**DHA Suffa University**



**Department of Computer Science**

**CS 2001L – Data Structures and Algorithms Lab**

**Fall 2019**

# Lab 08 – Stacks and Queues

**Objective:**

To learn about pushing and popping an element on a stack using linked list and arrays, use of stack in infix, postfix and prefix expressions, implementation of C++ templates and to learn what Queue Operations are and how to implement Queue using linked list and arrays.

**Stack**

A stack is a list of elements in which an element may be inserted or deleted only at one end, called the **top** of the stack. This means, in particular, that elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

Special terminology is used for two basic operations associated with stacks:

(A) "**Push**" is the term used to insert an element into a stack.

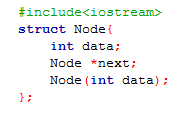
(B) "**Pop**" is the term used to delete an element from a stack.

The operation of adding (Pushing) an item onto a stack and the operation of removing (Popping) an item from a stack may be implemented, respectively, by the following procedures, called PUSH and POP.

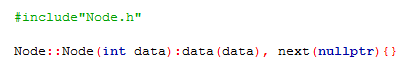
# In executing the procedure PUSH, one must first test whether there is room in the stack for the new item; if not, then we have the condition known as overflow. Analogously, in executing the procedure POP, one must first test whether there is an element in the stack to be deleted; if not, then we have the condition known as underflow.



