National University of Computer & Emerging Sciences, Karachi

Computer Science Department

Fall 2022, Lab Manual – 04

Course Code: CL-2001	**Course : Data Structures - Lab**
Instructor(s) :	**Abeer Gauher, Sobia Iftikhar**

LAB - 4

Single Linked List

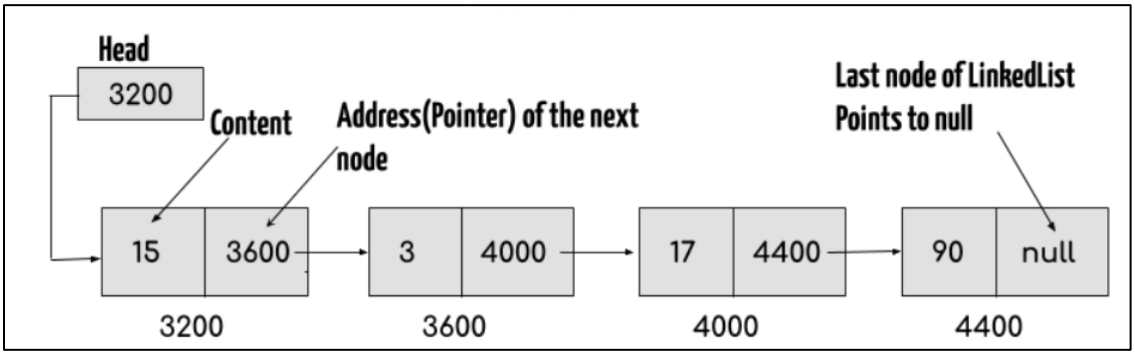
Linked List:

Like arrays, Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.

Why do we need a Linked List?

You must be aware of the arrays which is also a linear data structure but arrays have certain limitations such as:

- Size of the array is fixed
- Array elements need contiguous memory locations to store their values.
- Inserting an element in an array is performance wise expensive.
- Similarly deleting an element from the array is also a performance wise expensive operation because all the elements after the deleted element have to be shifted left

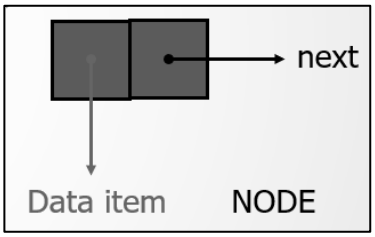


Types of Linked list

1. Single Linked list
2. Double Linked List
3. Circular Linked List
    a. Single Circular Linked List
    b. Double Circular Linked List

Implement a SinglyLinkedList class

- A singly linked list is a concrete data structure consisting of a series of nodes
- Each node stores
    - Data item
    - Link to the next node



Create A Linked List

we can represent a LinkedList as a class with its Node as a separate class. Hence this class will have a reference to the Node type.

```
class LinkedList {
   Node head; // list head
   class Node {
      int data;
      Node next;
      Node(int d) { data = d; }  //constructor to create a new node
   }
}
```

Implementation of single linked list

```
package DS;
class Node{
	int data;
	Node next;
}

public class SLIST {
	Node head;
	Node ptr;
	Node Ptrr;
	public void insert(int a) {

		Node newnode  = new Node();
		newnode.data = a;
		newnode.next =null;

		if (head==null) {
			head = newnode;
		}

		else {
			Node n = head;
			while(n.next != null) {
				n = n.next;
			}
			n.next = newnode;
		}
	}
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		SLIST sl = new SLIST();
		sl.insert(2);
		sl.insert(3);
		sl.insert(4);
		sl.insert(5);
	}

}
```

Traversal the linked list

```
	// Traversal
	public void SListPirnt() {
		Node temp = head;
		while(temp != null) {
			System.out.print(temp.data+" ");
			temp = temp.next;
		}
		System.out.println();
	}
```

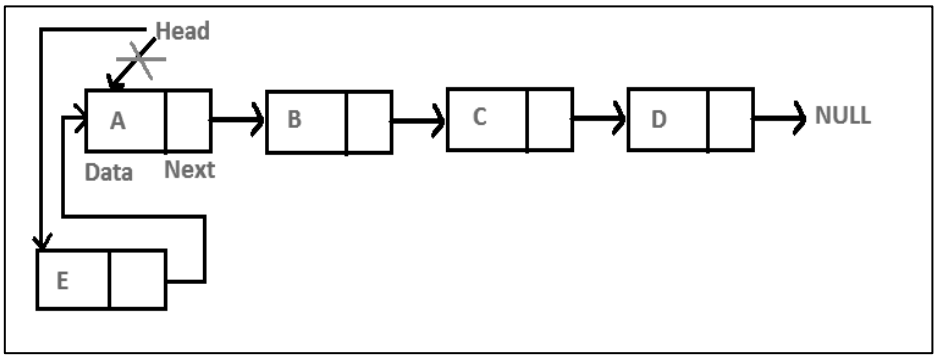
Add Node At Front

The new node is always added before the head of the given Linked List. And newly added node becomes the new head of the Linked List.

