Lab Manual

**CPCS204**

**Data Structures 1**

**1433/1434H**

**Lab - 3**

 **Learning Procedure**

1. Stage **J** (**Journey inside-out the concept**)
2. Stage **a1** (**apply the learned**)
3. Stage **v** (**verify the accuracy**)
4. Stage **a2** (**assess your work**)

**Laboratory 3:**

**Statement Purpose:**

This lab will give you practice with **linked lists**.

**Activity Outcomes:**

This lab teaches you the following topics:

* A linked list
* Implementation of linked list using java
* Insert Node at the beginning of the list
* Traverse List
* Delete Node from a list
* Doubly Linked List Implementation
* Circularly Linked List Implementation

**Instructor Note:**

As pre-lab activity, review Ch3, from the book Data Structures with Java by John R. Hubbard and also the relevant instructor’s slides.

**Names I.D.**

1. **.……………..………………………………. ………………………………**
2. **..…………………………………………….. ………………………………**
3. **.……………………………………………... ………………………………**
4. **.…………………………………………….. ..…………………………….**
5. **Stage J (Journey)**

**A linked list** is just a chain of nodes, with each subsequent node being a child of the previous one. Many programs rely on linked lists for their storage because these don't have any evident restrictions. For example, the array list we did earlier could not grow or shrink, but node based ones can! This means there is no limit (other than the amount of memory) on the number of elements they can store.

1. **Singly Linked Lists**

**Basics:**

* + - * **A *singly linked* list is a concrete data structure consisting of a sequence of nodes**
  + It has a *head* (or *first*) node pointer indicating the first node in list
  + It could have optionally a *tail* pointer node indication the last node in list
    - * **Each node stores**
  + Element (data)
  + Link to the next node
    - * **The null object (at the end) is denoted as ∅.**



1. **Doubly Linked Lists**

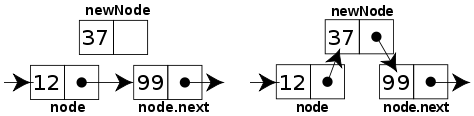
**Basics:**

* + - * **The *doubly linked* list allows us to go in both directions in a linked list:**
  + Forward
  + Reverse
    - * **Such lists allow for**
  + A variety of quick update operations, including insertion and removal at both ends, and in the middle.
    - * **A node in a doubly linked list stores two references**
  + **A next link**, which points to the next node in the list.
  + **A prev link**, which points to the previous node in the list.

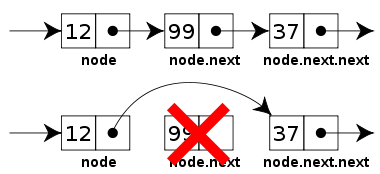


**The common operations of Singly/Doubly linked list are:**

1. **Insertion (or Add)**:
   1. Add first
   2. Add last
   3. Add after existing node e.g.:

****

1. **Deletion (or Remove)**:
   1. Delete first
   2. Delete last
   3. Delete after existing node e.g.:



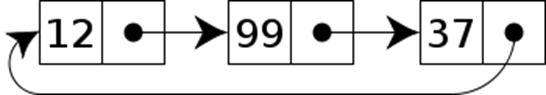
**Some common handling methods of Singly/Doubly linked list are:**

1. **Print (or show)**:
   1. Print all list elements
   2. Print certain node
2. **Search for an element**
3. **Find list size (if no size variable in list class)**
4. **Revers the linked list**
5. **Circular Linked Lists**

**Basics:**

* + - * **A *circularly linked* list has the same kind of nodes as a singly linked list.**

**cursor**



* + - * Each node in a circularly linked list has a next pointer and a reference to an element.
      * But there is no head (first) or tail (last) in a circularly linked list.
      * For instead of having the last node's next pointer be null, its last node points back to the first node.
      * Still there is some node to be marked as a special node used for traverse, which we call the ***cursor*** (or current), allows having a place to start form.
      * This cursor is used to remember this starting point, while a traversal of a circularly linked list it indicates the return to the starting point.
      * **By using cursor node, some simple update methods can be defined:**
  + **insert(*v*)**: Insert a new node *v* immediately after the cursor; if the list is empty, then *v* becomes the cursor and its next pointer points to itself.
  + **delete()**: Remove and return the node *v* immediately after the cursor (not the cursor itself, unless it is the only node); if the list becomes empty, the cursor is set to **null**.
  + **advance()**: Advance the cursor to the next node in the list.

