Assignment 4

Queues and Stacks

Purpose

You will need to use C++11 support for this exercise. For Eclipse and other GCC based compilers you will need to use the `-std=c++11` or `-std=c++0x` compiler option. This is not needed if you are using Visual Studio since it does not distinguish between the various versions of C++.

In assignment 3 you created a double linked list and an ordered double linked list. You will be using the **DoubleLinkedList** and **ListNode** classes to implement stack and queue classes. We really only need a linked list for this exercise, but we already have a nice double linked list you have written and we are going to make use of it. You will once again be using composition and not inheritance to reuse the **DoubleLinkedList** class.

Stack class

The **Stack** class needs to use the **DoubleLinkedList** to implement the stack. Your **Stack** class needs to be a template class:

```
template<class DataType>
class Stack
```

The **Stack** class needs to implement the following member functions:

```
bool empty() const
```

The empty member function will return true if the stack is empty.

```
std::size_t size() const
```

Returns the number of items in the stack.

```
DataType& top()
```

Returns a modifiable reference to the top item in the stack. Note, your member function can assume the stack has at least one item or you can throw an exception, the choice is yours.

```
const DataType& top() const
```

Returns a **const** reference to the top item in the stack. Note, your member function can assume the stack has at least one item or you can throw an exception, the choice is yours.

`void push(const DataType& value)`

Add a new item to the stack.

`void pop()`

Remove the top item from the stack.

Constructors and destructor

You will need a default constructor and a copy constructor and you will need a destructor.

The copy constructor needs to make a deep copy of the list. You should be able to use other member functions to do most of the work for you.

Make sure your destructor removes any remaining items from the stack.

Other information

You can add other member functions as needed. Make sure you free up any memory you allocate. Hopefully this is already being done for you by the **DoubleLinkedList** class.

You can inline trivial member functions in the **Stack** class (trivial being 1 or 2 lines of code).

You must implement all of the member functions shown here. You can implement additional private, and protected member functions if you need them.

Create a header file named **Stack.h** that contains your **Stack** class definition and any implementation code needed. Include the required header file include guards needed.

Please write a driver function (main) to test your Stack class. If you want to implement code that actually uses the stack feel free to do so (for example, you could have an application that prints out the contents of a linked list backwards).

Queue class

The **Queue** class needs to use the **DoubleLinkedList** to implement the queue. Your **Queue** class needs to be a template class:

```
template<class DataType>
class Queue
```

The **Queue** class needs to implement the following member functions:

`bool empty() const`

The empty member function will return true if the queue is empty.

`std::size_t size() const`

Returns the number of items in the queue.

`DataType& front()`

Returns a modifiable reference to the first item in the queue. Note, your member function can assume the queue has at least one item or you can throw an exception, the choice is yours.

`const DataType& front() const`

Returns a **const** reference to the first item in the queue. Note, your member function can assume the queue has at least one item or you can throw an exception, the choice is yours.

`DataType& back()`

Returns a modifiable reference to the last item in the queue. Note, your member function can assume the queue has at least one item or you can throw an exception, the choice is yours.

`const DataType& back() const`

Returns a **const** reference to the last item in the queue. Note, your member function can assume the queue has at least one item or you can throw an exception, the choice is yours.

`void push(const DataType& value)`

Add a new item to the back of the queue.

`void pop()`

Remove the first item from the queue.

Constructors and destructor

You will need a default constructor and a copy constructor and you will need a destructor.

The copy constructor needs to make a deep copy of the list. You should be able to use other member functions to do most of the work for you.

Make sure your destructor removes any remaining items from the queue.

Other information

You can add other member functions as needed. Make sure you free up any memory you allocate. Hopefully this is already being done for you by the **DoubleLinkedList** class.

You can inline trivial member functions in the **Queue** class (trivial being 1 or 2 lines of code).

You must implement all of the member functions shown here. You can implement additional private, and protected member functions if you need them.

Create a header file named **Queue.h** that contains your **Queue** class definition and any implementation code needed. Include the required header file include guards needed.

Update your driver function (main) to test your Queue class as well as your Stack class. If you want to implement code that actually uses the queue feel free to do so.

Submit all of your header and source files to eLearning to get credit for this project.

You should write stubs for all of the functions as you are creating the classes and then incrementally implement the member functions for the **Stack** and **Queue** classes. By doing this incremental development you will have a better idea which function is failing (because it is probably the code you just wrote). Also, make sure you test and fix a member function before you go onto the next one.

Use good formatting in your programs. Use meaningful variable names and comment your code. Were you happy with the comments you had put into the code you imported from assignment 3?

Let me know if you have any questions.