Documentation for *RPN Calc.*

(A Reverse Polish Notation Calculator for the Macintosh)

Document revision dates: 21Aug88, 25Sep88, 21Oct88, 7Aug89, 2Apr91, 5Oct91, 30Dec91, 20Mar92, 24Apr92, 7Apr93.

***RPN Calc* features**

Reverse Polish Notation.
Up to 18 digits of decimal accuracy with exponents as large as 4000.
Display numbers in any base from 2 to 16.
User specified output modes.
Logarithmic and trig functions.
Keyboard equivalents.
Remembers screen location and register values.
Color on Mac II.
On-line help.
Permutations & Combinations.
Financial functions (for cash flow analysis).

Installation

Always remove the old version of RPN Stuff/Prefs from your system folder. Using the Font DA Mover, install *RPN Calc* into the system file or a FONT/DA document for use with Suitcase (Trademark of Fifth Generation Systems Inc.). For System 7.0 simply open the rpnda suitcase document and drag RPN Calc to the system folder.

Acknowledgments

This desk accessory was written in ThinkC so portions are copyrighted by Symantec Co.

I'd like to thank the following individuals:

Who	Why
Nicholas Pisarro, Jr.	Bug reports and feature suggestions.
James Nutting	Help in debugging the interest rate function.
Will Singleton	Financial functions request, feature suggestions, and beta testing of version 8908xx.
many	Who reminded me to make and release a System 7.0 version.

Distribution

This desk accessory is $10 shareware. Anyone who finds themselves repeatedly using this desk accessory should send the shareware fee, in US funds, payable to ViviStar Consulting at the address shown below. If you find yourself in possession of *RPN Calc* but not repeatedly using it do **not** delete it - keep it in the event you might start to use it and send money then.

Jonathan Hess
ViviStar Consulting
7015 E Aster Dr
Scottsdale AZ 85254; USA
Electronically: Compuserve (73067,542) or Internet (73067.542@compuserve.com).

Change history

930407	*Corrected yet another closing bug. RPN Calc would cause a bus error if closed in an abnormal entry state.*
920424	Cosmetic change in credits: Addition of AppleLink address. Not actively uploaded to information services
920406	Cosmetic change in credits: Vividus -> ViviStar. Not actively uploaded to information services. Also converted this documentation file to Word 5.0.
911230	Cosmetic revision allowing "Keyboard Equivalents" help dialog to properly display on monitors of limited color depth (i. e. 1 and 2 bits). No code was changed.
911004a	This version is for System 7.0 compatibility. Several coloring changes including the addition of color icons. Preferences are automatically placed in the system folder's preference folder. System 7.0's absence of a SetWVariant function, the need for 32 bit and virtual memory compatibility, and limited programmer resources have resulted in much longer opening and close times. If response (in the form of registration) is overwhelming, a future version may provide near instantaneous opening and closing.
910402a	This version is an attempt to fix an elusive "closing bug" manifested mostly when not using MultiFinder. When closing, RPN Calc now de-hilites it's window, takes some time saving status information, and finally removes itself.

Reverse Polish Notation

The rest of this document briefly explains the use of Reverse Polish Notation (RPN). It is written with the *RPN Calc* desk accessory in mind but does not replace *RPN Calc*'s on-line documentation. *RPN Calc*'s on-line documentation assumes familiarity with RPN. This document provides that familiarity.

Introduction

For a calculator to solve a mathematical expression there must exist a way to express it to the calculator. Most mathematical expressions can be broken down into two elementary operations; binary operations and unary operations. Binary operations operate on two numbers and unary operations operate on one. Binary examples include addition, subtraction, multiplication, and division. Unary examples include square, reciprocal, and logarithm. For a simple calculation the calculator must be told the operation and one or two numbers.