**Node.h**

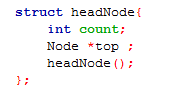
****

**Node.cpp**

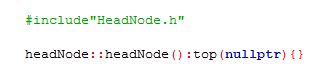
****

**HeadNode.h**

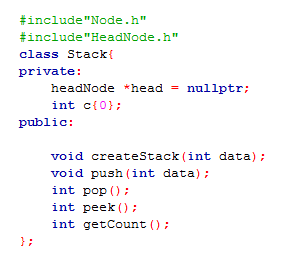
**#include “Node.h”**

****

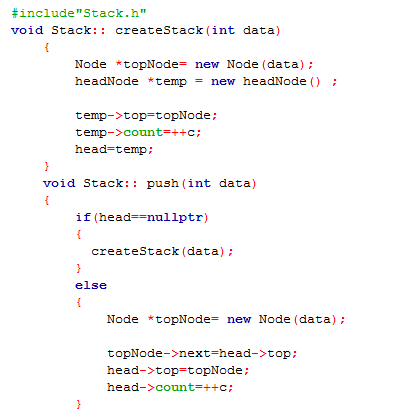
**HeadNode.cpp**

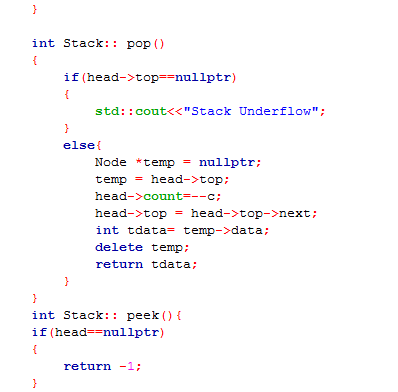
****

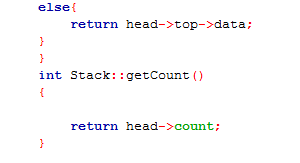
**Stack.h**

****

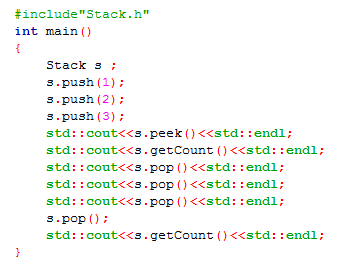
**Stack.cpp**

****

****



**driver.cpp**

****

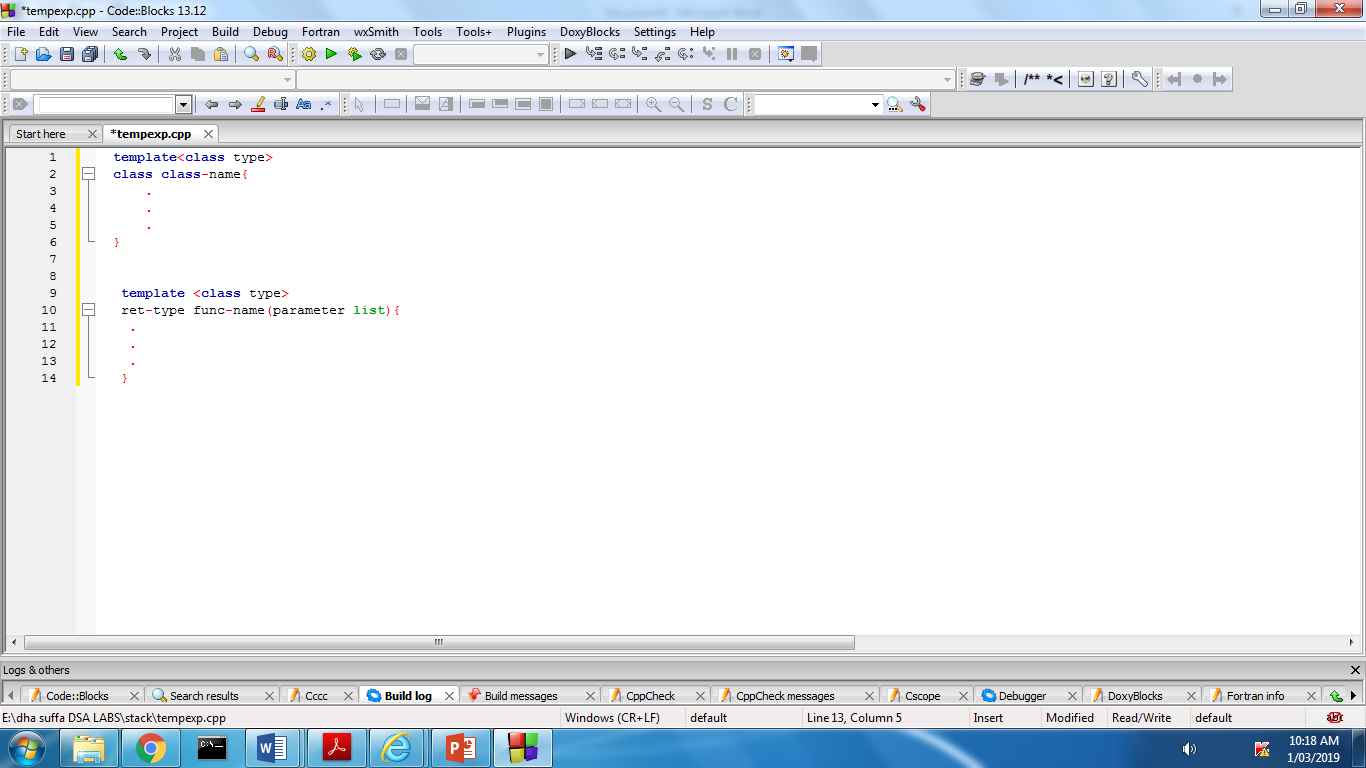
**Lab Task:** Implement function destroyStack( ) which deletes all data in stack.

**C++ Template**

A template is a blueprint or formula for creating a generic class or function. C++ template is a powerful tool, in which we pass data type as parameter. There is no need to write same code for different data types.

