2 / 4      2 [ENTER] 4 /
2 ^ 4      2 [ENTER] 4 ^
```

Here are some examples of problems involving one argument:

```
sin(1)      1 [SIN]
ln(.5)      .5 [LN]
```

One common mistake made by new RPN users is using [ENTER] when it is not necessary. For example, you could evaluate 2 + 4 like this:

```
2 [ENTER] 4 [ENTER] +
```

This will work, but the second [ENTER] is not necessary. You never need to press [ENTER] before pressing a function or operator key.

Substantial keystroke savings result on complicated expressions, because you don't need to enter and keep track of the parenthesis as you would with algebraic entry. For example, consider this calculation:

$$\frac{1 - \sqrt{\frac{\sin(.7)}{(2.3-4.6)^2}}}{\left(\frac{4^{\frac{7}{3}}}{((1.2-1).5)}\right) \cdot (6 \cdot 2.1 - 4)}$$

The basic principle in solving any complicated expression is to enter the expressions and operators in the same order as if you were solving the problem by hand. This means starting at the inner-most expressions. So, the keystrokes to solve this problem are:

```
1 [ENTER]
.7 [SIN]
2.3 [ENTER] 4.6 -
2 ^ / √ -
4 [ENTER] 7 [ENTER] 3 / ^
1.2 [ENTER] 1 - 5 * /
6 [ENTER] 2.1 * 4 - * /
```

Note that the left-hand display of the stack shows the expression as it is being entered, so you can verify that you are doing it correctly. The right-hand side of the stack display shows intermediate results, which can also help you find mistakes faster.

While RPN is good for entering numeric expressions, it is also very useful for entering symbolic expressions. This expression

$$\frac{\left(\frac{a}{b^2} + \frac{c}{d^2}\right)^2}{\ln(\sin(b^2 - d^2))}$$

is entered with these keystrokes

```
a [ENTER] b [ENTER] 2 ^ /
c [ENTER] d [ENTER] 2 ^ / + 2 ^
b [ENTER] 2 ^ d [ENTER] 2 ^ - [SIN] [LN] /
```

Note that you can verify each step, since the intermediate results are displayed.

This version of RPN does not have a true AUTO mode. In particular, RPN will not automatically set the mode when using the zeros(), cZeros(), Solve(), cSolve() and integral functions. However, AUTO mode will result in an approximate evaluation if there is a floating point value in one of the expressions.

Use [DIAMOND] [ENTER] to evaluate the last input in the opposite mode. For example, if the mode is set to EXACT, pressing [DIAMOND] [ENTER] will evaluate the last entry in APPROXIMATE mode.

RPN assumes the role of the normal-mode HOME screen while it is running. This means that pressing [HOME] returns control to RPN, when a built-in application has been started from RPN. See the section *Using built-in 89/92+ applications* for examples.

Using UNDO in RPN

RPN supports an UNDO feature, but only in RPN mode, not algebraic mode. The purpose of the UNDO feature is to restore the stack and command line contents prior to the last operation. The undo feature is executed by pressing [INS], or by selecting UNDO from the MATH STACK menu. You can undo up to 10 stack operations.

For example,

```
2 [ENTER] 4 [+]
```

results in 6 in stack level 1:. If you use UNDO at this point, then 2 is restored to stack level 1: and 4 is restored to the command line.

As another example, suppose you use these keystrokes to create a list:

```
1 [ENTER] 2 [ENTER] 3 [ENTER] 3 Stack►List
```

Using UNDO at this point results in this stack and command line:

```
3: 3
2: 2
1: 1
-----
3
-----
```

You can enable and disable the UNDO feature in the MODE Customize menu, with the *Enable UNDO* setting.

Cut, Copy, Paste and marking text

RPN supports cut, copy and paste features where appropriate. On the TI89, you use the labelled [CUT], [COPY] and [PASTE] keys. The TI92+ does not have labelled keys; instead, use these shortcuts:

```
[DIAMOND] [X] is Cut
[DIAMOND] [C] is Copy
[DIAMOND] [V] is Paste
```

All of these operations work on expressions in the command line. Only Copy can be used when an expression in the stack is marked. Mark the text to cut or copy with [SHIFT] and [LEFT] or [RIGHT]. [SHIFT] [UP] and [SHIFT] [DOWN] also mark text. [SHIFT] [UP] marks the text from the cursor to the beginning of the line. [SHIFT] [DOWN] marks the text from the cursor to the end of the line.

Note that Copy and Paste can be used to move expressions between RPN and the normal-mode command line. Just copy the expression in RPN, exit RPN, then paste the expression in the command line.

Copying stack expressions to the command line

To copy an expression from the stack display to the command line, use [UP] to choose the expression. Then use Copy to copy the expression. Use [DOWN] to move the cursor back to the command line, and paste the expression with Paste.

Another method is:

1. Push [F6] to switch to algebraic mode.
2. Use [UP] and [DOWN] to highlight the desired expression.
3. Push [ENTER] to copy the expression to the command line.
4. Push [F6] to switch back to RPN mode.

Entering conditional expressions

Some operations require conditional expressions, for example, $x > 0$. To enter this conditional expression, use

```
x [ENTER] 0 >
```

The *with* operator, |, is just as useful in RPN as it is in algebraic mode. In RPN, *with* is an operator like any other. It combines the expression in stack level 2: with the constraint in stack level 1:. To create an expression with this operator, enter the expression, then enter the constraint, then press [)]. For example, to enter

$$2*x+3|x=3$$

use these keystrokes:

```
[2] [ENTER] [x] [*] [3] [+]
[x] [ENTER] [3] [=]
[|]
```

Using RPN stack results in normal-mode operation

You may want to use an RPN stack result in 'normal' 89/92+ operation. The easiest method is to copy the expression in RPN, exit RPN, and paste the expression in the normal mode command line.

You can also define this function:

```
expr(main\rpnstk[dim(main\rpnstk)-nn*2+1])→ansrpn(nn)
```

`ansrpn(nn)` returns the expression in stack level *nn* to the command line in 'normal' mode. For example, if the stack contains

```
3: 33
2: 22
1: 11
```

then `ansrpn(1)` returns 11, and `ansrpn(3)` returns 33. Since the stack is saved in a list called `main\rpnstk`, you can also use the normal-mode Data/Matrix editor (under the [APPS] key) to copy and paste stack expressions.

Using normal-mode history results in RPN

You can copy the contents of the normal-mode history display to the RPN stack, by using the `ans()` function. Enter the number of the history entry on stack level 1:, then execute `ans()`. For example, suppose that you have the expression $ax^2 + b$ on the normal-mode history display level 1. Use this

```
1 [ENTER]
ans [ENTER]
```

and $ax^2 + b$ is pushed on the RPN stack on level 1:.

You can also copy the expression from the history area or command line in normal-mode operation, and paste the expression in RPN.

Finally, you can use the [ANS] key to paste `ans(1)` to the command line, the push [ENTER]. This pushes the expression in the normal-mode history level 1: to stack level 1:.

The command line and stack operations

You can edit entries in the command line until [ENTER] is pressed. The [LEFT] and [RIGHT] cursor keys move the cursor left and right. [BACKSPACE] deletes the character to the left of the cursor. [2nd] [LEFT] and [2nd] [RIGHT] move the cursor to the beginning and end of the entry.

In the command line, the [CLEAR] key operates as usual:

1. The current entry is deleted if the cursor is at the end of the expression.
2. Characters to the right of the cursor are deleted if the cursor is not at the end of the expression.

Use [BACKSPACE] when the command line is empty to delete the expression on stack level 1:.

The entire stack is cleared by pressing [DIAMOND] [CLEAR].

To copy the expression on stack level 1: to stack level 2:, press [ENTER] when the command line is empty. This is useful when you may want to repeat a calculation in a different mode, or with different argument.

Use the [UP] and [DOWN] cursor keys to display and select items that are on the stack, in the same way that the normal 89/92+ history display is used. When an item is highlighted in the stack display, you can use these keys:

- [ENTER] copies the highlighted expression to stack level 1:.
- [BACKSPACE] and [CLEAR] delete the highlighted expression and drop the remaining stack entries.
- [LEFT] and [RIGHT] scroll through expressions that are too large to be completely displayed.
- [2nd] [LEFT] and [2nd] [RIGHT] move the cursor to the beginning and end of large expressions.
- [SHIFT] [UP] and [SHIFT] [DOWN] scroll through large expressions on the stack.
- The Copy feature ([COPY] on the TI89, [DIAMOND] [C] on the TI92+) copies the marked stack expression.

RPN supports Exact, Auto and Approximate modes as used in normal-mode operation. Use [DIAMOND] [ENTER] to repeat a calculation in the opposite calculation mode. If the mode setting is Approx, then pressing [DIAMOND] [ENTER] will evaluate the expression on stack level 1: in Exact mode. If the mode setting is Auto, expressions are evaluated in either Exact or Approximate mode according to the same rules used in normal-mode operation. [DIAMOND] [ENTER] also operates in Auto mode. These two examples assume that the mode is set to Auto:

13 [ENTER] 7 /	<i>results in</i>	13/7	<i>in stack level 1:</i>
[DIAMOND] [ENTER]	<i>results in</i>	1.857	<i>in stack level 1:</i>

13. [ENTER] 7 /	<i>results in</i>	1.857	<i>in stack level 1:</i>
[DIAMOND] [ENTER]	<i>results in</i>	13/7	<i>in stack level 1:</i>

Note that the decimal point with 13 in the second example forces evaluation in Approx mode.

The size of the stack is limited by the available memory. The stack is saved when RPN is exited, but the maximum size of the saved stack is about 16K.

Entering strings

Strings are entered with the double-quote ["] key, as in normal-mode operation. However, RPN will automatically switch between RPN and algebraic mode as the string is being entered, so that all strings can be entered.

If the command line has no double quotes, or an even number of double quotes, then pressing ["] switches to algebraic mode. If the command line has an odd number of double quotes, then pressing ["] switches to RPN mode.

For example, enter the string "1+2=3":

["]	<i>RPN switches to algebraic mode</i>
1 + 2 = 3	<i>Command line now shows "1+2=3"</i>
["]	<i>RPN switches to RPN mode; the command line shows "1+2=3"</i>
[ENTER]	<i>"1+2=3" is pushed on stack level 1:</i>

As in normal-mode operation, double-quotes are embedded in strings by 'doubling' the double quotes. For example, to enter the string "1+2="3"", use

["] 1 + 2 = ["] ["] 3 ["] ["] [ENTER]

Swap function

It is often convenient to be able to swap the expressions on stack levels 1: and 2:. This operation is commonly called *swap* on RPN calculators. Function key [F7] is the Swap key. Swap can also be found in the MATH STACK menu.

For example, if the stack is

3: 3
2: 2
1: 1

and you press [F7], the stack becomes

3: 3
2: 1
1: 2

swap also works when the cursor is in the stack. In this case, *swap* exchanges the expression on the current level with the expression above. Again, if the stack is

3: 3
2: 2
1: 1

and the cursor is on level 2:, then *swap* results in this stack:

3: 2
2: 3
1: 1

Converting between lists, matrices and stack expressions

RPN has two functions to manipulate lists and matrices:

```
list►stk  
stk►list
```

These are both found in the MATH STACK menu.

list►stk decomposes a list or matrix into stack elements, and stk►list builds list elements into a stack or matrix. These functions must be used from the MATH STACK menu; they cannot be typed in.

To decompose a list on 1:, just execute list►stk. The number of list elements is returned to stack level 1:, and the list elements are returned on the other stack levels. For example, if the stack is

```
1: {d,c,b,a}
```

then list►stk returns

```
5: d  
4: c  
3: b  
2: a  
1: 4
```

If stack level 1: is a matrix, then list►stk returns the matrix rows as lists. For example, if stack level 1: is

```
1: [[a,b][c,d]]
```

then list►stk returns

```
3: {a,b}  
2: {c,d}  
1: 2
```

Note that when list►stk is used on a matrix, the number of rows is returned in 1:, not the total number of elements in the matrix.

stk►list performs the opposite function of list►stk: it converts individual stack elements to a list or matrix. To convert stack elements to a list, enter the list elements, then the number of list elements. For example, to convert the elements a, b, c and d to a list, the stack is

```
5: a  
4: b  
3: c  
2: d  
1: 4
```

then executing stk►list results in

```
1: {a,b,c,d}
```


To convert a set of lists to a matrix, enter the matrix rows as lists, then enter the number of rows, then execute `stk►list`. For example, to build the matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

enter this on the stack

```
3: {a,b}
2: {c,d}
1: 2
```

then execute `stk►list`.

Duplicating stack expressions

RPN has two functions that duplicate expressions on the stack: DUP2 and DUPN. These are in the [MATH] STACK menu. Use these to copy stack arguments before executing functions, in case you want to execute the function again in a different mode. To copy just the expression on stack level 1:, press [ENTER], and the expression is copied to stack level 2:.

DUP2 duplicates the expressions in 1: and 2:. For example, if the stack is

```
2: b
1: a
```

then executing DUP2 results in

```
4: b
3: a
2: b
1: a
```

DUPN is similar to DUP2, but DUPN duplicates the number of stack elements in (1:). For example, to duplicate the three stack elements a, b and c, the stack is

```
4: c
3: b
2: a
1: 3
```

then executing DUPN results in

```
6: c
5: b
4: a
3: c
2: b
1: a
```

PICK functions

The pick functions copy a stack level expression and push it onto stack level 1:. RPN has two PICK functions, PICK 2 and PICK n. PICK 2 takes no arguments and pushes the expression in stack level 2: to stack level 1. PICK n takes an argument 'n' in the command line, and pushes stack expression n: to stack level 1.

For example, if the stack is

		4: 3	
3: 3			3: 2
2: 2	and PICK 2 is executed, then the stack becomes		2: 1
1: 1			1: 2

As another example, pick the third stack expression with PICK n. Suppose the stack is

3: 3
2: 2
1: 1

then enter 3 in the command line, execute PICK n, then the stack is

4: 3
3: 2
2: 1
1: 3

Both PICK 2 and PICK n are in the MATH STACK menu. The argument for PICK n must be in the command line. If there is no expression in the command line, PICK n has no effect.

ROLL functions

The ROLL functions *cut* an expression from the stack, and push it on stack level 1:. The term *roll* is used because the effect is to roll the stack expressions without adding another expression to the stack. There are two ROLL functions, ROLL 3 and ROLL n. Both are found in the MATH STACK menu. ROLL 3 rolls the expression on stack level 3: to stack level 1:. ROLL n rolls the nth stack expression to stack level 1:.

As an example of ROLL 3, suppose the stack is

3: 3
2: 2
1: 1

When ROLL 3 is executed, the stack becomes

3: 2
2: 1
1: 3

As an example of ROLL n , if the stack is

```
5: 5
4: 4
3: 3
2: 2
1: 1
```

and 4 is entered on the command line, this is equivalent to ROLL 4. When ROLL n is executed, the stack becomes

