CS 201, Fall 2021
Homework Assignment 4

Due: 23:59, January 3, 2022

In this homework assignment, you are supposed to implement a program for manipulating algebraic expressions of infix, prefix, and postfix forms. Your implementation should enable the user to convert an expression of one form to another as well as to evaluate the expression.

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 into your implementation.

```
// It converts an infix expression exp to its equivalent prefix form.
string infix2prefix( const string exp );

// It converts an infix expression exp to its equivalent postfix form.
string infix2postfix( const string exp );

// It converts a prefix expression exp to its equivalent infix form.
string prefix2infix( const string exp );

// It converts a prefix expression exp to its equivalent postfix form.
string prefix2postfix( const string exp );

// It converts a postfix expression exp to its equivalent infix form.
string postfix2infix( const string exp );

// It converts a postfix expression exp to its equivalent prefix form.
string postfix2prefix( const string exp );

//It evaluates an infix expression.
double evaluateInfix( const string exp );

//It evaluates a prefix expression.
double evaluatePrefix( const string exp );

//It evaluates a postfix expression.
double evaluatePostfix( const string exp );
```

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 web site. 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).

In your algorithms, you may have the following assumptions:

- The input strings are syntactically correct expressions. For example, you may assume that the string is a valid infix expression when the `infix2prefix` function is called or you may assume that the string is a valid prefix expression when the `evaluatePrefix` function is called.
- An input string can include only digits, binary operators and space. You may assume that only the binary operators *, /, +, and - are allowed (no unary and exponentiation operators are allowed). Note that operands can contain multiple digits. Operands and operators are separated with empty space.

- In case of infix expressions, the string can also include parentheses in addition to digits, binary operators and space.

Here is an example test program that uses these functions. We will use a similar (but different) program to test your solution so make sure that you test your solutions thoroughly.

```
#include "AlgebraicExpression.h"

int main() {
    cout << infix2prefix("( 12 + 5 ) - 20 * 4") << endl;
    cout << infix2postfix("( 12 + 5 ) - 20 * 4") << endl;
    cout << evaluateInfix("( 12 + 5 ) - 20 * 4") << endl;
    cout << prefix2infix("* + * 100 2 4 - 12 4") << endl;
    cout << prefix2postfix("* + * 100 2 4 - 12 4") << endl;
    cout << evaluatePrefix("* + * 100 2 4 - 12 4") << endl;
    cout << postfix2infix("100 12 2 - 8 * + 4 /") << endl;
    cout << postfix2prefix("100 12 2 - 8 * + 4 /") << endl;
    cout << evaluatePostfix("100 12 2 - 8 * + 4 /") << endl;
    return 0;
}
```

Notes:

- This assignment is due by 23:59 on Monday, January 3, 2022. It will be graded by your TA Hakan Turkmenoglu (`h.turkmenoglu@bilkent.edu.tr`), you may ask your homework related questions to him.
- In this assignment, you must have separate interface and implementation files (i.e., separate `.h` and `.cpp` files). We will test your implementation by writing our own driver `.cpp` file which will include your header files. For this reason, your file names MUST BE `Stack.h`, `Stack.cpp`, `AlgebraicExpression.h` and `AlgebraicExpression.cpp`.
- You should put these files into a folder and zip the folder. 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. You must also write your own driver file, which must be different from the one that we gave above, to test each of your functions. **You MUST submit this test code** as well in the archive file.
- Before 23:59 on January 3rd, you need to upload this zipped file to Moodle.

  No hardcopy submission is needed. The standard rules about late homework submissions apply. Please see the course syllabus for further discussion of the late homework policy as well as academic integrity.
- You can reuse the codes and pseudocodes on the lecture slides if you need, but other than that, the code must be your own implementation.
- 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.
- You are free to write your programs in any environment (you may use Linux, Mac OS 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 properly work on the `dijkstra` machine, you would lose a considerable amount of points. Therefore, we recommend you to make sure that your program compiles and properly works on `dijkstra.ug.bcc.bilkent.edu.tr` before submitting your assignment. Note that you are NOT allowed to use `C++11` on the `dijkstra` machine, so do NOT use the `-std` compiler flag as we will test your implementation on the default `C++` version.