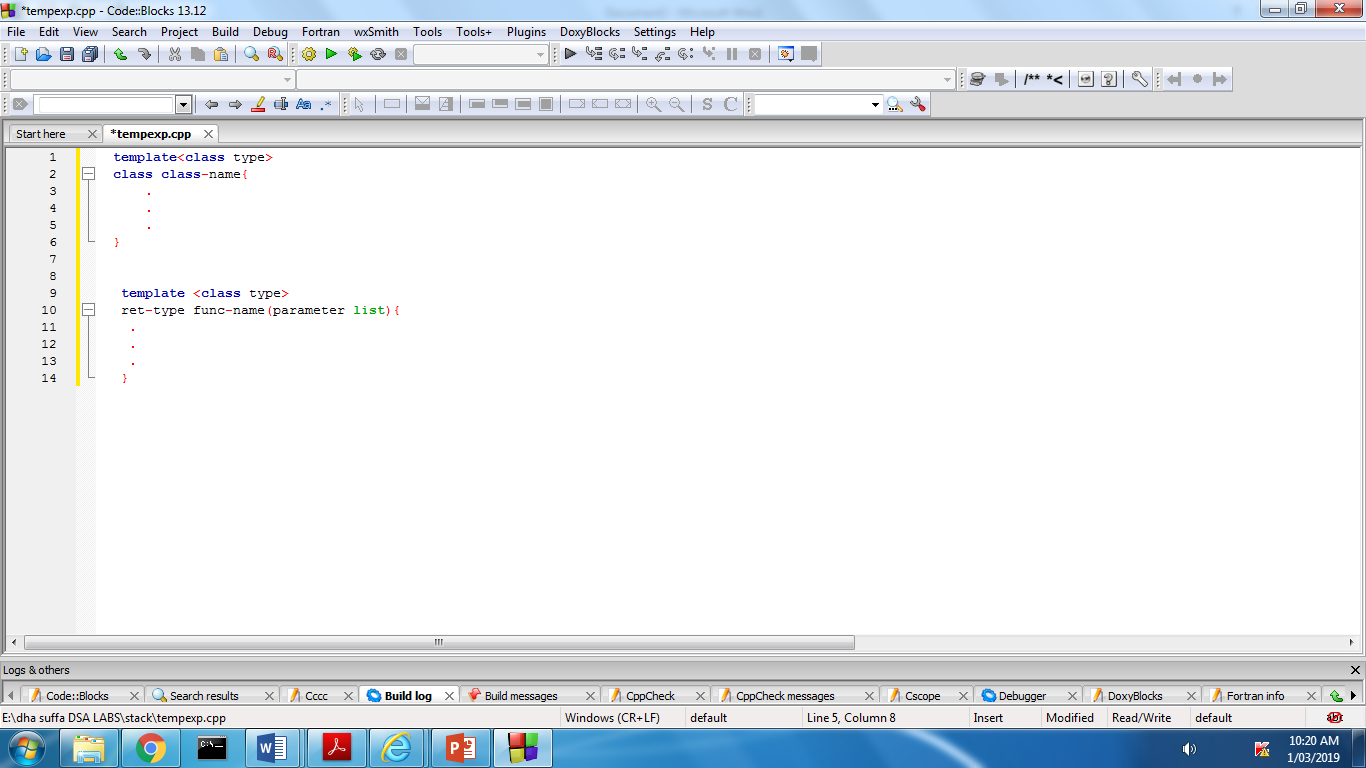
**CLASS TEMPLATE:**

The general form of a class template is shown here



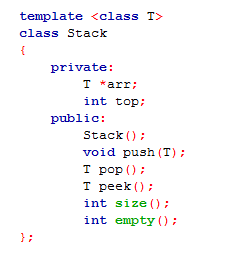
**FUNCTION TEMPLATE:**

The general form of a function template is shown here

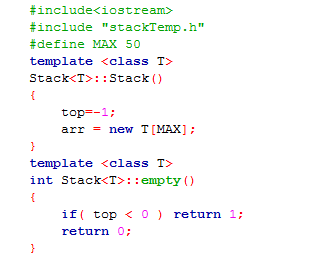


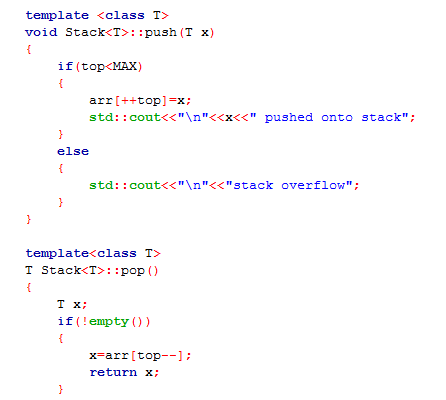
**CODE:**

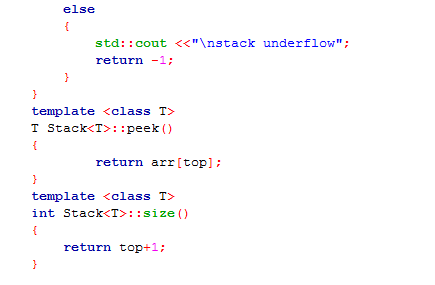
**StackTemp.h**

****

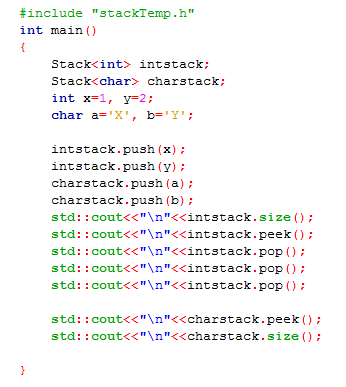
**StackTemp.cpp**

****

****

****

**driver.cpp**

****

**Use of stack in infix, postfix, prefix expressions:**

Infix, Postfix and Prefix notations are three different but equivalent ways of writing expressions. It is easiest to demonstrate the differences by looking at examples of operators that take two operands.

Infix notation: X + Y

Postfix notation: X Y +

Prefix notation: + X Y

**Precedence:**

When an operand is in between two different operators, which operator will take the operand first, is decided by the precedence of an operator over others. For example −

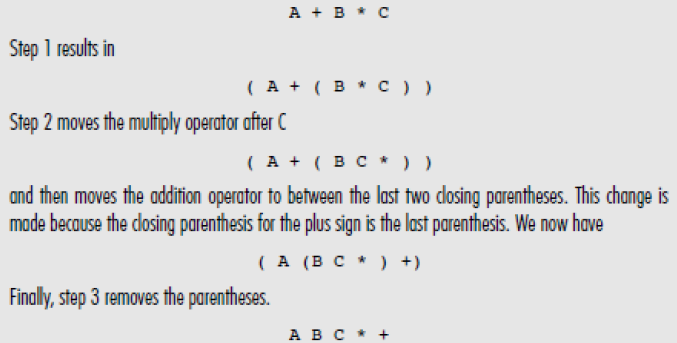
Operator Precendence

