Data Structures & Algorithms (Course & Lab) – Fall 2015

(BS-SE-F14 Morning & Afternoon)

PRACTICE Assignment # 1

Submission Deadline: None!!

Instructions

- **Although this is just a practice assignment, but still you need to work on it individually. Absolutely NO collaboration or discussion is allowed.**
- **You are not required to submit this assignment, but I will assume in quizzes and exams that you have completed this assignment.**

Task # 1

In this task you are going to write a program to implement the algorithm (that we have seen in class) for evaluating postfix expressions which involves the four basic arithmetic operators: **+**, **–**, *****, and **/**, and the unary minus operator (**~** will be used to indicate the unary minus operator, in order to distinguish it from the subtraction operator).

Since, this algorithm uses a stack; therefore, firstly you will have to implement a **`Stack`** class for storing **`double`** values (in class we have already implemented a stack of integers).

Once, you have implemented the **`Stack`** class, you need to get the postfix expression from the user and store it in a **`char`** array. After that you would implement the following function, which takes the postfix expression present in the **`char`** array (**`input`**) and evaluates it using the algorithm that we discussed in class:

```
void evaluatePostfixExpression ( char* input )
```

To trace the actions of the algorithm, this function should also display each token as it is encountered, and the actions (stack operations) that it is performing. For example, the output of your program should resemble the following (text shown in **`Red`** is entered by the user):

```
Enter the postfix expression: 12 3 ~ 1 - / 4.6 ~ *

Token = 12       Push 12
Token = 3        Push 3
Token = ~        Pop 3        Push -3
Token = 1        Push 1
Token = -        Pop 1        Pop -3       Push -4
Token = /        Pop -4       Pop 12       Push -3
Token = 4.6      Push 4.6
Token = ~        Pop 4.6      Push -4.6
Token = *        Pop -4.6     Pop -3       Push 13.8
End of input

Answer = 13.8
```

Note that in the above example, input provided by the user actually corresponds to the following infix expression: **`12 / (-3 – 1) * -4.6`**

Note the following important points regarding the input/output format:

- Your program should strictly follow the input/output format shown in the example above.
- As you can see from the above example, the numbers in the input can have more than one digit, and there might also be some floating point values in the input.
- You can assume that there will be **a space** after each operand or operator given in the input (except the last operator).
- You can assume that there are at most 200 characters (including spaces) in the input string entered by the user.
- You can also assume that the expression entered by the user is valid.

Hint: You might find following library functions helpful in your implementation:

- **`strtok`** in the header file **`<cstring>`**
- **`atof`** in the header file **`<cstdlib>`**
- **`isdigit`** in the header file **`<cctype>`**

Task # 2

Modify your program (from Task # 1) to handle the following cases in which the input is invalid. Your program should display an appropriate error message when one of the following conditions arises:

- Too few operands. For example: **`1 3 + *`**
- Too few operators. For example: **`3 4 5 +`**
- Invalid operator. For example: **`4 6 + 8 &`**

Task # 3

Now, modify your implementation to take input from a **text file** in which several postfix expressions are given (one on each line). Your program should evaluate these expressions, and put these expressions and their resulting values in an **output text file**. If some expressions are invalid, your program should store an appropriate error message with that expression.

☺ GOOD LUCK! ☺