7SENG010W Data Structures and Algorithms

Week 3 Tutorial Exercises: Linked Lists

These exercises cover: Singly and Doubly Linked Lists, and the .NET Framework doubly linked list.

The example Lists code classes used in the Week 3 Lecture are available on Blackboard in the Week 3 folder.

Exercise 1.

(a) Add a method to print out a string representation of a list node called `print()` to the `ListNode` class (`ListNode.cs`). It should include the value in the node and the value stored in its next node or "NULL" if there is no next node.

(b) Using the list from Lecture 3 - < 42, 10, 23, 60 >, draw "List" diagrams in the style of Lecture 3 for the following operations:

deleteHead( )	- delete the head node if it exists, e.g. result is < 10, 23, 60 >
insertAtTail(99)	- add an item to the tail of the list, e.g. result is < 10, 23, 60, 99 >

In addition think about how these operations would work on an empty list.

(c) Based on your list diagrams from part (b) implement them by add the following methods to the `LinkedList` class (`LinkedList.cs`):

```
public Object deleteHead() ;        // delete & return head item, or null if empty

public void insertAtTail( Object item ) ;    // insert item as tail node
```

(d) Test your implementation by defining a test program with code to test your three new methods from parts (a) and (b) using suitable test data.

To help with the testing amend the existing `LinkedList.printList()` method by using your `print()` method instead of `ToString()`.

Make sure that you also test when there is an empty list.

Exercise 2.

(a) With reference to the doubly linked list class `DLinkedList` class (`DLinkedList.cs`) used in the week 3 Lecture, see the current incomplete definition of the delete item method:

```
public bool DeleteItem( Object deleteItem ) { … }
```

Complete the method's definition by adding code at the 3 places in the method labelled by:

```
 // **************************************************
 // ****** TUTORIAL EXERCISE: INSERT CODE HERE ******
 // **************************************************
```

(b) Test your implementation by adding code to your test program to test your completed deletion operation. Remember to test the 3 possible deletion cases, see the comments in the code.

OPTIONAL Exercise 3.

See the Transport for London (TfL) tube map for details: https://tfl.gov.uk/maps/track/tube

(a) Use the .NET Framework Library LinkedListNode<T> and LinkedList<T> classes from System.Collections.Generics namespace and your TubeStation class from the Week 1 Tutorial to create an UndergroundLine class to represent a London underground tube line.

The class should have the following data members:

`Name`	- the name of the tube line
`Line`	- the list of the tube stations on the line.

(b) The UndergroundLine class should have the following methods:

constructor	- to create a named line with no stations
`addStation( TubeStation station )`	- add the station to the end of the tube line list
`printStationInfo(  string station )`	- print information about the tube station
`stationsOnLineFirstToLast(  )`	- prints stations names from the first to last stations
`stationsOnLineLastToFirst(  )`	- prints stations names from the last to first stations
`isOnLine( string station )`	- checks if the station is on the line

Hint: see the APIs for the LinkedListNode<T> and LinkedList<T> classes.

(c) Add additional code to your test program to:

(i) Create an instance of your `UndergroundLine` class for the Northern Line.

(ii) Add the 6 Northern Line tube stations between Warren Street and Embankment inclusive to the line.

(iii) Print the stations in order for both directions

(d) Can you think of any other data members or methods that would be useful to add to your `UndergroundLine` class? If yes then add them.