COMP108 Data Structures and Algorithms
Lab Exercises (Week 6)

Due: 19 March 2021, 5:00pm

Information

- Submission: Submit the file COMP108W06.java to SAM https://sam.csc.liv.ac.uk/COMP/CW_Submissions.pl?qryAssignment=COMP108-16
- Submission of lab/tutorial exercises contributes to 10% of the overall module mark. Submission is marked on a pass/fail basis - you will get full marks for submitting a *reasonable attempt.*
- Late submission is **NOT** possible. Individual feedback will not be given, but solutions will be posted promptly after the deadline has passed.
- These exercises aim to give you practices on the materials taught during lectures and provide guidance towards assignments.
- Relevant lectures: **Lecture 11, Video 1**
- You can refer to the guidance on how to use the web-based IDE https://ide.cs50.io/.

1. **Programming — Preparation**

   (a) Download three java files “COMP108W06App.java”, “COMP108W06.java” and “Node.java” from Canvas via the link “Labs & Tutorials” → “Week 6”.

   (b) Compile the programs by typing first **javac COMP108W06.java** and then **javac COMP108W06App.java**. There should be two files created: COMP108W06.class and COMP108W06App.class.

   (c) Run the program by typing **java COMP108W06App**.

   (d) **Every time you have edited COMP108W06.java, you have to (i) recompile by javac COMP108W06.java and then (ii) run by java COMP108W06App.**

2. **Linked List** This week we will work with basics of linked list mainly to **traverse the list** to report existence of key, counting occurrences, finding maximum/minimum.

   - An object `Node` is defined in Node.java to have three attributes: `data`, `next`, `prev`.
   - An object `COMP108W06` is defined in COMP108W06.java to have two attributes: `head`, `tail`, which point to the head and tail of a list.
   - Several auxiliary methods have been implemented to help, including
     `public void insertHead(Node newNode)` insert newNode to head of list;
     `public void insertTail(Node newNode)` insert newNode to tail of list;
     `public String headToTail()` goes through the list and return a String containing the data of each element of the list from head to tail;
     `public String tailToHead()` goes through the list and return a String containing the data of each element of the list from tail to head;

- The methods headToTail() and tailToHead() are meant only to convert the list to String for easy display. Do not use them to carry out your tasks.

Study these methods to see how to go through the linked list.

3. **Task 1: sequential search**

   The method seqSearchList() takes a parameter key and aims to find if key exists in the list. It should return true if exists and false otherwise.

   Complete the method (without changing its signature) and test it using test cases stated at the end of the document.

   Remarks: You are expected to go through the list using sequential search on list. Do not convert the list into an array to process it. Also, do not use the split method of the String class or the parseInt method of the Integer class to work on the String returned by headToTail() or tailToHead(). The latter two methods are only meant to ease printing from the list to help debugging.

4. **Task 2: counting occurrences**

   The method countList() takes a parameter key and aims to find the number of times key appears in the list. It should return the count as an integer. Return 0 if the key does not exist in the list.

   Complete the method (without changing its signature) and test it using test cases stated at the end of the document.

   The remarks in Task 1 also applies here.

5. **Task 3: finding minimum and maximum**

   The methods searchMin() and searchMax() aim to return the smallest and largest data, respectively, in the list.

   Note that it uses two values Integer.MAX_VALUE and Integer.MIN_VALUE.

   Complete the methods (without changing their signatures) and test them using test cases stated at the end of the document.

   The remarks in Task 1 also applies here.

6. **Test cases:**

input			expected return values			
# of int.	input integers	key	seqSearchList	countList	searchMin	searchMax
3	0 10 -10	10	true	1	-10	10
3	0 10 -10	20	false	0	-10	10
6	20 10 100 20 100 20	20	true	3	10	100
6	20 10 100 20 100 20	100	true	2	10	100
6	20 10 100 20 100 20	10	true	1	10	100
6	20 10 100 20 100 20	-20	false	0	10	100