CS 201, Spring 2022

Homework Assignment 4
Due: 23:59, May 13, 2022

Note: In this homework, you will create several global functions that utilize Stacks to solve various problems. For understanding the declaration and scope use of global functions, you can refer to this small tutorial.

In this homework assignment, you will provide solutions to several smaller Stack related problems. You will declare these functions as global functions in 'Solutions.h' and implement them in 'Solutions.cpp'

1. The first problem that you will solve is called the Mancunian And Fantabulous Pairs Problem. An array is called Fantabulous if (i) it contains at least two elements, and (i) if the second largest element in the array is to the left of the first largest element. So for example, arrays [1,4,6,3,12] and [2,1,6,9,13,2] are Fantabulous while arrays [13,3,4,9,0] and [1,3,4,12,22,19] are not. For this problem, you have to implement a global function, that when given a length of the array, and its contents finds the number of subarrays of that array that are Fantabulous.

   For example, candidate subarrays of the list [4,1,6,3,12] are {[4,1,6,3,12], [4,1,6,3], [1,6,3,12], [4,1,6], [1,6,3], [6,3,12], [4,1], [1,6], [6,3], [3,12]} of which only {[4,1,6,3,12], [4,1,6,3], [1,6,3,12], [4,1,6], [6,3,12], [3,12]} are Fantabulous. Therefore, the output for this input should be 6. Notice that candidate subarrays are not the same as a subset of a list. Meaning that you must preserve sequence and cannot include subsets such as [4,6,12] and etc. as candidates. You may assume that all elements in the list are unique.

2. The second problem that you are expected to solve is the merging overlapping intervals. That is, for a set of intervals [1:5], [2:7], [8:10], [11:16], [14:18] your output should be [1:7], [8:10], [11:18]. You will pass the intervals to the function as a single array which will be formatted as follows:

   For example, for a list of intervals [1:5], [2:7], [8:10], [11:16], [14:18] input will be passed as [1,5,2,7,8,10,11,16,14,18]. Here you can assume that the size of the array will always be even.

3. The third problem that you will solve is the conversion of an infix expression to a prefix expression. For example, for an input 3+5*6/(7-2) your implementation would output +3*5/6-72. You can assume that all input expressions are legitimate infix expressions. And that all operands are single-digit integers.

4. The fourth problem that you will solve using stacks is a simple calculator. Here the input to your program will be a single string mathematical expression of prefix format such as +3*5/6-72. And your function needs to print the output of that equation.

Your implementation MUST include the following global functions whose details are given below. Note that we will use these function prototypes to test your programs. Thus, you are not allowed to change the prototypes of these functions. On the other hand, you are free to add as many additional functions as you would like to your implementation

1.

```
void fantabulous(string list, int length)
```

The input to the program will be in the form of a string. So for example, for a list [1,4,6,3,12], the input will be "1,4,6,3,12". And the length variable will be 5. You can assume that the numbers will always be separated by commas and that the 'length' variable will always show the correct length of the list. As output, your program should print the number of fantabulous subarrays of the input list. (Please see the code given below and its output for the full signature of the method and format of the output.)

2.

```
void subset(string intervals, int length)
```

As with the previous function, the input to the program will be in the form of a string. So for example, for a list of intervals [1:5], [2:7], [8:10], [11:16], [14:18] the input will be passed as "1,5,2,7,8,10,11,16,14,18". Therefore you can assume that the length of the list will always be even and that all the integers will be separated by a comma. You can also assume that all intervals will be valid. So no intervals like [9, 4] and etc. You can also assume that the 'length' variable will always have the correct value. As the output, you should print the intervals in order. (Please see the code given below and its output for the full signature of the method and format of the output.)

3.

```
string infix2prefix(string expression, int length)
```

Your program should take an infix expression string and return a prefix expression string. You may assume that the input is always balanced and valid infix expression. (Please see the code given below and its output for the full signature of the method and format of the output.)

4.

```
void evaluatePrefix(string expression)
```

Your methods should take an infix expression and print the result. You may assume that the input expression is always a valid prefix expression. (Please see the code given below and its output for the full signature of the method and format of the output.)

You MUST use the ADT stack in your implementation. Thus, you will define and implement a class for the ADT stack and use it in your functions. Note that you are allowed to use the C++ codes that are given in your textbook but are not allowed to use any other stack implementation. For example, you cannot use the stack class defined in another book or on a website. Similarly, you are not allowed to use any stack-related functions (any other data structure-related functions or classes) in the C++ standard template library (STL). If you think that different problems in this homework require different stacks (string stack or int stack or etc.) then you may implement several stacks. But include all of them in your submission.

