# Lab 4: Linked List and its operations

# 1. Introduction

A linked list is a data structure that can store an indefinite amount of items. These items are connected using pointers in a sequential manner.

# Examples

1. A simple real life example is a Train, here each coach is connected to its previous and next coach (Except first and last). In terms of programming consider coach body as node value and connectors as links to previous and next nodes.

2. Consider the history section of web browsers, where it creates a linked list of web-pages visited, so that when you check history (traversal of a list) or press back button, the previous node's data is fetched.

# 2. Activity Time boxing

|  |  |  |  |
| --- | --- | --- | --- |
| **Task No.** | **Activity Name** | **Activity time** | **Total Time** |
| 1 | Introduction of Linked List | 10 mins |  |
| 2 | Basic Concept Link List and nodes | 10 mins |  |
| 3 | Insertion | 10 mins |  |
| 4 | Deletion | 20 mins |  |
| 5. | Lab Exercise. Demo | 140 mins. | 180 mins |

## 3. Lab Manual Lecture [Expected time = 40 minutes]

## 4. Objective

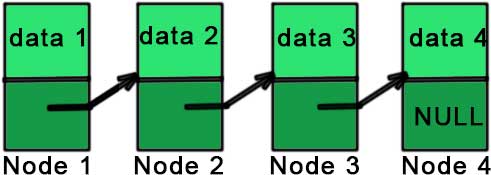
* Implementation of Link list
* Display Linked list
* Insertion
* Deletion

## Why Link List

We often face situations, where the data is dynamic in nature and number of data can’t be predicted or the number of data keeps changing during program execution. Linked lists are very useful in this type of situations.

## Implementation of Link List

The implementation of a linked list in C++ is done using pointers. A linked list is made up of many nodes which are connected in nature. Every node is mainly divided into two parts, one part holds the data and the other part is connected to a different node. It is similar to the picture given below.



Here, each node contains a data member (the upper part of the picture) and link to another node(lower part of the picture).

Notice that the last node doesn’t point to any other node and just stores NULL.

## Nodes

The elements of a linked list are called the nodes. A node has two fields

**1. Data**

The data field contains the data being stored in that specific node. It cannot just be a single variable. There may be many variables presenting the data section of a node.

**2. Next.**

The next field contains the address of the next node. So this is the place where the link between nodes is established.