1. **Stage a1 (apply)**

**Activity 1:**

Apply and test the Linked List implementation bellow:

// the code below is a simple example of a linked list that inserts a new link at the beginning of the list, deletes from the beginning of the list and loops through the list to print the links contained in it.

public class Node **{** **//Start of class Node**

public int data1;

public double data2;

public Node next;

**//Link constructor**

public Node (int d1, double d2) {

data1 = d1;

data2 = d2;

next=null;

}

**//Print Link data**

public void printLink() {

System.out.print("{" + data1 + ", " + data2 + "} ");

}

**} //End of class Node**

public class LinkList **{** **//Start of class LinkList**

private Node first;

//LinkList constructor

public LinkList() {

first = null;

}

//Returns true if the linked list is empty

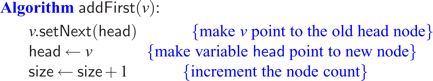
public boolean isEmpty() {

return first == null;

}

Now we must create the actual linked list by writing a method that will not only construct the node but put it at the end of the list (set the last node's next pointer to the new node). We already have a reference to the first and last node so by using that information we can tell one of two conditions: that the node we are creating is the first node or it is not. So:

//Inserts a new node at the first of the linked list



public void addFirst(int d1, double d2) {

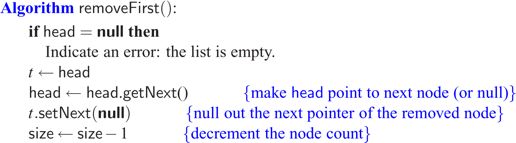
Node node = new Node(d1, d2);

node.next = first;

first = node;

}

//Deletes the node at the first of the linked list



public Node deleteFirst() {

Node temp = first;

first = first.next;

return temp;

}

//Prints the linked list data

public void printList() {

Node currentNode = first;

System.out.print("List: ");

while(currentNode!= null) {

currentNode.printLink();

currentNode = currentNode.next;

}

System.out.println("");

}

**} //End of class LinkList**

// Main LinkListTest Class

public class LinkListTest **{ //Start of class LinkListTest**

public static void main(String[] args) {

LinkList list = new LinkList();

list.addFirst(1, 1.01);

list.printList();

list.addFirst(2, 2.02);

list.printList();

list.addFirst(3, 3.03);

list.printList();

list.addFirst(4, 4.04);

list.printList();

list.addFirst(5, 5.05);

list.printList();

while(!list.isEmpty()) {

Node deletedLink = list.deleteFirst();

System.out.print("deleted: ");

deletedLink.printLink();

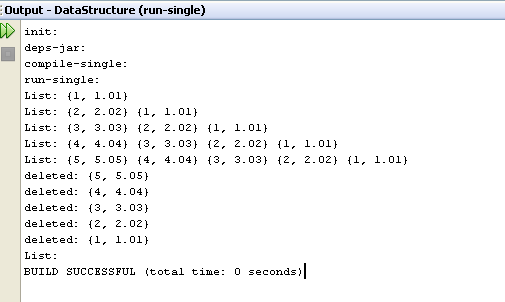
System.out.println("");

}

list.printList();

}

**} //End of class LinkListTest**



**Task 1: create three** Node**,** LinkList **and** LinkListTest **using the code of each below.**

**Task 2: write and test a method** public void addLast(int d1, double d2)  **to add the node to the last of the linked list.**

**Task 3: write and test a method** public Node deleteLast() **to remove the last node of the linked list.**

**Task 4: write and test a method** public int size() **to count the number of nodes in the linked list.**

**Task 5: write and test a method** public Boolean search(Node n)  **to find out whether the given data exists or not in the linked list.**

**Task 6: write and test a method** public void addAfter(int d1, double d2, Node n)  **to add the node after given node of the linked list.**

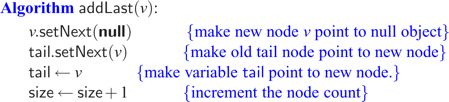
**Task 7: write and test a method** public Node deleteAfter(Node n)  **to delete the node after given node of the linked list.**

**EXTA TASKS:**

**Task 8: write and test a method** public void directAddLast(int d1, double d2)  **to add the node to the last of the linked list as given in the pseudo code below.**

//Inserts a new Link at the last of the list

**Algorithm directAddLast(v)**



**NOTE:** Instead of, A method called **directAddLast()** exists, but it runs in linear time, because every time it is called, it walks down the list to find the end. Without changing the meaning of this method or any other, modify the representation of a LinkedList and whatever methods are necessary to make **directAddLast ()** run in constant time.

Your LinkedList class will need to continually maintain a record of the last (**tail**) list in a LinkedList, and all LinkedList’s methods will have to ensure that this record stays current.

