Data Structure

Section #3 "Linked List"

Arrays

Index	0	1	2	3	4
number	10	20	30	40	50

Advantages of arrays

- In an array, accessing an element is very easy by using the index number.
- The search process can be applied to an array easily.
- 2D Array is used to represent matrices.
- For any reason, a user wishes to store multiple values of similar type then the Array can be used and utilized efficiently.

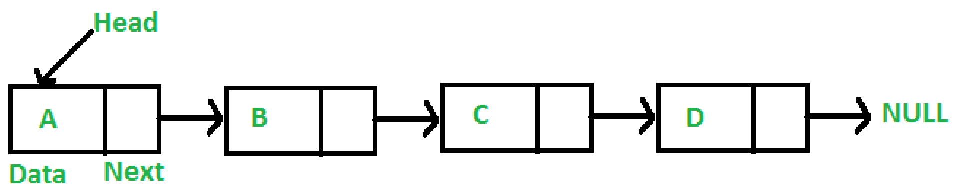
- **For example,** in a system, if we maintain a sorted list of IDs in an array id[].

id[] = [1000, 1010, 1050, 2000, 2040].

Disadvantages of arrays

- Array size is fixed.
- Array is homogeneous: The array is homogeneous, i.e., only one type of value can be store in the array.
- Array is Contiguous blocks of memory.
- Insertion and deletion are not easy in Array.

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:



In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

Advantages over arrays

1) Dynamic size
2) Ease of insertion/deletion

Drawbacks:

1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists efficiently with its default implementation.
2) Extra memory space for a pointer is required with each element of the list.
3)Not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Linked List Applications

- Dynamic memory allocation
- Implemented in stack and queue
- In **undo** functionality of softwares
- Hash tables, Graphs

Types of Linked List

Following are the various types of linked list.

- **Simple Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operations

Following are the basic operations supported by a list.

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Display** − Displays the complete list.
- **Search** − Searches an element using the given key.
- **Delete** − Deletes an element using the given key.

Representation:

A linked list is represented by a pointer to the first node of the linked list. The first node is called the **head**. If the linked list is empty, then the value of the head is **NULL**.
Each node in a list consists of at least two parts:

1. **data**
2. **Pointer** (Or **Reference**) to the next node s(C++).

In Java , we can represent a node using structures. Below is an example of a linked list node with integer data.
In Java, LinkedList can be represented as a class and a Node as a separate class. The LinkedList class contains a reference of Node class type.

1. Java program to implement LinkedList to add elements

```
class LinkedList {
  // create an object of Node class
  // represent the head of the linked list
  Node head;
  // static inner class
  static class Node {
    int value;
    // connect each node to next node
    Node next;
    Node(int d) {
      value = d;
      next = null;
    }
  }

  public static void main(String[] args) {

    // create an object of LinkedList
    LinkedList linkedList = new LinkedList();

    // assign values to each linked list node
```

```
    linkedList.head = new Node(1);
    Node second = new Node(2);
    Node third = new Node(3);

    // connect each node of linked list to next node
    linkedList.head.next = second;
    second.next = third;

    // printing node-value
    System.out.print("LinkedList: ");
    while (linkedList.head != null) {
      System.out.print(linkedList.head.value + " ");
      linkedList.head = linkedList.head.next;
    }
  }
}
```

Output

```
The linked list is: 1 2 3
```

2. Insert at the End

- Allocate memory for new node
- Store data
- Traverse to last node
- Change next of last node to recently created node

```
public int insertAtEnd(int data) {
    Node new_node = new Node(data);

    if (head == null) {
      head = new Node(data);
      return;
    }

    new_node.next = null;

    Node last = head;
    while (last.next != null)
      last = last.next;

    last.next = new_node;
```

```
    return;
  }
```

3. Insert at the Middle

- Allocate memory and store data for new node
- Traverse to node just before the required position of new node
- Change next pointers to include new node in between

```
public void insertAfter(Node prev_node, int data) {
    if (prev_node == null) {
      System.out.println("The given previous node cannot be null");
      return;
    }
    Node new_node = new Node(data);
    new_node.next = prev_node.next;
    prev_node.next = new_node;
  }
```

4. Delete from a given position

```
void deleteNode(int position) {
    if (head == null)
      return;

    Node node = head;

    if (position == 0) {
      head = node.next;
      return;
    }
    // Find the key to be deleted
    for (int i = 0; node != null && i < position - 1; i++)
      node = node.next;

    // If the key is not present
    if (node == null || node.next == null)
      return;

    // Remove the node
    Node next = node.next.next;

    node.next = next;
  }
```

5. Print Elements of list

```
public void printList() {
    Node node = head;
    while (node != null) {
      System.out.print(node.item + " ");
      node = node.next;
    }
  }
```

6. main Method

```
LinkedList llist = new LinkedList();

    llist.insertAtEnd(1);
    llist.insertAtBeginning(2);
    llist.insertAtBeginning(3);
    llist.insertAtEnd(4);
    llist.insertAfter(llist.head.next, 5);

    System.out.println("Linked list: ");
    llist.printList();

    System.out.println("\nAfter deleting an element: ");
    llist.deleteNode(3);
    llist.printList();
```

Task #3

Write a c++ program to delete element entered by user from linked list.

Write a c++ program to search an element entered by user using linked list.