```
5: 5
4: 3
3: 2
2: 1
1: 4
```

The argument n for ROLL n may be in the command line or the stack. If the command line is empty, the argument n is dropped from the stack before ROLL n is executed.

Delete functions

RPN has two functions to delete stack expressions: DEL and DEL n . DEL deletes the expression in stack level 1:, and drops the remaining stack elements. DEL n deletes n stack elements, and drops the remaining stack elements. Both of these functions are in the MATH STACK menu.

For example, if the stack is

```
3: 3
2: 2
1: 1
```

and DEL is executed, then the stack becomes

```
3:
2: 3
1: 2
```

The argument n for DEL n can be taken from the command line. For example, if the stack is

```
5: 5
4: 4
3: 3
2: 2
1: 1
```

and you want to delete the bottom three stack elements, enter 3 in the command line, and execute DEL n. The stack becomes

```
5:  
4:  
3:  
2: 5  
1: 4.
```

The argument n for DEL n may be in the command line or in stack level 1:. If the command line is empty, n is taken from stack level 1:. n is dropped from the stack before DEL n is executed.

Saving, recalling and deleting variables

To save the expression on stack level 1: in a variable, enter the variable name on the command line and press [STO]. For example, if 1.234 is on stack level 1:, then this

var1 [STO] or var1 [ENTER] [STO]

stores 1.234 in *var1* in the current folder.

To recall a variable to the stack, enter the variable name and press [ENTER]. Another way to recall a variable is to use the [RCL] menu, described below in the *[RCL] Menu* section. Variables can also be saved and recalled with custom menus. See the *Custom Menus* section.

There are two ways to delete variables in RPN. One method is to switch to algebraic mode with [F6], and use the delvar instruction as in normal-mode operation. The other method is to use the RPN DelVar functions. These two functions are executed from the RPN DelVar menu, which is displayed when [DEL] is pressed. DelVar deletes a single variable, and DelVar ..,n deletes *n* variables.

To use the DelVar functions, the variable names must be pushed on the stack. This is accomplished by:

1. Setting the [RCL] menu to line mode, with the MODE Customize menu.
2. Setting the [RCL] menu to display the variables.
3. Using [DIAMOND] [F1 - F5] to push the variable names on the stack.
4. Using DelVar or DelVar ..,n to delete the variables.

Push [DEL] to display the DelVar menu. The DelVar menu has two options:

1:DelVar
2:DelVar ..,n

These steps show how to set the [RCL] menu to a line menu, and select the Variables display:

1. Press [MODE], then [3] to choose Customize.
2. Press [2] to choose RCL menu.
3. Press [2] to choose Line, then press [ENTER].
4. Press [ESC], [ESC] to close the menus.
5. Press [RCL] to display the RCL menu.
6. Press the function key corresponding to the desired folder. For example, if the /main folder is shown as [F1] on my calculator, I press [F1].
7. Press [F1] to choose the VAR (variable) display.

At this point you can push the variable names on the stack with [DIAMOND] [F1] through [DIAMOND] [F5]. Use [DIAMOND] [LEFT] and [DIAMOND] [RIGHT] to scroll through the list of variables in the line menu.

For example, suppose you have five variables a, b, c, d and e in your /main folder. The Variable command line menu looks like this:

F1-a F2-b F3-c F4-d F5-e

If you push [DIAMOND] [F3], then the variable name *c* is pushed on the stack. Then, if you execute DelVar, the variable *c* is deleted.

To delete all five of these variables, you press

[DIAMOND] [F1]
[DIAMOND] [F2]
[DIAMOND] [F3]
[DIAMOND] [F4]
[DIAMOND] [F5]

and the stack is

5: a
4: b
3: c
2: d
1: e

Then use

[5]	<i>enter 5 on the command line, to delete 5 variables</i>
[DEL]	<i>display the DelVar menu</i>
[2]	<i>choose DelVar ...,n</i>

Using built-in 89/92+ applications

You can run most of the built-in 89/92+ applications from RPN. Push [APPS], and the application menu is shown:

1:FlashApps
2:Y= Editor
3:Window Editor
4:Graph
5:Table
6:Data/Matrix Editor
7:Program Editor
8:Text Editor
9:Numeric Solver
A:Home

Execute any of these applications by choosing them from the menu. To return to RPN, press [HOME]. [HOME] will also switch to the previous application from RPN. The previous application is shown by pressing [2ND] [APPS].

You can start these applications by pressing the associated key:

[MEM]	<i>memory display</i>
[VAR-LINK]	<i>VAR-LINK screen</i>
[CATALOG]	<i>built-in and user-defined functions and programs</i>

On the TI92+, these keys can be used to start the normal-mode applications from RPN:

[Y=]	<i>Y= editor</i>
[WINDOW]	<i>graph window settings</i>
[GRAPH]	<i>graph window</i>
[TblSet]	<i>table settings</i>
[TABLE]	<i>table editor</i>

These keys cannot always be used to start the applications on the TI89, because the keys are also the [DIAMOND] [F1] - [DIAMOND] [F5] function keys. In general, [F1] - [F5] can start the applications only when the keys are not otherwise in use. [F1] - [F5] are in use when:

1. The [RCL] line menu is active. In this case, [DIAMOND] [F1] - [DIAMOND] [F5] push the variable name on the stack.
2. The [UNIT] line menu is active. In this case, [DIAMOND] [F1] - [DIAMOND] [F5] are used for unit conversion.

When algebraic mode is active on the TI89, [DIAMOND] [F1] - [DIAMOND] [F5] always start the applications.

Executing functions, programs and commands

From RPN, you can execute functions, programs and commands. A *function* is a routine that may take arguments from the stack, and may return a result to the stack. A *program* may take arguments from the stack, but cannot return a result to the stack. A *command* is similar to a function. In normal-mode operation, a command takes arguments, but those arguments are not enclosed in parenthesis. Commands do not return results to the stack (or the history area, in normal-mode operation), but instead save the results to user variables or system variables. For example, LU and QR are commands, as are all the built-in regressions.

Using functions

In RPN, functions take their arguments from the stack. So, to find the sine of 1, enter

```
1 [SIN]
```

If a function takes more than one argument, all the arguments must be entered on the stack. To evaluate function *func1(a1,a2,a3)*, use

```
a1 [ENTER] a2 [ENTER] a3
```

to put the arguments a1, a2 and a3 on the stack, then execute *func1*.

There are three ways to specify a function in RPN. If the function is assigned to a key, like sine or cosine, it is evaluated by pressing the key. If the function is not assigned to a key, it can be executed by selecting it from the function menus. Or, the function can be executed simply by typing its name after the arguments are entered on the stack. For example, to use *nint()* to numerically integrate x^2 from 0 to 1, use

```
x [ENTER] 2 ^  
x [ENTER]  
0 [ENTER]  
1 [ENTER]  
[n] [i] [n] [t] [ENTER]
```

Before *nint()* is executed, the stack looks like this:

```
4 : x2  
3 : x  
2 : 0  
1 : 1
```

User functions are executed in the same way as built-in functions: put the arguments on the stack, then execute the function by typing the function name and pressing [ENTER]. You can also execute user functions by choosing them from the [RCL] menu, the [CAT] menu, or from a custom menu.

RPN can execute all built-in 89/92+ functions, even if those functions are not assigned to a key or in a menu. The function is executed by simply typing the function name at the command line. Any function arguments are on the stack as usual. For example, to execute the *avgRC()* function for $\sin(x)$, use

```
x [ENTER] [SIN]  
x [ENTER]
```



```
avgrc [ENTER]
```

In general, the stack arguments are specified as shown in the *89/92+ Guidebook*. Optional arguments are not supported for functions that can only be entered by typing the function name. For example, the avgRC() function is specified as

```
avgRC(expression,var,[h])
```

where *h* is an optional step value. Before executing avgRC(), the stack contents are

```
2: expression
```

```
1: var
```

and you cannot use the optional step size *h*.

Running TIBasic programs

You can run TIBasic programs from the command line, from the [RCL] menu, from a custom menu and from the [CATALOG] screen.

For example, suppose that the user program is called *aprog()*. To run the program from the command line, use

```
aprog [ENTER]
```

Include a folder name, if the program is not in the current folder. For example, if the program is in folder *abc*, use

```
abc\aprog [ENTER]
```

To run the program from the [RCL] menu, start the menu by pressing [RCL]. Select the folder; choose PROG, to choose PROGrams; then choose the program. The actual keystrokes used depend on whether the [RCL] menu is set as a line menu or a pop-up menu. Refer to the *[RCL] Menu* section for more details.

To run a program from the [CATALOG] screen, press [CATALOG] to display the screen, then press [F4] to choose User-Defined. Finally, choose the program with the [UP] and [DOWN] keys, press [ENTER] to paste the program name to the command line, then press [ENTER].

The programs execute as they would in normal-mode operation. When the program finishes, RPN resumes.

You can also debug TIBasic programs from RPN. If the program causes an error, or you stop the program with [ON], a dialog box is displayed. To edit the program, choose the *Enter=GOTO* option in the dialog box. The program is now loaded in the editor, but RPN resumes. To open the program editor, press [HOME]. The cursor is positioned at the line that caused the error, or the the line at which the program was stopped with [ON].

You cannot run a program that is open in the program editor, while the program editor is also open. You can use these methods to close the program editor:

1. Press [QUIT] to close the program editor.
2. Use [APPS] [7] to open a different program or function in the editor (or create a new one).

3. Use the [APPS] key to switch to a different built-in application, such as the Home screen.

Executing commands

RPN can execute most built-in 89/92+ commands. Commands can be executed from the command line, or from the MATH and Tools menus. If you run a command from the menus, the stack arguments correspond to the order given in the *89/92+ GuideBook*. For example, if the general form of a command is

```
command arg1,arg2,arg3
```

then the arguments are pushed on the stack like this:

```
3: arg1
2: arg2
3: arg3
```

Note that most commands only take variable names as arguments, and not expressions. The *[RCL] menu* section describes how to push the variable names on the stack.

You can run commands from the command line, and the arguments must also be in the command line. For example, to execute the *linreg* linear regression command on lists *xlist* and *ylist*, enter this in the command line:

```
linreg xlist,ylist
```

To run *linreg* from the menu, enter this stack:

```
2: xlist
1: ylist
```

then use [MATH] Stat Regres Lin LinReg ,.

Running assembly programs

RPN can run assembly programs in the same way as TIBasic programs. However, assembly programs that run from RPN *must* include a function argument test, which looks like this:

```
if(Number_of_Arguments < Required_Arguments)
  ER_throwVar (930);
```

Programs that do not include this test will not run reliably from RPN. Most assembly programs do not include this test, and cannot be run from RPN. Only no-stub assembly programs can be used. You should only use assembly programs that state RPN compatibility in the program documentation.

RPN is distributed with two examples of compatible assembly programs:

```
Laplace(f(var),var)    Laplace transform function
cexpand(f(var),var)    Complex partial fraction expansion
```

This information is provided to enable assembly programmers to write compatible programs.

Commands with variable number of arguments: ..,n

Some RPN functions can take a variable number of arguments from the stack. These commands are identified in the menus by the ..,n notation. *n* is the number of arguments. Some examples of this type of function are

```
DelVar ..,n
SortA ..,n
SortD ..,n
NewData ..,n
```

The number-of-arguments *n* is taken from the command line, or from stack level 1: if the command line is empty.

Consider using DelVar ..,n to delete the four variables a, b, c and d. If you enter the variable names to create this stack:

```
5: a
4: b
3: c
2: d
1: 4
```

and execute DelVar ..,n with an empty command line, the DelVar ..,n drops 4 from stack level 1:, and deletes the four variables whose name are now on stack levels 1: to 4: Alternatively, you can create this stack:

```
4: a
3: b
2: c
1: d
-----
4
-----
```

and enter 4 in the command line as shown, and execute DelVar ..,n. The four variables in stack levels 1: to 4: are deleted.

Keep in mind that ..,n function *drop the stack* to get *n* if the command line is empty. Repeating the example above, but with *n* in the stack, the stack looks like

```
5: a
4: b
3: c
2: d
1: 4
-----
-----
```

Using delayed evaluation and constraints

RPN uses a setting called *delayed evaluation* to execute expression with constraints. The [F8] function key switches delayed evaluation on and off. Delayed evaluation is shown on the stack display with the \blacklozenge character. For example, if delayed evaluated is switched off, the stack display looks like

```
3:
2:
1: 2*x+4
```

and when delayed evaluation is switched on, the stack display looks like

```
3:
2:
1 $\blacklozenge$  2*x+4
```

Use delayed evaluation to evaluate expressions with constraints. Delayed evaluation prevents the expression from being evaluated until you have added the constraints. This example shows the keystrokes and stack display to find the minimum of $z^3 - 3z^2 - 5z + 3$ for $z > 0$, in APPROX mode.

Build the expression:

```
z [ENTER] 3 [^]           1: z3
z [ENTER] 2 [^] 3 [x] [-] 1: z3 - 3z2
5 [ENTER] z [x] [-] 3 [+] 1: z3 - 3z2 - 5z + 3
```

Turn delayed evaluation on:

```
[F8]           1 $\blacklozenge$  z3 - 3z2 - 5z + 3
```

Apply fmin() function:

```
z [MATH] [4] [5]           1 $\blacklozenge$  fmin(z3 - 3z2 - 5z + 3,z)
```

Build the constraint z>0:

```
z [ENTER] 0 [>]           2 $\blacklozenge$  fmin(z3 - 3z2 - 5z + 3,z)
                          1: z>0
```

Apply the constraint:

```
[[]]           1 $\blacklozenge$  fmin(z3 - 3z2 - 5z + 3,z)|z>0
```

Turn delayed evaluation off:

```
[F8]           1: 2.63299
```

Although I used the fmin() function as an example, you can use delayed evaluation with other functions that use constraints, such as solve(), nSolve() and others.

Algebraic mode operation

While the purpose of RPN is to provide an RPN shell for the 89/92+, you may prefer to do some operations in algebraic mode. RPN supports an algebraic mode. You can switch between the two modes by pressing [F6], or by pressing [MODE], [9], [2]. To switch back to RPN mode, press [F6], or press [MODE], [8], [1].

You perform operations in algebraic mode just as you would in normal 89/92+ operation. All menus and functions work in both modes. In algebraic mode, choosing a function places the function name text at the insertion point in the command line. In RPN mode, choosing a function immediately executes the function.

Some operations which might be more convenient in algebraic mode include creating lists and matrices, and deleting variables.

You may also use algebraic mode to copy an expression on the stack to the command line, to edit it or use it again. See *Basic Operations in RPN Mode*, above.

Menu operation

Many features and functions in RPN are available through these menus.

[MODE]	<i>Customize formats and calculation modes</i>
[MATH]	<i>Math functions</i>
[RCL]	<i>Recall variables and variable names to the stack</i>
[CUSTOM]	<i>Display custom menu created by the user</i>
[DEL]	<i>Delete Variable menu</i>
[UNIT]	<i>Units, constants and unit conversion</i>

The menus are described in detail in the following sections. This section explains general menu operation.

RPN supports two types of menus: pop-up menus and command line menus. Pop-up menus appear over the stack display during use, then disappear after the item is selected. Command line menus, or just 'line menus', are shown in the menu line at the bottom of the display.

Pop-up Menus

A pop-up menu is displayed by pressing the appropriate key as shown above. Select items in the menu by pressing the number key associated with the menu item, or by selecting the menu item with [UP] and [DOWN], then pressing [ENTER]. You can press escape [ESC] at any time to remove the menu display, without making any changes. If a submenu is shown, pressing [ESC] displays the previous menu.

After a menu item is selected, another menu may be shown. In this case, make your choice from the new menu.

Command line menus

A command line menu is displayed at the bottom of the stack display. A command line menu is always displayed, unlike a pop-up menu, which disappears after you make a selection. The command menu line is divided in five sections. Each section corresponds to one of the function keys [F1] through [F5]. Pressing the displayed function key executes the menu item.

You can choose between pop-up and command line menus with MODE menu item A:Customise. Refer to the *MODE menu* section below for more details. For example, if the MATH menu type is set to Line, then pressing [MATH] results in this line menu, instead of a pop-up menu:

F1-Number F2-Test F3-Algebra F4-Calc F5-Complex

The first menu item is F1-Number. The small down-arrowhead after the menu label means that a sub-menu is available. So, in this case, pressing [F1] does not execute a function, but instead displays another line menu: F1-Exact, F2-abs, F3-round(,), and so on. Pressing F2:abs executes the abs function.

Only five menu items are displayed at once. If there are more menu items, small left- and right-arrowheads are shown at the ends of the menu line. Use [DIAMOND] [LEFT] and [DIAMOND] [RIGHT] to scroll through the menu items.

If a sub-menu is displayed, press [DIAMOND] [UP] or [DIAMOND] [DOWN] to go back to the previous menu.

You can create custom menus that display your own functions and variables. Refer to the section *Custom Menus* below.

MODE Menu

The MODE menu consists of these menu items:

- | | |
|-----------------|--|
| 1: Other | <i>display the normal-mode MODE menu</i> |
| 2: RPN/ALG (F6) | <i>switch between RPN and algebraic modes</i> |
| 3: Customize | <i>set menus to pop-up or line, and other settings</i> |
| 4: About | <i>display version number, author's email address</i> |

The rest of this section describes these menus.

1: Other

This menu item starts the normal-mode MODE display. RPN uses all of the appropriate settings in this mode. For example, use this mode menu to set the current folder, the display digits, the complex format, and all the other settings in the normal MODE screen. To close this screen, use [ENTER] or [ESC] as usual.

2: RPN/ALG [F6]

Use this menu item to switch between RPN mode and algebraic mode. The menu items are

- | | |
|--------|---------------------------|
| 1: RPN | <i>use RPN mode</i> |
| 2: ALG | <i>use algebraic mode</i> |

You can also use function key [F6] to switch between the two modes.

3: Customise

Use this menu item to customize the display of the menu items. This menu has these items:

- | | |
|-----------------------|---|
| 1: Math menu | <i>set [MATH] menu type to pop-up or command line</i> |
| 2: RCL menu | <i>set [RCL] menu type to pop-up or command line</i> |
| 3: Custom menu | <i>set [CUSTOM] menu type to pop-up or command line</i> |
| 4: CHAR menu | <i>set [CHAR] menu type to pop-up or command line</i> |
| 5: Show calculations | <i>enable calculations display on left-hand side of stack display</i> |
| 6: Enable undo | <i>enable UNDO feature</i> |
| 7: Enable system MATH | <i>enable normal-mode MATH menu in algebraic mode</i> |

The first four menu items all display this sub-menu:

- 1: Pop Up
- 2: Line

Select 1: Pop Up if you want the chosen menu to display as a pop-up menu. Select 2: Line if you want the menu items to be displayed as a command line menu at the bottom of the display.

Menu item 5: Show calculations is used to enable the display of the entries on the left-hand side of the stack display. Select Yes if you want the entries displayed, or No if you do not.

Menu item 6: Enable undo is used to enable and disable the UNDO feature. Choose Yes to enable the UNDO feature, and No to disable UNDO.

Menu item 7: `Enable system MATH` lets you choose to use the normal-mode MATH menu, instead of the default RPN MATH menu, but only when RPN is operating in algebraic mode. Choose *Yes* to use the normal-mode menu, or *No* to use the RPN menu.

These are the default Customize menu settings for RPN as it is distributed:

Math menu	Pop-up
RCL menu	Line
Custom menu	Line
CHAR menu	Line
Show calculations	Yes
Enable undo	Yes
Enable system MATH	No

4: About

This menu item shows a screen that includes the RPN version number, which calculator version is installed (89 or 92+), and the author's email address.

MATH Menu

Display the MATH menu by pressing [MATH]. The [MATH] menu may be shown as a pop-up menu or a command line menu. This menu consists of these items:

1: Number	<i>Number functions</i>
2: Test	<i>Conditional test functions</i>
3: Algebra	<i>CAS algebra functions</i>
4: Calc	<i>Calculus functions</i>
5: Complex	<i>Complex number functions</i>
6: Angle	<i>Degree and radian symbols; vector components</i>
7: Stack	<i>Stack manipulation functions</i>
8: List	<i>List functions</i>
9: Matrix	<i>Matrix functions</i>
A: Stat	<i>Statistics functions</i>
B: Prob	<i>Probability functions</i>
C: Hyperbol	<i>Hyperbolic functions</i>
D: log	<i>log() and ln() functions</i>
E: Base	<i>Number base functions</i>

In general, the function names are the same as those in normal 89/92+ operation, so you can use the *89/92+ User's Manual* for details. This manual section briefly describes the supported functions and specifies the stack arguments for the functions. In general, the stack arguments follow the same order as the arguments in normal-mode. For example, this function in normal algebraic mode

arcLen(expression, var, start, end)

would be executed with this stack:

4: expression
3: var
2: start
1: end

Most of the MATH menus can be shown with keyboard shortcuts:

[DIAMOND] [1]	Number
[DIAMOND] [2]	Test
[DIAMOND] [3]	Algebra
[DIAMOND] [4]	Calc
[DIAMOND] [5]	Complex
[DIAMOND] [6]	Stack
[DIAMOND] [7]	Matrix
[DIAMOND] [8]	Stat
[DIAMOND] [9]	Prob
[2nd] [7]	Int (integration)

You can use the normal-mode MATH menu, but only if RPN is operating in algebraic mode. Use [MODE], 3:Customize, 7:Enable system MATH to set this option.

The remainder of this section lists the submenu items in the MATH menu. The notation (*n*:) means the expression in stack level *n*:. For example, (1:) means the expression in stack level 1:.. *Boolean* means *true* or *false*.

1: Number

Number functions

1: exact	Exact evaluation functions:
1: exact()	Evaluate (1:) in exact mode, regardless of current mode setting
2: exact(,)	Evaluate (2:) in exact mode, with tolerance in (1:)
2: abs	Return absolute value of (1:)
3: round(,)	Round (2:) to digits after decimal point in (1:)
4: iPart	Return integer part of (1:)
5: fPart	Return fractional part of (1:)
6: floor	Return floor of (1:)
7: ceiling	Return ceiling of (1:)
8: sign	Return sign of (1:)
9: mod	Return (1:) modulo (2:)
A: remain	Return remainder of (1:) with respect to (2:)
B: lcm	Return least common multiple of (1:) and (2:)
C: gcd	Return greatest common denominator of (1:) and (2:)

2: Test

Conditional test functions

1: >	Return Boolean of (2:) greater than (1:)
2: <	Return Boolean of (2:) less than (1:)
3: ≥	Return Boolean of (2:) greater than or equal to (1:)
4: ≤	Return Boolean of (2:) less than or equal to (1:)
5: =	Return Boolean of (1:) equal to (2:)
6: ≠	Return Boolean of (1:) not equal to (2:)
7: not	Return Boolean of NOT(1:)
8: and	Return Boolean (1:) AND (2:)
9: or	Return Boolean of (1:) OR (2:)
A: xor	Return Boolean of (1:) exclusive-OR (2:)
B: isPrime	Return <i>true</i> if (1:) is prime; <i>false</i> otherwise

3: Algebra

Algebra and CAS functions

1: solve	Solve (2:) for real solutions of (1:). If (2:) and (1:) are lists, solve system of equations in list in (2:) for variables in list in (1:)
2: Factor	Factor functions:
1: factor()	Return factors of (1:) if (1:) is a rational number, otherwise return (1:) factored with respect to all variables
2: factor(,)	Return (2:) factored with respect to (1:)
3: Expand	Expand functions:
1: expand()	Return (1:) expanded with respect to all its variables
2: expand(,)	Return (2:) expanded with respect to (1:)
4: zeros	Return real values of (1:) which are zeros of (2:) if (1:) and (2:) are not lists, otherwise return real values of list elements of (1:) which are zeros of expressions in list in (2:)
5: approx	Evaluate (1:) in Approximate mode, regardless of current mode setting

6: comDen	Common denominator functions:
1: comDenom()	Return (1:) as reduced ratio of expanded numerator over denominator
2: comDenom(,)	Return (2:) as reduced ratio of expanded numerator over denominator with respect to (1:)
7: Prop	Proper fraction functions:
1: propFrac()	If (1:) is a rational number, return (1:) as sum of an integer and proper fraction, otherwise return (1:) as a proper fraction expression with respect to the most main variable.
2: propFrac(,)	Return (2:) expanded as a proper fraction with respect to (1:)
8: nSolve	Numerically solve (2:) for variable or guess in (1:) (Use delayed evaluation to apply constraints)
9: Trig	Trigonometric expansion and collection functions:
1: tExpand	Return trigonometric expansion of (1:)
2: tCollect	Return trigonometric collection of (1:)
A: Complex	Complex solve and factor functions:
1: cSolve	Solve (2:) for complex solutions of (1:). If (2:) and (1:) are lists, solve system of equations in list in (2:) for variables in list in (1:).
2: cZeros	Return candidate real and complex solutions to (2:) for variables in (1:). If (2:) and (1:) are lists, returns solutions to system of equations in list in (2:) for variables in list in (1:).
3: cFactor()	Return (1:) factored with respect to all variables over a common denominator
4: cFactor(,)	Return (2:) factored with respect to (1:)
B: Extract	Symbolic expression extraction functions
1: getNum	Return numerator of (1:)
2: getDenom	Return denominator of (1:)
3: left()	Return left-hand side of conditional expression in (1:)
4: right()	Return right-hand side of conditional expression in (1:)

4: Calc Calculus functions

1: Int	Integration functions:
1: $\int(,)$	Return anti-derivative of (2:) with respect to (1:)
2: $\int(,,)$	Return anti-derivative of (3:) with respect to (2:) with constant of integration (1:)
3: $\int(,,,)$	Return integral of (4:) with respect to (3:) with lower bound of (2:) and upper bound of (1:)
4: nInt	Return numerical integral of (4:) with respect to (3:) from lower bound (2:) to upper bound (1:)
2: Limit	Limit functions:
1: limit(,,)	Return limit of (3:) with respect to (2:) at point (1:)
2: limit(,,,)	Return limit of (4:) with respect to (3:) at point (2:) in direction (1:)
3: Σ	Return sum of (4:) with respect to (3:) from (2:) to (1:)
4: Π	Return product of (4:) with respect to (3:) from (2:) to (1:)
5: fMin	Return values for variable in (1:) that minimize (2:)

6: fMax	Return values for variable in (1:) that minimize (2:)
7: arcLen	Return arc length of (4:) with respect to (3:) from (2:) to (1:)
8: Taylor	Taylor series functions:
1: taylor(,,)	Return Taylor series expansion of (3:) with respect to variable (2:) of order (1:). Zero is the evaluation point.
2: taylor(,,,)	Return Taylor series expansion of (4:) with respect to variable (3:) of order (2:) evaluated at point (1:)
9: nDeriv	Numerical derivative functions:
1: nDeriv(,)	Return numerical derivative of (2:) with respect to (1:), with step value of 0.001
2: nDeriv(,,)	Return numerical derivative of (3:) with respect to (2:) with step value (1:)
A: nInt	Return numerical integral of (4:) with respect to (3:) from lower bound (2:) to upper bound (1:)
B: deSolve	Return solution to 1st or 2nd order ODE in (3:) with independent variable (2:) and dependent variable (1:)

5: Complex Complex number functions

1: conj	Return complex conjugate of (1:). (1:) may be an expression, list or matrix
2: real	Return real part of (1:). (1:) may be an expression, list or matrix
3: imag	Return imaginary part of (1:). (1:) may be an expression, list or matrix
4: angle	Return the angle of (1:), where (1:) is a complex number. (1:) may be an expression, list or matrix
5: abs	Return the absolute value of (1:). (1:) may be an expression. list or matrix. If (1:) is complex, abs() returns the modulus.

6: Angle Angle units and conversion functions:

1: °	In Radian angle mode, multiply (1:) by $\pi/180$. (1:) is unchanged in degree mode
2: °	In Degree angle mode, multiply (1:) by $180/\pi$. (1:) is unchanged in radian mode
3: P►Rx	Return the x-coordinate of (r,θ) vector, with r in (2:) and θ in (1:). (1:) and (2:) may be expressions, lists or matrices. Returns $r*\cos(\theta)$.
4: P►Ry	Return the y-coordinate of (r,θ) vector, with r in (2:) and θ in (1:). (1:) and (2:) may be expressions, lists or matrices. Returns $r*\sin(\theta)$.
5: R►Pr	Return the r-coordinate of (x,y) vector, with x in (2:) and y in (1:). (1:) and (2:) may be expressions, lists or matrices. Returns $\sqrt{x^2+y^2}$
6: R►Pθ	Return the θ-coordinate of (x,y) vector, with x in (2:) and y in (1:). (1:) and (2:) may be expressions, lists or matrices. (1:) and (2:) may be expressions, lists or matrices. In Degree mode, returns $90 \cdot \text{sign}(y) - \tan^{-1}\left(\frac{x}{y}\right)$. In Radian mode, returns $\frac{\pi \cdot \text{sign}(y)}{2} - \tan^{-1}\left(\frac{x}{y}\right)$

- 7: ►DMS Return (1:) expressed as an angle in DMS format DDDDD°MM'SS.ss". (1:) may be an expression, list or matrix. Converts to degrees if Angle mode is radians.
- 8: ►DD Return (1:) expressed as a decimal angle, where (1:) is in DMS format. (1:) may be an expression, list or matrix. (1:) is interpreted as radians or degrees depending on the Angle mode setting.

7: Stack Stack functions (there are no equivalent 89/92+ functions)

These functions are described in detail in the section *The command line and stack operations* above.

- 1: DUP 2 Duplicate (1:) and (2:): copy (1:) to (3:) and (2:) to (4:).
 2: SWAP Swap the contents of (1:) and (2:). You can also use [F7].
 3: PICK 2 Copy (2:) and push it on (1:)
 4: ROLL 3 Cut (2:) and push it on (1:)
 5: DEL Delete (1:) and drop the stack.
 6: DUP n Duplicate number of stack elements in (1:)
 7: PICK n Copy (n:) and push it on (1:)
 8: ROLL n Cut (n:) and push it on (1:)
 9: DEL n Delete the bottom *n* stack expressions and drop the stack
 A: UNDO Undo the most recent operation by restoring the stack to its previous contents.
 B: List►Stk Decompose a list or matrix in (1:) to individual stack expressions.
 C: Stk►List Combine stack expressions to a list or matrix. The number of elements in the list is in the command line, or in (1:) if the command line is empty

8: List List functions

- 1: newList Create a new list with *n* elements. *n* is taken from the command line, or from 1: if the command line is empty. The list elements are all zero.

- 2: seq Sequence functions:

- 1: seq(,,) Returns a list from these stack arguments:

- 4: *expr*
 3: *var*
 2: *low*
 1: *high*

where expression *expr* is evaluated for each value of *var*, as *var* is incremented from *low* through *high* by 1. *var* is not changed, and may not be a system variable.

- 2: seq(,,,) Returns a list from these stack arguments:

- 5: *expr*
 4: *var*
 3: *low*
 2: *high*
 1: *step*

where expression *expr* is evaluated for each value of *var*, as *var* is incremented from *low* through *high* by *step*. *var* is not changed, and may not be a system variable.

- 3: min Return the minimum of two expressions, a list, or a matrix:
- 1: min() If (1:) is a list, return the minimum element of the list.
 If (1:) is a matrix, return a row vector where each element is the minimum of each matrix column.
- 2: min(,) Return the minimum of the expressions in (1:) and (2:). If (1:) and (2:) are lists or matrices, return a list or matrix whose elements are the minimum of each corresponding pair of elements.
- 4: max Return the maximum of two expressions, a list or a matrix:
- 1: max() If (1:) is a list, return the maximum element of the list.
 If (1:) is a matrix, return a row vector where each element is the maximum of each matrix column.
- 2: max(,) Return the maximum of the expressions in (1:) and (2:). If (1:) and (2:) are lists or matrices, return a list or matrix whose elements are the maximum of each corresponding pair of elements.
- 5: SortA Sort elements of list or vector variables in ascending order:
- 1: SortA Sort list or vector elements in ascending order. The variable *name* is in (1:)
- 2: SortA ..,n Sort list or vector elements in ascending order. The number of lists or vectors is in (1:) or the command line. The list or vector names are in (1:) to (n:) if *n* is in the command line. The list or vector names are in (2:) to ((n+1):) if *n* is in (1:).

Here is an example of sorting three lists *list1*, *list2* and *list3*, with *n* on the stack:

```
4: list1
3: list2
2: list3
1: 3
```

When SortA ..,n is executed with this stack, the elements *list1* are sorted in ascending order, and the corresponding elements of *list2* and *list3* are moved to match the position of the corresponding elements in *list1*.

Here is the same example, with the argument *n* in the command line:

```
3: list1
2: list2
1: list3
-----
3
-----
```

6: SortD Sort elements of list or vector variables in descending order:

1: SortD Sort list or vector elements in descending order. The variable *name* is in (1:)

2: SortD ...,n Sort list or vector elements in descending order. The number of lists or vectors is in (1:) or the command line. The list or vector names are in (1:) to (n:) if *n* is in the command line. The list or vector names are in (2:) to ((n+1):) if *n* is in (1:).

Here is an example of sorting three lists *list1*, *list2* and *list3*, with *n* on the stack:

```
4: list1
3: list2
2: list3
1: 3
```

When SortA ...,n is executed with this stack, the elements *list1* are sorted in descending order, and the corresponding elements of *list2* and *list3* are moved to match the position of the corresponding elements in *list1*.

Here is the same example, with the argument *n* in the command line:

```
3: list1
2: list2
1: list3
-----
3
-----
```

7: sum If (1:) is a list, return the sum of the list elements. If (1:) is a matrix, return a row vector whose elements are the sum of the matrix column elements.

8: cumSum If (1:) is a list, return a list whose elements are the cumulative sum of the elements of the list in (1:). If (1:) is a matrix, return a matrix whose columns are the cumulative sums of the column elements of (1:), from top to bottom.

9: product If (1:) is a list, return the product of the elements in the list. If (1:) is a matrix, return a row vector whose elements are the products of the column elements of (1:).

A: left(,) If (2:) is a list and (1:) is an expression, return the left-most (1:) elements of (2:). If (2:) is a string and (1:) is an expression, return the left-most (1:) characters of (1:).

B: mid(,,) Return middle elements of a string or list. In general, (3:) is a string, list, or list of strings. (2:) is a *start* expression and (1:) is a *count* expression.

If (3:) is a string, return *count* characters starting with the character at *start*. If *count* is zero, return an empty string. If *count* is greater than the number of characters in the string, return the entire string.

If (3:) is a list, return *count* elements starting with the element at *start*. If *count* is zero, return an empty list. If *count* is greater than the number of elements in (3:), return the entire list.

If (3:) is a list of strings, return *count* strings beginning at element *start*.

C: right(,) If (2:) is a list and (1:) is an expression, return the right-most (1:) elements of (2:).
If (2:) is a string and (1:) is an expression, return the right-most (1:) characters of (1:).

D: polyeval (2:) is a list of expressions interpreted as the coefficients of a descending-degree polynomial. (1:) may be an expression or a list. If (1:) is an expression, return the expression found by evaluating the polynomial for (1:). For example, executing polyeval with this stack

2: {3,2,1}
1: x

returns $3x^2 + 2x + 1$.

If (1:) is a list, return a list whose elements are the polynomial evaluated at each element of (1:). For example, executing polyeval with this stack

2: {3,2,1}
1: {x,y}

returns $\{3x^2 + 2x + 1, 3y^2 + 2y + 1\}$

E: list►m Convert lists to matrices:

1: list►mat() Convert a list in (1:) to a row vector.

2: list►mat(,) Convert a list in (2:) to a matrix with (1:) elements in each row. Zeros are added if there are not enough list elements to fill the matrix.

F: mat►list Return a list whose elements are the elements of the matrix in (1:), row by row.

9: Matrix **Matrix functions:**

The matrix menu functions are organized in these sub-menus:

1: Solve *transpose, determinants, ref, rref and simult*
2: Extr *diagonals, eigenvalues and vectors, LU and QR, submatrices, norms and dim*
3: New *identity, random matrix, new matrix and fill*
4: Ops *row, element and vector operations*
5: augment *matrix augment functions*

Matrix Solve functions

1: τ Return complex conjugate transpose of matrix in (1:)

- 2: det Return determinant of square matrix in (1:):
 1: det() Square matrix is in (1:)
 2: det(,) Square matrix is in (2:), tolerance is in (1:)
- 3: ref Return row echelon form of matrix in (1:):
 1: ref() Matrix is in (1:)
 2: ref(,) Matrix is in (1:), tolerance is in (2:)
- 4: rref Return reduced row echelon form of matrix in (1:):
 1: rref() Matrix is in (1:)
 2: rref(,) Matrix is in (2:), tolerance is in (1:)
- 5: simult(,) Return column vector that is solution to simultaneous equation system:
 1: simult(,) Square coefficient matrix in (2:) and constant vector in (1:)
 2: simult(,,) Square coefficient matrix in (3:), constant vector in (2:), tolerance in (1:)

Matrix Extr functions

- 1: diag If (1:) is a list, row matrix or column matrix, return square matrix with (1:) as diagonal elements.
 If (1:) is a square matrix, return row matrix which is main diagonal of (1:)

- 2: eigVl Return list of the eigenvalues of square matrix in (1:)

- 3: eigVc Return matrix of eigenvectors for square matrix in (1:)

- 4: LU Perform Dolittle lower-upper decomposition on a real or complex matrix

- 1: LU ,, These are the stack arguments:

- 4: *matrix*
- 3: *lMatName*
- 2: *uMatName*
- 1: *pMatName*

matrix is the matrix on which the LU decomposition is done. *lMatName* is the name of the matrix in which the lower-triangular matrix is stored. *uMatName* is the name of the matrix in which the upper triangular matrix is stored. *pMatName* is the name of the matrix in which the permutation matrix is stored. The LU decomposition results in matrices that satisfy this equation:

$$lMatName * uMatName = pMatName * matrix$$

- 2: LU ,,,, These are the stack arguments:

- 5: *matrix*
- 4: *lMatName*
- 3: *uMatName*
- 2: *pMatName*
- 1: *tol*

matrix is the matrix on which the LU decomposition is done. *lMatName* is the name of the matrix in which the lower-triangular matrix is stored. *uMatName* is

the name of the matrix in which the upper triangular matrix is stored.
pMatName is the name of the matrix in which the permutation matrix is stored.
 Any matrix less than *tol* is treated as zero.

The LU decomposition results in matrices that satisfy this equation:

$$lMatName * uMatName = pMatName * matrix$$

5: QR Perform Householder QR factorization of a real or complex matrix:

1: QR ,, The stack arguments are

- 3: *matrix*
- 2: *qMatName*
- 1: *rMatName*

matrix is the matrix on which the QR factorization is performed. The unitary Q matrix is stored to *qMatName*. The upper triangular R matrix is stored to *rMatName*.

2: QR ,,, The stack arguments are

- 4: *matrix*
- 3: *qMatName*
- 2: *rMatName*
- 1: *tol*

matrix is the matrix on which the QR factorization is performed. The unitary Q matrix is stored to *qMatName*. The upper triangular R matrix is stored to *rMatName*. Elements are treated as zero if they are less than *tol*.

6: subMat Return specified submatrix of matrix in (1:). Defaults for start row and start column are 1. Defaults for end row and end column are the last row and column

- 1: subMat(,) Return submatrix of matrix in (2:) and start row in (1:)
- 2: subMat(,,) Return submatrix of matrix in (3:), start row in (2:) and start column in (1:)
- 3: subMat(,,,)
- 4: subMat(,,,,) Return submatrix of matrix in (5:), start row in (4:), start column in (3:), end row in (2:) and end column in (1:)

7: norms Matrix norms functions:

- 1: norm Return Frobenius norm of matrix in (1:)
- 2: rowNorm Return the maximum of the sums of the absolute values of elements of the rows of matrix in (1:)
- 3: colNorm Return the maximum of the sums of the absolute values of elements of the columns of matrix in (1:)

8: Dim Matrix and list dimension functions:

1: dim	If (1:) is a list, return number of elements of (1:) If (1:) is a matrix, return number of rows and columns as list {rows, columns} If (1:) is a string, return number of characters in (1:)
2: rowDim	Return the number of rows of matrix in (1:)
3: colDim	Return the number of columns of matrix in (1:)

Matrix New functions

1: identity	Return identity matrix with dimension of expression in (1:)
2: randMat	Return a matrix with integer random elements between -9 and 9, with (1:) rows and (2:) columns
3: newMat	Return a matrix with zero elements of (1:) columns and (2:) rows
4: Fill	Fill a matrix or list with an expression. The matrix or list must already exist. The expression is in (2:), and the list or matrix name is in (1:).

Matrix Ops functions

1: Row	Matrix row swap and arithmetic functions:
1: rowSwap	Return matrix in (3:) with rows (2:) and (1:) swapped
2: rowAdd	Return matrix in (3:) with row (2:) replaced by sum of rows (2:) and (1:)
3: mRow	Return matrix in (2:) with row (1:) multiplied by expression in (3:)
4: mRowAdd	Return matrix in (3:) with each element of row (1:) replaced with (4:) * (2:) + (1:). For example, if the stack is

4: n
3: [a,b;c,d]
2: 1
1: 2

then mRowAdd returns $\begin{bmatrix} a & b \\ a \cdot n + c & b \cdot n + d \end{bmatrix}$

2: Element	Matrix element arithmetic:
1: .+	Return matrix which is (2:) + (1:). If (1:) and (2:) are both matrices then the returned matrix is the sum of the corresponding elements of the matrices. If (1:) or (2:) is an expression, then the returned matrix is the sum of the expression and each element of the matrix.
2: .-	Return matrix which is (2:) - (1:). If (1:) and (2:) are both matrices then the

returned matrix is the difference between the corresponding elements. If (1:) or (2:) is an expression, then the returned matrix is the difference between each element of the matrix and the expression.

- 3: .* Return matrix which is (2:) * (1:). If (1:) and (2:) are both matrices then the returned matrix is the product of the corresponding elements. If (1:) or (2:) is an expression, then the returned matrix is the difference between each element of the matrix and the expression.
- 4: ./ Return matrix which is (2:) / (1:). If (1:) and (2:) are both matrices then the returned matrix is the quotient of the corresponding elements. If (1:) or (2:) is an expression, then the returned matrix is the quotient of each element of the matrix and the expression.
- 5: .^ Return matrix which is (2:) ^ (1:). If (1:) and (2:) are both matrices then the elements of (2:) are the exponents for the elements in (1:). If (1:) is a matrix and (2:) is an expression, then (2:) is the exponent of each element in (1:). If (1:) is an expression and (2:) is a matrix, then the elements of (2:) are the exponents of (2:).

3: Vector Vector functions and conversions:

- 1: unitV Return unit vector for vector in (1:). (1:) must be a single-column or single-row matrix.
- 2: crossP Return cross product of (1:) and (2:). Returns a list if (1:) and (2:) are lists. Returns a vector if (1:) and (2:) are vectors, which are single-row or single-column matrices.
- 3: dotP Return dot product of (1:) and (2:). Returns a list if (1:) and (2:) are lists. Returns a vector if (1:) and (2:) are vectors, which are single-row or single-column matrices.
- 4: ►Polar Return polar form of (1:). (1:) may be a row- or column-vector of dimension 2, or a complex value.
- 5: ►Rect Return rectangular form of (1:). (1:) may be a complex value, or a row- or column vector of dimension 2 or 3.
- 6: ►Cylind Return cylindrical form of (1:). (1:) must be a row or column vector of dimension 3.
- 7: ►Sphere Return spherical form of (1:). (1:) must be a row or column vector of dimension 3.

Matrix augment functions

- 1: augment(,) If (1:) and (2:) are lists, append elements of (2:) to (1:)
If (1:) and (2:) are matrices, append (2:) to (1:) as columns
- 2: augment(;) Append matrix (1:) to matrix (2:) as rows.
If (1:) and (2:) are lists, return list which is (2:) appended to (1:)
If (1:) and (2:) are matrices, append (2:) to (1:) as columns

A: Stat functions

1: OneVar *Calculate one-variable statistics and update the system statistics variables*

1: OneVar (1:) is the list for which the statistics are calculated.

2: OneVar , (2:) is the list for which the statistics are calculated.
(1:) is the list of frequencies

3: OneVar ,, (3:) is the list for which the statistics are calculated.
(2:) is the list of frequencies
(1:) is the list of category codes

4: OneVar ,,, (4:) is the list for which the statistics are calculated.
(3:) is the list of frequencies
(2:) is the list of category codes
(1:) is the category include list

2: Two Var *Calculate two-variable statistics and update the system statistics variables*

1: TwoVar , (2:) is the xlist
(1:) is the ylist

2: TwoVar ,, (3:) is the xlist
(2:) is the ylist
(1:) is the list of frequencies

3: TwoVar ,,,, (5:) is the xlist
(4:) is the ylist
(3:) is the list of frequencies
(2:) is the list of category codes
(1:) is the category include list

3: Regres *Calculate regression equation coefficients*

1: Lin Linear regression $y = a + b*x$

1: LinReg , (2:) is the xlist
(1:) is the ylist

2: LinReg ,, (3:) is the xlist
(2:) is the ylist
(1:) is the frequency list

3: LinReg ,,,, (5:) is the xlist
(4:) is the ylist
(3:) is the frequency list
(2:) is the category code list
(1:) is the category include list

2: Exp Exponential regression $y = a \cdot b^x$

- 1: ExpReg , (2:) is the xlist
 (1:) is the ylist
- 2: ExpReg ,, (3:) is the xlist
 (2:) is the ylist
 (1:) is the frequency list
- 3: ExpReg ,,,, (5:) is the xlist
 (4:) is the ylist
 (3:) is the frequency list
 (2:) is the category code list
 (1:) is the category include list

3: Quad Quadratic regression $y = a \cdot x^2 + b \cdot x + c$

- 1: QuadReg , (2:) is the xlist
 (1:) is the ylist
- 2: QuadReg ,, (3:) is the xlist
 (2:) is the ylist
 (1:) is the frequency list
- 3: QuadReg ,,,, (5:) is the xlist
 (4:) is the ylist
 (3:) is the frequency list
 (2:) is the category code list
 (1:) is the category include list

4: Power Power regression $y = a \cdot x^b$

- 1: PowerReg , (2:) is the xlist
 (1:) is the ylist
- 2: PowerReg ,, (3:) is the xlist
 (2:) is the ylist
 (1:) is the frequency list
- 3: PowerReg ,,,, (5:) is the xlist
 (4:) is the ylist
 (3:) is the frequency list
 (2:) is the category code list
 (1:) is the category include list

5: Ln Logarithmic regression $y = a + b \cdot \ln(x)$

- 1: LnReg , (2:) is the xlist
 (1:) is the ylist
- 2: LnReg ,, (3:) is the xlist
 (2:) is the ylist
 (1:) is the frequency list

3: LnReg ,,,, (5:) is the xlist
(4:) is the ylist
(3:) is the frequency list
(2:) is the category code list
(1:) is the category include list

6: MedMed Median-median line regression $y = a + b*x$

1: MedMed , (2:) is the xlist
(1:) is the ylist

2: MedMed ,, (3:) is the xlist
(2:) is the ylist
(1:) is the frequency list

3: MedMed ,,,, (5:) is the xlist
(4:) is the ylist
(3:) is the frequency list
(2:) is the category code list
(1:) is the category include list

7: Cubic Cubic regression $y = a*x^3 + b*x^2 + c*x + d$

1: Cubic , (2:) is the xlist
(1:) is the ylist

2: Cubic ,, (3:) is the xlist
(2:) is the ylist
(1:) is the frequency list

3: Cubic ,,,, (5:) is the xlist
(4:) is the ylist
(3:) is the frequency list
(2:) is the category code list
(1:) is the category include list

8: Quart Quartic regression $y = a*x^4 + b*x^3 + c*x^2 + d*x + c$

1: QuartReg , (2:) is the xlist
(1:) is the ylist

2: QuartReg ,, (3:) is the xlist
(2:) is the ylist
(1:) is the frequency list

3: QuartReg ,,,, (5:) is the xlist
(4:) is the ylist
(3:) is the frequency list
(2:) is the category code list
(1:) is the category include list

9: Sin Sine regression $y = a*\sin(b*x + c) + d$

- 1: SinReg , (2:) is the xlist
(1:) is the ylist
- 2: SinReg ,, (3:) is the xlist
(2:) is the ylist
(1:) is the maximum number of iterations
- 3: SinReg ,,, (4:) is the xlist
(3:) is the ylist
(2:) is the maximum number of iterations
(1:) is the estimate for the period
- 4: SinReg ,,,,, (6:) is the xlist
(5:) is the ylist
(4:) is the maximum number of iterations
(3:) is the estimate for the period
(2:) is the category code list
(1:) is the category include list

A: Logistic Logistic regression equation $y = a/(1 + b \cdot e^{c \cdot x}) + d$

- 1: Logistic , (2:) is the xlist
(1:) is the ylist
- 2: Logistic ,, (3:) is the xlist
(2:) is the ylist
(1:) is the maximum number of iterations
- 3: Logistic ,,, (4:) is the xlist
(3:) is the ylist
(2:) is the maximum number of iterations
(1:) is the frequency list
- 4: Logistic ,,,,, (6:) is the xlist
(5:) is the ylist
(4:) is the maximum number of iterations
(3:) is the frequency list
(2:) is the category code list
(1:) is the category include list

- 4: mean If (1:) is a list, return the mean of all the list elements.
If (1:) is a matrix, return a row vector of the means of the column elements.
- 5: variance If (1:) is a list, return the variance of the list elements.
If (1:) is a matrix, return a row vector of the variances of the column elements.
- 6: stdDev If (1:) is a list, return the standard deviation of the list elements.
If (1:) is a matrix, return a row vector of the standard deviations of the column elements.
- 7: median If (1:) is a list, return the median of the list elements.
If (1:) is a matrix, return a row vector of the medians of the column elements.

8: ShowStat Show the dialog box containing the last computed statistics results, if valid. The statistics results are cleared if the data that generated them has changed.

B: Prob Probability functions

1: ! Return the factorial of (1:). (1:) may be an expression, list or matrix.

2: nPr Number of permutations. The stack is

(2:) *expr1*
(1:) *expr2*

expr1 and *expr2* can be integers or expressions. If $expr1 \geq expr2 \geq 0$, then nPr returns the number of permutations of *expr1* items taken *expr2* at a time.

If $expr2 = 0$, then nPr returns 1.

If *expr2* is a negative integer, then nPr returns

$$1/(expr1 + 1) * (expr1 + 2) * \dots * (expr1 - expr2)$$

If *expr2* is a positive integer, then nPr returns

$$expr1 * (expr1 - 1) * (expr1 - 2) * \dots * (expr1 - expr2 + 1)$$

If *expr2* is a non-integer, then nPr returns

$$expr1! / (expr1 - expr2)!$$

If (1:) and (2:) are lists, return a list of permutations of the corresponding pairs of elements in the two lists.

If (1:) and (2:) are matrices, return a matrix of permutations of the corresponding pairs of elements in the two matrices.

3: nCr Number of combinations (binomial coefficient). The stack is

(2:) *expr1*
(1:) *expr2*

expr1 and *expr2* can be expressions or integers. If $expr1 \geq expr2 \geq 0$, then nCr returns the number of combinations of *expr1* items taken *expr2* at a time.

If $expr2 = 0$, then nCr returns 1.

If *expr2* is a negative integer, then nCr returns 0.

If *expr2* is a positive integer, the nCr returns

$$expr1 * (expr1 - 1) * \dots * (expr1 - expr2 + 1) / expr2!$$

If *expr2* is a non-integer, then nCr returns

$$expr1! / ((expr1 - expr2)! * expr2)$$

If *expr1* and *expr2* are lists, *nCr* returns a list of combinations of the corresponding elements of the lists.

If *expr1* and *expr2* are matrices, *nCr* returns a matrix of combinations of the corresponding matrix elements.

- 4: *rand* Return a random integer *n* in the interval specified by (1:). If *n* > 0, the interval is [1,*n*]. If *n* < 0, the interval is [-*n*, -1]. To return a random number on the interval [0,1], execute *rand* from the command line.
- 5: *randNorm* Return normally distributed random number specified by
- 2: *mean*
 - 1: *stdDev*
- 6: *randSeed* Reseed random number generator. If (1:) is 0, reseed to factory defaults. If (1:) is not equal to zero, use (1:) to generate two seeds stored in system variables *seed1* and *seed2*.
- 7: *randMat* Return matrix of random numbers between -9 and 9, based on stack arguments:
- 2: *numberOfRows*
 - 1: *numberOfColumns*
- 8: *randPoly* Return a polynomial based on these stack arguments:
- 2: *var*
 - 1: *order*

var is the polynomial variable, and *order* is the order of the polynomial. The coefficients are from -9 to 9, and the leading coefficient is not zero.

C: Hyperbol **Hyperbolic trigonometric functions:**

- 1: *sinh* Return hyperbolic sine of (1:)
- 2: *cosh* Return hyperbolic cosine of (1:)
- 3: *tanh* Return hyperbolic tangent of (1:)
- 4: \sinh^{-1} Return inverse hyperbolic sine of (1:)
- 5: \cosh^{-1} Return inverse hyperbolic cosine of (1:)
- 6: \tanh^{-1} Return inverse hyperbolic tangent of (1:)

D: log **Logarithms:**

- 1: *ln* Return natural logarithm of (1:). If (1:) is an expression or list, return the natural logarithm of the expression, or each expression in the list. If (1:) is a square diagonalizable matrix, return the matrix natural logarithm. This is *not* the same as the natural logarithm of each matrix element.
- 2: *log* Return base-10 logarithm of (1:). If (1:) is an expression or list, return the base-10 logarithm of the expression, or each expression in the list. If (1:) is a

square diagonalizable matrix, return the matrix base-10 logarithm. This is *not* the same as the base-10 logarithm of each matrix element.

E: Base

Base conversion functions:

- 1: ▶Bin
- 2: ▶Dec
- 3: ▶Hex

Convert integer in (1:) to binary (base 2)
Convert integer in (1:) to decimal (base 10)
Convert integer in (1:) to hexadecimal (base 16)

RCL Menu

The RCL menu provides basic operations for variables and folders. The RCL menu takes its name from the fact that it is displayed by pressing [RCL], and it is used to recall items to the stack. It is not intended to replace the VAR-LINK menu in normal 89/92+ operation, with all of its functions. You can use the RCL menu to

- Change the current folder
- Recall user variables and variable names to the stack
- Execute user functions and programs

The RCL menu can be set to be shown as a pop-up menu or a line menu. Choose the menu type by pressing [MODE], Customize, RCL menu, then choosing Pop Up or Line.

The RCL menu operates the same as a pop-up or line menu, with one exception. To push a variable *name* to the stack, instead of the variable contents, you must use the RCL menu in line mode.

For the remainder of this description I assume that the RCL menu is shown as a pop-up menu.

Display the RCL menu by pressing [RCL]. The folders menu is displayed, for example