**Task 9: write and test a method** public void directDeleteLast()  **to delete the last node of the linked list directly using (tail) as used in Task 8 above.**

**Task 10: write and test a method** public void swapNodes(Node n)  **to search for a node containing a given data (as search keyword) and then swap this node with the next node. E.g. if the list 2->5->7->9 the call of the method swapNodes(5) will rearrange the list as 2->7->5->9, Note that each digits presents a complete node NOT the data saved inside.**

**Task 11: write and test a method** public void reverse() **to reverse the order of the nodes in the linked list. E.g. if the list a->b->c->d the call of the method reverse() will rearrange the list as d->c->b->a.**

**Activity 2:**

Apply and test Circular Linked List **(**you can name it **CLinkList)** using the previous implementation of class Node then implement the following methods:

1. **insert(*v*)**: Insert a new node *v* immediately after the cursor; if the list is empty, then *v* becomes the cursor and its next pointer points to itself.
2. **delete()**: Remove and return the node *v* immediately after the cursor (not the cursor itself, unless it is the only node); if the list becomes empty, the cursor is set to **null**.
3. **advance()**: Advance the cursor to the next node in the list.
4. **printList()**: Print out the all nodes of this ClinkList;
5. Change the data type of the CircularLinkList content from the current data type to be as generic data type E.
6. **Stage v (verify)**

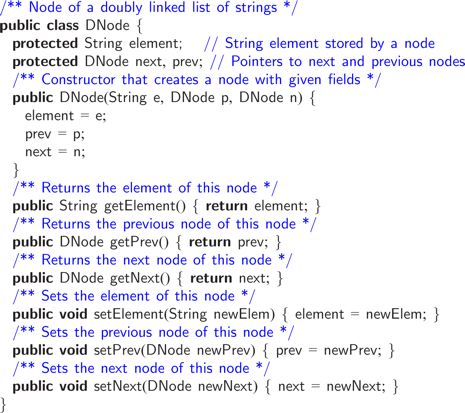
**Programming Exercise**

**Ex-1:**

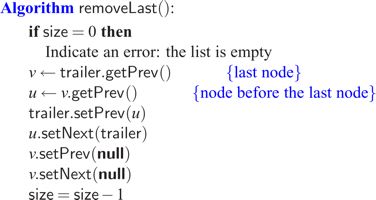
**Reapply Activity 1 but after changing class Node to be nested in LinkedList class then make needed change if any accordingly.**

**Ex-2 (home activity):**

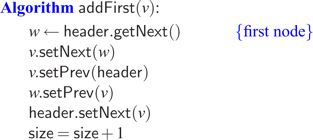
**Implement and test Doubly linked List using the following given Class:**



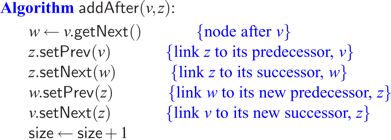
1. **Write java method to remove the last node from a doubly linked list**



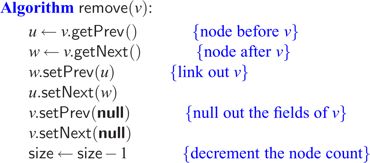
1. **Write and test a method** public Node removeFirst() **to remove the first node of the linked list.**
2. **Write and test a method** public Node removeAfter(String s) **to remove the first node of the linked list.**
3. **Write java method to Add a node to the first of a doubly linked list**



1. **Write and test a method** public void addLast(String s) **to add the node to the last of the linked list.**
2. **Write java method to Add after a node of a doubly linked list**



1. **Write java method to remove certain node from a doubly linked list**



**Task 3: write and test a method** public Node deleteLast() **to remove the node to the last of the linked list as given in the pseudo code below.**

**Task 4: write and test a method** public int size() **to count the number of nodes in the linked list.**

**Task 5: write and test a method** public Boolean search(Node n)  **to find out whether the given data exists or not in the linked list.**

**Task 6: write and test a method** public void addAfter(int d1, double d2, Node n)  **to add the node after given node of the linked list.**

**Task 7: write and test a method** public Node deleteAfter(Node n)  **to delete the node after given node of the linked list.**

1. **Stage a2 (assess)**

**Lab Work:**

In each laboratory you are assessed on your work within lab session based on your participation, discussions and achievement of lab activities. Thus, each lab has a portion of the (LAB WORK MARK). Therefore, a checklist of each lab is used to evaluate your work. This checklist accounts the following criteria:

* + - * Following the lab manual step by step
      * Answering given questions concisely and precisely
      * Practicing and implementing given examples correctly
      * Writing code of required programming tasks
      * Being focused, positive, interactive and serious during lab session
      * Asking good questions or answering instructor questions if any

**Note:** performing given home activities or extra programming is highly recommended to improve your understanding, capability and programming skills.