Data Structures & Algorithms Problem Sheet 2

THIS ASSIGNMENT WILL BE CONVERTED TO AUTO-MARKING, AS ANNOUNCED IN THE LECTURE. HOWEVER, PLEASE START AS EARLY AS POSSIBLE – YOU WILL BE ABLE TO USE YOUR ANSWERS FAIRLY EASILY FOR THE CONVERTED ASSIGNMENT ONCE THE CONVERSION IS DONE.

Worth 7% of your final grade
Due Monday 22/11, 7pm in Moodle
Submit the following files in a single ZIP archive to Moodle:
`LinkedList.java`, `DoublyLinkedList.java`, `SkipList.java`, `SkipListNode.java`

Motivation: References (or "pointers") are everywhere in software systems. And they are tricky! They are the most common source of errors in programs (e.g. the dreaded `NullPointerException`). That's why knowing how to work with them is incredibly valuable. This coursework will give you good practice and also give you the opportunity to achieve pointer mastery with skip lists. Skip lists are not easy. But they are a great advanced exercise – if you are aiming for a first in D&A give them a try!

Note:

- If you have questions about this coursework please ask them in our Moodle forum. You will get answers more quickly as the whole teaching team can answer, and you are helping your classmates.
- Marks will be given based on correctly working code and correct answers submitted. You do not need to comment your code, unless there is a problem with it (then comments may give you partial marks).
- Do not copy code or answers from others. More about plagiarism and how to avoid it here: http://www.bath.ac.uk/library/help/infoguides/plagiarism.html
- If you submit code that does not compile, you can only get a maximum of 60% of the marks for it. If your code does not work, it is much better to make it compile and leave comments why you think it does not work in the code.
- For this problem sheet you should implement your own algorithms and data structures and not use pre-defined ones such as the ones from the Java API. You may use any type of array, e.g. `Object[]`. You may not use predefined data structures such as `ArrayList` or API methods such as in `java.util.Arrays`.
- Do not change the method signature (i.e. parameters, name, types) when implementing a given method. Changing the method signature may make it impossible for the marker to test it correctly. **Change only the parts in the given files that are marked for your code.**
- You can assume that your methods will only be used with correct inputs, i.e. your code does not need to handle errors caused by incorrect inputs.
- Please respect the Java conventions for naming of methods and classes, e.g. the use of upper/lower case: http://www.oracle.com/technetwork/java/codeconventions-135099.html
- Please ensure that the Java files you submit are recognized as correct text files (beware of copy-pasting special characters from the lecture slides as they can confuse the Java compiler).
- If you are struggling with Java or programming tools such as IntelliJ, have a look at the Topic 0 resources such as the video tutorials on YouTube. Many students found them useful.
- After submitting, please **check your submission on Moodle to make sure it is there and you have submitted the right files!!!** Also, have you submitted them to the right assignment?

- We aim to provide feedback no later than three weeks after the submission date.
- If you need a deadline extension, please apply to your Director of Studies. Please email submissions with extended deadline to the lecturer instead of submitting them on Moodle.

Question 1: Singly Linked Lists (35% of marks)

a) (15% of marks) Consider the Java source code for a singly linked list on the slides of the lecture. Based on the given source code, create a class `LinkedList` which implements the following functions, as illustrated in the lecture: `addFirst` (adding an element at the beginning of the list), `get` (getting the i^{th} element of the list), `insert` (inserting a given element at a given index position) and `remove` (removing the element at a given index position). Tip: simply copy and paste the code from the lecture slides. Your implementation should use the class `ListNode` from the lecture, which you can download from Moodle. On Moodle there is an executable class `LinkedListTestA`, which you can use to test your implementation.

b) (10% of marks) As suggested in lecture 8, add a `tail` reference to class `LinkedList` to make inserting at the end faster. Add a new method called `add` to the class, which inserts an element at the end of the list. Tip: you need to consider the special case that the list is empty, similar to the way we did in the lecture for the `insert` method. On Moodle there is an executable class `LinkedListTestB`, which you can use to test your implementation.

c) (10% of marks) The `tail` reference from part b) needs to be updated whenever the last element in the list is changed. Add code to the methods `addFirst`, `insert` and `remove` so that they update the `tail` reference correctly if the last element was added, inserted or removed. On Moodle there is an executable class `LinkedListTestC`, which you can use to test your implementation.

