**Assignment 6**

**Q1:-** Write a function to find the maximum element in the stack.

**Answer:-** The different functions designed to push and pop elements from the stack.  
**Push(x)** : Inserts x at the top of stack.

* If stack is empty, insert x into the stack and make maxEle equal to x.
* If stack is not empty, compare x with maxEle. Two cases arise:
  + If x is less than or equal to maxEle, simply insert x.
  + If x is greater than maxEle, insert (2\*x – maxEle) into the stack and make maxEle equal to x. For example, let previous maxEle was 3. Now we want to insert 4. We update maxEle as 4 and insert 2\*4 – 3 = 5 into the stack.

**Pop() :**Removes an element from top of stack.

* Remove element from top. Let the removed element be y. Two cases arise:
  + If y is less than or equal to maxEle, the maximum element in the stack is still maxEle.
  + If y is greater than maxEle, the maximum element now becomes (2\*maxEle – y), so update (maxEle = 2\*maxEle – y). This is where we retrieve previous maximum from current maximum and its value in stack. For example, let the element to be removed be 5 and maxEle be 4. We remove 5 and update maxEle as 2\*4 – 5 = 3.
  + Stack doesn’t hold actual value of an element if it is maximum so far.
  + Actual maximum element is always stored in maxEle.

**Program:-**

#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(6);

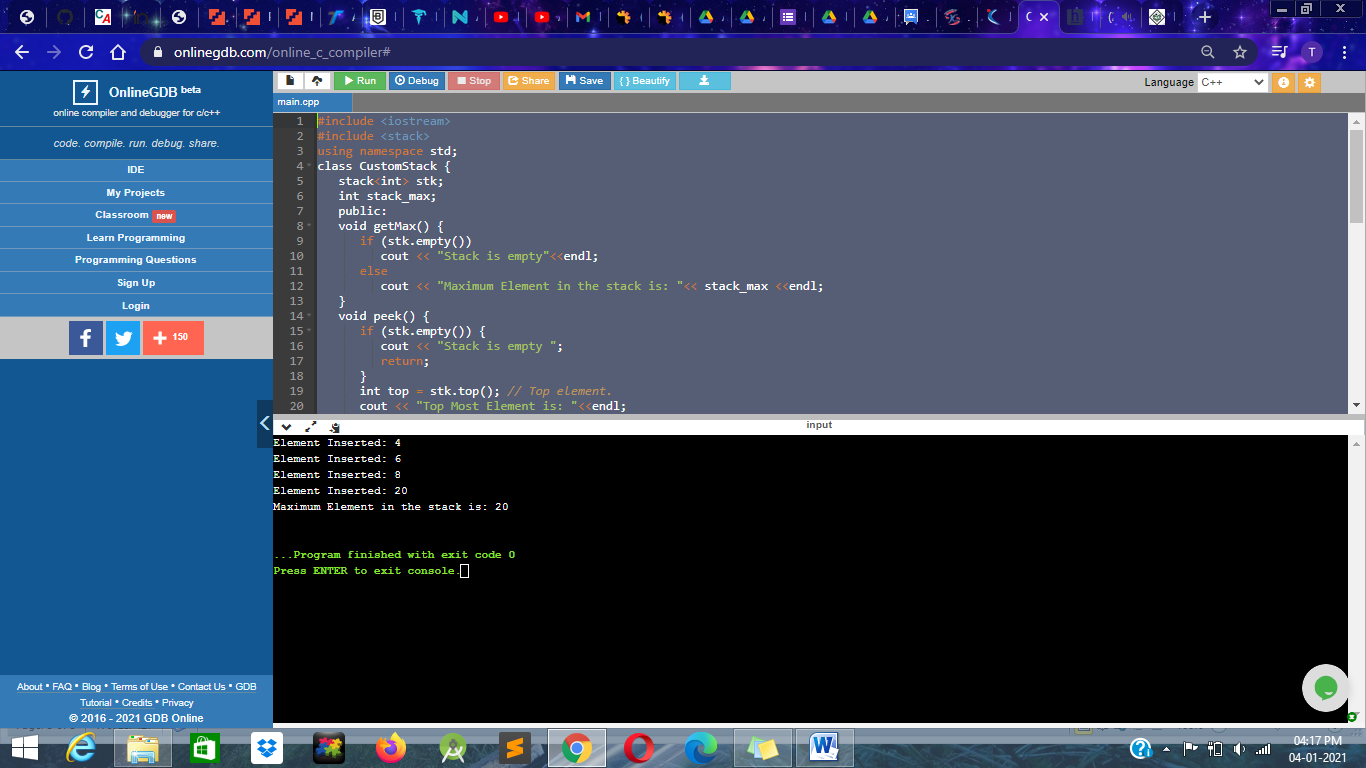
stk.push(8);

stk.push(20);

stk.getMax();

}

Output :-



Q2 :- Write a function to find the minimum element in the stack.

Answer :-

Program :-

#include <iostream>

#include <stdlib.h>

using namespace std;

class Stack {

private:

static const int max = 100;

int arr[max];

int top;

public:

Stack() { top = -1; }

bool isEmpty();

bool isFull();

int pop();

void push(int x);

};

bool Stack::isEmpty()

{

if (top == -1)

return true;

return false;

}

bool Stack::isFull()

{

if (top == max - 1)

return true;

return false;

}

int Stack::pop()

{

if (isEmpty()) {

cout << "Stack Underflow";

abort();

}

int x = arr[top];

top--;

return x;

}

void Stack::push(int x)

{

if (isFull()) {

cout << "Stack Overflow";

abort();

}

top++;

arr[top] = x;

}

class SpecialStack : public Stack {

Stack min;

public:

int pop();

void push(int x);

int getMin();

};

void SpecialStack::push(int x)

{

if (isEmpty() == true) {

Stack::push(x);

min.push(x);

}

else {

Stack::push(x);

int y = min.pop();

min.push(y);

if (x < y)

min.push(x);

else

min.push(y);

}

}

int SpecialStack::pop()

{

int x = Stack::pop();

min.pop();

return x;

}

int SpecialStack::getMin()

{

int x = min.pop();

min.push(x);

return x;

}

int main()

{

SpecialStack s;

s.push(10);

s.push(20);

s.push(30);

cout << s.getMin();

return 0;

}

Output :--