```
1: aaa
2: main
3: rpnc
4: test
```

In this case four folders are available: aaa, main, rpnc and test. Choose one of these menu items to make the folder current. Next, this menu is shown:

```
1: VAR
2: FUNC
3: PROG
```

Choose VAR to display the variables, FUNC to display the functions, and PROG to display the programs in the current folder. If the RCL menu is shown as a line menu, you can use [DIAMOND] [LEFT] and [DIAMOND] [RIGHT] to scroll through the line menu items.

If the RCL menu is shown as a line menu, you can use [DIAMOND] [UP] or [DIAMOND] [DOWN] to display the previous menu level. For example, if you have the variables displayed, and you want to see your functions, press [DIAMOND] [UP], then this line menu is shown:

```
F1-VAR      F2-FUNC      F3-PROG
```

Push [F2] for the functions menu.

To change the current folder:

Push [RCL] to display the folders menu. Choose the folder you want. This folder is made current. If you are using the line menu, no further action is needed. If you are using the pop-up menu, choose VAR, FUNC or PROG. Push [ESC] as needed to close all the menus. The folder you chose is now the current folder.

Note that you can also use [MODE], Other to use the built-in mode screen to change the current folder.

To recall a variable to the stack:

Push [RCL] to display the folders menu. Choose the folder for the variable you want. Choose VAR. Choose the variable from the menu. The variable's value is pushed onto stack level 1:.

To recall a variable name to the stack:

This can only be done if the RCL menu is shown as a line menu. Push [RCL] to display the folders menu. Choose the folder for the variable you want, then choose F1:VAR. Push [DIAMOND], then the function key for the desired variable. The variable name, not the value, is pushed onto stack level 1:. You can also use this method to push a folder name on the stack.

To run a user function or program:

Push [RCL] to display the folders menu. Choose the folder for the function you want. Choose FUNC for functions, PROG for programs. Choose the function or program from the menu. Functions are executed with the input arguments taken from the stack and the results returned to the stack. Programs are executed, the RPN resumes when the program finishes.

As an example, suppose I have a function called func1() in my *test* folder. This function has three input arguments a, b and c. The function concatenates the three arguments as strings. To call func1("a","b","c"), put the arguments on the stack:

```
3: "a"  
2: "b"  
1: "c"
```

then when func1() is executed, the result is returned to the stack:

```
1: "abc"
```

TOOLS Menu

The TOOLS menu is displayed by pushing [DIAMOND] [(-)]. The TOOLS menu consists of functions for graphics, variables, and folders, as well as NewData and NewProb commands. The main menu is

- 1: Graphics
- 2: Var
- 3: Folder
- 4: NewData ...,n
- 5: NewProb

The rest of this sections describes these menus in detail.

Graphics menu

1: Graph

- 1: Graph x Graph the expression in (1:). The independent variable must be x
- 2: Graph , Graph an expression which requires two stack arguments. For function and polar graphing, the stack is
 - 1: expr *expression to graph*
 - 2: var *independent variable*
- 3: Graph ,, Graph an expression which requires three stack arguments. For parametric graphing, the stack is
 - 3: xExpr *the expression for x*
 - 2: yExpr *the expression for y*
 - 1: var *the parametric variable*

For 3D graphing, the stack is

 - 3: expr *the expression to graph*
 - 2: xVar *the independent x-variable*
 - 1: yVar *the independent y-variable*

2: Draw

- 1: DrawFunc Draw the expression in (1:), using x as the independent variable
- 2: DrawInv Draw the inverse of the expression in (1:), by plotting the x -values on the y -axis and the y -values on the x -axis
- 3: Parm Draw parametric equation graphs:
 - 1: DrawParm ,
 - 2: DrawParm ,,
 - 3: DrawParm ,,,

4: DrawParm ,,,, Draws the graph of two parametric equations with these stack arguments:

5: *expr1* (function of t)
4: *expr2* (function of t)
3: *tmin* (minimum value of t)
2: *tmax* (maximum value of t)
1: *tstep* (step value of t)

4: Pol Draw polar equation graphs:

1: DrawPol x Draw the polar graph of the expression in (1:), using θ as the independent variable. Use the current window settings.

2: DrawPol , Draw a polar graph with θ as the independent variable with these stack arguments:

2: *expr* (function of θ)
1: θ *min* (minimum θ)

3: DrawPol ,, Draw a polar graph with θ as the independent variable with these stack arguments:

3: *expr* (function of θ)
2: θ *min* (minimum θ)
1: θ *max* (maximum θ)

4: DrawPol ,,, Draw the graph of a polar equation with these stack arguments:

4: *expr* (function of θ)
3: θ *min* (minimum θ)
2: θ *max* (maximum θ)
1: θ *step* (step value of θ)

5: DrawSlp Draw the line through the point (x1,y1) with a specified slope with these stack arguments:

3: x1
2: y1
1: slope

The equation for the line is $y = \text{slope} * (x - x1) + y1$

6: DrwCtour Draw contours on the current 3D graph at the z-values specified in the list or expression in (1:). 3D graph mode must be set. The contours drawn by DrwCtour are added to those specified by the *ncontour* window variable. Set *ncontour* to zero to turn off the default contours.

3: BldData Build a data variable from the current graph information.

1: BldData Build the data variable and store it in the system variable *sysData*.

2: BldData x Build the data variable and store it in the variable in (1:).

4: Table Build table operations:

- 1: Table x Build a table from a function or polar function. If the current graph mode is set to Function, the independent variable must be x . If the current graph mode is set to Polar, the independent variable must be θ .
- 2: Table , If the graph mode is Function or Polar, build a table from a function or polar function. In either case, the function is in (2:) and the independent variable is in (1:).
- If the graph mode is parametric, build a a table from these stack arguments:
- 2: $xExpr$
1: $yExpr$
- The independent parametric variable must be t .
- 3: Table ,, Build a table from a parametric function with these stack arguments:
- (3:) $xExpr$ (The x-expression)
(2:) $yExpr$ (The y-expression)
(1:) $tVar$ (The parametric independent variable)
- 5: Clr Clear operations:
- 1: ClrDraw Clear the Graph screen and reset the 'Smart Graph' feature
- 2: ClrGraph Clear any functions or expressions graphed with the Graph command or created with the Table command.
- 3: ClrHome Clear all items in the normal-mode history display. Reset arbitrary constant and integer suffix to 1.
- 4: ClrIO Clear the Program I/O screen.
- 5: ClrTable Clear all table values.
- 6: Zoom Graph screen zoom operations
- 1: ZoomBox Display the graph screen, accept a box that defines a new viewing window, then updates the window.
- 2: ZoomData Adjust window settings so that all statistical plot data point are shown, then displays the Graph screen.
- 3: ZoomDec Adjust the window settings so that $\Delta x = \Delta y = 0.1$, and so that the origin is centered in the graph screen.
- 4: ZoomFit Display the graph screen and adjust the window settings for the dependent variable so that all plotted points are shown for the independent variables.
- 5: ZoomIn Display the graph screen, accept a zoom center point, then update the graph screen. The zoom magnitude depends on $xFact$ and $yFact$, and also on $zFact$ in

- 3D mode.
- 6: ZoomInt Display the graph screen, accept a zoom center point, then adjust the window settings so that each pixel on all axes is an integer.
- 7: ZoomOut Display the graph screen, accept a zoom center point, then update the graph screen. The zoom magnitude depends on $xFact$ and $yFact$, and also on $zFact$ in 3D mode.
- 8: ZoomPrev Display the graph screen, restore the window settings prior to the last zoom, then update the graph screen.
- 9: ZoomRcl Display the graph screen, restore the windows settings last saved with ZoomSto, then update the graph screen.
- A: ZoomSqr In the 2D graph modes, adjust the x or y axis window settings so each pixel represents an equal width and height in the coordinate system, then update the graph screen.
- In 3D graph mode, the shortest two axes are lengthened to the same length as the longest axis.
- B: ZoomStd Set the window settings to standard values shown below, then update the graph screen.
- Function graph:
 $x: [-10, 10, 1], y: [-10, 10, 1], xres = 2$
- Parametric graph:
 $t: [0, 2\pi, \pi/24], x: [-10, 10, 1], y: [-10, 10, 1]$
- Polar graph:
 $\theta: [0, 2\pi, \pi/24], x: [-10, 10, 1], y: [-10, 10, 1]$
- Sequence graph:
 $x: [-10, 10, 1], y: [-10, 10, 1], nmin = 1, nmax = 10, plotStrt = 1, plotStep = 1$
- 3D graph:
 $eye\theta^\circ = 20, eye\phi^\circ = 70, eye\psi^\circ = 0, x: [-10, 10, 14], y: [-10, 10, 14], z: [-10, 10], ncontour = 5$
- Differential equation graph:
 $t: [0, 10, .1, 0], x: [-1, 10, 1], y: [-10, 10, 1], ncurves = 0, Estep = 1, difftol = .001, fldres = 20, dtime = 0$
- C: ZoomSto Save the current window settings in the zoom memory. Restore them with ZoomRcl.
- D: ZoomTrig Display the graph screen. Set $\Delta x = \pi/24, xscl = \pi/2$, set the y settings to $[-4, 4, 5]$ and update the graph screen.
- 7: FnOn Function-on operations
- 1: FnOn Select all $Y=$ functions for the current graph mode

2: FnOn ...,n Select specified Y= functions for the current graph mode. Specify the numbers of the functions to select in the stack, and the number of functions to select in 1: or the command line. For example, to select Y3(x) and Y(5), enter this stack:

3: 3 *function number for Y3(x)*
2: 5 *function number for Y5(x)*
1: 2 *total number of Y= functions to select*

then execute FnOn ...,n

8: FnOff Function-off operations

1: FnOff Deselect all Y= functions for the current graph mode

2: FnOff ...,n Deselect specified Y= functions for the current graph mode. Specify the numbers of the functions to deselect in the stack, and the number of functions to deselect in 1: or the command line. For example, to deselect Y3(x) and Y(5), enter this stack:

3: 3 *function number for Y3(x)*
2: 5 *function number for Y5(x)*
1: 2 *total number of Y= functions to select*

then execute FnOff ...,n

Var menu

1: Define Define a function with these stack arguments:

2: func(var) *function name and arguments*
1: expr *function expression*

For example, to define $f(x) = 1/\sin(x)$, enter this stack

2: f(x)
1: 1/sin(x)

then execute Define.

2: DelVar Delete variable commands

1: DelVar x Delete a single variable whose name is in (1:)

2: DelVar ...,n Delete multiple variables whose names are on the stack, and the number of variables to delete, n , is in (1:) or the command line. For example, to delete the three variables a, b and c, enter this stack:

4: a
3: b
2: c
1: 3

then execute DelVar ...,n

- 3: Rename Rename a variable, using this stack:
- 2: *varName* name of existing variable
1: *newName* new name for variable
- 4: CopyVar Copy a variable, using this stack:
- 2: *varName* name of existing variable
1: *newName* name of copy of variable
- 5: MoveVar Move a variable to a different folder, using this stack:
- 3: *varName* name of variable to move
2: *oldFolder* name of current folder
1: *newFolder* name of folder to move *varName*

Use the [RCL] menu in command line mode with [DIAMOND] [F1] - [DIAMOND] [F5] to push the folder names on the stack.

6: Lock Variable lock operations

- 1: Lock x Lock a single variable whose name is in (1:) or the command line.
- 2: Lock ...,n Lock one or more variables whose names are on the stack, and the number of variables to lock is in (1:) or the command line. For example, to lock three variables a, b and c, enter this stack

4: a
3: b
2: c
1: 3

then execute Lock ...,n.

7: Unlock Variable unlock operations

- 1: Unlock x Unlock a single variable whose name is in (1:) or the command line.
- 2: Unlock ...,n Unlock one or more variables whose names are on the stack, and the number of variables to unlock is in (1:) or the command line. For example, to unlock three variables a, b and c, enter this stack

4: a
3: b
2: c
1: 3

then execute Unlock ...,n.

8: Archive Variable archive operations

- 1: Archive x Archive a single variable whose name is in (1:) or the command line.

2: Archive ...,n Archive one or more variables whose names are on the stack, and the number of variables to archive is in (1:) or the command line. For example, to archive three variables a, b and c, enter this stack

```
4: a
3: b
2: c
1: 3
```

then execute Archiv ...,n.

9: Unarchiv Variable unarchive operations

1: Unarchiv x Unarchive a single variable whose name is in (1:) or the command line.

2: Unarchiv ...,n Unarchive one or more variables whose names are on the stack, and the number of variables to unarchive is in (1:) or the command line. For example, to unarchive three variables a, b and c, enter this stack

```
4: a
3: b
2: c
1: 3
```

then execute Unarchiv ...,n.

Folder menu

1: NewFold Create a new folder whose name is in (1:) or the command line.

2: DelFold Delete a single folder whose name is in (1:) or the command line.

3: DelFold ...,n Delete one or more folders whose names are on the stack, and the number of folders to delete is in (1:) or the command line. For example, to delete three folders aaa, bbb and ccc, enter this stack:

```
4: aaa
3: bbb
2: ccc
1: 3
```

then execute DelFold ...,n

NewData ...,n Create a new data variable whose name and list elements are on the stack, and the number of arguments is in (1:) or the command line. For example, to create a data variable called *adata* with three columns {1,2,3}, {4,5,6} and {7,8,9}, enter this stack:

```
5: adata
4: {1,2,3}
3: {4,5,6}
```

2: {7,8,9}
1: 4

then execute NewData ...,n.

NewProb

Clear the calculator state without resetting memory. All single-character variable names are cleared, unless they are locked or archived. Turn off all functions and statistical plots in the current graph mode. Execute ClrDraw, ClrErr, ClrGraph, ClrHome, ClrIO and ClrTable.

Custom Menus

RPN supports custom menus. A custom menu shows only the items that you choose, unlike the RCL menu, which shows all variables and functions. Each folder may have its own custom menu. The custom menus can be shown as pop-up or line menus. This description assumes that you have set the menu to be shown as a pop-up menu. Custom line menus operate in the same way.

The menu is displayed when you press [CUSTOM]. If a variable named *custom_* exists in the current folder, RPN makes a custom menu with the strings in the list in *custom_*.

For example, to create a custom menu with the variables *var1*, *var2* and *var3*, and the functions *func1*, *func2* and *func3*, use these keystrokes in RPN mode:

```
{ "var1", "var2", "var3", "func1", "func2", "func3" } [ENTER]
custom_ [STO]
```

Use the \ folder specifier to refer to variables and functions that are not in the current folder. For example, this *custom_* variable

```
{ "folder1\var1", "folder1\var2", "folder1\var3" }
```

specifies that the variables are all in *folder1*.

You can include variables in the custom menu that are not defined. Undefined variables will be shown in the custom VAR menu. Built-in functions will be shown in the custom FUNC menu. System commands cannot be shown in the menus.

These variable types are not evaluated when chosen from the menu:

PIC	DATA	TEXT
GDB	MACRO	FIGURE

Instead, for these types, the name of the variable is pushed on the stack.

To display the custom menu, press [CUSTOM]. This menu is shown:

```
1: VAR      display variables
2: FUNC     display functions
3: PROG     display programs
```

Choose the appropriate option to display variables, functions or programs, then another menu is shown which lists those items. For example, if you have defined the variables *a*, *b*, and *c*, then when VAR is selected, this menu is shown:

```
1:a
2:b
3:c
```

Some additional features are available when a custom menu is shown as a line menu. Use [DIAMOND] [UP] or [DIAMOND] [DOWN] to display the previous menu. With a line menu, you can either push a variable's contents or its name on the stack. For the example above, the custom variable menu is displayed as

F1-var1 F2-var2 F3-var3

To push the contents of *var1* on the stack, push [F1]. To push the name *var1* on the stack, push [DIAMOND] [F1]. Pushing the variable name on the stack is convenient for storing new values to existing variables. The new value can be stored without typing the variable name. For example, to store 1.234 to the variable *var2*:

1.234	<i>enter 1.234</i>
[DIAMOND] [F2]	<i>push var2 on stack</i>
[STO]	<i>store 1.234 to var2</i>

You will also need to push a variable's name on the stack if you want to copy it, rename it, move it, archive or unarchive it, and lock or unlock it.

You can remove a custom menu by deleting the *custom_* variable.

Custom key assignments

The custom key assignment is an RPN feature that lets you assign your own functions to the keys. This lets you use your calculator more efficiently, by quickly executing functions that you often use.

There are two types of key assignments:

1. You can assign functions to unused keys, then execute the function just by pressing the key. This is called an *unused-key* assignment.
2. You can assign functions to any key, then execute the function by pressing [DOWN] [key]. This is called an *alternate-key* assignment.

The two sections that follow describe each assignment method.

Alternate-key assignments

This is the method to assign custom key functions:

1. Enter the function you want to assign, as a string.
2. Push a keycode variable on the stack, using [2nd] [DOWN] [key], where *key* is the key you want to use.
3. Push [STO] to store the string to the keycode variable.

To execute a custom key function, press [DOWN], then the key.

In the examples that follow, I use the TI92+. The procedure is the same on the TI89, except that the actual key code variable names will be different. For example, the key code variable for the [SIN] key is `rpnc\key259` on the TI92+, and `rpnc\key4185` on the TI89.

As an example, assign `sinh()` to the [SIN] key:

"sinh" [ENTER]	<i>enter the function to assign</i>
[2nd] [DOWN] [SIN]	<i>get the keycode variable for the [SIN] key</i>
[STO]	<i>save the function string in the keycode variable</i>

To use this key assignment to find `sinh(1)`, use

1 [DOWN] [SIN]

which returns 1.1752.

To display a key assignment, use

[2nd] [DOWN] [key]	<i>get the keycode variable for [key]</i>
[DIAMOND] [ENTER]	<i>recall the key assignment string</i>

For the example above:

[2nd] [DOWN] [SIN] *pushes rpnc\key259 to (1:)*

[DIAMOND] [ENTER] *displays "sinh" in (1:)*

To remove a key assignment, delete the key assignment variable with DelVar:

[2nd] [DOWN] [key] *get the keycode variable for [key]*
[DIAMOND] [DEL] [1] *delete the keycode variable*

To delete the sinh() assignment made above:

[2nd] [DOWN] [SIN] *get the keycode variable for [SIN]*
[DIAMOND] [DEL] [1] *delete the keycode variable*

In general, almost any function or command that can be executed from the command line can be assigned to a custom key. This includes user programs and functions in TIbasic or assembly. You can also assign expressions to a key. You cannot assign built-in commands to a key.

Assignments can be made to modified keys, where the modifiers are [SHIFT], [2nd] and [DIAMOND]. For example, to push the key code variable for [DIAMOND] [8], use

[2nd] [DOWN] [DIAMOND] [8]

which pushes rpnc\key8248 on the stack.

You can create a key menu by assigning a list of strings to a key. In this case, the strings define a menu that acts much the same as a custom menu. This is called a key menu. The difference is that key menus are global; not local to a folder as the custom menus are. The key menu is displayed as a pop-up menu or a line menu depending on the current setting for custom menus, found in [MODE], Customize, Custom menu. For example, suppose you store this list to the [sin] key code variable:

```
{"var1", "var2", "func1", "func2", "prog1", "prog2"}
```

After this assignment, when [DOWN] [SIN] is pressed, this line menu is shown:

