CS 201, Fall 2019

Homework 2

DUE: November 1, Friday @ 23:59.

Please check the submission rules towards the end of the document. Points will be deducted in case of a violation of these rules!

Description: In this assignment you will write a C++ program that **evaluates an arithmetic expression** (represented with an **infix** notation), then outputs this expression in **postfix** form and also outputs the result of the calculation. The program will first convert the input infix expression to a postfix expression (using the Stack ADT) and then calculate the result (again, using the **Stack ADT**). The details are provided in the following sections.

1) Input and Expected Output

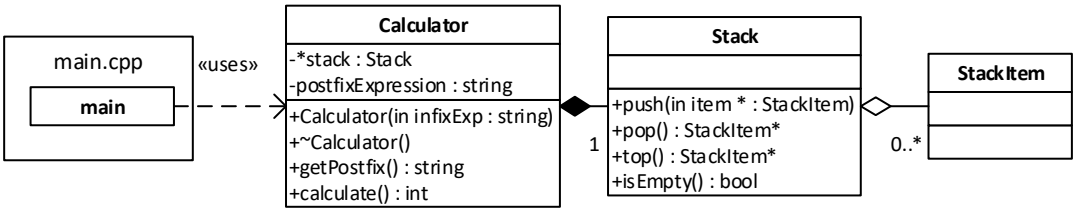
The program will ask for an arithmetic expression in infix form. All the operators and operands of the input expression should be **separated with a space character** and the expression should **end with a ';' character**. The input format is strict to be able to parse the expression easily. The program will first print out the input expression in postfix form, following the same formatting rules. Then, the result of the arithmetic expression will be printed. A sample run of the program can be as follows. More test cases are provided at the end of the document. The parts with **bold face** represent the user input.

```
Enter an arithmetic expression: 4 * 1 + 5 + 6 * 1 ;
Input expression in postfix form: 4 1 * 5 + 6 1 * + ;
The result is: 15
```

The program should support the basic four operators (i.e., **+**, **-**, ***, /) and the use of parenthesis.

2) The Design

In this assignment you will implement a **stack** using a **linked list**. Depending on the design choice, the implementation can be different. However, you should make sure that the standard methods (i.e., push, pop, top, isEmpty) are supported by your stack implementation. In the following, first the overall design of the program is discussed. Then, we will describe the linked list implementation of the stack. The overall design is depicted below.



Hereby, the **main** function consumes an arithmetic infix expression as input (from the console). Then it creates a *Calculator* object to convert the input expression to postfix form, and then to calculate the result. It prints out both the postfix expression and the result of the calculation on the screen. The implementation of the main function (i.e., the *main.cpp* file) *will be provided* to you.

The **Stack** class will implement a standard Stack ADT interface. Your stack implementation should store objects of type **StackItem**. *StackItem* is a class, which can store an operator or an operand of an expression. The implementation of the *StackItem* class (*StackItem.h* and *StackItem.cpp*) *will be provided* to you. The implementation of this class and the *Stack* class is discussed in the following section.

The **Calculator** class has two member attributes; *postfixExpression* of type string that stores an expression in postfix form, and *stack,* a pointer variable of type *Stack*. The class has also two methods; get*Postfix* and *calculate*.

The *constructor* of the *Calculator* class gets an infix expression as an argument. First, it should initialize the *stack* object. Then, it should convert the infix expression to postfix form using this stack. The resulting expression should be assigned to the *postfixExpression* attribute.

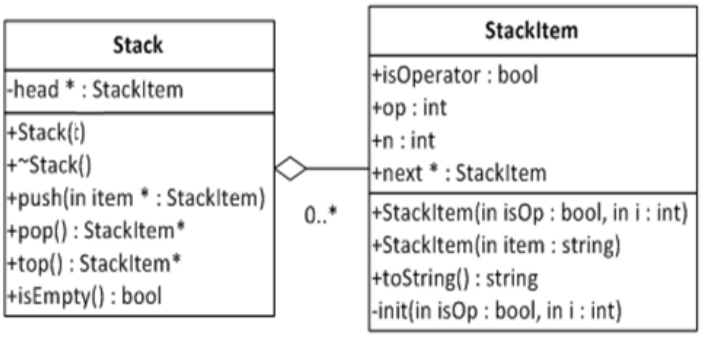
The *calculate* method makes use of *stack* to evaluate the postfix expression and return the result.

