Date: 15/01/2021

Practical 1

2CSDE75 - Advanced Data Structures

Name: Shrey Viradiya

Roll No: 18BCE259

Aim:

Implement shadow copying technique for the STACK data structure to solve the MAXZISE problem.

Code:

Stack.h

```
#pragma once
#include <iostream>

// This header file contains the Stack Class implementation for the
// self growable and self shrinking data structure
// with shadow copying

class Stack{
private:
    double threshold;
    int size;
    int top;
    int *stack;
public:
    Stack();
    Stack(int s);
    ~Stack();
    void push(int n);
    int pop();
    int isEmpty();
    int isFull();
    void setThreshold(double t);
    void displayItems();
};

Stack::Stack(){
        threshold = 0.75;
        size = 5;
        top = -1;
        stack = new int[size];
}

Stack::Stack(int s){
        threshold = 0.75;
        size = s;
        top = -1;
        stack = new int[size];
}

Stack::~Stack(){
    delete[]stack;
}

int Stack::isEmpty(){
```

```
    if(top==-1)
        return 1;
    else
        return 0;
}

void Stack::setThreshold(double t){
    threshold = t;
}

int Stack::isFull(){
    if(top==(size-1))
        return 1;
    else
        return 0;
}

void Stack::push(int n){
    using namespace std;
    if(top >= 0.75*(size-1)){
        size += size;
        int *stackNew = new int[size];
        for (int i = 0; i <= top; i++)
        {
            stackNew[i] = stack[i];
        }
        delete[]stack;
        stack = stackNew;
    }
    ++top;
    stack[top]=n;
}

int Stack::pop(){
    using namespace std;

    if(isEmpty()){
        cout << "Empty Stack.. Returning 0" << endl;
        return 0;
    }
    int temp = stack[top--];

    if(top < (1-threshold)*size){
        size = (size / 2) + 1;
        int *stackNew = new int[size];
        for (int i = 0; i <= top; i++)
        {
            stackNew[i] = stack[i];
```

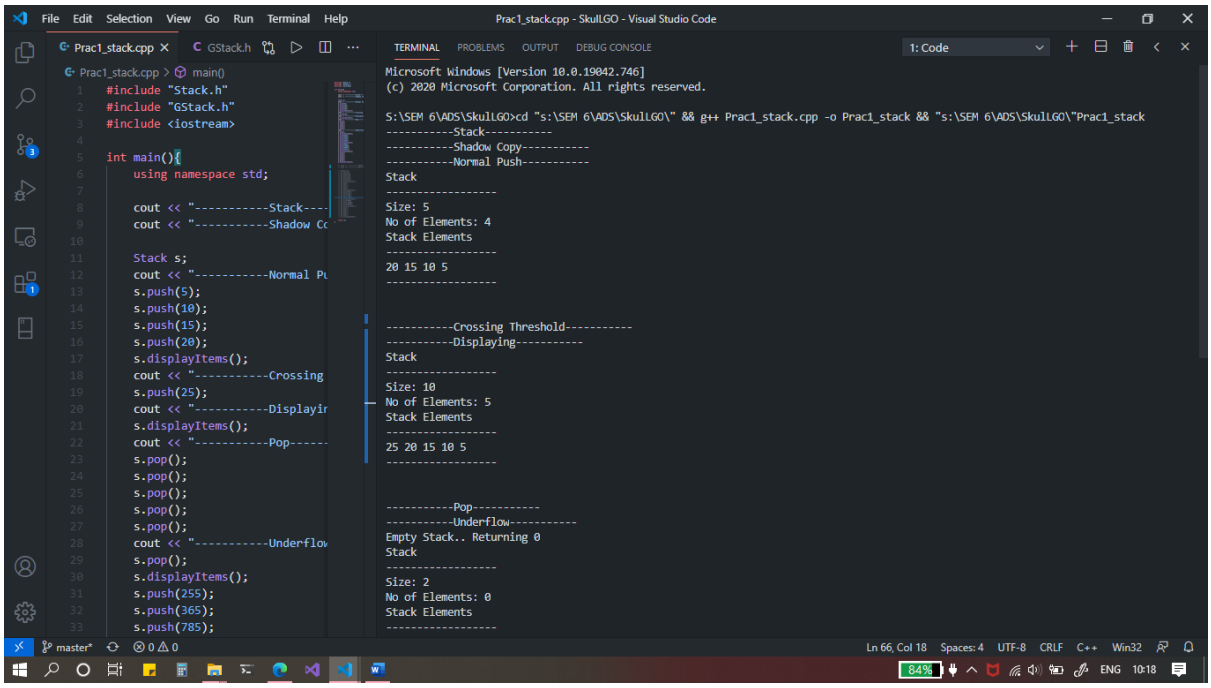
```
        }
        delete[]stack;
        stack = stackNew;
    }
    return temp;
}

void Stack::displayItems(){
    using namespace std;
    cout << "Stack \n------------------" << endl;
    cout << "Size: " << size << endl;
    cout << "No of Elements: " << top+1 << endl;
    cout << "Stack Elements \n------------------" << endl;
    for (int i = top; i >= 0; i--)
    {
        cout << stack[i] << " ";
    }
    cout << "\n------------------\n\n" << endl;
}
```

