CS121 - Computer Science II
Lab Assignment #4
Fall 2011

The purpose of this exercise is to give you experience with stacks - both in operation and implementation.

The original HP calculators (including my trusty old HP15!) used a notation called "Reverse Polish Notation" (RPN). While it takes some getting used to, it is actually easier to use than a "standard" calculator, especially for long scientific equations. And, RPN is quite easy to implement using a stack. For the record, a standard calculator uses *infix* notation, whereas HP calculators use *postfix* notation.

The rules for implementing RPN is quite simple - if the next input is a number it is pushed on the stack. If the next input is an operator, then the appropriate number of values are popped off the stack (some operators are unary, meaning they only require a single operand, while others are binary, needing two operands), the operation specified by the operator is performed, and the result is pushed back on the stack. Parentheses are not needed with RPN.

You are to implement an RPN calculator. Your program should accept input from the keyboard - this input should consist of either numbers or operators. Multiple items can be entered on a single line, separated by white space. The value on the top of the stack should be displayed at all times as a prompt to the user. For example, to compute the value $(3.2 + 2.4)$, the following sequence (including the starting dialogue from the program) might happen:

```
$ ./rpn
RPN Calculator
RPN (empty) >  3.2 2.4 +
RPN 5.6 >
```

Then to compute the value $(1.5 * (2.4 + 3))$:

```
RPN 5.6 > 1.5 2.4 3 + *
RPN 8.1 >
```

You could also split up the above input into several lines:

```
RPN 5.6 > 1.5
RPN 1.5 > 2.4 3 +
RPN 5.4 > *
RPN 8.1 >
```

And one more (using some of the operators described below): $\sqrt{(5.2-3.1)^2+(7.9-2.4)^2}$:

```
RPN 8.1 > 5.2 3.1 - sq 7.9 2.4 - sq + sqrt
RPN 5.887 > ps
    5.887  <---top
    8.1
    5.6
RPN 5.887 > quit
$
```

For this assignment, your calculator should implement the four basic functions, **+ - * /**, along with the following operators: **sq** (square), **sqrt** (square root), **yx** (computes y^x, where x is the top of stack and y is second from top, **dup** (duplicate the top of stack), **swap** (swaps the top two elements of the stack), **clear** (empties the stack), **ps** (print stack), and **quit**. Calculations should be performed as doubles. Your program should detect any errors in the input or calculation, including trying to pop an empty stack or inputting an illegal number or operator.

You should implement your own stack class using dynamic memory. The stack data itself should be `private`, so all stack manipulations must be performed using methods you write. The stack class you write should include separate class declaration (ie, `.h`) and method implementation (`.c`) files. In other words, your stack class files should be completely separate from the code you write for this particular problem, so that your stack class can be reused without modification in another application. Your class should include constructor and destructor methods.

Your cscheckin submission should be a zip or tar file containing your class files as well as the `main` program and supporting code.