Stack as linked list Lab

Steps:

1. Create new project with name (stackAsLinkedList)

2. Create inside this project two file:

   - Class header file with name (linkedStack.h)

   - Test prog file with name (testProglinkedStack.cpp)

3. Fill the files with the bellow code.

linkedStack.h

```
#ifndef H_StackType
#define H_StackType

#include <iostream>

template <class Type>
struct nodeType
{
       Type info;
       nodeType<Type> *link;
};

template<class Type>
class linkedStackType
{
private:
    nodeType<Type> *stackTop;
public:
   linkedStackType();

    void initializeStack();
    bool isEmptyStack();

    bool isFullStack();

    void push(const Type& newItem);

    void pop(Type& poppedElement);

    void destroyStack();

    ~linkedStackType();

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

       linkedStackType(const linkedStackType<Type>& otherStack);

};
```

```
template<class Type>
linkedStackType<Type>::linkedStackType()
{
       stackTop = NULL;
}

template<class Type>
void linkedStackType<Type>::destroyStack()
{
       nodeType<Type> *temp;

       while(stackTop != NULL)
       {
          temp = stackTop;
          stackTop = stackTop->link;
          delete temp;
       }
}
template<class Type>
void linkedStackType<Type>:: initializeStack()
{
    destroyStack();
}


template<class Type>
bool linkedStackType<Type>::isEmptyStack()
{
       return(stackTop == NULL);
}

template<class Type>
bool linkedStackType<Type>:: isFullStack()
{
   return 0;
}

template<class Type>
void linkedStackType<Type>::push(const Type& newElement)
{
       nodeType<Type> *newNode;

       newNode = new nodeType<Type>;
```

```
        newNode->info = newElement;
        newNode->link = stackTop;
        stackTop = newNode;
}

template<class Type>
void linkedStackType<Type>::pop(Type& poppedElement)
{
   nodeType<Type> *temp;

   poppedElement = stackTop->info;

   temp = stackTop;
   stackTop = stackTop->link;
   delete temp;
}

template<class Type>
linkedStackType<Type>::linkedStackType(const linkedStackType<Type>& otherStack)
{
        nodeType<Type> *newNode, *current, *last;

        if(otherStack.stackTop == NULL)
                stackTop = NULL;
        else
        {
                current = otherStack.stackTop;


                stackTop = new nodeType<Type>;
                stackTop->info = current->info;
                stackTop->link = NULL;

                last = stackTop;
                current = current->link;

                while(current != NULL)
                {
                        newNode = new nodeType<Type>;
                        newNode->info = current->info;
                        newNode->link = NULL;
                        last->link = newNode;
                        last = newNode;
                        current = current->link;
                }
        }
}


template<class Type>
linkedStackType<Type>::~linkedStackType()
{
        nodeType<Type> *temp;

        while(stackTop != NULL)
        {
                temp = stackTop;
                stackTop = stackTop ->link;
                delete temp;
        }
}

template<class Type>
const linkedStackType<Type>& linkedStackType<Type>::operator=
                        (const linkedStackType<Type>& otherStack)
{
        nodeType<Type> *newNode, *current, *last;

        if(this != &otherStack)
        {
                if(stackTop != NULL)
                        destroyStack();

                if(otherStack.stackTop == NULL)
                        stackTop = NULL;
                else
                {
                        current = otherStack.stackTop;
                        stackTop = new nodeType<Type>;
                        stackTop->info = current->info;
                        stackTop->link = NULL;
                        last = stackTop;
                        current = current->link;

                        while(current != NULL)
                        {
```

```
                newNode = new nodeType<Type>;
                newNode->info = current->info;
                newNode->link = NULL;
                last->link = newNode;
                last = newNode;
                current = current->link;
            }
        }
    }

    return *this;
}


#endif
```

testProglinkedStack.cpp

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

void testCopy(linkedStackType<int> OStack);

int main()
{
	linkedStackType<int> stack;

	int num;

	stack.push(34);
	stack.push(43);
	stack.push(27);

	while(!stack.isEmptyStack())
	{
		stack.pop(num);
		cout<<num<<endl;
	}

	cout<<endl;
	linkedStackType<int> otherStack;
	linkedStackType<int> newStack;
	newStack = stack;

	cout<<"After the assignment operator, newStack: "<<endl;

	while(!newStack.isEmptyStack())
	{
		newStack.pop(num);
		cout<<num<<endl;
	}
	otherStack = stack;

	cout<<"Testing the copy constructor"<<endl;

	testCopy(otherStack);
```

```
        cout<<"After the copy costructor, otherStack: "<<endl;

        while(!otherStack.isEmptyStack())
        {
               otherStack.pop(num);
               cout<<num<<endl;
        }

        system("pause");
        return 0;
}
void testCopy(linkedStackType<int> OStack)

{
        int num;

        cout<<"Stack in the function testCopy:"<<endl;

        while(!OStack.isEmptyStack())
        {
               OStack.pop(num);
               cout<<num<<endl;
        }
}
```