A Guide to rpncalc

Ian Duncan

January 24, 2017

Contents

0.1 Introduction to rpncalc

Rpncalc is a reverse Polish notation calculator that supports a variety of pre-calculus, calculus, and statistics operations. Rpncalc uses currently tied to a shell-like interactive interface, but is evolving to become heavily scriptable. Rpncalc's functionality can be easily extended through C-language plugins that are compiled as shared object libraries and loaded at runtime.

0.2 Installing rpncalc

Rpncalc is currently distributed in source code form, but building it is easy. Rpncalc relies only on the C standard library, as well as the `dlfcn` set of functions to provide support for plugins.

If you are building rpncalc on a non-UNIX system, you will have to find a library that provides these functions in order to build rpncalc.

Once you have satisfied all of rpncalc's build dependencies, you can build and install it by running the following commands:

```
$ make
$ sudo make install
```

After building and installing rpncalc, you can run it by invoking the `rpncalc` command.

0.3 Introduction to Reverse Polish Notation

Unlike many calculators, rpncalc makes use of reverse Polish notation. This means that all operations are specified after their operands. For example, to add 4 to 5 in reverse Polish notation, you would write `4 5 +`.

Reverse Polish Notation is based on the idea of a stack. A stack is a kind of data storage structure that only supports two operations: push and pop. To push an item onto the stack means to place an item at the top of the stack. To pop an item off the stack means to retrieve and remove the top item from the stack.

Rpncalc makes use of a stack to perform reverse Polish notation calculations. In order to better understand this, let's take a look at how rpncalc would evaluate the expression `4 5 + 3 *`.

The first item rpncalc sees is the number `4`. This number is pushed to the stack, which you can see below.

```
1: 4
```

Next the `5` is pushed to the stack.

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

Next comes the `+` operator, used to add two numbers. Two numbers are popped from the stack, added, and the result is pushed back onto the stack.

```
1: 9
```

The number `3` is then pushed to the stack.

```
1: 3
2: 9
```

Finally, the two numbers are multiplied.

```
1: 27
```

As you can see, the reverse Polish notation expression `4 5 + 3 *` is the equivalent of the infix notation expression `(4 + 5) * 3`. Don't worry if this seems a bit confusing. As you use rpncalc, you will find that reverse Polish notation is just as valid and easy-to-use as infix notation. You may even begin to think that reverse Polish notation is superior for performing calculations electronically.

0.4 Getting Started with rpncalc

Go ahead and start rpncalc by running the command `rpncalc`. You should be greeted by the following prompt.

```
rpnCalc

Initialized stack with 1000 maximum entries.
Initialized stat stack with 1000 maximum entries.
Loaded 0 plugins.


>
```

The presence of the `>` prompt at the bottom means that rpncalc is ready to accept your input. After you are done composing each line of input, you should press the return key to tell rpncalc to evaluate your input.

Save for a few exceptions, after evaluation the value of the topmost item in the stack is displayed.

```
rpnCalc

Initialized stack with 1000 maximum entries.
Initialized stat stack with 1000 maximum entries.
Loaded 0 plugins.

> 4
4.000000
> 5
5.000000
> +
9.000000
> 3
3.000000
> *
27.000000
> 4 5 + 3 *
27.000000
>
```

You can always view the contents of the stack at any time by running the `inspect` command.

```
> 4 5
5.000000
> inspect
1: 5.000000
2: 4.000000
>
```

0.4.1 Basic Math Operations

As you have seen above, the operators `+` and `*` are used for addition and multiplication, respectively.

```
> 36 3 +
39.000000
> 3 4 *
12.000000
>
```

The operators `-` and `\`, as you might expect, are used for subtraction and division, respectively.

```
> 36 3 -
33.000000
> 12 4 /
```

```
3.000000
>
```

0.4.2 Roots, Exponents, and Logarithms

The `sq` command can be used to take the square root of a number.

```
> 16 sq
4.000000
>
```

The `rt` command can be used to take any root of a number.

```
> 27 3 rt
3.000000
>
```

The `^` operator can be used to raise numbers to a power.

```
> 4 2 ^
16.000000
>
```

It's also easy to raise numbers to fractional powers.

```
> 4 1 2 / ^
2.000000
>
```

In order to help better understand how this works, let's break this down.

```
> 4
4.000000
> 1 2
2.000000
> inspect
1: 2.000000
2: 1.000000
3: 4.000000
> /
0.500000
> inspect
1: 0.500000
2: 4.000000
> ^
2.000000
>
```

You can use the `log` command to take the base-10 logarithm of a number.

```
> 1000 log
3.000000
>
```

You can use the `ln` command to take the natural logarithm of a number. Note that rpncalc provides `e` as a constant as well.

```
> e ln
1.000000
> e 2 ^ ln
2.000000
>
```

You can use the `log2` command to take the base-2 logarithm of a number.

```
> 512 log2
9.000000
>
```