COSC1285/2123: Algorithms & Analysis
Laboratory 1

Topic

Introduction to laboratory environment, debugging, linked lists and coding to a prescribed interface.

Objective

- Become familiar with the departmental programming environment.
- Be able to compile and modify a Java template.
- Understand and implement a singly linked list.
- Understand and implement a doubly linked list.

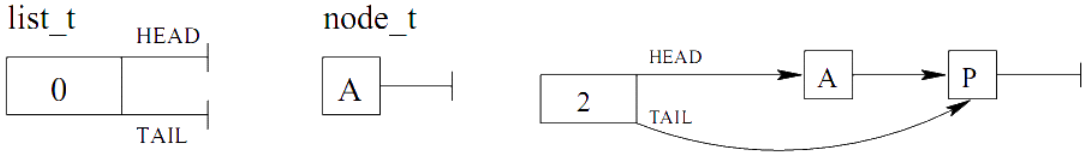
Environment

- Look at last week's lab if you are unfamiliar with Unix and compiling Java on the core teaching servers.

Linked Lists

Linked list is a basic type of data structure commonly used to implement abstract data types of collections, such as sequences and adjacency lists.

As a data structure, it consists of a group of **nodes**, where each node stores an element of the collection (see Figure 1a, right figure). In addition, each node contains a reference to the next node in the list. For example, in Figure 1b, we have a linked list representing a collection of two elements, 'A' and 'P'. Node **A** is a node in the linked list that contains the data **A** and is pointing to node **P**. If a node is the only one or the last element in the list, as it has no "next" element, it can contain a NULL reference (not pointing to any other node). In Figure 1a, node **A** contains a NULL reference as it is the only node in the list. Typically, a list is accessed through a structure (e.g., structure *list_t* in left figure of Figure 1a). Usually, the list structure holds three types of information: (i) the number of nodes in a list, (ii) **HEAD** - the reference to the first node, (iii) **TAIL** - the reference to the last node. In Figure 1b, **HEAD** is pointing to the first node **A** and **TAIL** is pointing to the last node **P**.



(a) Empty list (list_t) data structure with first node (node_t) A

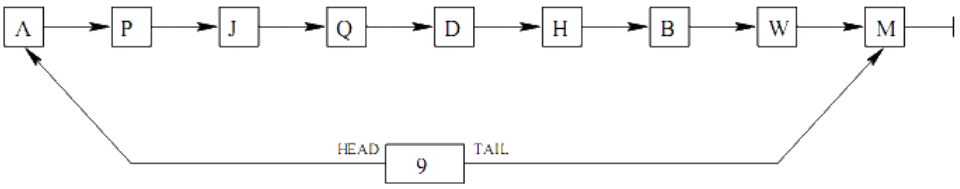
(b) List with two nodes *A* and *P*

Figure 1: Examples of linked list.

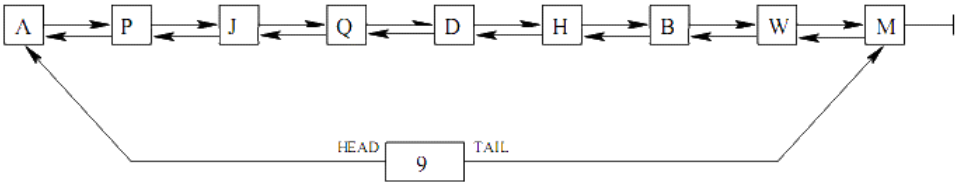
There are two main types of linked list implementations – singly linked list and doubly linked list. In singly linked list, each node has one pointer/reference to the next node in

the list (see Figure 2a). In Figure 2a, the **HEAD** is pointing to the first node **A** and TAIL is pointing to the last node **M**. Singly linked list enables fast insertion (at head or tail). However, search can be slow as it may require traversing all the nodes from **HEAD** to **TAIL** in a singly link list .

In doubly linked list, each node has two pointers/references, one to the next node and the other to the previous node. In figure (d), the node **D** has the reference to its next node **H** and the reference to its previous node **Q**. The doubly linked list uses more space (need to store the extra pointer/reference per node), but for some operations can be faster (which operations are for you to find out in this lab) – this is an example of space-time tradeoff.



(a) A singly linked list with nine nodes



(b) A doubly linked list with nine nodes

Figure 2: Examples of singly and doubly linked list.

Implementation

Although Java contains many inbuilt data structures, it is very informative to implement some of the basic data structures ourselves.

Download the Java source code from `week 2` module in Canvas. It contains a simple program for typing list operations from stdin, that are then executed on either a singly linked list (only forward pointer/reference per node) or doublyLinkedList (have both forward and backward pointers/references).

file	description
`LinkedListDemo.java`	Code that reads in list operations from stdin then executes those on the list. No need to modify this file.
`MyList.java`	Interface for list operations. No need to modify this file.
`SimpleLinkedList.java`	Code that implements a singly linked list. Complete the implementation (implement parts labelled "YOUR IMPLEMENTATION").
`DoubleLinkedList.java`	Code that implements a doubly linked list. Complete the implementation (implement parts labelled "YOUR IMPLEMENTATION").

Try to compile the code using the following command:

```
javac *.java
```

We suggest for you to use the core teaching servers to complete these labs, as your assignments will be partially tested on these servers. However, if you are familar with using Java on the core teaching servers, please feel free to use your favourite environment for development and testing. Remember the testing code uses input from the command prompt.

Exercise

The java file for the singly linked list (`SimpleLinkedList.java`) has most of the functionality implemented. Your task is to complete the implementation for the two `remove` methods. You are to code these methods for a singly linked list. You should not change the interface (types, method names, return types, parameters).

Afterwards, open and examine the skeleton for a doubly linked list (`DoubleLinkedList.java`). Implement the two `add` methods, two `remove` methods and the `reversePrint` method.

Afterwards, use the interactive interface of LinkedListDemo to test your implementations. To test the singly linked list, type:

```
java LinkedListDemo single
```

To test the doubly linked list, type:

```
java LinkedListDemo double
```

Questions

- Which type of linked list is faster for printing the list in reverse (the reversePrint() method)?
- For an unsorted douby linked list, could you implement a faster search than the one provided?

Discuss with your demonstrator if you are unsure.