Below is an example of the main.cpp file that we will use to test your program.
Note: It is crucial that you do not change the 'include' statements in the main code when you create your own. We will use a different main.cpp file to test your program, but it will import the same exact libraries like this one. So the code you submit needs to work with that configuration. Whatever additional libraries you need to include, do so outside of main.cpp file.

```
#include <iostream>
using namespace std;
#include "Solutions.h"


int main() {

	// Testing add parking lot
	cout << endl;
	cout << "Testing Fantabulous" << endl;
	cout << endl;

	fantabulous("1,7,4,12,14",5);
	fantabulous("1,15,2,5,12,14",6);
	fantabulous("3,4,12,18,21,100,4,2,1,99",10);

	cout << endl;
	cout << "Testing Subset Problem" << endl;
	cout << endl;

	subset("1,7,10,12,11,17,15,16,19,22",10);
	subset("3,6,4,12,15,19,21,25,21,44,45,47,90,92",14);
	subset("2,4,3,10,12,18,19,20",8);


	cout << endl;
	cout << "Testing Infix to Prefix" << endl;
	cout << endl;

	cout << infix2prefix("(7-2)*3-6") << endl;
	cout << infix2prefix("9/2-(6-4)*2+9") << endl;
	cout << infix2prefix("9/3+6/2-(4-2)") << endl;
```

```
        cout << endl;
        cout << "Testing evaluate Prefix" << endl;
        cout << endl;

        evaluatePrefix("-*-7236");
        evaluatePrefix("-/92+*-6429");
        evaluatePrefix("+/93-/62-42");

        return 0;
}
```

Sample Output:

```
Testing Fantabulous

The number of Fantabulous subsequences: 8
The number of Fantabulous subsequences: 6
The number of Fantabulous subsequences: 33

Testing Subset Problem
Joined subset: (1:7), (10:17), (19:22)
Joined subset: (2:12), (15:19), (21:44), (45:47), (90:92)
Joined subset: (2:10), (12:18), (19:20)

Testing Infix to Prefix

Preix form of (7-2)*3-6 is -*-7236
Preix form of 9/2-(6-4)*2+9 is -/92+*-6429
Preix form of 9/3+6/2-(4-2) is +/93-/62-42

Testing evaluate Prefix

Result of -*-7236 is: 9
Result of -/92+*-6429 is: 9.5
Result of +/93-/62-42 is: 4
```

NOTES ABOUT IMPLEMENTATION AND SUBMISSION:

This assignment is due by 23:59 on May 13, 2022. **This homework will be graded by your TA Aydamir Mirzayev (aydamir.mirzayev[at]bilkent.edu.tr). Please direct all your homework-related questions to him.**

1. Your code must not have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct. To detect memory leaks, you may want to use Valgrind which is available at http://valgrind.org.

2. Otherwise stated in the description, you may assume that the inputs for the functions (e.g., the chemical location) are always valid so that you do not need to make any input checks.

3. Make sure that each file that you submit (each and every file in the archive) contains your name and student number at the top as comments.

4. In this assignment, you must have a separate interface and implementation files (i.e., separate .h and .cpp files) for your class. The file names should be "Solutions.h" and "Solutions.cpp". You should also submit other .h and .cpp files if you implement additional classes. We will test your implementation by writing our own main function. Thus, you should not submit any file that contains the main function. Although you are not going to submit it, we recommend you write your own driver file to test each of your functions. However, you SHOULD NOT submit this test code (we will use our own test code)

5. You should put your "Solutions.h" and "Solutions.cpp" (and additional .h and .cpp files if you implement additional classes) into a folder and zip the folder (in this zip file, there should not be any file containing the main function). The name of this zip file should conform to the following name convention: secX-Firstname-Lastname-StudentID.zip where X is your section number. The submissions that do not obey these rules will not be graded.

6. You are free to write your programs in any environment (you may use Linux, Mac OS X, or Windows). On the other hand, we will test your programs on "dijkstra.ug.bcc.bilkent.edu.tr" and we will expect your programs to compile and run on the Dijkstra machine. If we could not get your program to work properly on the Dijkstra machine, you would lose a considerable amount of points. Therefore, we recommend you make sure that your program compiles and properly works on "dijkstra.ug.bcc.bilkent.edu.tr" before submitting your assignment.