As multiplication operation has precedence over addition, b \* c will be evaluated first. A table of operator precedence is provided here.

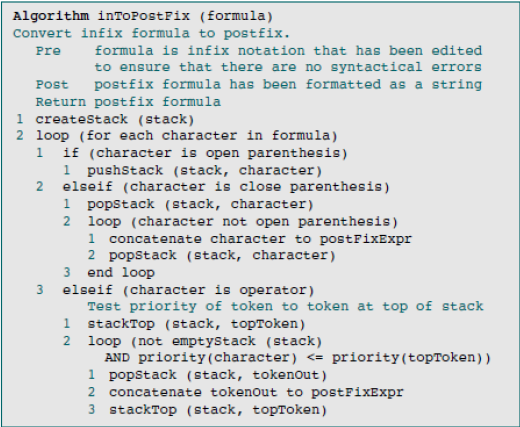
|  |  |  |
| --- | --- | --- |
| **S.No** | **Operator** | **Precedence** |
| 1 | Exponentiation ^ | Highest |
| 2 | Multiplication ( ∗ ) & Division ( / ) | Second Highest |
| 3 | Addition ( + ) & Subtraction ( − ) | Lowest |

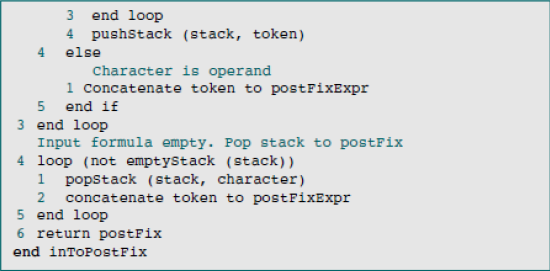
**Infix to Postfix Conversion**

In infix notation, we have to use parenthesis to control evaluation of operators.



**ALGORITHM:**

****

****

**Queue:**

A queue is a list of elements with insertions permitted at one end – called the rear, and deletions permitted from the other end – called the front. This means that the removal of elements from a queue is possible in the same order in which the insertion of elements is made into the queue.

Queue data structure possesses the FIFO (first in first out) property. Insert and delete are the two operations that are provided for insertion of elements and the removal of elements from the queue, respectively.

