ENGR 200 FALL 2015

Homework 7: Reverse Polish Notation Calculator

EC Nov. 5th at Class time, Due Nov. 10th at 11:59pm POINTS: 50

Objective

Practice using functions, switch statements, arrays, and loops.

How a reverse polish notation (RPN) calculator works

This information is based on the wikipedia description of a RPN calculator, but similar descriptions exist in textbooks and on other web sites. RPN calculator operation is relatively straightforward:

- While there is information to read from the user
  - Read in the information in the form of a string
  - If the string contains a number
    - If the stack is full
      - print "(Error) Stack overflow."
      - continue the loop
    - Add the number onto the top of the stack. This operation is referred to as pushing a number onto the stack.
  - If the string is not a number, check to see if it is a known operator. (+,-,*./,p,l,q)
    - If the operation is 'q' break the loop.
    - If it is a known operator check to make sure there are enough numbers, called operands, on the stack to perform the operation. ([+, -, *, /] takes 2 operands, [p] takes 1 operand, [l, q] require no operands)
      - If there are fewer operands in the stack than required for the operation the program should print "(Error) there are not sufficient values in the expression".
      - continue the loop
    - Remove the operands from the stack
    - Perform the operation, with the values as arguments.

    - Push the result, if any, back onto the stack.
- If the string contains an unknown operator or symbol
    - Print "(Error) Unknown operator"
    - continue the loop

Example

The expression "`5 + ((1 + 2) * 4) - 3`" can be written down like this in RPN:

```
5
1
2
+
4
*
+
3
-
```

The table below describes what should be happening in your program:

Input	Operation	Stack	num_operands	Comment
5	Push operand	5	1	
1	Push operand	5, 1	2	
2	Push operand	5, 1, 2	3	
+	Add	5, 3	2	Pop two values $(1,\ 2)$ and push result (3)
4	Push operand	5, 3, 4	3	
*	Multiply	5, 12	2	Pop two values $(3,\ 4)$ and push result (12)
+	Add	17	1	Pop two values $(5,\ 12)$ and push result (17)
3	Push operand	17, 3	2	
-	Subtract	14	1	Pop two values $(17,\ 3)$ and push result (14)

The stack is an array. The num_operands is used to keep track of how many numbers (operands) are in the stack at any time. One important thing to realize is that to remove a number from the stack all one has to do is subtract one from the num_operands. The value does not have to be "cleared" or removed in any other way. The num_operands represents how many valid operands there are on the stack. The contents of the rest of the stack do not matter.

A RPN calculator can be found at `http://www.arachnoid.com/lutusp/calculator.html`

I have provided an executable file. It should work on most windows computers. If it does not work, you are welcome to look at any number of online RPN calculators.

Assignment

1. Get the partial source file included with this document. The provided code will read data from the user and determine if it is a number or an operator. Add the appropriate code where indicated. The code added to the program must process numbers and operators from the list above appropriately. **It should not be possible to crash the calculator.** That requires that you make sure there are enough operands on the stack and the operation is defined. Also, the user must not be able to preform operations that generate errors such as dividing by zero or overflowing the stack. If the user attempts a illegal action, the command should be ignored and an error message describing the error should be printed to the screen.

2. Each item in the table below, with the exception of quit, should be a function. The only two functions that should modify the stack are pop and push. The list function will access the stack to print the values, but not change the number of items in the stack or the stack itself.

entry	command	operands reqired	description
+	add	2	pops two numbers, adds them, and push result
-	subtract	2	pops two numbers, subtracts the last from the first, and push result
*	multiply	2	pops two numbers, multiplies them, and push result
/	divide	2	pops two numbers, divides second number popped by first, and push result
p	pop	1	removes a number from the stack
l	list	0	prints all the numbers in the stack
q	quit	0	stops the program
0–9	push	0	adds the number to the stack

Start by implementing three functions: push, pop, and list. Push and pop are the building blocks for the remaining functions and list will tell you what is in the stack. The pop function should remove a value from the stack and return the value. It is important to note that removing a number from the stack only requires a change in the stack size. The push function should add a value to the stack and does not need to return anything.

An example function:

```
void divide()
{
   double result;
```

```
   /*Check to ensure that there are two numbers on the stack and
     we are not dividing by zero*/
   if(idx<2)
   {
      printf("not enough operands on the stack\n");
      return;
   } else if ( fabs(stack[idx-1]) < 1e-50) {
      printf("will not divide by zero\n");
      return;
   }
   /*do the division.*/
   result = 1/pop()*pop();
   /*push the result back onto the stack*/
   push(result);

   return;
}
```

3. Have two people try to crash your program. Make sure to note if they could. Also place there names in the header.

Extra Credit

This will not be accepted unless the assignment is complete and works.

3 points Make it possible to use sine, cosine, and tangent.

3 points Extend the program to do arc sine, arc cosine, and arc tangent.

How to complete the assignment:

1. Begin by creating the code to push numbers onto the stack. In simple terms create the code needed to add numbers to the stack.
2. To make sure that numbers are being added to the stack correctly, implement the code that prints all the numbers on the stack.
3. Now implement the function pop. It should remove one number from the stack and return its value. To remove a number from the stack, you simply reduce the count of how many numbers are on the stack. There is no need to manipulate the stack array itself.
4. Use push and pop to create the operators. For example add is simply push(pop()+pop()); .