Lab 3: Linked-List

Objective(s)

This lab aims to:

- Introduce the linked list.
- Advantages of Linked List.
- Need of Linked List
- How to Create Linked List in Java
- How to deal with Linked List in case of: insertion, Deletion, searching

Tool(s)/Software

Java programming language with NetBeans IDE.

Description

Advantages of Linked List

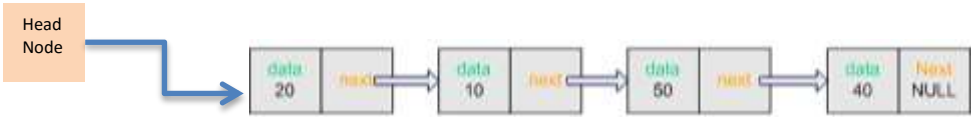
- Linked List is a Dynamic data Structure .Linked List can grow and shrink during run time.
- Insertion and Deletion Operations are easier compared to arrays.
- Efficient Memory Utilization ,i.e. no need to pre-allocate memory
- Linear Data Structures such as Stack, Queue can be easily implemented using Linked list

Need of Linked List

- Suppose you are writing a program which will store marks of 100 students in math. Then our logic would be like this during compile time –
  int marks[100];
- Now at run time i.e. after executing program if number of students are 101 then how you will store the address of 101th student ?
- Or if you need to store only 40 students then again you are wasting memory unnecessarily.
- Using linked list you can create memory at run time or free memory at run time so that you will able to fulfill your need in efficient manner

What's Linked-List?

- A linked data structure consists of capsules of data known as *nodes* that are connected via *links*
  - Links can be viewed as arrows and thought of as one way passages from one node to another
- In Java, nodes are realized as objects of a node class
  - The data in a node is stored via instance variables
- The links are realized as references
  - A reference is a memory address, and is stored in a variable of a class type
  - Therefore, a link is an instance variable of the node class type itself



How to Create Linked-List in Java

There are **3-steps** approach to create Linked-List in Java

Step 1: Declare class for the **Node** – forming the structure of the node

```
private static class Node<E>{
    private E element;
    private Node<E> next;
    public Node(E e, Node<E> n){
        this.element = e;
        this.next = n;
    }
    public E getElement(){
        return this.element;
    }
    public Node<E> getNext(){
        return this.next;
    }
    public void setNext(Node<E> n){
        this.next=n;
    }
}
```

Step 2: Declare the **SinglyLinkedList** class that includes the Node class.

```
public class SinglyLinkedList<E> {

    // local class for Node with <E>,a generic
    private static class Node<E>{...18 lines }
    //instance variables of the singlyLinkedLis
    private Node<E> head = null;
    private Node<E> tail = null;
    private int size = 0;
    public SinglyLinkedList(){}

    //Access methods
    public int size(){...3 lines }
    public boolean isEmpty(){...3 lines }
    public E first(){...4 lines }
    public E last(){...4 lines }
    public void display(){...11 lines }
    public void find(E e){...11 lines }

    //Update methods
    public void addFirst(E e){...6 lines }
    public void addLast(E e){...8 lines }
    public E removeFirst(){...9 lines }
```

Display Method:

```
Source  History
56      public void display(){
57          Node <E> Current;
58          Current = head;
59          int counter = 0;
60          System.out.println("\n ---------- display Method ----------");
61          while(Current!=null){
62              counter++;
63              System.out.println(" LinkedList( " + counter +" ): " + Current.getElement());
64              Current = Current.getLink();
65          }
66      }
linkedlist.LinkedList > main >
```

Step 3: Define the object of **SinglyLinkedList** class

```
SinglyLinkedList myList=new SinglyLinkedList();
myList.addFirst("IAU");
myList.addFirst(2322);
```

Note: *Please refer to lecture slides for the algorithms of the main operations on linked list.*

Tasks/Assignments(s)

1. Create SinglyLinkedList class. Apply all the following operations as discussed in the lecture:
    - **addFirst**: to add new node to the beginning of the linked list.
    - **addLast**: to add new node to the end of the linked list.
    - **addNode**: to add new node to a specific position in the linked list.
    - **removeFirst**: to remove the first node in the linked list.
    - **findNode**: to find a node with specific given value.
    - **removeNode**: to remove a specific node from the list.

2. (**HW**) Add a Person class that holds the name and address of a person with accessor and mutator methods. Then, test the class by creating a linked list of Person objects. Add and remove objects from the linked list using linked-list operations defined in Task 1.

Deliverables(s)

You are required to implement and deliver a Java program as described in the previous section.