RPN programmable calculator

for Transputer

guillaume.tello@orange.fr

Table of contents

Overview

RPN is a calculator using the Reverse Polish Notation, as most Hewlett Packard machines until the 90's.

Every calculation is made on a stack :

- You push your arguments on the stack
- The operation/fiunction pops them from the stack
- the result is pushed back on the stack.

This calculator uses a stack of 20 floating points in double precision.

You have access to 26 individual variables named from a to z in double precision too.

It features arithmetic operations, trigonometric, logarithms, exponentials, stack movement, memory access, structured loops and some more functions as SOLVE, INTEG.

Invoking the calculator

On the command line, just type:

```
>RPN ...operations... [ENTER]
```

And the commands will be executed. At the end, RPN displays the value at the top of the stack (if any and if not in silent mode). But you can display intermediate results when needed.

All **operations are separated by a space**.

There are **two operating modes** : Interactive and Non-Interactive.

In the **Non-Interactive mode**, only the commands from the **command line** are executed and then the program exits.

In **Interactive mode**, when the command line instructions are executed (or if the line is empty), then a **prompt appears** and wait for a new set of commands. *Enter an empty line to exit.*

`inter+` enter interactive mode (default)

`inter-` quit interactive mode

If nothing happens when you run RPN, force interactive mode from a command line:

```
>RPN inter+
```

Arithmetic operations

Here is a list of the basic operations you can perform:

+	pop two values on the stack and push the sum
-	pop two values on the stack and push the difference
*****	pop two values on the stack and push the product
/	pop two values on the stack and push the quotient
mod	pop two values on the stack and push the remainder of the division
sqr	replace the value on the stack with its square root
x2	replace the value on the stack with its square
1/x	replace the value on the stack with its inverse
+/-	replace the value on the stack with its opposite
abs	replace the value on the stack with its absolute value
int	replace the value on the stack with its floor
frac	replace the value on the stack with its decimal part
rnd	returns a random number into [0 ; 1 [,
.	pop the value from the stack and display it

Examples:

$(5+6)\times 4$	`>RPN 5 6 + 4 * .`
$\frac{1}{0{,}5^2+2{,}3^2}$	`>RPN 0.5 x2 2.3 x2 + 1/x .`
$\frac{-2+\sqrt{15}}{5}$	`>RPN -2 15 sqr + 5 / .`
$\frac{6}{7}-\frac{8}{15}$	`>RPN 6 7 / 8 15 / - .`

Stack movements

dup	duplicate the topmost value on the stack
drop	remove the topmost value from the stack
swap	exchange the two topmost values on the stack
over	duplicate the second value over the topmost value on the stack
clr	empty stack
stack	display stack content

Examples:

- **>RPN 5 7 over + / .**

  over duplicate 5 over 7, so you have: `5 7 5`

  + add 7 and 5

  and then / computes the final result $\frac{5}{7+5}$

- **>RPN 15 dup * .**

  dup duplicates 15, so you have: `15 15`

  * computes the final result $15 \times 15 = 15^2$

  This is equivalent to :

  `>RPN 15 x2 .`

- **>RPN 8 11 swap / .**

  values are swapped, so you have `11 8`

  then the quotien $\frac{11}{8}$ is computed

  This is equivalent to

  `>RPN 8 11 / 1/x .`

  or faster

  `>RPN 11 8 / .`

Variable operations

Every variable operation has the form : **$vo**

$ is the dollar sign to specify a variable operations

v is the variable name from **a** to **z**

o is one of the operation in the list **+ * - / 0 @ !**

Examples:

$a!	pop the topmost value from the stack and store in a
$g@	push the current value of g on the stack.
$x+	pop the topmost value from the stack and add to the content of x
$z-	pop the topmost value from the stack and subtract to the content of z
$b*	pop the topmost value from the stack and multiply the content of b
$n/	pop the topmost value from the stack and divide the content of n
$w0	set to zero the variable w.

There are **six shortcuts** for specific variables used in **loops** *(i, j and k)* or in **functions** *(x, y and z).*

i	replace **$i@** to recall the content of variable **i**
j	replace **$j@** to recall the content of variable **j**
k	replace **$k@** to recall the content of variable **k**
x	replace **$x@** to recall the content of variable **x**
y	replace **$y@** to recall the content of variable **y**
z	replace **$z@** to recall the content of variable **z**

Some useful words:

var	display all non zero variables
clrv	set all variables to zero

Arrays

Every variable can be the first element of an array.

For example:

- a is A(1)
- b is A(2), or B(1)
- c is A(3), or B(2), or C(1)
- and so on...

To specify that you want to talk about an array element, just use an **upper case letter.**
In this case, the **topmost value** on the stack is used as **index in the array**.

