

**CSE-225  
 Assignment-1**

Name: Md. Najmus Sakib.

ID: 1813345642.

Section: 14.

**Ans. To The Ques. No. 01**

**Application of stack:-1**

The stack data structure will be used in the Infix to Postfix Converting utilizing Stack in C++ procedure. When we acquire any operands from scanning the infix expression from left to right, we simply add them to the postfix form, and for the operator and parenthesis, we add them to the stack while keeping their precedence.

Sample input: 10+3\*(2^3-1)^(45+2\*1)-3

Sample output: 10323^1-4521\*+^\*+3-

**Application of stack:-2**

To find a parentheses of a string is balanced or not. When the parentheses (“(”, ”{“, ”[“ ) comes we push the bracket to stack otherwise not. For other parentheses (“)”, ”}“, ”]“ ) we will eliminate the relevant parentheses from the stack. When we see the stack remains empty after going through each parentheses, we can say the string is balanced otherwise not.

Sample input: ((()()))

Sample output: Balanced.

**Application of queue:-1**

Priority queue is the application of queue. Element can be inserted arbitrarily but only smallest element can be removed. For example, suppose there is an array having elements 4, 2, 8 in the same order. So, while inserting the elements, the insertion will be in the same sequence but while deleting, the order will be 2, 4, 8.  
  
Sample input: 10, 30, 5, 20

Sample output: 5, 10, 20, 30

**Application of queue:-2**

Double Ended Queue is also a Queue data structure in which the insertion and deletion operations are performed at both the ends (front and rear). That means, we can insert at both front and rear positions and can delete from both front and rear positions.

Sample input: 10, 30, 5, 20

Sample output: “Data deleted from the last is 20”

**Ans. To The Ques. No. 02**

**2.** **Here is the code of “main.cpp” driver file**

#include <iostream>

#include "StackType.cpp"

using namespace std;

int main() {

StackType<char> stack; // defining object of that class.

string parentheses; // holds the string

char top;

bool b=true; // it finds whether it is balanced or not

cout << "Enter string of parentheses: "; //taking parentheses string

getline(cin, parentheses);

for(int i = 0; i < parentheses.length(); i++) { // loop for every character of that string

if(parentheses[i] == '(') { /\*check every character of that string and if we found "(" then we will push it to stack\*/

stack.Push(parentheses[i]);// push function called to push the parentheses.

continue;

}

if(parentheses[i] == ')') { /\*check every character of that string and if we found ")" then we will push it to stack.\*/

if(stack.IsEmpty()) /\* we will see if the stack is empty or not and if it is empty then the boolean variable b is set to false which means not balanced.\*/

b= false;

else{ // if the stack is not empty then we will pop the "(" from the stack top position.

top = stack.Top();

stack.Pop(); // pop out the value of the stack

}

}

}

if(stack.IsEmpty() && b==true) { /\*in decision making, we will see whether the boolean is true also the stack is empty or not.\*/

cout << "Balanced\n"; /\* since stack is empty and boolean variable holds true so the stack is balanced\*/

}

else { // otherwise the string is not balanced.

cout << "Not Balanced\n";

}

return 0;

}

**Here is the “stack.h” header file-**

#ifndef STACKTYPE\_H\_INCLUDED

#define STACKTYPE\_H\_INCLUDED

const int MAX\_ITEMS = 5;

class FullStack

{};

class EmptyStack

{};

template <class ItemType>

class StackType //defining class

{

public: //public variables and functions starts

StackType();

bool IsFull();

bool IsEmpty();

void Push(ItemType);

void Pop();

ItemType Top();

private: //private variables and functions starts

int top;

ItemType items[MAX\_ITEMS];

};

#endif

**Here is the code of “stack.cpp” file where all the functions are implemanted-**

#include "StackType.h"

template <class ItemType> //Standard template class can take any input value.

StackType<ItemType>::StackType()

{

top = -1;

}

template <class ItemType>

bool StackType<ItemType>::IsEmpty()

{

return (top == -1);

}

template <class ItemType>

bool StackType<ItemType>::IsFull()

{

return (top == MAX\_ITEMS-1);

}

template <class ItemType>

void StackType<ItemType>::Push(ItemType newItem)

{

if( IsFull() ) throw FullStack();

top++;

items[top] = newItem;

}

template <class ItemType>

void StackType<ItemType>::Pop()

{

if( IsEmpty() ) throw EmptyStack();

top--;

}

template <class ItemType>

ItemType StackType<ItemType>::Top()

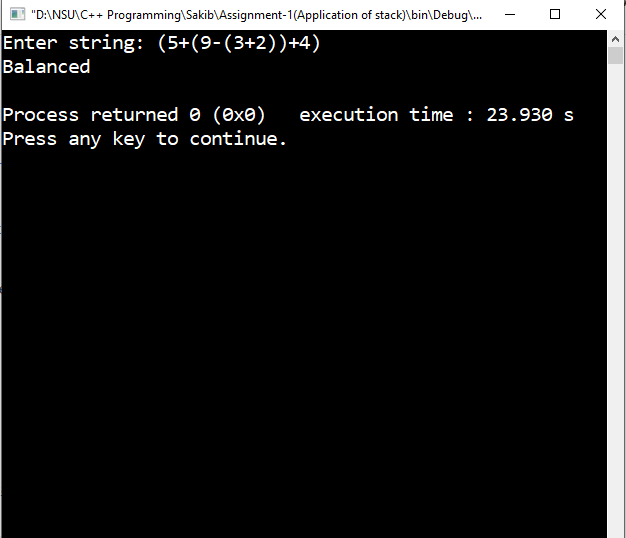
{

if (IsEmpty()) throw EmptyStack();

return items[top];

}

**Sample input output:-1**



**Sample input output:-2**