Pre, in, and postfix

There are three ways of expressing mathematical formulas: prefix, infix, and postfix. The prefixes in these names indicate where the operation is specified. With prefix the operation precedes the two numbers, with infix it is specified between the two numbers, and with postfix it follows the two numbers. Consider the division of 28 by 7.

prefix:	/	28	7
infix:	28	/	7
postfix:	28	7	/

Please note the order of the numbers is important; the division of 28 by 7 is not the division of 7 by 28.

Prefix is not widely used and will not be considered further. Most calculators use infix. Postfix is just another name for RPN. The rest of this document describes postfix and compares it with infix.

The juxtaposition of 28 and 7 brings the question "how does the calculator know when one number ends and another number begins?" An enter key, signified by ¬, solves this problem for the postfix calculator. The actual key sequence is "28¬7/." After pressing the / the calculator will immediately respond with 4. An ¬ isn't needed after the 7 because the division symbol implies the second number was complete.

This appears to take more keystrokes than infix because of the ¬. However, this is not true. Infix suffers from the same problem but it occurs at the end. For an infix calculator the key sequence "28/7" does nothing until the = is pressed. The = implies the end of the second number. As with postfix, any operator signifies the end of the previous number.

Infix was originally chosen for calculators because it more naturally follows the algebraic order of operations. As a result of this pre-bias, learning to use an RPN calculator is often avoided. When mastered, RPN is faster and more intuitive. It also becomes chore to use an infix calculator again. The benefits of postfix show themselves when doing operations on a series of numbers or when working with an expression that contains many operations.

Repetitive expressions

Consider the addition of the series 1, 2, 3, 4. The infix expression is

1 + 2 = + 3 = + 4 =.

The postfix expression is

1 ¬ 2 + 3 + 4 +.

As an aid in explaining this postfix key sequence, and to prepare for more complicated problems, the stack concept must be introduced.

The register stack

A computer or calculator stack can be compared to a stack of dishes. Barring insertions or deletions in the middle of the stack, the first item on the stack is the last item off the stack and the last item on the stack is the first item off. *RPN Calc* has an upside down stack as shown below. The fixed size of this stack is four and each position or register has a name associated with it. All values enter and leave the stack from the x register. Also, the calculator always displays the value in the x register.

t
r
y
x

Note: This is the same naming convention used by scientific calculators made by Hewlett - Packard. Hewlett - Packard financial calculators call the r register z.

With this organization new numbers, entered by the enter key or implicitly by an operation key, are pushed into the x position and all other values move up one register. The number that was in the t register is lost. When an operand is pulled from the stack it comes from the x register and all other values move down one register. Since there is no register above t, the value in t after a pull is the value that was originally in t.

During an operation the calculator takes the required number of arguments off the stack, performs the operation, and pushes the result back on. Referring back to the 28/7 example the following stack sequence occurs.

	28¬	7†	/††	/†††
t	…	…	…	…
r	…	…	…	…
y	…	28	…	…
x	28	7	…	4

Explanation of sequence:
†: The enter implied by the / operator.
††: The calculator pulled the arguments off the stack.
†††: The calculator pushes the answer onto the stack.
…: This register does contains a value but it is not involved with the given problem
In subsequent examples step /†† will be skipped.

Returning to the 1 + 2 + 3 + 4 the following stack sequence occurs:

	1¬	2	+	3	+	4	+
t	…	…	…	…	…	…	…
r	…	…	…	…	…	…	…
y	…	1	…	3	…	6	…
x	1	2	3	3	6	4	10.

