COP 3503 Programming Fundamentals for CIS Majors II, Fall 2014

Programming Assignment 3

Out: Oct. 29 (Wednesday), 2014

Due: 5pm on Nov. 10 (Monday), 2014

Problem Description

In this exercise, you should implement a simple calculator using C++. The operations that the calculator is expected to take care of ranges from simple operations, e.g., addition and multiplications, to more complicated operations, e.g., sinus and logarithm.

Your code should input a string from user, which contains the expression he/she wants to compute. Then, your code should parse the input, detect the operators and their argument(s), and return the result. For simplicity, you can assume that the input expression is fully parenthesized (see below for more details). It is expected that you use the stack data structure to pass arguments and compute the result.

With this assignment, you will get practice with the stack implementation which is one of the most widely used data structures. Besides, you will be familiar with string processing and input parsing, which are of crucial importance in most C++ projects.

Details

1. (Data Structures:) You need to implement a stack data structure to keep track of the argument(s) for each operator and do the computations, whenever it is necessary. Adding elements to the stack (push) and removing objects from it (pop) are two essential methods that must be implemented. You can use any data structure to implement the stack, e.g., arrays, linked-lists, etc.

2. (Input Format:) The input is a string from the user containing the mathematical expression to be computed. You can assume that the expression is fully parenthesized, e.g., "2 + 4*6" will be represented by "(2 + (4*6))" by the user. But, there might be any number of spaces between two characters, e.g., " (2 + 4)" must be treated the same way as "(2+4)". Besides, your code is responsible to check if the parenthesis are balanced, and should return error, otherwise, e.g., "(2 + 4" is not a valid expression, hence can not be computed.

3. (Mathematical Operators:) To be consistent, the following is a list of the operators that your code should take care of, along with their corresponding symbols:

   - Addition (+), Example: "(2 + 3)".

- Subtraction ($-$), Example: "$(2-3)$".
- Multiplication ($*$), Example: "$(2*3)$".
- Division ($/$), Example: "$(2/3)$".
- Sine (sin), Example: "$\sin(30)$".
- Cosine (cos), Example: "$\cos(60)$".
- Logarithm (log), which is based 2, Example: "$\log(8)$".
- Power ($\wedge$), Example: "$2 \wedge 3$".
- Square Root ($sqrt$), Example: "$sqrt(16)$".

4. (Algorithms:) Once the input expression is given, your calculator should decide which arguments should be inserted to the stack, and when the (sub)result needs to be computed. The easiest way is to compute (sub)results, whenever a left parenthesis, ")", is reached. Of course, this is based on the assumption that the left and right parenthesis are well-balanced.

5. (Hint:) Once an operation is completed, its argument(s) are removed from the stack, and the result is replaced at the top. This way the chain computations are automatically taken care of. For instance, for the input "$(2 + (3 * 4))$", the following items are added to the stack, in order: "2", " + ", "3", " * " and "4". Once the first left parenthesis, ")", is reached, the operator and its arguments from the top of stack are removed and the operation is computed, i.e., $3 * 4 = 12$.

   Then, the result is added to the stack, i.e., the stack now looks like: "2", " + ", "12". Finally, the second left parenthesis is seen, and again the operation seen at the top of stack is computed, i.e., $2 + 12 = 14$. Therefore, the final result is 14 which is inserted to the stack.

Example Run

Try to keep your output as close to the given format as possible:

```
> ./pa3.out
Please enter the expression:

((6 + (4 * 12))/6)

The result is: 9
-----------------------------------------
> ./pa3.out
```

```
Please enter the expression:

((2*sin(30)) + cos(60))

The result is: 1.5
-----------------------------------------
> ./pa3.out
Please enter the expression:

(sqrt(16) + (3^2))

The result is: 13
-----------------------------------------
> ./pa3.out
Please enter the expression:

((2*sin(30) + (2-4))

Error: Unbalanced parenthesis!!!
-----------------------------------------
> ./pa3.out
Please enter the expression:

(7*8)/(1-sin(90))

Error: Division by zero!!!
```

Hints

1. You should get your stack data structure working well before implementing the calculator.

2. The string processing to parse the input is an essential part of this assignment. You should make sure that you parse the input correctly, and you take care of all edge cases, e.g., more than one spaces between characters, no spaces, etc.

3. You should take care of bad user inputs. For instance the input expression may not have equal number of left or right parenthesis or they may not be balanced, e.g., ")2+4(". Other examples are division by zero and the square root of a negative number.

Submission Guide-lines

1. You must finish this assignment with your individual effort. Your C++ source code file MUST be named as "pa3.cpp" and C++ header/class file (if there is any) should be named as "pa3.h".

2. Please test your program via g++ compiler (i.e., g++ -Wall) on the CISE machines to make sure it runs correctly.

3. Please upload the source code file(s) to SAKAI system as the attachment(s). Please submit the source code file(s) ONLY. NO need to compress the source code file(s) if the size is small.

4. Make sure you submit your assignment BEFORE the deadline. Late submission will NOT be graded !!

Grading Criteria

1. Successful Compilation (30%): Your source code should be able to compile using "g++ -Wall" command without any error or warning. The output should be a valid executable. See lab tutorial for the commends. Please note that we will be using g++ compiler on Linux to grade your programs. If you are using other compilers or IDE (e.g., Visual C++), it is recommended that you test the source codes with g++ before the SAKAI submission (i.e., make sure there is no warning).

2. Program Correctness (40%): The executable should be able to run correctly by giving out the required output.

3. Programming Style (30%): Good coding style is a key to efficient programming. We encourage you to write clear and readable codes. You should adopt a sensible set of coding conventions, including proper indentation, necessary comments and more. Here are some guidelines of good programming style: http://en.wikipedia.org/wiki/Programming_style

Final Notes

1. AGAIN, remember to start the programming assignments as soon as possible. Unlike the conventional assignments, programming assignments sometimes take un-predictable amount of time to finish. Thus, have the code running first, then polish it later with the extra time before the deadline.

2. Remember you should always write your own code and never copy-and-paste from

other students' work or other sources. There are indeed many tools (like Stanford Moss) to detect the code similarity.

3. Programming assignments are usually designed by TAs. If you have any question or concern, please feel free to contact TAs. Our goal is to let you experience the fundamentals of computer science. If you like the programming experience, you are one of us !!