Submit your file `LinkedList.java` to Moodle. No other files need to be submitted for this question (the `ListNode` class does not need to be submitted).

Question 2 on next page…

Question 2: Doubly Linked Lists (30% of marks)

a) (10% of marks) Implement a Java class `DoublyLinkedList` which implements a doubly linked list, as described in lecture 8. The class should make use of the following class `ListNode2` which can be downloaded from Moodle:

```
class ListNode2 {
    Object element;
    ListNode2 prev = null;
    ListNode2 next = null;

    ListNode2(Object e, ListNode2 p, ListNode2 n) {
        element = e;
        prev = p;
        next = n;
    }
}
```

To start with, `DoublyLinkedList` should have the following two methods: `addFirst` (adding an element at the beginning of the list) and `get` (getting the i^{th} element of the list). Tip: the code for this is similar to your solution for question 1 a) – you only need to update the `prev` pointer of the old first element (if there was one) in the `addFirst` method. On Moodle there is an incomplete file `DoublyLinkedList.java` with a `print` method and an executable class `DoublyLinkedListTestA`, which you can use to test your implementation.

b) (10% of marks) Add an `insert` method to `DoublyLinkedList`, which inserts a given element at a given index position. Tip: the code for this is similar to your solution for question 1 a). On Moodle there is an executable class `DoublyLinkedListTestB`, which you can use to test your implementation.

c) (10% of marks) Add a `remove` method to `DoublyLinkedList`, which removes the element at a given index position. Tip: the code for this is similar to your solution for question 1 a). On Moodle there is an executable class `DoublyLinkedListTestC`, which you can use to test your implementation.

Submit your file `DoublyLinkedList.java` to Moodle. No other files need to be submitted for this question (the `ListNode2` class does not need to be submitted).

Question 3 on next page…

Question 3: Skip List (35% of marks)

Given the following incomplete Java class `SkipList`, which implements a skip list of lexicographically ordered `String`s and can be downloaded from Moodle:

```
class SkipList {
  private SkipListNode[] head;
  private int n = 0; // list size

  public SkipList() {
    // TODO implement this
  }

  public void createTestList() {
    // TODO implement this
  }

  public void print() {
    // TODO implement this
  }

  public boolean inList(String s) {
    // TODO implement this
    return false;
  }
}
```

a) (5% of marks) Implement the `SkipListNode` class and the `SkipList` constructor, which creates a new skip list, as described in the lecture. For this implementation, the maximum number of lanes in the skip list should be 5.

b) (5% of marks) Implement method `createTestList`, which should set up a skip list with 5 `String` elements "Anne", "Ben", "Charlie", "Don" and "Ernie" programmatically for testing. This test skip list should have three lanes: the lowest (i.e. the slowest) connecting all elements, the next one up connecting "Anne", "Charlie", and "Ernie", and the highest (fastest) one connecting "Anne" and "Ernie". Note you need to add another constructor to `SkipListNode` that allows you to set the number of "lanes" a new node should have, so you can build the test list exactly as specified. Tip: The array `head` contains the references for the start of the different lanes, e.g. `head[0]` (the first pointer for the slowest lane) should point to the first `SkipListNode` if the list is not empty. The `SkipList` class contains a main method which you can use for testing.

c) (5% of marks) Implement the `print` method, which prints out the list elements of each of the lists that make up the skip list, in ascending order. `print` should first print out the elements of the highest lane (i.e. the one with the fewest nodes in it), separated by commas and without line breaks. It should then proceed downwards to the lower lanes analogously, starting a new line for each lane. For example, the first line printed should be: `Anne, Ernie`

d) (20% of marks) Implement the `inList` method, which checks whether the given object `o` is part of the list and returns true if it is, otherwise false. The method should make use of the different lanes to optimize runtime, as discussed in the lecture. Tip: use the `compareTo` method of `String`.

Submit your files `SkipList.java` and `SkipListNode.java` to Moodle.