Need to build as > g++ Main.cpp MyStack.cpp

MyStack.h

```
#define STACKMAXSIZE 100

typedef struct MyStack {
      unsigned int currentSize;
      unsigned int maximumSize;
      int * elementList;
} MyStack;

void Initialize(MyStack &, unsigned int = STACKMAXSIZE);
void Deinitialize(MyStack &);
int Push(MyStack &, const int &);
int Push(MyStack &, const int &, const int &);
int Pop(MyStack &, int &);
inline unsigned int MaximumSize(const MyStack & theStack)
{
    return theStack.maximumSize;
}
inline unsigned int CurrentSize(const MyStack & theStack)
{
    return theStack.currentSize;
}
int IsEmpty(const MyStack &);
void Display(const MyStack &);

const int ResultSuccess  = 1;
const int ResultFailure = 0;
```

MyStack.cpp

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

using namespace std;

void Initialize(MyStack & theStack, unsigned int maximumSize)
{
    theStack.currentSize = 0;

    // Boundary checking, make it default in case invalid size is provided
    if((0 == maximumSize) || (maximumSize > STACKMAXSIZE))
    {
        maximumSize = STACKMAXSIZE;
    }

    theStack.maximumSize = maximumSize;

    theStack.elementList = new int[maximumSize];

    for(int i = 0; i < maximumSize; i++)
    {
        theStack.elementList[i] = 0;
    }
```

```
}

void Deinitialize(MyStack & theStack)
{
    delete []theStack.elementList;
    theStack.currentSize = 0;
    theStack.maximumSize = 0;
}

int Push(struct MyStack & theStack, const int & element)
{
    if(theStack.currentSize == theStack.maximumSize)
    {
        // LOG an error message
        cout << "PUSH operation failed : Stack is full" << endl ;
        return ResultFailure;
    }

    theStack.elementList[theStack.currentSize++] = element;

    return ResultSuccess;
}

int Push(MyStack & theStack, const int & element1, const int & element2)
{
    if((theStack.currentSize+2) > theStack.maximumSize)
    {
        // LOG an error message
        cout << "PUSH operation failed : No space available for two elements" <<
endl ;
        return ResultFailure;
    }

    theStack.elementList[theStack.currentSize++] = element1;
    theStack.elementList[theStack.currentSize++] = element2;

    return ResultSuccess;
}

int Pop(MyStack & theStack, int & element)
{
    if(0 == theStack.currentSize )
    {
        // LOG an error message
        cout << "POP operation failed : The Stack is empty" << endl ;
        return ResultFailure;
    }

    element = theStack.elementList[--theStack.currentSize];

    return ResultSuccess;
}

int IsEmpty(const MyStack & theStack)
{
    return theStack.currentSize ? ResultFailure : ResultSuccess;
}

void Display(const MyStack &theStack)
```

```
{
    cout << "Stack Maximum Size : " << theStack.maximumSize << endl;

    if(0 == theStack.currentSize )
    {
        cout << "The Stack is Empty" << endl;
        return;
    }

    if(theStack.currentSize == theStack.maximumSize)
    {
        cout << "The Stack is full : " << theStack.currentSize << ". Stack elements
: ";
    }
    else
    {
        cout << "The Stack current size : " << theStack.currentSize << ". Stack
elements : ";
    }

    for(int i = 0; i < theStack.currentSize; i++)
    {
        cout << theStack.elementList[i] << " " ;
    }
    cout << endl;
}
```

Main.cpp

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

using namespace std;

// A sample client program to demonstrate Stack functionalities
int main()
{
    MyStack myStackOne;
    MyStack myStackTwo;

    Initialize(myStackOne, 42);
    Initialize(myStackTwo);

    Display(myStackOne);
    Display(myStackTwo);

    Push(myStackOne, 4);
    Push(myStackOne, 3, 9);

    cout << "StackOne current size : " << CurrentSize(myStackOne) << endl;

    cout << "StackTwo maximum size : " << MaximumSize(myStackTwo) << endl;

    Push(myStackTwo, 6, -7);

    Display(myStackOne);
    Display(myStackTwo);
```

```
    int elem = 0;
    if(ResultSuccess == Pop(myStackOne, elem))
    {
        cout << "Popped element from StackOne is : " << elem << endl;

        // Push the popped element from StackOne into StackTwo
        Push(myStackTwo, elem);
    }

    Display(myStackOne);
    Display(myStackTwo);

    Deinitialize(myStackOne);
    Deinitialize(myStackTwo);

    MyStack myStackThree;
    Initialize(myStackThree, 4);
    Push(myStackThree, 3, 9);
    Push(myStackThree, 7);
    Push(myStackThree, 8, -11);
    Push(myStackThree, 5);
    Display(myStackThree);

    Deinitialize(myStackThree);

    return 0;
}
```