**Queue Operations:**

The main operations in a queue are as follows:

* **Enqueue** (x) – inserts the data element x to the rear-end of the queue
* **Dequeue** () – deletes the element from the front-end of the queue

**DEQUEUE**

**Front Rear**

## Image result for enqueue dequeue

**ENQUEUE**

Figure 8.1 - Queue

**Implementation Of Queue Using Linked List:**

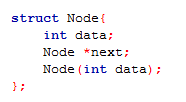
**enqueue(element):** This function is used to insert an element into the queue. In a queue, the new element is always inserted at Rear position.

**dequeue():**This function is used to delete an element from the queue. In a queue, the element is always deleted from front position.

****

Figure 8.2 – Queue Front and Rear

**Node.h**

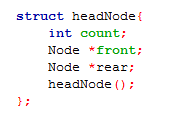
****

**Node.cpp**

****

**HeadNode.h**

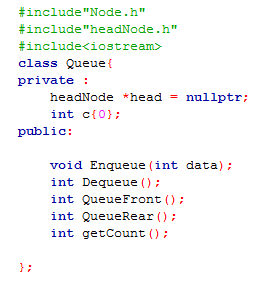
**#include “Node.h”**

****

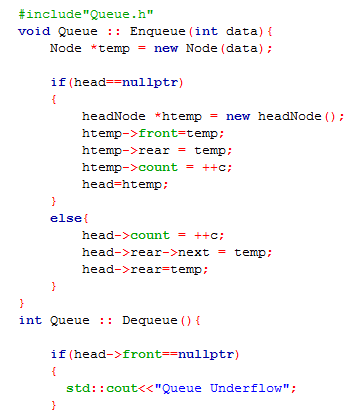
**HeadNode.cpp**

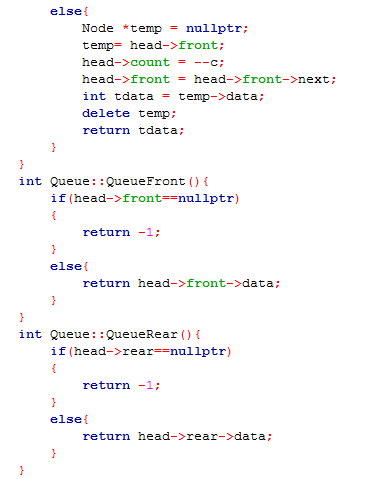
****

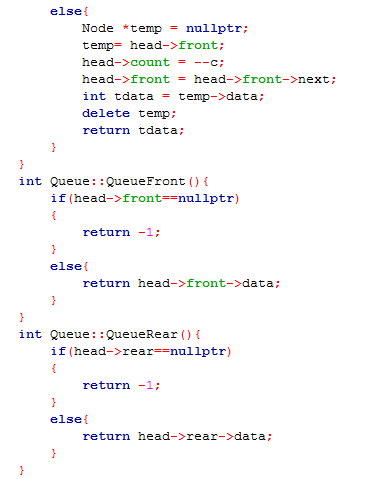
**Queue.h**

****

**Queue.cpp**

****

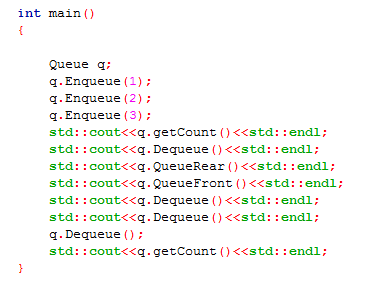
****

****

****

**driver.cpp**

**#include”Queue.h”**

****

**Lab Task:** Implement the queue using arrays. Use templates to implement your queue so that user can insert values of different data types.

**Assignment:**

**Q.1)** Using stack, create a program for the conversion of infix expression to postfix expression.

**Q.2)** Write a program that reads a text file, one line at a time, and prints the line as it was read

and then prints the line with its text reversed. Print a blank line after each reversed line.

Use stack to implement this program.

**Q.3)** Perform the following tasks and implement Queue using Link list.

a. Reverse the first K elements of a Queue.

b. Implement a double ended queue. Double Ended Queue is a Queue data structure in which the insertion and deletion operations are performed at both the ends (front and rear).

# Submission Guidelines

* **Write C++ code , separate function for each operation.**
* **Place your file in a folder named with your rollNo (cs172xxx) where xxx is your 3 digit rollno.**
* **Upload it on LMS.**