```
F1-VAR      F2-FUNC      F3-PROG
```

Choose F1 to display the variables *var1* and *var2*. Choose F2 to display the functions *func1* and *func2*. Choose F3 to display the programs *prog1* and *prog2*.

In algebraic mode, executing a custom key assignment pastes the text in the command line at the cursor position.

All custom key definitions are saved in the \rpnc folder.

If a program or function is named with a key code variable name and stored in the \rpnc folder, that function will run when the custom key is pressed. For example, the key code variable name for the [SIN] key is rpnc\key259, so if you have a program or function called key259(), it will run when [DOWN] [SIN] is pressed.

Unused-key assignments

Most of the previous description for alternate-key assignments applies to unused-key assignments, as well. The assignments are made in the same way, by storing the function string to a key-code

variable. However, an unused-key assignment function is executed by just pressing the key, instead of pressing [DOWN][key].

For example, you may assign a function to [DIAMOND] [A] on the 92+, since that key combination is not used by RPN or the 92+ in normal mode. However, you cannot assign a function to [DIAMOND] [1]. Even though that combination is unused by the 92+, it is used by RPN as a shortcut to the MATH Number menu.

The tables below show unused keys for the TI89 and the TI92+.

TI89 unused keys:

[DIAMOND] [,]	[2nd] [ENTER]
[DIAMOND] [UP]	[2nd] [UP]
[DIAMOND] [DOWN]	
[DIAMOND] [LEFT]	
[DIAMOND] [RIGHT]	

TI92+ unused keys:

[DIAMOND] [A]	[DIAMOND] [MODE]	[DIAMOND] [^]
[DIAMOND] [B]	[DIAMOND] [ESC]	[DIAMOND] [(]
[DIAMOND] [D]	[DIAMOND] [STO]	[DIAMOND] [)]
[DIAMOND] [H]	[DIAMOND] [LEFT]	[DIAMOND] [,]
[DIAMOND] [I]	[DIAMOND] [RIGHT]	[DIAMOND] [/]
[DIAMOND] [J]	[DIAMOND] [UP]	[DIAMOND] [*]
[DIAMOND] [K]	[DIAMOND] [DOWN]	
[DIAMOND] [L]		[2nd] [ENTER]
[DIAMOND] [M]	[DIAMOND] [LN]	[2nd] [UP]
[DIAMOND] [N]	[DIAMOND] [SIN]	[2nd] [4]
[DIAMOND] [O]	[DIAMOND] [COS]	[2nd] [B]
[DIAMOND] [S]	[DIAMOND] [TAN]	
[DIAMOND] [U]		
[DIAMOND] [Z]	[DIAMOND] [θ]	
[DIAMOND] [SPACE]		

Constants, units and unit conversion

RPN supports constants, units and unit conversion. In RPN mode, an additional conversion method is available which is usually faster than using the normal-mode conversion method. The UNIT menu can only be displayed as a line menu, not a pop-up menu.

In general, the units are the same as those in normal-mode operation. However, RPN includes more units for area, volume and other categories, and some of the normal-mode categories have been combined, for example, time and frequency.

In algebraic mode, units are converted in the same way as in normal-mode operation. For example, to convert three feet to meters, use this key sequence:

3 [UNITS]	<i>enter 3 and display the units menu</i>
[F2]	<i>select the LENG menu</i>
[DIAMOND] [RIGHT]	<i>scroll to the menu that shows the _ft unit</i>
[F1]	<i>enter the _ft unit</i>
[▶]	<i>enter the conversion operator</i>
[F5]	<i>enter the _m unit</i>
[ENTER]	<i>do the conversion resulting in 0.9144_m</i>

In RPN mode, it is not necessary to use the conversion operator. The units are displayed in a line menu, and pressing [DIAMOND] before pressing the unit key performs the conversion. Repeating the example in RPN mode:

3 [UNITS]	<i>enter 3 and display the units menu</i>
[F2]	<i>select the LENG menu</i>
[DIAMOND] [RIGHT]	<i>scroll to the menu that shows the _ft unit</i>
[F1]	<i>enters the _ft unit and puts 3_ft on stack level 1:</i>
[DIAMOND] [F5]	<i>convert 3_ft to 0.9144_m on stack level 1:</i>

The numeric values that are displayed for constants and units depend on the current Unit System setting. This can be changed by pressing [MODE] [8], then choosing the desired unit system.

To display the constants/units line menu, press [UNITS]. The menu is:

F1: CONST	<i>Constants</i>
F2: LENG	<i>Length units</i>
F3: AREA	<i>Area units</i>
F4: VOL	<i>Volume units</i>
F5: TIME	<i>Time and frequency units</i>
F1: SPEED	<i>Speed units</i>
F2: MASS	<i>Mass units</i>
F3: FORCE	<i>Force units</i>
F4: ENRG	<i>Energy units</i>
F5: POWR	<i>Power units</i>
F1: PRESS	<i>Pressure units</i>
F2: TEMP	<i>Temperature units</i>
F3: ELECT	<i>Electricity units</i>

F4: LIGHT	<i>Light units</i>
F5: VISC	<i>Viscosity units</i>
F1: MAGN	<i>Magentism units</i>

Use [DIAMOND] [LEFT] and [DIAMOND] [RIGHT] to scroll the menu displays. The rest of this section describes the constants and units submenus.

Constants menu

Push [UNITS] [F1] to display the constants menu. Pressing a function key [F1] to [F5] multiplies the expression in the command line by the constant. If the command line is empty, pressing the function key for a constant enters the constant.

For example, to enter the speed of light constant, press

[UNITS]	<i>select UNITS menu display</i>
[F1]	<i>select CONST menu display</i>
[F1]	<i>select _c constant</i>

and `_c` is entered on stack level 1:.

If the command line is not empty when the unit is selected, the expression in the command line is multiplied by the constant. For example,

2 [UNITS] [F1] [F1]

results in `2*_c` in stack level 1:.

Note that the units associated with a constant depend on the current *Unit System* setting, which is found in the MODE menu. For example, if the *Unit System* is set to *SI*, then `_c` is displayed as `2.998E8 _m/_s`. If the *Unit System* is set to *ENG/US*, then `_c` is displayed as `9.836E8 _ft/_s`.

These constants are available:

F1: <code>_c</code>	<i>Speed of light in a vacuum</i>
F2: <code>_Cc</code>	<i>Coulomb constant</i>
F3: <code>_g</code>	<i>Acceleration of gravity</i>
F4: <code>_Gc</code>	<i>Gravitational constant</i>
F5: <code>_h</code>	<i>Planck's constant</i>
F1: <code>_k</code>	<i>Boltzman's constant</i>
F2: <code>_Me</code>	<i>Electron rest mass</i>
F3: <code>_Mn</code>	<i>Neutron rest mass</i>
F4: <code>_Mp</code>	<i>Proton rest mass</i>
F5: <code>_Na</code>	<i>Avogadro's number</i>
F1: <code>_q</code>	<i>Electron charge</i>
F2: <code>_Rb</code>	<i>Bohr radius</i>
F3: <code>_Rc</code>	<i>Molar gas constant</i>
F4: <code>_Rdb</code>	<i>Rydberg constant</i>
F5: <code>_Vm</code>	<i>Molar volume</i>

F1: ϵ_0	<i>Permittivity of a vacuum</i>
F2: σ	<i>Stefan-Boltzmann constant</i>
F3: ϕ_0	<i>Magnetic flux quantum</i>
F4: μ_0	<i>Permeability of a vacuum</i>
F5: μ_B	<i>Bohr magneton</i>

LENG (length) units menu

These units are displayed in the length units menu. They are the same units that are available in normal mode operation.

F1: \AA	Angstrom	F1: ft	foot (ENG/US)
F2: au	astronomical unit	F2: in	inch
F3: cm	centimeter	F3: km	kilometer
F4: fath	fathom	F4: lyr	light year
F5: fm	fermi	F5: m	meter (SI)
F1: mi	mile	F1: rod	rod
F2: mil	1/1000 inch	F2: yd	yard
F3: mm	millimeter	F3: μ	micron
F4: nm	nautical mile	F4: \AA	angstrom
F5: pc	parsec		

AREA (area) units menu

These units are displayed in the area units menu. Note that there are more units available here than in the normal-mode area units menu.

F1: acre	acre	F1: km^2	square kilometer
F2: cm^2	square centimeter	F2: mi^2	square mile
F3: ft^2	square foot	F3: mil^2	square mil
F4: ha	hectare	F4: mm^2	square millimeter
F5: in^2	square inch	F5: m^2	square meter
F1: yd^2	square yard		

VOL (volume) units menu

These units are displayed in the volume units menu. This menu includes a few units that are not available in normal-mode operation.

F1: cm^3	cubic centimeter (cc)	F1: gal	gallon (US)
F2: cup	cup	F2: galUK	gallon (British)
F3: ft^3	cubic foot	F3: l	liter
F4: floz	fluid ounce (US)	F4: m^3	cubic meter
F5: flozUK	fluid ounce (British)	F5: ml	milliliter
F1: mm^3	cubic millimeter	F1: yd^3	cubic yard
F2: pt	pint		
F3: qt	quart		

F4: _tbsp	tablespoon
F5: _tsp	teaspoon

TIME (time and frequency) units menu

This menu combines the normal-mode time and frequency menus. The available units are

F1: _Hz	hertz	F1: _ μ s	microseconds
F2: _kHz	kilohertz	F2: _ms	milliseconds
F3: _MHz	megahertz	F3: _s	seconds
F4: _GHz	gigahertz	F4: _min	minutes
F5: _ns	nanoseconds	F5: _hr	hours
F1: _day	day		
F2: _week	week		
F3: _year	year		

SPEED (speed) units menu

This menu is called the velocity menu in normal-mode operation. It is called SPEED in RPN, because velocity is actually a vector, and numbers with speed units are not vectors. The speed menu includes an additional unit not available in normal-mode operation, which is m/s. The available units are

F1: _knot	knot
F2: _kph	kilometer per hour
F3: _mph	miles per hour
F4: _m/s	meters per second

MASS (mass) units menu

F1: _amu	atomic mass unit	F1: _mol	mole (amount of substance)
F2: _gm	gram	F2: _mton	metric ton
F3: _kg	kilogram	F3: _oz	ounce
F4: _lb	pound (ENG/US)	F4: _slug	slug
F5: _mg	milligram	F5: _ton	ton
F1: _tonne	metric ton		
F2: _tonUK	long ton		

FORCE (force) units menu

F1: _dyne	dyne
F2: _kgf	kilogram force
F3: _lbf	pound force
F4: _N	newton
F5: _tonf	ton force

ENRG (energy) units menu

F1: _Btu	British thermal unit	F1: _J	joule
F2: _cal	calorie	F2: _kcal	kilocalorie
F3: _erg	erg	F3: _kWh	kilowatt-hour
F4: _eV	electron volt	F4: _latm	liter-atmosphere
F5: __ftlb	foot-pound		

POWR (power) units menu

F1: -hp	horsepower
F2: _kW	kilowatt
F3: _W	watt

PRESS (pressure) units menu

F1: _atm	atmosphere	F1: _mmHg	millimeters of mercury
F2: _bar	bar	F2: _Pa	pascal
F3: _inH2O	inches of water	F3: _psi	pounds per square inch
F4: _inHg	inches of mercury	F4: _torr	millimeters of mercury
F5: _mmH2O	millimeters of water		

TEMP (temperature) units menu

F1: _°C	degree Celsius
F2: _°F	degree Fahrenheit
F3: _°K	degree Kelvin
F4: _°R	degree Rankine

ELECT (electricity) units menu

This menu combines several categories of normal mode operation into these submenus:

F1: A (current units)	F1: _kA F2: _A F3: _mA F4: _μA	kiloampere ampere milliampere microampere
F2: V (voltage units)	F1: _kV F2: _V F3: _volt F4: _mV	kilovolt volt volt millivolt
F3: R (resistance units)	F1: _MΩ F2: _kΩ F3: _ohm F4: _Ω	megaohm kilo ohm ohm ohm

F4: C (capacitance units)	F1: _F F2: _ μ F F3: _nF F4: _pF F5: _coul	farad microfarad nanofarad picofarad coulomb
F5: L (inductance units)	F1: _henry F2: _mH F3: _nH F4: _ μ H	henry millihenry nanohenry microhenry
F1: 1/R (conductance units)	F1: _mho F2: _mmho F3: _ μ mho F4: _seimens	mho millimho micromho seimens

LIGHT (light) units menu

F1: _cd candela

VISC (viscosity) units menu

F1: _P poise (dynamic viscosity)
F2: _St stokes (kinematic viscosity)

MAGN (magnetism) units menu

This menu combines units for magnetic field strength, flux and flux density.

F1: _Oe oersted (magnetic field strength)
F2: _Gs gauss (magnetic flux density)
F3: _T tesla (magnetic flux density)
F4: _Wb weber (magnetic flux)

Special characters and the CHAR menu

RPN supports special characters, which are entered with the CHAR menu. Press [CHAR] to display the character menu:

- | | |
|------------|-----------------------------------|
| 1: Greek | <i>Greek alphabet</i> |
| 2: Math | <i>Mathematics symbols</i> |
| 3: Punct | <i>Punctuation symbols</i> |
| 4: Special | <i>Special calculator symbols</i> |

These are the four categories of special characters. To enter a character at the cursor position in the command line, choose the character from the submenu. For example, to enter the α character, press [CHAR], [1], [1], then α is inserted at the cursor position.

The CHAR menu can be customized as a pop-up menu or a command line menu. Use

[MODE], 3:Customize, 4:CHAR menu

to set this option.

Supported key functions

This version of RPN supports these function keys:

[ON]	Break a running Tlbasic program, or turn the calculator off if no program is running.
[OFF]	Turn the calculator off
[DIAMOND] [ON]	Turn the calculator off
[HOME]	Switch between RPN and last normal-mode application
[+], [-], [x], [/], [^]	Basic arithmetic functions
[x ⁻¹]	Reciprocal operator (TI92+ only)
[(-)]	Negation (change sign)
[EE]	Enter exponent
[sin], [cos], [tan], [cos ⁻¹], [sin ⁻¹], [tan ⁻¹]	Trigonometric functions
[√], [e ^x], [LN], [π]	Square root, natural logarithm functions and pi
[d], [∫]	Differentiation and integration
[=], [<], [>]	Conditional test operators
[DIAMOND] [=]	Not-equal operator ≠
[DIAMOND] [<]	Less-than-or-equal operator ≤