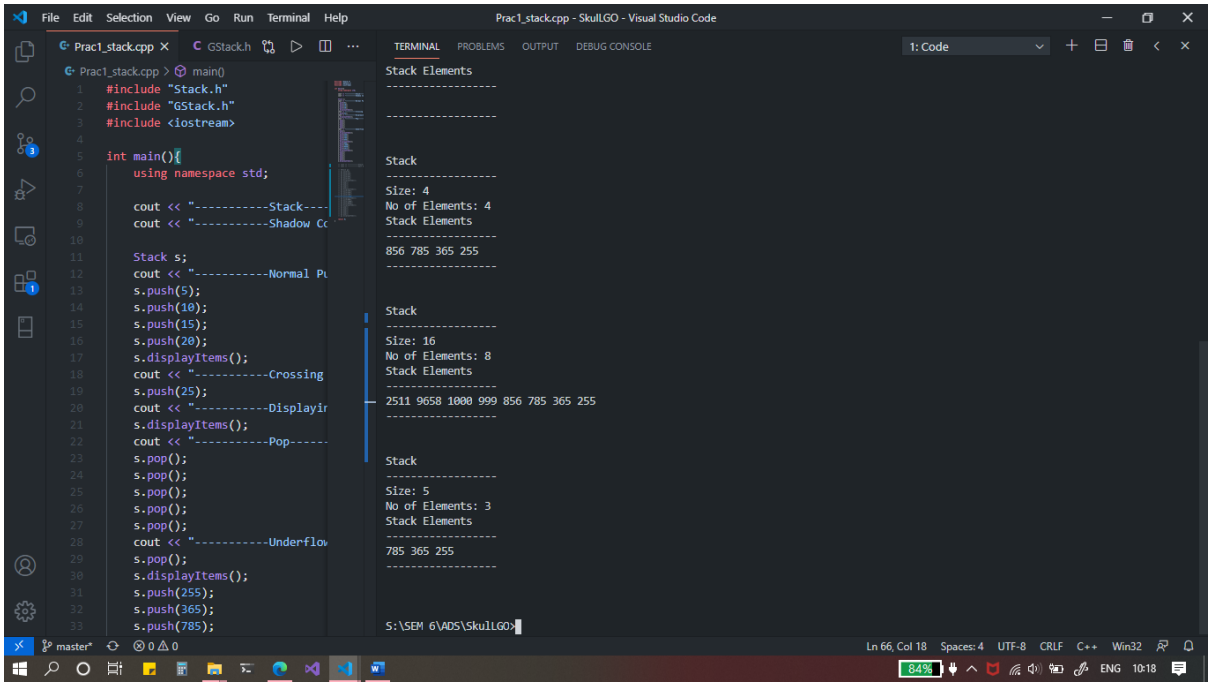
Prac1_Stack.cpp

```
#include "Stack.h"
#include <iostream>

int main(){
    using namespace std;

    cout << "-----------Stack-----------" << endl;
    cout << "-----------Shadow Copy-----------" << endl;

    Stack s;
    cout << "-----------Normal Push-----------" << endl;
    s.push(5);
    s.push(10);
    s.push(15);
    s.push(20);
    s.displayItems();
    cout << "-----------Crossing Threshold-----------" << endl;
    s.push(25);
    cout << "-----------Displaying-----------" << endl;
    s.displayItems();
    cout << "-----------Pop-----------" << endl;
    s.pop();
    s.pop();
    s.pop();
    s.pop();
    s.pop();
```

```
    cout << "-----------Underflow-----------" << endl;
    s.pop();
    s.displayItems();
    s.push(255);
    s.push(365);
    s.push(785);
    s.push(856);
    s.displayItems();
    s.push(999);
    s.push(1000);
    s.push(9658);
    s.push(2511);
    s.displayItems();
    s.pop();
    s.pop();
    s.pop();
    s.pop();
    s.pop();
    s.displayItems();

    return 0;
}
```

Snapshot of the output:





Conclusion:

With the shadow copy, we can solve the problem of the MAXSIZE problem of the stack. When the number of elements reaches a threshold value size, stack memory is increased. In this practical, I also implemented the code to reduce memory, if the number of elements is less than (1-threshold) of size.