CS 319 Applied Programming

Fall 2018

Programming Assignment # 1: Linked Lists, Stacks, Queues

Instructor: Muhammad Saqib Ilyas (saqib.ilyas@nu.edu.pk)

TAs:

MS (A): Waleed Mazhar (l176137@lhr.nu.edu.pk)

MS (B): Razi Uddin (l176115@lhr.nu.edu.pk)

MS (DS): Muhammad Kashif Arif (l176102@lhr.nu.edu.pk)

Introduction

In this assignment, you will implement the doubly linked list data structure. Using the linked list, you will then implement a stack and a queue.

Document conventions: Filenames are given in ***bold italic***. Class names are given in **bold** type. Code fragments are given in `monospaced font`.

Part I: Implementing a doubly linked list

Skeletal code for this part is provided to you in the files ***list.h*** and ***list.cpp***. The ***list.h*** file contains the class declarations for the generic **Node** class which represents a node in the linked list and the generic **List** class, which represents a doubly linked list itself. The implementation for class **Node** is already given, whereas you must provide the implementation for the **List** class in ***list.cpp***. WARNING: You are not allowed to make any changes to ***list.h***.

The following provides a description of the functions to be implemented for the **List** class.

```
List()
```

This is the default constructor for the doubly linked list

```
List(const List<T>& otherList)
```

This is the copy constructor for the doubly linked list

```
~List()
```

This is the destructor for the doubly linked list. It should delete all nodes in the linked list and deallocate all dynamically allocated memory.

```
void InsertAtHead(T item)
```

This function should insert a new node with the value stored in the variable "`item`", at the front of the linked list. Remember to handle any **List** class variables affected by this operation.

```
void InsertAtTail(T item)
```

This function should insert a new node with the value stored in the variable "`item`", at the rear of the linked list. Remember to handle any **List** class variables affected by this operation.

```
void InsertAfter(T toInsert, T afterWhat)
```

This function should insert a new node with the value stored in the variable "`item`", right after the node that has a value equal to the value of the variable "`afterWhat`". If no such node exists in the linked list, you may insert the new node at the end of the linked list.

```
Node<T> *GetHead()
```

This function returns a pointer to the first node in the linked list

```
Node<T> *GetTail()
```

This function returns a pointer to the last node in the linked list

`Node<T> *SearchFor(T item)`

This function searches for a node in the linked list that has value equal to that stored in the variable "`item`". If such a node is found, its address is returned, otherwise, `NULL` is returned.

`void DeleteElement(T item)`

This function searches for a node in the linked list that has value equal to that stored in the variable "`item`". If such a node is found, it is deleted from the list. Be sure to update any pointers affected by this operation.

`void DeleteHead()`

This function deletes the node at the front of the linked list. Be sure to update any pointers affected by this operation.

`void DeleteTail()`

This function deletes the node at the rear of the linked list. Be sure to update any pointers affected by this operation.

`int Length()`

This function returns the number of nodes presently in the linked list.

Compiling and testing

To test this part, open a command prompt, switch to the directory containing the files list.cpp, list.h and task1.cpp, then run the following command:

```
g++ task1.cpp –o task1.exe
```

If you encounter any compilation errors, go back to the editor and fix them, then repeat the compilation. Once the compilation is successful, run the program by issuing the following command from the command prompt in the same directory:

```
task1
```

The output will show which test cases passed and which failed. If all goes well, your output will look like the following:

```
Starting Tests
Testing List Initialization .. Passed! 5 / 5
Testing Insertion at Head . Passed! 5 / 5
Testing Insertion at Tail . Passed! 10 / 10
Testing Insertion After . Passed! 10 / 10
Testing Delete ... Passed! 20 / 20
Testing Copy . Passed! 5 / 5
Total Points: 55 / 55
```

Part II: Implementing a stack and a queue

Skeletal code for this part is provided to you in the files ***stack.h***, ***stack.cpp***, ***queue.h***, ***queue.cpp*** and ***task2.cpp***. You should not make any changes to any of the ***.h*** files or ***task2.cpp***. You need to provide implementations of the relevant functions in ***stack.cpp*** and ***queue.cpp***. Internally, our stack and queue implementations are instantiating a linked list that you implemented in task 1. You just need to delegate

each of the stack and queue functions to one or more functions in the linked list class to achieve the desired behavior.

The following is a summary of the functions that you must implement for the **Stack** class:

`Stack()`

The default constructor for the **Stack** class.

`Stack(const Stack<T>& otherStack)`

The copy constructor for the **Stack** class.

`~Stack()`

The destructor for the **Stack** class.

`void Push(T item)`

This function is used to add elements to the stack. The elements should be added to the underlying linked list so that they can be later retrieved efficiently according to the LIFO order.

`T Peek()`

This function returns the value currently stored at the top of the stack, without actually removing it.

`T Pop()`

This function removes the element from the top of the stack and returns it.

`int Length()`

This function returns the number of elements presently on the stack.

The following is a summary of the member functions of the **Queue** class:

`Queue()`

The default constructor for the **Queue** class.

`Queue(Queue<T>& otherQueue)`

The copy constructor for the **Queue** class.

`~Queue()`

The destructor for the **Queue** class.

`void Enqueue(T item)`

Add an item to the tail of the queue.

`T Peek()`

Return the value of the item stored at the head of the queue without removing it from the queue.

`T Dequeue()`

Remove the item stored at the head of the queue and return its value.

```
int Length()
```

Return the number of items in the queue.

Compiling and testing

To test this part, open a command prompt, switch to the directory containing the files ***list.cpp***, ***list.h***, ***stack.h***, ***stack.cpp***, ***queue.h***, ***queue.cpp*** and ***task2.cpp***, then run the following command:

```
g++ task2.cpp –o task2.exe
```

If you encounter any compilation errors, go back to the editor and fix them, then repeat the compilation. Once the compilation is successful, run the program by issuing the following command from the command prompt in the same directory:

```
task2
```

The output will show which test cases passed and which failed. If all goes well, your output will look like the following:

```
Starting Tests
Testing Stack ..... Passed! 5 / 5
Testing Queue ..... Passed! 5 / 5
Total Points: 10 / 10
```

Submission instructions

Zip only the source code files and not any object or executable files into one archive. Submit this archive to slate. The submission must be done before 5 pm on October 29, 2018.