**C++ Workshop – 150018**

**Homework Assignment #8**

**Stacks and Queues**

**Before you begin, make sure that your program follows the following rules:**

1. Your program should be easy to read. This means to make sure that you are using correct indentation and blank lines to make your code more readable
2. Make sure that you use constants whenever possible.
3. Make sure that the names that you give your identifiers are meaningful.
4. Break your program down into smaller sub-problems and use functions to solve each sub-problem.
5. Document your code as you were taught in the lecture. You are to write an abstract at the beginning of the program, comments before each method and function, and comment complicated sections of code.
6. Make sure that you understand the assignment and write a program that does exactly what the assignment requires.
7. As a general rule, it is better to pass object as cbr (and not cbv). When passing it cbr – it does not call the copy constructor which saves both time and memory.
8. Make sure that the function/method heading uses the right value return type. If the function is not supposed to calculate a single value then have your function return void. If your function/method is supposed to return an object then make sure that it is actually returning the object and not a reference to the object
9. Add an example of the output of the program in comments at the end of your program.
10. All solutions must be submitted in to Moodle according to the date that your instructor gave you which may not be the date listed in Moodle. You are given exactly one week to complete the assignment. You may work in teams of two. If you choose to do so, then one student must upload the program. The comments on the program must contain the names of both students. The other student must upload a Word document that lists the names of both students who worked on the program.

**Question 1**

In this question you will write a program that will receive a mathematical expression (infix) as a string from the user, and calculate the value of the expression according to the order of precedence of mathematical operations. Your program will use a stack to perform this calculation. **You must solve this problem by using a template class.**

**Part A.**

Add the template class Vector that was taught in the class (and which appears in the code examples). (You are creating a template of the class Vector, because you will need to use the class Vector to sometimes hold elements of type char and sometimes hold elements of type int. Details will follow.)

* Add the method **isEmpty** to the class Vector<T>. The method should return **true**  if the vector is empty and otherwise return **false**. The method must be of type const.

Aka use the following signature for the method:

template <class T> bool Vector<T>::isEmpty() const

**Part B.**

Add the template class **StackVector<T>** that will be similar to the class **StackVector** that we learned in the lecture (as well as appears in the code examples) except that StackVector<T> is a template class and in lecture we studied a regular class.

* The template class will have only one attribute by the name of **data.** The attribute **data** will be of type **StackVector<T>**
* Update all of the methods so that they can now work with an attribute of type **StackVector<T>**
* When you write the function **infixToPostfix** you will use the stack that contains characters (aka **StackVector<char>)**. Details will follow.
* When you write the function **calcPostfix** you will use the stack that contains whole numbers (aka **StackVector<int>)**. Details will follow.

**Part C.**

Write a global function named **infixToPostfix** that is passed by parameter a string which contains an expression in infix form and returns a new string representing the expression but this time in the postfix form.  
 (The algorithm is given below).

For example, for the string

**"(5+3)\*((20/10)+(8-6))"**

the function returns the string:

**"5 3 + 20 10 / 8 6 - + \*"**

Note:

* + - The quotation marks are not part of the expression
    - Expressions can contain numbers that have more than one digit.
    - Add a single space between any two components in the postfix expression that is returned

**The algorithm to convert infix to postfix**

This following algorithm accepts a string in infix form, transforms it into a postfix form, and returns the postfix expression. The expression that we receive as input may have round parentheses only.

1. Create an empty string 🡪 *str*
2. Create an empty stack of chars 🡪 *stk*
3. Read the first character from input 🡪 *ch*
4. As long as there is still is input:
   1. if *ch* is an left parenthesis then push it on the stack
   2. if *ch* is a right parenthesis, then pop the contents of the stack up until and not including the left parenthesis and append each element of the stack to *str*.

Then pop the left parenthesis from the stack.

* 1. if *ch* is an operator, then pop from the stack all operators with higher precedence and append them to *str*.

Then, push *ch* onto the stack.

* 1. if *ch* is a number, then append it to str
  2. read the next character from the input 🡪 *ch*

1. As long as the stack is not empty, pop the contents of the stack and append to *str*

**Reminder:** You need to have a blank space between every two components in the postfix expression. Therefore, when appending values to the string *str* you need to also append blanks spaces. You can use the following hints as to where and how to add the blank spaces.

* In sections 4b, 4c, and 5 – after each character that you add to the string *str* add a blank space
* In section 4.4 after each number (**not digit**) that you add to the string *str* add a blank space.

