Assignment 4

Problem 1: *Implement a Stack by using two Queues. In other words, implement push, pop, top, empty functions by using enqueue, dequeue, front and empty functions.*

```
//Main.cpp
#include "QList.h"
#include <iostream>
using namespace std;

class StQueue {
private:
	QList Q1, Q2;
public:
	void push(int);
	void pop();
	int top();

	int size();

	friend ostreamfi operator<<(ostreamfi, StQueuefi);
};

void StQueue::push(int val) {

	Q1.Enqueue(val);

}

void StQueue::pop() {
	int temp = 0;

	if (Q1.isEmpty()) {
		cout << "Stack is empty";
		return;
	}

	if (!(Q1.isEmpty()))
	{
		while (Q1.size() != 1) {
			temp = Q1.Front();
			Q2.Enqueue(Q1.Front());
			Q1.Dequeue();
		}

		Q1.Dequeue();

		while (!(Q2.isEmpty())) {
			temp = Q2.Front();
			Q1.Enqueue(Q2.Front());
			Q2.Dequeue();
		}
	}

}
```

```
int StQueue::top() {
	int temp = 0,
		front = 0;

	if (Q1.isEmpty()) return 0;

	if (!(Q1.isEmpty()))
	{
		while (Q1.size() != 1) {
			temp = Q1.Front();
			Q2.Enqueue(Q1.Front());
			Q1.Dequeue();
		}

		front = Q1.Front();

		while (!(Q2.isEmpty())) {
			temp = Q2.Front();
			Q1.Enqueue(Q2.Front());
			Q2.Dequeue();
		}
	}
	return front;
}

int StQueue::size() {
	return Q1.size();
}

ostreamfi operator<<(ostreamfi out, StQueuefi S) {
	out << S.Q1;

	return out;
}

int main() {
	StQueue SQ;
	for (int i = 0; i < 5; i++) { SQ.push(i); cout << SQ; }
	for (int i = 0; i < 3; i++) { SQ.pop(); cout << SQ; }

	return 0;
}
```

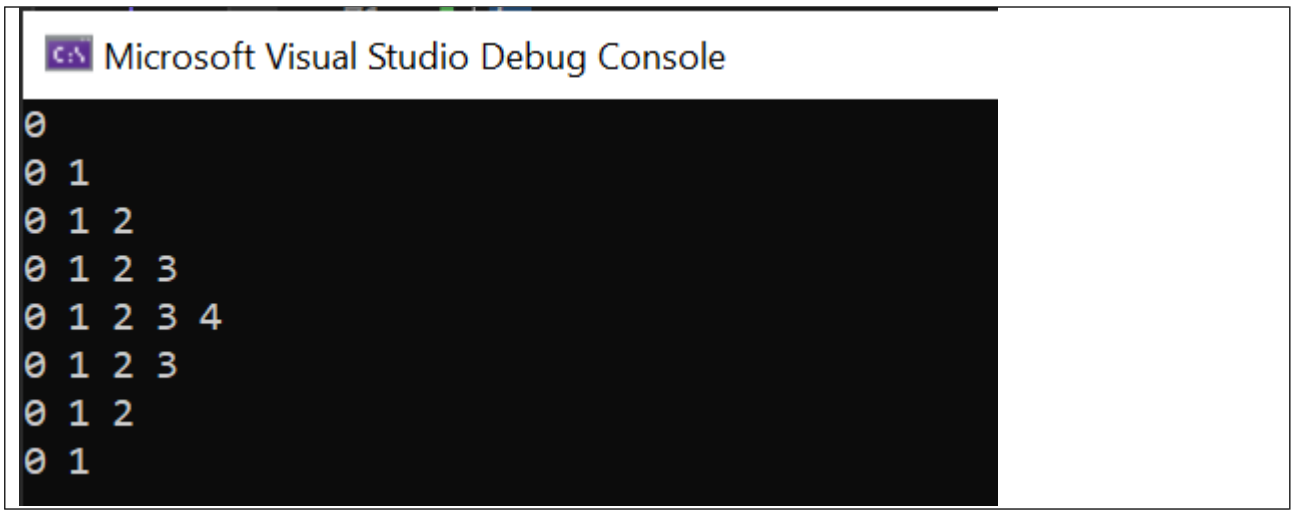
```
//QList.h - Linked List Queue
#pragma once
#include <iostream>
using namespace std;

class QList
{
private:

	class Node {
	public:
		int value;
		Node* next;
		Node() : value(0), next(0) {};
		Node(int item): value(item), next(0){}
	};

	Node* myFront;
	Node* myBack;
	int mySize;

public:
```

```
        QList();
        QList(int); //takes capacity
        QList(QListfi);
        ~QList();

        int size();
        bool isEmpty();

        void Enqueue(int);
        void Dequeue();
        int Front();

        QList operator=(QListfi);
        friend ostreamfi operator<<(ostreamfi, QListfi);
};
```

```
//QList.cpp
#include "QList.h"

QList::QList() : myFront(0), myBack(0), mySize(0){}

QList::QList(int val) : myBack(0), mySize(0)
{
        myFront = new Node(val);
}

QList::QList(QListfi q)
{
        QList();
        Node* temp = q.myFront;
        while (temp != 0) {
                Enqueue(temp->value);
                temp = temp->next;
        }

}

QList::~QList()
{
        Node* temp;
        while (myFront != 0) {
                temp = myFront;

                myFront = myFront->next;
                delete temp;
        }
        myBack = 0;
        mySize = 0;
}

int QList::size()
{
        return mySize;
}

bool QList::isEmpty()
{
        return (myFront == 0);
}

void QList::Enqueue(int val)
{
        if (myFront == 0) {
```

```
                myFront = myBack = new Node(val);
        }
        else {
                myBack->next = new Node(val);
                myBack = myBack->next;
        }
        mySize++;
}

void QList::Dequeue()
{
        if (myFront == 0) cout << "Queue is empty\n";
        else {
                if (myFront == myBack) {
                        delete myFront;
                        myFront = myBack = 0;
                }
                else {
                        Node* temp = myFront;
                        myFront = myFront->next;
                        delete temp;
                }
                mySize--;
        }

}

int QList::Front()
{
        if (myFront == 0) { cout << "Queue is empty\n"; return 9999999; }
        else { return myFront->value; }
}

QList QList::operator=(QListfi q)
{

        QList::~QList();

        mySize = q.mySize;
        QList();
        Node* temp = q.myFront;
        while (temp != 0) {
                Enqueue(temp->value);
                temp = temp->next;
        }
        return *this;
}

ostreamfi operator<<(ostreamfi out, QListfi q)
{
        QList::Node* temp = q.myFront;
        while (temp != 0) {
                out << temp->value << " ";
                temp = temp->next;
        }

        cout << endl;
        return out;
}
```

