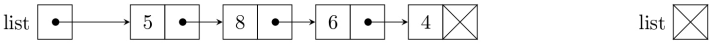
Questions

Data Structures & Algorithms Labs
Week 4: Lists and Linked Lists

These exercises will allow you to test your knowledge of the lecture material from week 3 (linked lists).

1 Theory Exercises

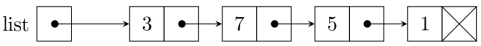
A **singly linked list** is an ordered sequence of elements each connected to its next one. Two examples, one non-empty and one empty, are



In a singly-linked list, the following commands are available:

- `list` points to the first node, if any. If there are no nodes in the list, `list` has the value END and trying to access a node through END will cause an error.
- `list.next` points to the second node, if any. Appending the command with another `.next` will output the third node if any, and so on;
- `e.value` returns the value of the node `e`;
- if `e` is the last element, `e.next` returns END.

► **1.1.** Consider the following singly linked list.



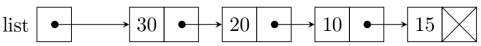
Do the following:

1. write the command that points to the node with value 3;
2. write the command that points to the node with value 5;
3. write the command that outputs the value of the node with value 3; and
4. write the command that outputs the value of the node with value 7. ◄

It is possible to remove any number of elements at the start of a singly linked list. This is done by changing the element the `list` node points to and uses the following command:

- `list=list.next` points the `list` node to the first element, if any, otherwise it causes an error. Appending the command with another `.next` will point it to the second element, if any, and so on.

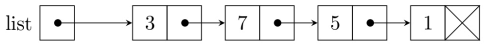
► **1.2.** Consider the following singly linked list.



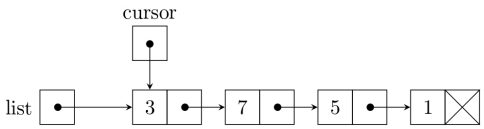
Do the following:

1. write the command that removes the node with value 30. Draw the resulting list; and
2. write the command that removes the nodes with values 30, 20 and 10. Draw the resulting list. ◄

Sometimes we need to remove a specific element from a linked list (e.g. the first, the last, or of a specific value). To do this, it is often useful to introduce a new pointer that traverses the list in order to find the desired element. For example, consider the following singly linked list.



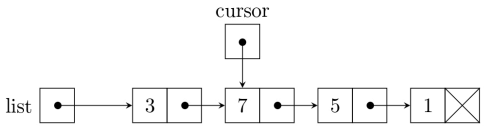
The command `cursor=list` introduces a variable named `cursor` that points to the first element of a list, if any, If the list is empty, then list will have the value END, and the assignment will assign END to `cursor`. The new list looks like this:



The new node can the be made to point at different elements of the list. For example

- `cursor=cursor.next` points the `cursor` to the next element of the one it is currently pointing, if any, otherwise returns error; appending the command with another `.next` will point it to the second element, if any, and so on.

The new list after `cursor=cursor.next` will look like this:



The new pointer can be used to check if an element has a desired property, without changing the position of the `list` node. Using this technique, we can execute the following commands:

- `delete_beg` deletes the first element of a non-empty list and
- `delete_end` deletes the last element of a non-empty list.

► **1.3.** Consider a singly linked list of arbitrary size. Write pseudo-code that executes `delete_beg`. You can use loops (e.g. if-else, while, for) but be careful when dealing with empty lists. ◄

► **1.4.** Consider a singly linked list of arbitrary size. Write pseudo-code that executes `delete_end`. You can use loops (e.g. if-else, while, for) but be careful when dealing with empty lists. ◄

2 Programming Exercises

Feel free to use the following `Node` class to implement the required data structures in these exercises.

```
public class Node {
    int value;
    Node next;
    Node(int value) {
        this(value, null);
    }
    Node(int value, Node next) {
        this.value = value;
        this.next = next;
    }
}
```

► **2.1.** Using the given `Node` class, or otherwise, implement a linked list of integers and define the following operations on it:

- Return the value at the beginning of the linked list

- Add a new node with a given value at the beginning of the linked list.
- Remove the node at the beginning of the linked list.
- Print all the values in the linked list.
- Return the number of nodes in the linked list.

You're free to solve this exercise however you like - such as with recursive functions acting on `Nodes`, or by defining a `myLinkedList` class that contains a node and the given operations as methods. ◀

▶ **2.2.** Given a single linked list of length `n` find its middle element. What is the (average/worst case) time and space complexity of your solution? ◀

▶ **2.3.** Let the `List` ADT be an ordered sequence of integers with the following operations (represented as a Java Interface):

```
public interface myList {

    boolean isEmpty();
    int size();
    void print();

    // Return the value at a given position
    int getFirst(int value);
    int getLast(int value);
    int getAt(int value, int index);

    // Insert a value at a given position
    void insertFirst(int value);
    void insertLast(int value);
    void insertAt(int value, int index);

    // Delete the value at a given position
    void deleteFirst();
    void deleteLast();
    void deleteAt(int index);
}
```

Using the given `Node` class, or otherwise, implement the operations above with a linked list. You're free to solve this exercise however you like - such as with recursive functions acting on `Nodes`, or with a `myLinkedList` class that contains a node and the given methods. If you want to create a class which implements the above interface then you can do so as follows (make sure to override every abstract method or you'll get errors):

```
public class myLinkedList implements myList {
    myLinkedList(); // Constructor, very important!

    public boolean isEmpty() {
        // Override abstract method with implementation
    }

    // Override rest of abstract methods
}
```

◀