The *getPostfix* method just returns *postfixExpression*.

All the memory that is allocated within the methods should be freed before their return. Also, the *destructor* of the *Calculator* class should free all the memory (e.g., *stack* object) that is allocated within its *constructor*. The header file of the Calculator class (*Calculator.h*) *will be provided* to you.

3) The Stack Implementation

The linked list based implementation of the Stack class is depicted below.



In linked list implementation of the Stack, the size does not need to be fixed. Hence, the constructor of the Stack class will not have to take any arguments. The constructor of the *Calculator* class can initialize the *stack* without specifying any predefined size.
The *Stack* class only keeps a pointer to the head node of the linked list. The list is empty when this pointer points to NULL.

The **StackItem** class has four attributes; *isOperator*, *op*, *n* and *next*. A *StackItem* object can either store an operator or an operand. If the object stores an operator, the *isOperator* boolean variable is set to *true* and the *op* attribute is set to an integer constant that represents an operator (e.g., 1 represents a plus operator, "+"). If the object stores an operand, then the *isOperator*boolean variable is set to *false* and the *n* attribute is set to the value of the operand. *next* keeps a pointer to another *StackItem*object.

The *StackItem* class has two constructors. *One of these constructors* takes as arguments the value of the *boolean* attribute, and the value of *op* or *n* depending on the *boolean* attribute. The other *constructor* takes a string (e.g., "+", "34", "(", "-", "5", etc.) as input and parses the string to set the attributes of the class accordingly. The *toString* method returns a string that includes the operator or operand stored in the object. The *init* method is a private method that initializes the member variables. It is reused by the two constructors of the class. The implementation of the *StackItem* class *will be provided* to you.

4) Submission

Youwill submit this homework via the LMS system. You should follow the file-naming conventions and guidelines below.

- You should submit your source files as a **ZIP** archive file (**NOT** RAR or other formats). The name of the file should be in format "**<USER-ID>_hw<HOMEWORK-NR>.zip**". For example, if your username is vy1043, then the name of the submitted file should be "vy1043_hw2.zip". Pay attention that all

the letters are in lower-case. ZIP archive is supposed to contain **just the source files**, no folders are allowed by any means.

- The contents of the ZIP file should be as follows:
  - **main.cpp** (includes the *main* function)
  - **Calculator.h** (Calculator class definition)
  - **Calculator.cpp** (Calculator class implementation)
  - **Stack.h** (Stack class definition)
  - **Stack.cpp** (Stack class implementation)
  - **StackItem.h** (StackItem class definition)
  - **StackItem.cpp** (StackItem class implementation)

- Late submissions and C++ files that do not compile are **not** accepted.

- You can resubmit your homework (until the deadline) if you need to.

- Make sure that your program does **not** include commands specific to a development environment, e.g., `system("pause")` or `#pragmaonce` in Visual Studio.

5) Sample Test Cases

```
Test Case 0;
Input: 4 * 1 + 5 + 6 * 1 ;
Output: 4 1 * 5 + 6 1 * + ;
Result: 15
---------------------------------------------------------------------------------------
Test Case 1;
Input: 4 + 3 * 6 + 5 ;
Output: 4 3 6 * + 5 + ;
Result: 27
---------------------------------------------------------------------------------------
Test Case 2;
Input: ( 4 + 3 ) * ( 6 + 5 ) ;
Output: 4 3 + 6 5 + * ;
Result: 77
---------------------------------------------------------------------------------------
Test Case 3;
Input: 1 + ( ( 4 + ( 4 * 4 ) / ( 4 + 3 + 1 ) - 4 ) + ( 6 + ( 3 * 1 ) ) ) * ( ( ( 1 + 2 ) * 4 ) - 3 ) ;
Output: 1 4 4 4 * 4 3 + 1 + / + 4 - 6 3 1 * + + 1 2 + 4 * 3 - * + ;
Result: 100
---------------------------------------------------------------------------------------
Test Case 4;
Input: 3555 + ( ( 40 + ( 40 * 4 ) / ( 5 + 32 + 3 ) - 40 ) + ( 63 + ( 3 * 1 ) ) ) * ( ( ( 1 + 211 ) * 4 ) - 35 ) ;
Output: 3555 40 40 4 * 5 32 + 3 + / + 40 - 63 3 1 * + + 1 211 + 4 * 35 - * + ;
Result: 60465
---------------------------------------------------------------------------------------
```