Programming Assignment II
COP3502H

Due: 2/7/07 11:59pm to Web CT

The CEO of the Computer Sciences Corporation, Dr. Link Queue has decided that the first product in the company's arsenal of applications will be a text-based numerical calculator. In this assignment, you will develop a simple calculator that takes mathematical expressions from the user and outputs the results. Your calculator will utilize stacks and postfix notation to perform the computations.

Requirements

Users of your calculator should be able to enter a mathematical numerical expression as input and the program should print out the result as output. You can do this in one of two ways. The first way is to make the calculator a command line program. Thus, the expression would be passed to the program when the executable is run. For example, if your program is named 'calc', it might look something like this:

```
>calc 2+5+7
>Result: 14
```

The second way is to make the calculator more interactive. You would run the 'calc' program and see something like this:

```
>calc

Welcome to the Calculator (type 'Q' to quit)

Enter Expression:  2+2
Answer: 4

Enter Expression 5*8
Answer: 40

Enter Expression: Q

>
```

You have the option of choosing either method.

As for expressions, we will assume that all expressions are mathematically valid. Thus, you will not have to worry if someone writes an expression such as '2+5(5'. You can do error checking of valid expressions for extra credit.

Your calculator must support addition, subtraction, multiplication, and division as well as parentheses. Your operands are limited to the integers, but the output of your calculator may be a real number.

Extra Credit
If you finish these requirements, for extra credit you can add the following

1. The exponentiation operator – '^'
2. The unary minus operator to allow for negative numbers
3. Error checking of invalid expressions
4. The trigonometric functions sin, cos, and tan.

Approach

There are basically 4 parts to this assignment.

1. Create a stack library.
2. Be able to read in the mathematical expression and break it up into pieces.
3. Using the stack library, convert the expression to postfix notation.
4. Using the stack library, evaluate the postfix expression and print out the result.

In part one, you will need to create a stack library. You should use the linked list routines from your last assignment to build the library. The structure you should use is

```
struct stackNode {
     char token[10];
     struct stackNode *next;
};
```

You will need to implement the push, pop, isEmpty, and returnTop functions we discussed in class.

In part two, you will need to be able to read in an expression for further processing. You can use a character array so you can examine individual symbols.

In part three, you will have to write a function that takes the expression and converts it to postfix notation. You should use the pseudocode we discussed in class as a template for your implementation, making use of the functions you wrote in part one. Also, make sure you print out the postfix expression for testing purposes.

Finally, in part four, you will have to write an evaluator function that takes the postfix expression as input and calculates its value. Use the stack library and the procedure we discussed in class for your implementation. The functions found in the header file, "ctype.h" may be useful for determining whether a character is a digit, punctuation, etc…

You should end up having to create 5 new files, stack.h, stack.c, calculator.h, calculator.c, and main.c.

Documentation

You must document your routines. Each function should be commented with what the function does, its input parameters, and what it returns. Comments are very important and will account for 20% of your grade.

Testing

You must thoroughly test you're your calculator to show that it works. You will have to create some test expressions (about 10 or so) that show the requirements have been met and print out the results to a file. Make sure you print out both the input expression and the result to your test data file.

Deliverables

You must submit all source files in addition to your test files to WebCT by the due date. Also, include a README file explaining how you tested your code and what problems you encountered. Note the code must be able to be compiled and executed on the Olympus unix system.

Grading

Grading will be based on the following:

80% correct functionality
20% documentation