Examples:

```
>RPN pi 5 $A!
```

Store pi into e, because e=A(5).

```
>RPN clrv 1000 [ 1 rnd 10 * 1 + int $L+ ] var
```

`clrv 1000 [`	variables are cleared and we start a one-thousand-loop
`1`	one, waiting on the stack
`rnd 10 * 1 + int`	one value from 1 to 10 as index in L table
`$L+`	add the one into L(index)
`] var`	display the table from L(1)=l to L(10)=u

So you can se the repartition of random values with ten intervals.

Scientific operations

ln	replace the value on the stack with its natural logarithm
exp	replace the value on the stack with its exponential
log	replace the value on the stack with its decimal logarithm
xy	replace the two topmost values with the second to the power of the topmost
pi	push on the stack the value of π=3,1415...
deg	set the angle unit to degree
rad	set the angle unit to radian
cos	replace the angle value on the stack with its cosine
sin	replace the angle value on the stack with its sine
tan	replace the angle value on the stack with its tangent
acos	replace the value on the stack with its arccosine
asin	replace the value on the stack with its arcsine
atan	replace the value on the stack with its arctangent
atan2	pop two values on the stack and push atan(y/x). The order is: **y x atan2**

Examples :

```
>RPN x dup ln * .
```

return the value of $x \times \ln(x)$

```
>RPN rad pi 5 / cos 3 xy .
```

Return the value of $\cos\left(\frac{\pi}{5}\right)^3$

Compare instructions

All those instructions return either 1 (TRUE) or 0 (FALSE).
Actually, every non zero value is considered as TRUE.

eq pop the two topmost values and return 1 if they are =

ne pop the two topmost values and return 1 if they are not =

gt pop the two topmost values and return 1 if the second is > than the topmost

lt pop the two topmost values and return 1 if the second is < than the topmost

ge pop the two topmost values and return 1 if the second is >= than the topmost

le pop the two topmost values and return 1 if the second is <= than the topmost

Examples:

```
>RPN x y eq .
```

Return 1 if x=y

```
>RPN i 0 lt i 5 gt + .
```

Return a non zero value if i<0 OR i>5

```
>RPN x 9 ge x 20 le * .
```

Return 1 only if x>=9 and x<=20.

Loops

Simple loops with counter or Infinite loops

Loops are enclosed in brackets [and]. Before a loop starts, **a value must be prepared** on the top of the stack.

If this value is **n = 0**, then it's an **infinite loop**.

If this value is **n > 0**, then it's a **loop with a counter** from **n down to 1**.

```
>RPN 0 [ ]
```

is an unending nothing, use Ctrl+C to abort the program.

```
>RPN 10 [ ]
```

does nothing, but only ten times!

When you open **a first loop with counter**, the counter is stored into **variable i**.

```
>RPN 10 [ i . ]
```

display all values from 10 downto 1.

```
>RPN 0 100 [ i + ] .
```

sum all integers from 1 to 100 and display result.

If you open a **second loop** inside the first one, then the new counter is stored into **variable j**. The same applies if you open a **third nested loop**, its **counter will be k**. You can nest as much as 10 loops.

Note: from variable l, there is no shortcut, you must use $l@ to recall its content.

```
>RPN 0 10 [ 5 [ i j * + ] ] .
```

return the sum of every $i \times j$ for i into [1 ; 10] and j into [1 ; 5]

Conditional loops with or without counter

Two conditions can be used inside a loop:

?n ***"if next"***, pop the topmost value from the stack and if it's TRUE (non zero), then jump to the end of the loop for the next value.

?e ***"if exit"***, pop the topmost value from the stack and if it's TRUE (non zero), then jump to the instruction after the end of the loop, the loop is finished.

So now, you can understand the presence of infinite loops. They can be considered as loops while a condition is FALSE, or until a contition is TRUE.

Examples:

>RPN 1000 0 [dup 17 mod 0 eq ?e 1 +] .

search for the first multiple of 17 starting at 1000.

Details:

`1000` is on the stack, then `0 [` starts an infinite loop.

`Dup`, duplicates `1000` and `17 mod` returns the remainder of `1000/17`.

The remainder is tested to know if it's equal to zero.

If so, the `?e` exit the loop and the value is displayed.

If not, then `1` is added to `1000`, and the loop goes on with `1001`, etc, until a remainder of zero is found. This means that the number is multiple of 17.

>RPN 0 10 [10 [i j eq ?n i j * +]] .

return the sum of every $i \times j$ for i and j into [1 ; 10] when $i \neq j$.

Details:

`0` on the stack to init the sum.

`10 [ 10 [` open two loops from 10 down to 1.

`i j eq ?n` test for equality between i and j, if so, skip sum and jump to the next value

`i j * +` sum a new product to the value on the stack.

>RPN t+ 1 2499 [dup * sqr ln exp atan tan 1 +] dup 2500 - . .

The **savage benchmark** that tests precision and speed of scientific functions.

Returns the error beetween 2500 and actual value.

Functions

You can add a prefix to the **`n [ ]`** loop structure to use some functions facilities.

