Queue as Linked based list Lab

Steps:

1. Create new project with name (QueueAsLinkedList)
2. Create inside this project two file:
   - Class header file with name (linkedQueue.h)
   - Test prog file with name (testProgLinkedQueue.cpp)
3. Fill the files with the bellow code.

linkedQueue.h

```
#include <iostream>
#include <assert.h>

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

template<class Type>
class linkedQueueType
{
private:
	nodeType<Type> *queueFront;
	nodeType<Type> *queueRear;
public:
	bool isEmptyQueue();
	bool isFullQueue();
	void destroyQueue();
	void initializeQueue();
	Type front();
	Type back();
	void addQueue(const Type& newElement);
	void deQueue(Type& deqElement);
	linkedQueueType();
	~linkedQueueType();
};

template<class Type>
linkedQueueType<Type>::linkedQueueType()
{
	queueFront = NULL;
	queueRear = NULL;
}
template<class Type>
bool linkedQueueType<Type>::isEmptyQueue()
{
	return(queueFront == NULL);
}

template<class Type>
bool linkedQueueType<Type>::isFullQueue()
{
	return false;
}

template<class Type>
void linkedQueueType<Type>::destroyQueue()
```

```
{
	nodeType<Type> *temp;

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

	queueRear = NULL;
}

template<class Type>
void linkedQueueType<Type>::initializeQueue()
{
	nodeType<Type> *temp;

	while (queueFront != NULL)
	{
		temp = queueFront;
		queueFront = queueFront->link;
		delete temp;
	}
	queueRear = NULL;
}
template<class Type>
Type linkedQueueType<Type>::front()
{
	assert(queueFront != NULL);
	return queueFront->info;
}

template<class Type>
Type linkedQueueType<Type>::back()
{
	assert(queueRear != NULL);
	return queueRear->info;
}

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

	newNode = new nodeType<Type>;
	newNode->info = newElement;
	newNode->link = NULL;

	if (queueFront == NULL)
	{
		queueFront = newNode;
		queueRear = newNode;
	}
	else
	{
		queueRear->link = newNode;
		queueRear = queueRear->link;
	}
}
template<class Type>
void linkedQueueType<Type>::deQueue(Type& deqElement)
{
	nodeType<Type> *temp;
	deqElement = queueFront->info;
	temp = queueFront;
```

```
        queueFront = queueFront->link;
        delete temp;

        if (queueFront == NULL)
                queueRear = NULL;
}

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

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

        queueRear = NULL;
}
```

testProgLinkedQueue.cpp

```
#include <iostream>

#include "linkedQueue.h"
using namespace std;


int main()

{

        linkedQueueType<int> queue;

        int x, y;

        queue.initializeQueue();

        x = 4;

        y = 5;

        queue.addQueue(x);

        queue.addQueue(y);

        queue.deQueue(x);

        queue.addQueue(x + 5);

        queue.addQueue(16);

        queue.addQueue(x);

        queue.addQueue(y - 3);

        cout<<"Queue Elements: ";

        while(!queue.isEmptyQueue())

        {

                queue.deQueue(y);

                cout<<" "<<y;

        }

        cout<<endl;

        system("pause");

        return 0;             }
```