Assignment No. 6 Data Structures and Algorithms

**Question 1:**

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

**Answer:**

**#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();**

**}**

**Question 2:**

**Write a function to find the minimum element in the stack.**

**Answer:**

**#include<stdio.h>**

**#include<conio.h>**

**int main\_stack[100] , supporting\_stack[100];**

**int push(int element , int \*top , int \*stack){**

**\*top = \*top + 1;**

**stack[\*top] = element;**

**}**

**int pop(int \*stack , int \*top){**

**int element;**

**if(\*top > -1){**

**element = stack[\*top];**

**\*top = \*top - 1;**

**return element;**

**}**

**else {**

**printf("\n== STACK EMPTY == \n");**

**return -99999; // means nothing is popped**

**}**

**}**

**int main()**

**{**

**int choice , element , top\_main=-1 , top\_support=-1 ,i ,supp\_stack\_pop\_element, popped\_element;**

**printf("Enter the operation : \n 1. Push \n 2.Pop \n 3.check minimum \n 4.STOP \n");**

**scanf("%d",&choice);**

**while(choice != 5)**

**{**

**if (choice == 1)**

**{**

**printf("\nEnter the number to be pushed");**

**scanf("%d",&element);**

**push(element , &top\_main , main\_stack);**

**if (top\_support >= 0 && element < supporting\_stack[top\_support])**

**{**

**push(element , &top\_support , supporting\_stack);**

**}**

**else if (top\_support == -1){**

**push(element , &top\_support , supporting\_stack);**

**}**

**}**

**else if (choice == 2){**

**popped\_element = pop(main\_stack , &top\_main);**

**if (popped\_element != -99999)**

**printf("\n Popped element = %d",popped\_element);**

**if (popped\_element != -99999){**

**if (popped\_element == supporting\_stack[top\_support]){**

**supp\_stack\_pop\_element = pop(supporting\_stack , &top\_support);**

**}**

**}**

**}**

**else if( choice == 3){**

**if (top\_support > -1)**

**printf("\nMinimum element in the stack = %d \n\n" , supporting\_stack[top\_support] );**

**else**

**printf("\n ==== Stack Empty ======");**

**}**

**else if (choice == 4){**

**if (top\_main > -1){**

**printf("\n === Main Stack === \n ");**

**for (i=top\_main;i>=0;i--)**

**{**

**printf("\n%d",main\_stack[i]);**

**}**

**}**

**else**

**printf(" \n ==== STACK EMPTY === \n ");**

**printf("\n top\_support = %d", top\_support);**

**for (i=top\_support;i>=0;i--)**

**{**

**printf("\n%d",supporting\_stack[i]);**

**}**

**}**

**printf("\n\nEnter the operation : \n 1. Push \n 2.Pop \n 3.check minimum \n 4.See Full Stack \n 5.STOP \n");**

**scanf("%d",&choice);**

**}**

**return 0;**

**}**