Output:



Problem 2: *Implement a Queue by using Two Stacks. In other words, implement enqueue, dequeue, front and empty functions by using push, pop, top and empty functions.*

```
//Main.cpp
#include <iostream>
#include "LStack.h"
using namespace std;

class QuStack {
private:
	LStack S1, S2;
public:
	void Enqueue(int);
	void Dequeue();
	int Front();

	friend ostreamfi operator<<(ostreamfi out, QuStackfi Q) {
		out << Q.S1;
		return out;
	}
};

void QuStack::Enqueue(int item) {
	S1.push(item);
}

void QuStack::Dequeue() {
	int temp = 0;

	if (S1.isEmpty()) {
		cout << "Queue is empty";
		return;
	}

	while (!(S1.isEmpty())) {
		temp = S1.top();
		S2.push(temp);
		S1.pop();

	}
	S2.pop();
	while (!(S2.isEmpty())) {
		temp = S2.top();
		S1.push(temp);
```

```
                S2.pop();

        }
}

int QuStack::Front() {
        int front = 0,
                temp = 0;

        if (S1.isEmpty()) return 0;

        while (!(S1.isEmpty())) {
                front = S1.top();
                S2.push(front);
                S1.pop();
        }
        while (!(S2.isEmpty())) {
                temp = S2.top();
                S2.push(temp);
                S2.pop();
        }
        return front;
}

int main() {
        QuStack QS;
        for (int i = 0; i < 5; i++) { QS.Enqueue(i); cout << QS; }
        for (int i = 0; i < 3; i++) { QS.Dequeue(); cout << QS; }

        return 0;
}
```

```
//LStack.h - Linked List Stack
#pragma once
#include <iostream>
using namespace std;

class LStack
{
private:

        class Node
        {
        public:
                int value;
                Node* next;

                Node() : value(0), next(NULL) {};
                Node(int item) : value(item), next(NULL) {};
        };

        Node* myArray;
        Node* myTop;
        int mySize;

public:
        LStack();
        LStack(int);
        LStack(LStackfi);
        ~LStack();

        void push(int);
        void pop();
        int top();
```

```
        int size();
        bool isEmpty();

        Node* newNode(int);

        LStack operator=(LStackfi);
        friend ostreamfi operator<<(ostreamfi, LStackfi);

};
```

```
//LStack.cpp
#ifndef LStack_H
#define LStack_H
#include "LStack.h"
#include <iostream>
using namespace std;

LStack::LStack() : myArray(0), myTop(0), mySize(0) {}

LStack::LStack(int c)
{
        myTop = NULL;
        myArray = new Node[c];
}

LStack::LStack(LStackfi ls)
{
        myArray = new Node(ls.myArray->value);
        Node* duplicate = myArray;
        Node* currnode = ls.myArray->next;

        while (currnode != 0) {
                duplicate->next = new Node(currnode->value);
                currnode = currnode->next;
                duplicate = duplicate->next;
        }

        mySize = ls.mySize;
}

LStack::~LStack()
{
        LStack::Node* temp;
        while(myTop != 0){
                temp = myTop;
                myTop = myTop->next;
                delete temp;
        }
}


void LStack::push(int val)
{
        Node* temp = new Node(val);
        if (myTop == 0) { myTop = temp; }
        else {
                temp->next = myTop;
                myTop = temp;
        }
        mySize++;
}

void LStack::pop()
{
```

```
	if (!(isEmpty())) {
		Node* temp = myTop;
		myTop = myTop->next;
		delete temp;
		mySize--;
	}
	else {
		cout << "LStack is Empty" << endl;
		return;
	}
}

int LStack::top()
{
	if (!(isEmpty())) {
		return myTop->value;
	}
	else {
		cout << "LStack is Empty: Error: " << endl; return 9999999;
	}

}

int LStack::size()
{
	return mySize;
}

bool LStack::isEmpty()
{
	return (myTop == NULL);
}

LStack LStack::operator=(LStackfi LStack)
{
	delete[] myArray;

	myArray = new Node(LStack.myArray->value);

	Node* duplicate = myArray;
	Node* currnode = LStack.myArray->next;

	while (currnode != 0) {
		duplicate->next = new Node(currnode->value);
		currnode = currnode->next;
		duplicate = duplicate->next;
	}

	mySize = LStack.mySize;

	return *this;

}

ostreamfi operator<<(ostreamfi out, LStackfi ls)
{
	LStack::Node* temp = ls.myTop;
	while (temp != 0) {
		out << temp->value << " ";
		temp = temp->next;
	}
	out << endl;

	return out;
}
```

```
#endif
```

Output: