DAY 6 ASSIGNMENT

1. Write a function to find the maximum element in the stack.

Ans)

#include <iostream>

#include <stack>

using namespace std;

class CustomStack {

stack<int> stk;

int stack\_max;

public:

void getMax() {

if (stk.empty())

cout << "Stack is empty"<<endl;

else

cout << "Maximum Element in the stack is: "<< stack\_max <<endl;

}

void peek() {

if (stk.empty()) {

cout << "Stack is empty ";

return;

}

int top = stk.top(); // Top element.

cout << "Top Most Element is: "<<endl;

(top > stack\_max) ? cout << stack\_max : cout << top;

}

void pop() {

if (stk.empty()) {

cout << "Stack is empty"<<endl;

return;

}

cout << "Top Most Element Removed: ";

int top = stk.top();

stk.pop();

if (top > stack\_max) {

cout << stack\_max <<endl;

stack\_max = 2 \* stack\_max - top;

} else

cout << top <<endl;

}

void push(int element) {

if (stk.empty()) {

stack\_max = element;

stk.push(element);

cout << "Element Inserted: " << element <<endl;

return;

}

if (element > stack\_max) {

stk.push(2 \* element - stack\_max);

stack\_max = element;

} else

stk.push(element);

cout << "Element Inserted: " << element <<endl;

}

};

int main() {

CustomStack stk;

stk.push(4);

stk.push(6);

stk.getMax();

stk.push(8);

stk.push(20);

stk.getMax();

stk.pop();

stk.getMax();

stk.pop();

stk.peek();

}

Q) Write a function to find the minimum element in the stack.

Ans)

# Class to make a Node

**class** Node:

    # Constructor which assign argument to nade's value

**def** \_\_init\_\_(self, value):

        self.value **=** value

        self.next **=** None

    # This method returns the string representation of the object.

**def** \_\_str\_\_(self):

**return** "Node({})".format(self.value)

    # \_\_repr\_\_ is same as \_\_str\_\_

    \_\_repr\_\_ **=** \_\_str\_\_

**class** Stack:

    # Stack Constructor initialise top of stack and counter.

**def** \_\_init\_\_(self):

        self.top **=** None

        self.count **=** 0

        self.minimum **=** None

    # This method returns the string representation of the object (stack).

**def** \_\_str\_\_(self):

        temp **=** self.top

        out **=** []

**while** temp:

            out.append(str(temp.value))

            temp **=** temp.next

        out **=** '\n'.join(out)

**return** ('Top {} \n\nStack :\n{}'.format(self.top,out))

    # \_\_repr\_\_ is same as \_\_str\_\_

    \_\_repr\_\_**=**\_\_str\_\_

    # This method is used to get minimum element of stack

**def** getMin(self):

**if** self.top **is** None:

**return** "Stack is empty"

**else**:

**print**("Minimum Element in the stack is: {}" .format(self.minimum))

    # Method to check if Stack is Empty or not

**def** isEmpty(self):

        # If top equals to None then stack is empty

**if** self.top **==** None:

**return** True

**else**:

        # If top not equal to None then stack is empty

**return** False

    # This method returns length of stack

**def** \_\_len\_\_(self):

        self.count **=** 0

        tempNode **=** self.top

**while** tempNode:

            tempNode **=** tempNode.next

            self.count**+=**1

**return** self.count

    # This method returns top of stack

**def** peek(self):

**if** self.top **is** None:

**print** ("Stack is empty")

**else**:

**if** self.top.value < self.minimum:

**print**("Top Most Element is: {}" .format(self.minimum))

**else**:

                print("Top Most Element is: {}" .format(self.top.value))

    # This method is used to add node to stack

**def** push(self,value):

**if** self.top **is** None:

            self.top **=** Node(value)

            self.minimum **=** value

**elif** value < self.minimum:

            temp **=** (2 **\*** value) **-** self.minimum

            new\_node **=** Node(temp)

            new\_node.next **=** self.top

            self.top **=** new\_node

            self.minimum **=** value

**else**:

            new\_node **=** Node(value)

            new\_node.next **=** self.top

            self.top **=** new\_node

**print**("Number Inserted: {}" .format(value))

    # This method is used to pop top of stack

**def** pop(self):

**if** self.top **is** None:

            print( "Stack is empty")

**else**:

            removedNode **=** self.top.value

            self.top **=** self.top.next

**if** removedNode < self.minimum:

                print ("Top Most Element Removed :{} " .format(self.minimum))

                self.minimum **=** ( ( 2 **\*** self.minimum ) **-** removedNode )

**else**:

                print ("Top Most Element Removed : {}" .format(removedNode))

# Driver program to test above class

stack **=** Stack()

stack.push(3)

stack.push(5)

stack.getMin()

stack.push(2)

stack.push(1)

stack.getMin()

stack.pop()

stack.getMin()

stack.pop()

stack.peek()