In this case, the brackets enclose the description of a function. The value of the variable will be x. As for the loops with counters, if another function structure is opened within the first one, then the second variable will be y, then z. You can't nest more than three functions. And the sum of nested loops and functions can't exceed 10.

Solve

The general form is that:

`>RPN solve x0 [ function ]`

`solve` is the prefix

`x0` a value on the stack used as first approximation for the solution

`function` any calculations using x that return f(x), a single value on the stack.

This operation **solves for f(x)=0** and return on the stack (and in variable x), the solution.

Example :

`>RPN solve 5 [ x ln - 1 ] .`

Solve for $\ln(x)-1=0$, starting with x=5, so will return x = e = 2,7182818...

`>RPN 10 [ solve 1 [ x x2 i - ] . ]`

Solve every equation $x^2-i=0$ for i in [1 ; 10] and display the result. So, you'll get the list of the first ten square roots.

Integ

The general form is that:

`>RPN integ n [ function ]`

But, you have to fill variables a and b that are the limits of the integral.

So, a more complete for will be:

`>RPN start $a! end $b! integ n [ function ]`

This operation computes $\int_a^b f(x)\,dx$ with the Gauss method with 3 points.

The value n is the number of intervals that you want to use to get a better approximation.

Actually, the method is exact for every polynom up to degree 5 with n=1.

For other functions, using a higher value of n can return a better value of the integral.

Examples:

- **>RPN $a0 1 $b! integ 1 [x 5 xy].**

Computes $\int_0^1 x^5\,dx$, the value returned is exact.

- **>RPN $a0 1 $b! integ 1 [x dup ln *] .**

Computes $\int_0^1 x \times \ln(x)\,dx$, the value returned should be -0,25.

So you can consider using a higher number of intervals, let's say 100:

>RPN $a0 1 $b! integ 100 [x dup ln *] .

- **>RPN $a0 solve 5 [x $b! integ 100 [y dup ln *]] .**

Tries to find the value x that makes $\int_0^x y \times \ln(y)\,dy = 0$, starting with x=5.

The value is $x = \sqrt{e}$.

Diff+ and Diff-

The general form is:

>RPN diff+ x0 [function]

It returns an approximation of the derivative of f for x=x0, f ' (x0).

Use **diff+** to get the right derivative and **diff-** to get the left derivative.

Examples:

>RPN diff+ 5 [x 1 + sqr 1/x] .

Using $f(x) = \frac{1}{\sqrt{x+1}}$, it returns f ' (5).

>RPN diff+ 0 [x abs] . diff- 0 [x abs] .

Using $f(x) = |x|$, this shows the differences of f ' (0) from right to left.

If you **solve for f '(x) = 0**, you can get the **maximum or minimum** points of a function.

>RPN solve 0.5 [diff+ x [y dup ln *]] .

Use the fiunction $f(y)=y\times\ln(y)$ and solves f '(y) = 0 starting with x=0,5.

The aswer is $x=\frac{1}{e}$.

Note : both solve and diff+/diff- return approximations of the real solution. So the value returned when you search a maximum is an approximation of an approximation.

>RPN 10 [diff+ i [x cos exp] .]

Using $f(x)=e^{\cos(x)}$, it returns a table of values from f '(10) down to f '(1).

Suppose you want a table with a step of 0,1 from 2 to 3, you'll have to use another variable.

>RPN 2 $z! 11 [diff+ z [x cos exp] . 0,1 $z+]

z is initialized with value of 2.

Then a loop with 11 iterations is started.

F '(z) is displayed.

Finally, z is incremented by 0,1.

So you get a table of f '(2), f' '(2,1) ... up to f '(3) .

Settings

As seen with trigonometric functions, the calculator can be set to work with degrees or radians.

`deg` set the angle unit to degree

`rad` set the angle unit to radian (default)

If you're fed up with the status displayed at every call, you can use this:

`s+` set the silent mode

`s-` set the verbose mode (default)

Another setting is the timer information. Upon exit, RPN can display the duration of the calculus.

`t+` display time upon exit

`t-` don't display time upon exit (default)

Note: *even in silent mode (s+) you can still get the time (t+), they are independent.*

You can select a smart display of numbers or force the scientific display:

`sci+` set the display to scientific $mantissa \times 10^{exp}$

`sci-` back to smart display : unused zeros are removed, scientific display only when needed. (default)

You can set the max number of decimals displayed in both previous modes:

`dec` pop a value on the stack and set accordingly the number of decimals.
(default is **`6 dec`**)

`help` display useless help (try it).

File **RPNsaved.mem**.

All of those settings, the contents of the stack and of the 26 varaibles are saved when RPN quits. A file RPNsaved.mem is created. This same file is opened when you run again RPN so your calculator is restored is the state you left it. (To restore to defaul, simply delete the file).

The status is displayed when RPN starts is a summmary line.