Stack as array list Lab

Steps:

1. Create new project with name (StackAsArray)
2. Create inside this project two files:
    a. Class header file with name (myStack.h)
    b. Main method file with name (testProgStack.cpp)
3. Fill each of the above files with its code.

```
#ifndef H_StackType
#define H_StackType

template <class Type>
class stackType
{
private:
    int maxStackSize;
    int stacktop;
    Type *list;
public:
   void initializeStack();
    bool isEmptyStack();

    bool isFullStack();

    void destroyStack();
    void push(const Type& newItem);
    void pop(Type& poppedItem);

    void displaystack();

       Type top();

       stackType(int stackSize = 100);
    ~stackType();

         const stackType<Type>& operator=(const stackType<Type>&);

         stackType(const stackType<Type>& otherStack);

};


template<class Type>
void stackType<Type>::initializeStack()
{
       stacktop = 0;
}

template<class Type>
void stackType<Type>::destroyStack()
{
```

```
        stacktop = 0;
}

template<class Type>
bool stackType<Type>::isEmptyStack()
{
        return(stacktop == 0);
}

template<class Type>
bool stackType<Type>::isFullStack()
{
        return(stacktop == maxStackSize);
}

template<class Type>
stackType<Type>::stackType(int stackSize)
{
        if(stackSize <= 0)
        {
                cout<<"The size of the array to hold the stack must "
           <<"be positive."<<endl;
                cout<<"Creating an array of size 100."<<endl;

                maxStackSize = 100;
        }
        else
                maxStackSize = stackSize;

        stacktop = 0;
        list = new Type[maxStackSize];

}

template<class Type>
stackType<Type>::~stackType()
{
   delete [] list;
}

template<class Type>
stackType<Type>::stackType(const stackType<Type>& otherStack)
{
   int j;

   maxStackSize = otherStack.maxStackSize;
```

```
   stacktop = otherStack.stacktop;
   list = new Type[maxStackSize];

   if(stacktop != 0)
    for(j = 0; j < stacktop; j++)
      list[j] = otherStack.list[j];
}
template<class Type>
const stackType<Type>& stackType<Type>::operator=
                                    (const stackType<Type>& otherStack)
{
       int j;

       if(this != &otherStack)
       {
              if(maxStackSize != otherStack.maxStackSize)
                     cout<<"Cannot copy. The two stacks are of "
                            <<"different sizes."<<endl;
              else
              {
                     stacktop = otherStack.stacktop;

                     if(stacktop != 0)
                            for(j = 0; j < stacktop; j++)

                                   list[j] = otherStack.list[j];
              }
       }

     return *this;
}

template<class Type>
void stackType<Type>::push(const Type& newItem)
{
       if (!isFullStack())
         {
              list[stacktop] = newItem;
              stacktop++;
         }
       else
              cout << "Cannot add to a full stack." << endl;
}

template<class Type>
void stackType<Type>::pop(Type& poppedItem)
```

```
{
	if (!isEmptyStack())
	  {
		stacktop--;
		poppedItem = list[stacktop];
	  }
	else
		cout << "Cannot remove from an empty stack." << endl;

}


template<class Type>
void stackType<Type>::displaystack()
{
	int i;

	for(i = 0; i < stacktop; i++)
		cout<<list[i]<<" ";
	cout<<endl;
}

template <class Type>
Type stackType<Type>::top()
{
	assert(stacktop != 0);

	return list[stacktop - 1];

}

#endif
```

```
#include <iostream>
#include <assert.h>
#include "myStack.h"
using namespace std;

void testCopyConstructor(stackType<int> otherStack);

int main()
{
        stackType<int> stack(50);
        stackType<int> copyStack(50);
        stackType<int> dummyStack(100);
        int x;

        stack.push(66);
        stack.push(33);
        stack.push(88);
        stack.push(55);

        cout<<"display the contants of Stack\n";
        cout<<"from first element to last "<<endl;
        stack.displaystack();
    cout<<"\nThe Top Element in the Stack is "<<stack.top();
        cout<<endl;

    cout<<"\ndisplay the contants of Stack \n";
        cout<<"from last element to first \n";
        cout<<"with the using of POP function "<<endl;
        while(!stack.isEmptyStack())
        {
                stack.pop(x);
                cout<<" "<<x;
        }

        cout<<"\n\n\n"<<endl;

        copyStack = stack;

        while(!copyStack.isEmptyStack())
        {
                copyStack.pop(x);
                cout<<"Inside copyStack "<<x<<endl;
        }
        copyStack = stack;
        testCopyConstructor(stack);
```

```
        if(!stack.isEmptyStack())
        {
                cout<<"Original stack is not empty"<<endl;
                stack.pop(x);
                cout<<"Top element of the original stack :"<<x<<endl;
        }

        dummyStack = stack;

        system("pause");
        return 0;
}
void testCopyConstructor(stackType<int> otherStack)
{
        int x;

        if(!otherStack.isEmptyStack())
        {
                cout<<"Other stack is not empty"<<endl;
                otherStack.pop(x);
                cout<<"Top element of the other stack: "<<x<<endl;
        }
}
```