Operator precedence

To make the transition to more complex problems consider 1 + 2 * 3. This is ambiguous, it is unclear which operation is to be performed first. Algebraic rules of precedence set the order of operation as multiplication first and then the addition. Nearly all infix calculators follow the algebraic order of precedence and would evaluate this expression as 1 + (2 * 3) = 1 + 6 = 7. A new postfix user may try the sequence 1 ¬ 2 + 3 *. This evaluates as (1 + 2) * 3 = 9 and is probably not the desired answer. The correct postfix expression is 1 ¬ 2 ¬ 3 * +. To show that this works consider the stack sequence

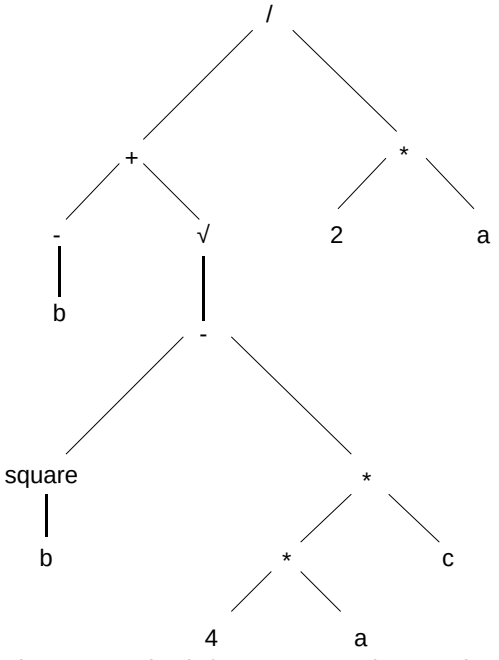
	1¬	2¬	3	*	+
t	…	…	…	…	…
r	…	…	1	…	…
y	…	1	2	1	…
x	1	2	3	6	7.

Where the infix calculator assumes the order of operations, a postfix calculator requires the user to know the order of the operations wanted. It will be shown that with the order of operations known, postfix is quicker.

Evaluating complicated expressions

The quadratic equation is a good example of more involved expressions.

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$



The tree to the left represents the quadratic equation. Note that the tree representation leaves no ambiguities as to the ordering of operations.

The tree shows that any operator has at most two sub expressions. For postfix, evaluating both subtrees of an operator allows evaluation of the operation. This principle of evaluating subtrees first, when applied recursively, allows evaluation of the entire tree.

This principle can be restated as follows. Evaluate the left subtree first, leave the result on the stack, evaluate the right subtree, and then specify the operator involved between the two subtrees.

The entire tree is evaluated when the operation at the top is performed. This can be done when both its subtrees are evaluated. The subtree on the left is topped by a +. To evaluate the + both its subtrees must be evaluated. Its left most subtree is the unary operation which changes the sign of b (chs). The postfix string is

b ¬ chs.

Now the right subtree of the + needs evaluation. It consists of the unary operation square root with the value determined by the tree below it. The tree below the square root is topped by the -. To evaluate the left subtree of the - we have the unary square b. The string is now

b ¬ chs b square.

The right subtree of - consists of the two subtrees topped by multiplication operators. The expression for the left most multiplication is 4 ¬ a *. This is appended to the string which is now

b ¬ chs b square 4 ¬ a *.

To evaluate the upper *, c * is appended resulting in

b ¬ chs b square 4 ¬ a * c *.

Since the right subtree of the - is evaluated the - operation is specified and the string is

b ¬ chs b square 4 ¬ a * c * -.

Proceeding back up the tree the operators √ and + are appended resulting in

b ¬ chs b square 4 ¬ a * c * - √ +.

Now the right subtree of / must be evaluated. It is the binary operation 2 ¬ a *. This is appended onto the string with the / for the complete postfix expression

b ¬ chs b square 4 ¬ a * c * - √ + 2 ¬ a * /.

Comparing this with the infix expression for a calculator that has parenthesis

(- b + (b square - 4 * a * c) √) / (2 * a) =

shows that the postfix expression has 4 fewer key strokes. With experience postfix becomes more intuitive and it is easier to evaluate complicated expressions because its not necessary to have to match parenthesis.