Please Note: the function infixToPostfix uses a stack that will need to hold **characters**, therefore you will need to use **StackVector<char>**. In other words, when you build an instance of your stack you will need to make sure that your attribute **data** is of type **Vector<char>**  so that it can store **characters.**

**Part D.**

Write a global function called **calcPostfix** that receives as a parameter a string representing an expression in postfix format. The expression might include add, subtract, multiply, and divide operations. The function must calculate and return the result of the expression. (The algorithm is given below).

For example, for the input:

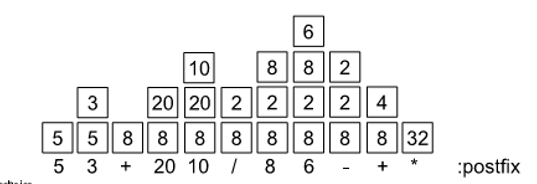
**5 3 + 20 10 / 8 6 - + \***

thefunction will return: 32.

Please Note: the function calcPostfix uses a stack that will need to hold whole numbers, therefore you will need to use **StackVector<**int>. In other words, when you build an instance of your stack you will need to make sure that your attribute **data** is of type **Vector<**int**>**  so that it can store whole numbers.

**The algorithm for computing a postfix arithmetic expression using a stack:**

1. start with empty stack
2. iterate over the elements of the expression from left to right:
   1. if the next element is an operand add it to the stack
   2. if the next element it is an operator execute the operator on the two top elements of the stack and push the result onto the stack as seen in the figure below.



**Part E.**

Use the following main program to check the correctness of the program you have written.

int main() {

string exp;

cout << "enter an infix expression as a string" << endl;

cin >> exp;

string postfix = infixToPostfix(exp);

cout << "in postfix form: " << postfix << endl;

cout << "calculated value: " << calcPostfix(postfix) << endl;

return 0;

}

**Question 2**

In the lecture, we defined the abstract class Queue as follows:

class Queue

{

public:

virtual void clear() = 0;

virtual void enqueue(int value) = 0;

virtual int dequeue() = 0;

virtual int front () = 0;

virtual bool isEmpty() const = 0;

};

We gave 2 different implementations for a queue, one using an array and the other using a list. In this exercise we would like to give the queue a third implementation, this time using a stack (and another helper stack).

class QueueStack : public Queue

{

protected:

Stack\* data;

public:

QueueStack();

void clear() override;

int dequeue() override ;

void enqueue(int value) override;

int front() override;

bool isEmpty() const override;

};

You need to implement the class **QueueStack**.

Hints:

The following classes can help you solve the problem:

1. The class **Queue** as defined above (note that you will need to only hand in the header file for this class since there is no implementation of its methods.)
2. The class **QueueStack** (both the header and cpp files) You need to make sure that this class implements a queue (FIFO – first in first out) even though in reality, it contains only one attribute which is of type stack.
3. The class **Stack** (both the header and cpp files) You can write your own class Stack or use the class that you were taught in the lecture. If you choose the class that was taught in the lecture, then you are going to need to choose if you want to implement your stack by using a list or by using a vector.
4. If in part 3, you decided that you wish to use the class **Stack** that was taught in the lecture, then
   1. If you decide to implement your stack by using a list, you will need to include the class **List** (both the header and cpp files)
   2. If you decide to implement your stack by using a vector, you will need to include the class **Vector** (both the header and cpp files)

Use the following main program to check the correctness of the class you wrote.

#include <iostream>

#include "QueueStack.h"

using namespace std;

int main() {

Queue\* Q;

int num;

Q = new QueueStack();

cout << "enter 5 numbers: "<< endl;

try {

for (int i = 0; i < 5; i++) {

cin >> num;

Q->enqueue(num);

}

}

catch (const char\* msg)

{

cout << msg;

}

cout << "first in queue is: " << Q->front() << endl;

cout << "removing first 2 elements:" << endl;

cout << Q->dequeue() << ' ';

cout << Q->dequeue() << endl;

cout << "first in queue is: " << Q->front() << endl;

Q->enqueue(8);

Q->enqueue(9);

cout << "emptying out the queue: " << Q->front() << endl;

while (!Q->isEmpty())

cout << Q->dequeue() << " ";

cout << endl;

return 0;

}

Example :

enter 5 numbers:

1 2 3 4 5

first in queue is: 1

removing first 2 elements:

1 2

first in queue is: 3

3 4 5 8 9

GOOD LUCK!