[DIAMOND] [>]	Greater-than-or-equal operator ≥
[]	Conditional "where" operator
[°], [►], [∠], [i], [∞], [□]	Degree symbol, unit conversion, angle, complex "i", infinity and units operator
[QUIT]	Close a built-in application and return to RPN
[ESC]	Close pop-up menus
[BACKSPACE]	Delete the character to the left of the cursor
[CHAR]	Display the special character menu
[UNITS]	Display the units and constants menu
[STO►]	"Store" operator
[RCL]	Display the RCL menu

[INS]	Perform Undo, if Undo feature is enabled.
[DEL]	Display the Delete menu
[ANS]	Paste <i>ans(1)</i> in the command line
[COPY]	Copy the marked expression in the stack or command line.
[CUT]	Cut the marked expression in the command line.
[PASTE]	Paste the expression in the command line.
[UP], [DOWN], [LEFT], [RIGHT]	Cursor movement and stack movement
[2nd] [DOWN] [<i>key</i>]	Push key code variable name for <i>key</i> in RPN mode. Paste key code variable name for <i>key</i> in algebraic mode.
[DOWN] [<i>key</i>]	Execute custom key assignment for <i>key</i>
[2ND] [LEFT], [2ND] [RIGHT]	Move cursor to beginning and end of command line or stack elements
[SHIFT] [UP]	Mark to beginning of line, if cursor is in command line. Scroll large objects if cursor is in the stack.
[SHIFT] [DOWN]	Mark to end of line, if cursor is in command line. Scroll large objects if cursor is in the stack.
[DIAMOND] [C] [DIAMOND] [X] [DIAMOND] [V]	COPY on the TI92+ CUT on the TI92+ PASTE on the TI92+
[DIAMOND] [LEFT], [DIAMOND] [RIGHT]	Scroll the command line menu to display more choices
[DIAMOND] [UP], [DIAMOND] [DOWN]	Move up one menu level in these line menus: RCL, custom, custom key, MATH, UNIT and CHAR.
[DIAMOND] [F1], ... [DIAMOND] [F5]	Push the variable <i>name</i> , not the variable value on the stack. This works in both the custom line menus and the custom key line menus.
[DIAMOND] [CLEAR]	Clear stack
[DIAMOND] [ENTER]	Evaluate (1:) in Exact mode if mode setting is Approximate, or in Approximate mode if the setting is Exact. If the mode setting is Auto, evaluate (1:) in Exact mode if the last evaluation was Approximate, and vice versa. Display a custom key assignment if (1:) is a key code variable.
[DIAMOND] [+] [DIAMOND] [-]	Increase display contrast Decrease display contrast

[DIAMOND] [(-)]	Display the Tools menu
[DIAMOND] [1]	Display Math Number menu
[DIAMOND] [2]	Display Math Test menu
[DIAMOND] [3]	Display Math Algebra menu
[DIAMOND] [4]	Display Math Calc menu
[DIAMOND] [5]	Display Math Complex menu
[DIAMOND] [6]	Display Math Stack menu
[DIAMOND] [7]	Display Math Matrix menu
[DIAMOND] [8]	Display Math Stat menu
[2nd] [7]	Display Math Calc Int menu
[F1], [F2], [F3], [F4], [F5]	Command line menu keys
[F6]	Switch between RPN and algebraic modes
[F7]	Swap function
[F8]	Switch delayed evaluation on and off
[MODE]	Display mode settings menu
[MATH]	Display MATH menu
[MEM]	Display normal-mode memory screen
[VAR-LINK]	Display normal-mode Var-Link screen
[CATALOG]	Display normal-mode function Catalog screen
[APPS]	Display normal-mode APPS screen
[2nd] [APPS]	Switch to the previous normal-mode application
[DIAMOND] [APPS]	Display the normal-mode Flash applications menu

Note that the five keys below work on the TI89 only if the line menus for RCL, custom and Unit menus are not active. These keys always work on the TI92+.

[Y=]	Display normal-mode Y= editor
[WINDOW]	Display normal-mode Window screen
[GRAPH]	Display normal-mode Graph screen
[TblSet]	Display normal-mode TblSet screen
[TABLE]	Display normal-mode Table screen
[CLEAR]	Clear command line or delete stack entry
[CUSTOM]	Display custom user menu

Alphabetic List of Functions

The table below shows all built-in 89/92+ commands, as well as the functions that are only in RPN. This table serves two purposes: you can find the menu in which an operation can be found, and you can quickly determine if an 89/92+ operation is supported by RPN. By *operation* I mean a function or command.

[keyboard] is shown if the operation is on the 89/92+ keyboard, not in a menu.

[type] is shown if the operation can only be executed by typing the function name.

[command line] is shown if the operation can be executed from the command line.

Some 89/92+ operations are not supported. These operations are used only in TIBasic programming and are not useful in RPN. Operations that are not implemented are shown in a smaller font size, in blue, and there is no description beside the function.

<u>Function</u>	<u>Menu location</u>
abs()	Math Number, Math Complex
and	Math Test
AndPic	[command line]
angle()	Math Complex
ans()	[type] Does not support argument; can only return ans(1). Also use [ANS]
approx()	Math Algebra
Archive	Tools, Archiv
arcLen()	Math Calc
augment()	Math Matrix Augment
avgRC()	[type]
►Bin	Math Base
BldData	Tools, Graphics, BldData
ceiling()	Math Number
cFactor	Math Algebra Complex
char()	[type]
Circle	[command line]
ClrDraw	Tools, Graphics, Clr
ClrErr	
ClrGraph	Tools, Graphics, Clr
ClrHome	Tools, Graphics, Clr
clrIO	Tools, Graphics, Clr
ClrTable	Tools, Graphics, Clr
colDim()	Math Matrix Extr dim
colNorm()	Math Matrix Extr norms
comDenom()	Math Algebra ComDen
conj()	Math Complex
CopyVar	Tools Var
cos()	[keyboard]
cos ⁻¹ ()	[keyboard]
cosh()	Math Hyperbol
cosh ⁻¹ ()	Math Hyperbol

crossP()	Math Matrix Ops Vector
cSolve()	Math Algebra Complex
CubicReg	Math Stat Regress Cubic
cumSum()	Math List
CustmOff	[command line]
CustmOn	[command line]
Custom	
Cycle	
CyclePic	
►cylind	Math Matrix Ops Vector
cZeros	Math Algebra Complex
d()	[keyboard]
►DD	Math Angle
►Dec	Math Base
Define	Tools Var
DEL	Math Stack
DEL n	Math Stack
DelFold	Tools Folder
DelVar	Tools Var
deSolve()	Math Calc
det()	Math Matrix Solve
diag()	Math Matrix Extr
Dialog	
dim()	Math Matrix Extr dim
Disp	[command line]
DispG	[command line]
DispHome	[command line]
DispTbl	[command line]
►DMS	Math Angle
dotP()	Math Matrix Ops Vector
DrawFunc	Tools Graphics Draw
DrawInv	Tools Graphics Draw
DrawParm	Tools Graphics Draw Parm
DrawPol	Tools Graphics Draw Pol
DrawSlp	Tools Graphics Draw
DropDown	
DrwCtour	Tools Graphics Draw
DUP 2	Math Stack
DUP N	Math Stack
E	[keyboard]
e^()	[keyboard]
eigVc	Math Matrix Extr
eigVI()	Math Matrix Extr
Else	
Elseif	
EndCustm	
EndDlog	
EndFor	
EndFunc	
EndIf	
EndLoop	
EndPrgm	
EndTBar	
EndTry	

EndWhile	
entry()	[type] Does not support argument, can only return entry(1)
exact()	Math Number exact
Exec	
Exit	
exp►list()	[type]
expand()	Math Algebra
expr()	[type]
ExpReg	Math Stat Regres Exp
factor()	Math Algebra
Fill	Math Matrix New
floor()	Math Number
fMax()	Math Calc
fMin()	Math Calc
FnOff	Tools Graphics FnOff
FnOn	Tools Graphics FnOn
For	
format()	[type] Does not support format string; uses current display format
fPart()	Math Number
Func	
gcd()	Math Number
Get	
GetCalc	
getConfig()	[type]
getDenom()	Math Algebra Extract
getFold()	[type]
getKey()	[type] Always returns 0
getMode()	[type]
getNum()	Math Algebra Extract
getType()	[type]
getUnits()	[type]
Goto	
Graph	Tools Graphics Graph
►Hex	Math Base
Identity()	Math Matrix New
if	
imag()	Math Complex
Input	
InputStr	
InString()	[type] <i>start</i> argument not supported
int()	(same as Floor)
intDiv()	[type]
integrate	Math Calc Int
iPart()	Math Number
isPrime()	Math Test
Item	
Lbl	
lcm()	Math Number
left()	Math Algebra Extract, or Math List
limit()	Math Calc Limit
Line	[command line]

LineHorz	[command line]
LineTan	[command line]
LineVert	[command line]
LinReg	Math Stat Regres Lin
Δlist	[command line]
list►mat()	Math List List►m
list►Stk	Math Stack
ln()	[keyboard] or Math log
lnReg	Math Stat Regres Ln
Local	
Lock	Tools Var
log()	Math log
Logistic	Math Stat Regres Logistic
Loop	
LU	Math Matrix Extr
mat►list()	Math List
max()	Math List
mean()	Math Stat
median()	Math Stat
MedMed	Math Stat Regres MedMed
mid()	Math List
min()	Math List
mod()	Math Number
MoveVar	Tools Var
mRow()	Math Matrix Ops Row
mRowAdd()	Math Matrix Ops Row
nCr()	Math Prob
nDeriv()	Math Calc
NewData	Tools NewData ...,n
NewFold	Tools Folder
newList()	Math List
newMat()	Math Matrix New
NewPic	[command line]
NewPlot	[command line]
NewProb	Tools NewProb
nInt()	Math Calc Int
norm()	Math Matrix Extr Norms
not	Math Test
nPr()	Math Prob
nSolve()	Math Algebra
OneVar	Math Stat
or	Math Test
ord()	[type]
Output	[command line]
P►Rx()	Math Angle
P►Ry()	Math Angle
part()	[type] Does not support 'level' argument
PassErr	
Pause	
PICK 2	Math Stack
PICK n	Math Stack

PlotsOff	[command line]
PlotsOn	[command line]
►Polar	Math Matrix Ops Vector
polyEval()	Math List
PopUp	
PowerReg	Math Stat Regres Power
Prgm	
product()	Math List
Prompt	
propFrac()	Math Algebra Prop
PtChg	[command line]
PtOff	[command line]
PtOn	[command line]
ptTest()	[type]
PtText	[command line]
PxlChg	[command line]
PxlCrcl	[command line]
PxlHorz	[command line]
PxlLine	[command line]
PxlOff	[command line]
PxlOn	[command line]
pxlTest()	[type]
PxlText	[command line]
PxlVert	[command line]
QR	Math Matrix Extr
QuadReg	Math Stat Regres Quad
QuartReg	Math Stat Regres Quart
R►Pθ()	Math Angle
R►Pr()	Math Angle
rand()	Math Prob
randMat()	Math Matrix New, also Math Prob
randNorm()	Math Prob
randPoly()	Math Prob
RandSeed	Math Prob
RclGDB	[command line]
RclPic	[command line]
real()	Math Complex
►Rect	Math Matrix Ops Vector
ref()	Math Matrix Solve
remain()	Math Number
Rename	Tools Var
Request	
Return	
right()	Math Algebra Extract, also Math List
ROLL 3	Math Stack
ROLL n	Math Stack
rotate()	[type] Does not support #ofRotations argument
round()	Math Number
rowAdd()	Math Matrix Ops Row
rowDim()	Math Matrix Extr Dim
rowNorm()	Math Matrix Extr Norms
rowSwap()	Math Matrix Ops Row
RplcPic	[command line]

rref()	Math Matrix Solve
Send	
SendCalc	
SendChat	
seq()	Math List
setFold()	[type]
setGraph()	[type]
setMode()	[type]
setTable()	[type]
setUnits()	[type]
Shade	[command line]
shift()	[type] Does not support <i>#ofShifts</i> argument
ShowStat	Math Stat
sign()	Math Number
simult()	Math Matrix Solve
sin()	[keyboard]
sin ⁻¹ ()	[keyboard]
sinh()	Math Hyperbol
sinh ⁻¹ ()	Math Hyperbol
SinReg	Math Stat Regres Sin
solve()	Math Algebra
SortA	Math List
SortD	Math List
►Sphere	Math Matrix Ops Vector
stdDev()	Math Stat
Stk►List	Math Stack
StoGDB	[command line]
Stop	[command line]
StoPic	[command line]
Store	[keyboard]
string()	[type]
Style	[command line]
subMat()	Math Matrix Extr
sum()	Math List
SWAP	Math Stack
switch()	[type] Does not support window number parameter
τ (transpose)	Math Matrix Solve
Table	Tools Graphics Table
tan()	[keyboard]
tan ⁻¹ ()	[keyboard]
tanh()	Math Hyperbol
tanh ⁻¹ ()	Math Hyperbol
taylor	Math Calc Taylor
tCollect()	Math Algebra Trig
tExpand()	Math Algebra Trig
Text	
Then	
Title	
tmpCnv()	[type] or use [DIAMOND] [F1] - [DIAMOND] [F4] in the UNITS TEMP menu
Δ tmpCnv()	[type]
Toolbar	
Trace	
Try	

TwoVar	Math Stat
Unarchiv	Tools Var
UNDO	Math Stack or [INS]
unitV()	Math Matrix Ops Vector
Unlock	Tools Var
variance	Math Stat
when()	[type] Does not support <i>falseResult</i> or <i>unknownResult</i> arguments
While	
"With"	[keyboard]
xor	Math Test
XorPic	
zeros()	Math Algebra
ZoomBox	Tools Graphics Zoom
ZoomData	Tools Graphics Zoom
ZoomDec	Tools Graphics Zoom
ZoomFit	Tools Graphics Zoom
ZoomIn	Tools Graphics Zoom
ZoomInt	Tools Graphics Zoom
ZoomOut	Tools Graphics Zoom
ZoomPrev	Tools Graphics Zoom
ZoomRcl	Tools Graphics Zoom
ZoomSqr	Tools Graphics Zoom
ZoomStd	Tools Graphics Zoom
ZoomSto	Tools Graphics Zoom
ZoomTrig	Tools Graphics Zoom
+	[keyboard]
- (subtract)	[keyboard]
*	[keyboard]
/	[keyboard]
- (negate)	[keyboard]
%	[command line]
=	[keyboard] or Math Test
≠	[keyboard] or Math Test
<	[keyboard] or Math Test
≤	[keyboard] or Math Test
>	[keyboard] or Math Test
≥	[keyboard] or Math Test
.+	Math Matrix Ops Element
.-	Math Matrix Ops Element
.*	Math Matrix Ops Element
./	Math Matrix Ops Element
.^	Math Matrix Ops Element
!	[keyboard] or Math Prob
&	[command line]
f()	[keyboard] or Math Calc Int
√()	[keyboard]
Π()	Math Calc
Σ()	Math Calc

^	[keyboard]
# (indirection)	[command line]
ʳ (radian)	Math Angle
° (degree)	Math Angle
∠ (angle)	[keyboard]
° , ' , "	[keyboard]
' (prime)	[keyboard]
_ (underscore)	[keyboard]
► (convert)	[keyboard]
10^()	[keyboard]
x ⁻¹	[keyboard]
"with"	[keyboard]
→ (store)	[keyboard]
Ⓒ (comment)	
0b, 0h	[keyboard]