Tsirekidze mariam

Staque.h

```
#pragma once
#include <iostream>
#ifndef STAQUE
#define STAQUE
using namespace std;
typedef int Staqueelement;
class Staque
{
public:
	Staque();                          //constructor
	Staque(const Staque& original);  //definition of stack copy constructor
	~Staque();                         //destructor
	const Staque& operator= (const Staque& rightHandSide); //assignment operator
	bool empty()const;                      //check if stack is empty
	void push(const Staqueelement& value);//push odd push even
	void display(ostream& out) const;
	//display stack values
	Staqueelement top() const;           //retreive value at top
	void pop(char i);                          // delete top and end of staque
private:
	class Node
	{
	public:
		Staqueelement data;
		Node* next;
		Node* top;
		Node(Staqueelement value, Node * first = 0, Node* link = 0)
			:data(value), top(first),next(link)
		{}
	};
	typedef Node* NodePtr;
	NodePtr myTop;
	NodePtr myEnd;
};

ostream& operator<<(ostream& out, const Staque& aStaque);

#endif
```

Staque.cpp

```
#include <iostream>
#include <new>
#include "Staque.h"
using namespace std;
Staque::Staque()
	:myTop(NULL), myEnd(NULL)
{};
ostream& operator<<(ostream& out, const Staque& aStaque) {
	aStaque.display(out);
	return out;
};
Staque::Staque(const Staque& original)
{
	myTop = NULL; myEnd = NULL;
	if (!original.empty())
	{
		myTop = new Node(original.top());
		Staque::NodePtr lastPtr = myTop,
			origPtr = original.myTop->next;
		while (origPtr != 0)
		{
			lastPtr->next = new Staque::Node(origPtr->data, NULL, lastPtr);
			lastPtr = lastPtr->next;
			origPtr = origPtr->next;

		}
		myEnd = lastPtr;
	}
}
Staque::~Staque()
{
	Staque::NodePtr currPtr = myTop,
		nextPtr;
	while (currPtr != 0)
	{
		nextPtr = currPtr->next;
		delete currPtr;
		currPtr = nextPtr;
	}
	delete myEnd;
	delete myTop;
}

const Staque& Staque :: operator=(const Staque& rightHandSide)
{
	if (this != &rightHandSide)
	{
		this->~Staque();
		if (rightHandSide.empty())
			myTop = 0;
		else
		{
			myTop = new Staque::Node(rightHandSide.top());
			Staque::NodePtr lastPtr = myTop,
				rhsPtr = rightHandSide.myTop->next;
			while (rhsPtr != 0)
```

```
                        {
                                lastPtr->next = new Staque::Node(rhsPtr->data, NULL, lastPtr);
                                lastPtr = lastPtr->next;
                                rhsPtr = rhsPtr->next;

                        }
                        myEnd = lastPtr;
                }
        }
        return *this;
}

bool Staque::empty() const
{
        return(myTop == 0);
}
void Staque::push(const Staqueelement& value)
{

        if (!empty()) {
                if (value % 2 == 0)
                {

                        myTop->top = new Staque::Node(value, NULL, myTop);
                        myTop = myTop->top;
                }

                else
                {
                        myEnd->next = new Staque::Node(value, myEnd, NULL);
                        myEnd = myEnd->next;
                }
        }

        else
        {

                myTop = myEnd = new Staque::Node(value, NULL, NULL);

        }
}
void Staque::display(ostream& out) const
{
        Staque::NodePtr ptr;
        if (!empty()) {
                for (ptr = myTop; ptr != 0; ptr = ptr->next)
                        out << ptr->data << endl;
        }
        else
        {
                cerr << "staque is empty\n";
        }
}


Staqueelement Staque::top()const
{
        if (!empty())
```

```
                return (myTop->data);
        else
        {
                cerr << "staque is empty ";
                return 0;


        }
}
void Staque::pop(char i)
{
        if (!empty())
        {
                if (myTop->next == NULL)
                {
                        delete myTop;
                        myTop = NULL;
                        cout << "the staque is empty" << endl;
                }
                else if (i == 'e')
                {
                        Staque::NodePtr ptr = myTop;
                        myTop = myTop->next;
                        delete ptr;

                }
                else if(i == 'o')
                {
                        Staque::NodePtr ptr1 = myTop;
                        while (ptr1->next->next != NULL)
                        {
                                ptr1 = ptr1->next;
                        }
                        delete ptr1->next;
                        ptr1->next = NULL;
                }
                else
                {
                        cout << "you should press 'o' for odd removal and 'e' for even
removal" << endl;
                }
        }
        else
        {
                cout << "Empty list" << endl;
        }
}
```

Driver.cpp

```cpp
#include <iostream>
#include "Staque.h"
using namespace std;
int main()
{
	int numberofNodes;
	Staque s;
	cout << "Check if my staque is constructed. \n" << boolalpha<<s.empty()<< endl;

	s.push(1);
	s.push(2);
	s.push(3);
	s.push(9);
	s.push(6);
	s.push(8);
	cout << "display our staque s: \n" << endl;
	s.display(cout);
	cout << "delete one odd and two even number\n" << endl;
	s.pop('o');
	s.pop('e');
	s.pop('e');
	cout << "check if staque is empty" << endl;
	cout << s.empty() << endl;
	cout << "display our staque" << endl;
	s.display(cout);
	cout << "if we pass to pop function nor 'o' neither 'e', we get" << endl;
	s.pop('p');

	Staque t;
	cout << "How many elements do you want?" << endl;
	cin >> numberofNodes;
	for (int i = 1; i < numberofNodes; ++i)
	{
		t.push(i * 11);
	}

	cout << "display our staque t" << endl;
	t.display(cout);

	Staque m;
	m.push(1);
	m.push(3);
	m.push(4);
	m.push(1000);
	m.push(15);
	m.push(17);

	cout << "display Staque m" << endl;
	m.display(cout);
	cout << "remove leave one element" << endl;
	m.pop('e');
	m.pop('e');
	m.pop('o');
	m.pop('o');
	m.pop('o');
```

```
        cout << "check element" << endl;
        m.display(cout);



        return 0;

}
```