Tutorial #10 Exercises

Linked Lists

Week of November 10 - 14

Exercise #1: Given a sorted singly linked list, write a method which takes an element and returns the list with that element inserted such that the list is still sorted.

$$< 1, 2, 5, 6, 8, 9 > +7 \rightarrow < 1, 2, 5, 6, 7, 8, 9 >$$

Hint: you should use the following specification to create this function

```
public static LinkedList<Integer>
    insertSorted(LinkedList<Integer> list, Integer number) {
```

Exercise #2: Given a singly linked list, write a method which takes two index values and swaps the values contained within those indices in the list. For this exercise, just use a list of integers.

$$< 1, 2^*, 5, 6, 8^*, 9 > \rightarrow < 1, 8, 5, 6, 2, 9 >$$

Hint: you should use the following specification to create this function

```
public static LinkedList<Integer>
    swap(LinkedList<Integer> list, int first, int second) {
```

Exercise #3: Given a singly linked list, write a method which traverses the list searching for a specific element. If that element is found, then the method returns *true*, otherwise it returns *false*.

Hint: you should use the following specification to create this function

```
public static boolean
    search(LinkedList<Integer> list, Integer term) {
```

Exercise #4: Let's try something new. A *doubly linked list* is a type of linked list where the nodes point both forward to the next element and backward to the previous element. Therefore the element at index 2 points back to index 1 but also points forward to index 3 (assuming it is not the tail!) Your job is to create your own implementation of a doubly linked list. Your class should have the ability to add, find, and delete elements from the list.

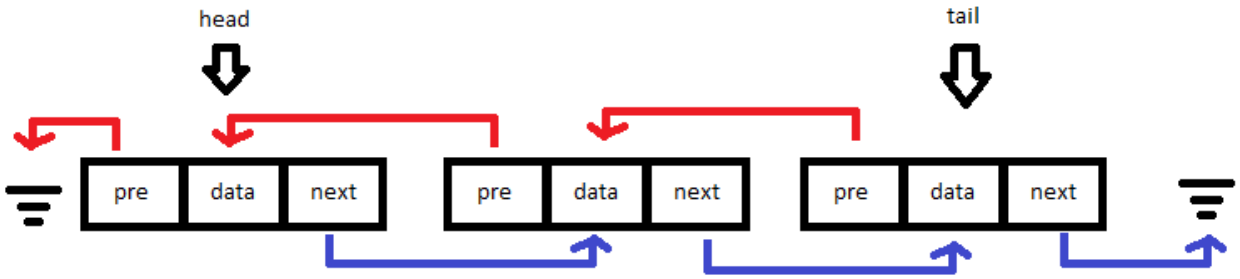

Figure 1: Illustration of a doubly linked list. Note the pointers to the previous element!

Hint: Start by creating a class *DoublyLinkedList* with a subclass *Node*. Use these two classes to implement the same linked lists we have been using, except with the added property that the nodes have information about the previous node in the list.

Exercise #5: There's more where that came from! A *circular linked list* is a type of linked list where the tail points around back to the head of the list. Your job is create your own implementation of a circular linked list and add the same functionality as above!

Hint: Start by creating a class *CircularLinkedList* with a subclass *Node*. Use these two classes to implement the same linked lists we have been using, except with the added property that the tail points to the head of the list. Not as hard as it sounds!