For example, if the given Linked List is 2->3->4->5 and we add an item 1 at the front, then the Linked List becomes 1->2->3->4->5.

Let us call the function that adds at the front of the list is push ().

The push () must receive a pointer to the head pointer, because push must change the head pointer to point to the new node.




```
// Append at Front
public void AtFront(int a) {

        Node newNode = new Node(); // Create Newnode
        newNode.data = a;   // set the data value
        newNode.next = head; // link the previous top node with new node
        head  = newNode; // update the head value now point out to newnode as  head
}
```

Add a node after a given node in a Singly Linked List

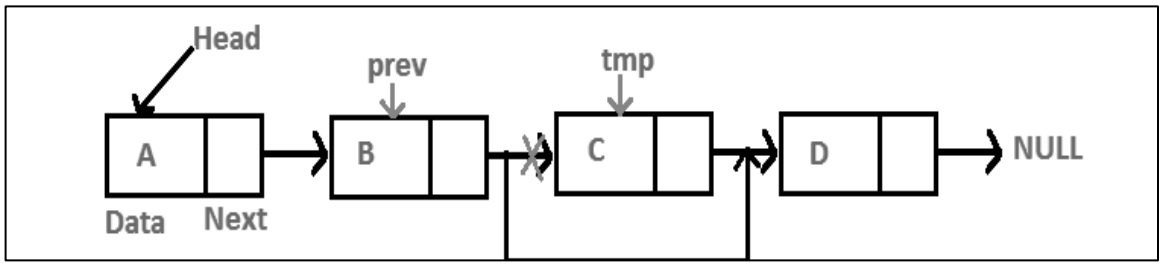
To insert a new node after some node, create a void function named insertNodeAfter () carrying two arguments, one for the key of the node after which the insertion is to be done, the new node.

```
public void insertNodeAfter(int newValue, int CounterValue) {
            Node Ptemp = head;
            while(Ptemp.data != CounterValue) {

                  Ptrr = Ptemp;
                  Ptemp  = Ptemp.next;
            }
            Node newNode = new Node();
            newNode.data = newValue;
            newNode.next = Ptemp.next;
            Ptemp.next = newNode;

     }
```

Delete any Node




```
// any node delete
public void AnyNodeDelete(int a) {
        Node Dtemp =head;

        while(Dtemp.data != a) {
                ptr = Dtemp;
                Dtemp = Dtemp.next;
        }
        ptr.next = Dtemp.next;
        Dtemp =  null; // free the memory

}
```

Task 01 :

a. Create a linked list from of first 10 even number.
b. Remove the 5^{th} even number.
c. Insert first odd number after the 4^{th} even number.
d. Swap the 5^{th} value with first value.
   Example:
   - 2, 4,6,8,10,12,14,16, 18, 20
   - 2, 4,6,8,12,14,16, 18, 20
   - 2, 4,6,8,1,12,14,16, 18, 20
   - 1, 4,6,8,2,12,14,16, 18, 20

Task 02 :

a. Test whether a particular item is in a linked list. So if the item is 4 and the list is [1,4,2,5], the method returns true; if the item is 6 and the list is [1,4,2,5], the method returns false.
b. Delete the first occurrence of an item from a linked list. So if the item is 7 and the list is [1,3,7,4,3,7,2], the result is [1,3,4,3,7,2].
c. Delete all occurrences of an item from a linked list. So if the item is 7 and the list is [1,3,7,4,3,7,2], the result is [1,3,4,3,2].
d. Delete the last occurrence of an item from a linked list. So if the item is 7 and the list is [1,3,7,4,3,7,2], the result is [1,3,7,4,3,2].
e. Merge two linked lists of numbers, but if a number occurs in both lists, only include it once in the final list. So if the lists are [1,3,4,7,12] and [1,5,7,9], the result is [1,3,4,5,7,9,12].

Task 03

Write a program that will invoke the method and take two input from user as length for two linked list, so that the nodes of the list passed as a parameter are inserted as every other node of the current list (i.e., first node of first list, first node of second list, second node of first list, second node of second list, ...)

Example input and output:

Input: 5 and 8
List 1: 0-> 2->4->6->8
List 2: 1->3->5->7->9->11->13->15
Output: 0->1->2->3->4->5->6->7->8->9->11->13->15

Input: 5 and 3
List 1: 0-> 2->4->6->8
List 2: 1->3->5
Output: 0->1->2->3->4->5->6->8

Input: 0 and 8
List 1:
List 2: 1->3->5->7->9->11->13->15
Output: 1->3->5->7->9->11->13->15

Input: 0 and 8
List 1:
List 2: 1->3->5->7->9->11->13->15
Output: 1->3->5->7->9->11->13->15