For a complete analysis of the quadratic equation a symbolic stack sequence is shown below.

	b¬	-	b	^2	4¬	a	*	c	*
t	…	…	…	…	…	-b	-b	b-	-b
r	…	…	…	…	-b	b^2	-b	b^2	-b
y	…	…	-b	-b	b^2	4	b^2	4a	b^2
x	b	-b	b	b^2	4	a	4a	c	4ac

-	√	+	2¬	
t	-b	-b	-b	-b
r	-b	-b	-b	-b
y	-b	-b	-b	-b+√(b^2 - 4ac)
x	b^2 - 4ac	√(b^2 - 4ac)	-b+√(b^2 - 4ac)	2

a	*	/	
t	-b	-b	-b
r	-b+√(b^2 - 4ac)	-b	-b
y	2	-b+√(b^2 - 4ac)	-b
x	a	2a	(-b+√(b^2 - 4ac))/(2a)

More on the stack and complicated expressions

As previously stated, when something is pushed onto the stack, the value in the r register replaces the value in the t register. The previous t value is **lost!** A postfix expression that would overflow the xyrt stack is not hard to imagine. Consider the postfix expression 1¬2¬3¬4¬5++++. Although a valid postfix expression for the addition of 5 numbers, it will overflow the calculator stack and result in the wrong answer. The stack sequence is as follows.

	1¬	2¬	3¬	4¬	5	+	+	+	+
t	…	…	…	1	2	2	2	2	2
r	…	…	1	2	3	2	2	2	2
y	…	1	2	3	4	3	2	2	2
x	1	2	3	4	5	9	12	14	16

When 5 is pushed onto the stack the value in t, which is 1, is lost. This constitutes a stack overflow (every stack push loses information but in this case it is needed information). When the last + is executed a 1 should be pulled from the stack. This value was lost when the 5 was pushed onto the stack. When the pull for a 1 is executed a 2 is received since the t register, lacking the knowledge of what value it should assume, assumes its previous value.

A postfix expression which works is 1¬2+3+4+5+. Practically all mathematical expressions are expressible in a postfix form which will not overflow the stack. A rule of thumb: Pressing the enter key repeatedly without intervening binary operations is a sure way to overflow the stack.

If stack overflow is a concern, intermediate values can be stored and recalled from memory registers. This is a good idea in general because it breaks large expressions into smaller ones. If one portion of the expression was evaluated incorrectly, the other portions need not be re-evaluated.

RPN discussion conclusion

By knowing how the stack operates and what the operator keys do to the stack, an RPN calculator can be made to evaluate an expression exactly in the manner required.

Financial functions on *RPN Calc*

Besides the standard function keys, *RPN Calc* includes five financial keys. The financial functions deal with cash flow analysis with compounded interest. Internal to the calculator are five financial registers which hold values for the cash flow problem at hand. An unknown quantity can be found if the other four registers contain the known information.

The financial keys are dual purpose in nature†; they either store a value in the given register, or calculate the given financial register from the values in the other four registers. The rule determining when a value is stored and when it is calculated it quite simple:

> The value in the display is stored in the specified register unless the previous key pressed was a financial key (pv, pmt, fv, i, n).

This means the required sequence of events is to enter the number for a given financial factor and store it in the given register by pressing the associated key. This process repeats until the four unknown factors are stored. At this point the last key pressed was a financial key. The unknown factor is calculated by immediately pressing the associated key.

If the known registers have appropriate values but it is not known if the last key pressed was a financial key, then just press the desired key twice. If the last key pressed was a financial key this action just calculates the unknown twice. If it wasn't, the first press will store the display value into the register and the second press will calculate the true value from the other 4 registers.

Please note:

- All registers retain their previous value unless specifically changed.
- A financial register cannot be assumed 0 unless it is set to 0.
- The sign convention† used is cash received is positive and cash paid out is negative.
- Store and Recall also work on the financial keys.
- *The interest rate (i) is expressed in percentage points, not fractions. Ten percent is 10 not .1.*

† These follow Hewlett - Packard's convention.