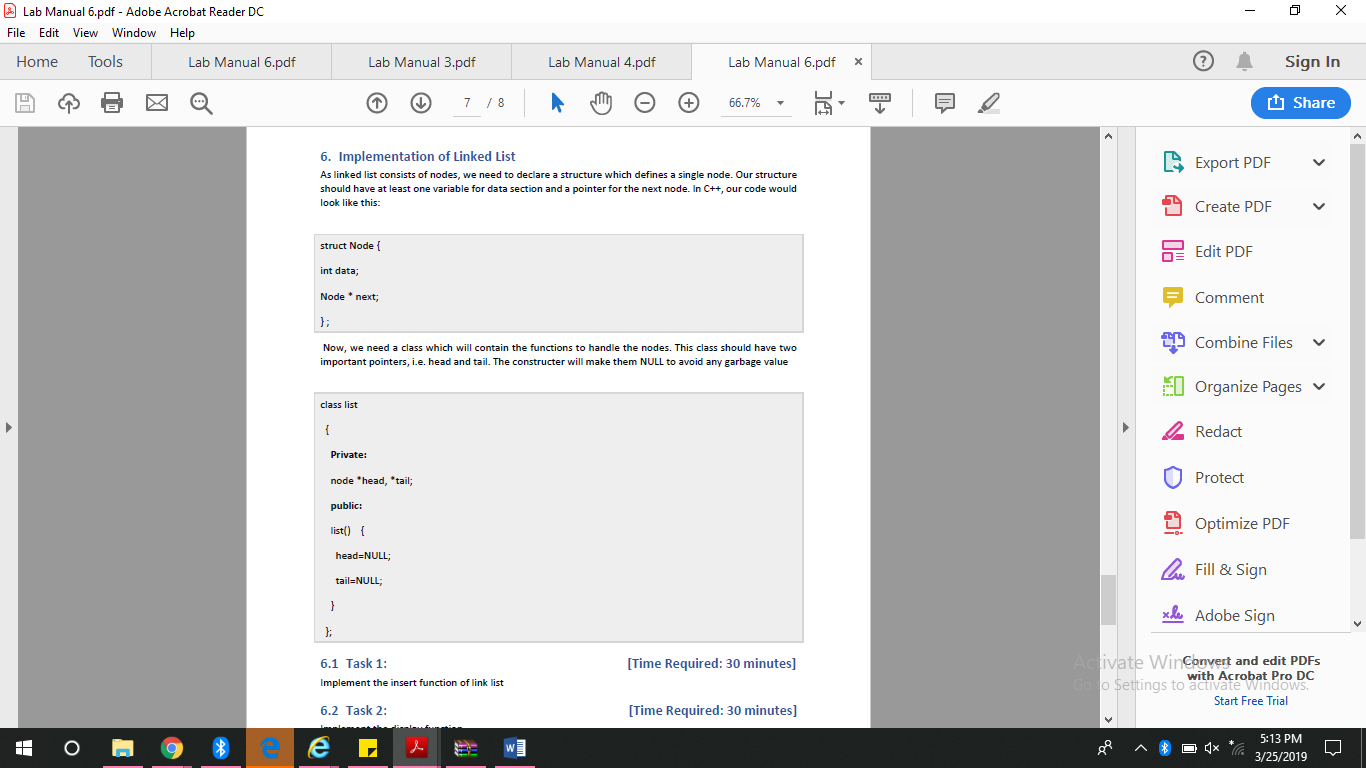
## 5.1. Head and Tail Nodes

No matter how many nodes are present in the linked list, the very first node is called head and the last node is called the **tail**. If there is just one node created then it is called both head and tail.



## 6**. Implementation of Linked List**

As linked list consists of nodes, we need to declare a structure which defines a single node. Our structure should have at least one variable for data section and a pointer for the next node. In C++, our code would look like this:



So, if we have access to the first node then we can access any node of the linked list. For example, if ‘a’ is a node then a->next is the node next to the ‘a’ (the pointer storing the address of the next node is named ‘next’).

### **Operations:**

Just like the other data structures, we can perform various operations for the linked list as well. But unlike arrays, in which we can access the element using subscript directly even if it is somewhere in between, we cannot do the same random access with a linked list.

In order to access any node, we need to traverse the linked list from the start and only then we can access the desired node. Hence accessing the data randomly from the linked list proves to be expensive.

**We can perform various operations on a linked list as given below:**

#### **1) Insertion**

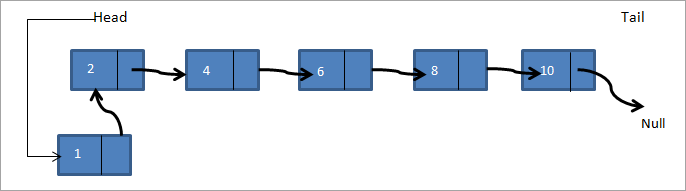
Insertion operation of linked list adds an item to the linked list. Though it may sound simple, given the structure of the linked list, we know that whenever a data item is added to the linked list, we need to change the next pointers of the previous and next nodes of the new item that we have inserted.

The second thing that we have to consider is the place where the new data item is to be added.

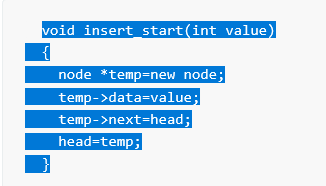
**There are three positions in the linked list where a data item can be added.**

**1) At the beginning of the linked list**

A linked list is shown below 2->4->6->8->10. If we want to add a new node 1, as the first node of the list, then the head pointing to node 2 will now point to 1 and the next pointer of node 1 will have a memory address of node 2 as shown in the below figure.

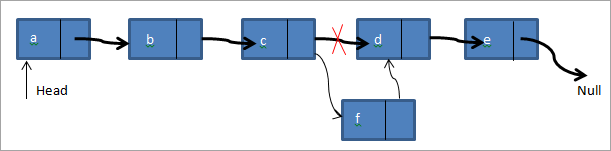
[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/Insertion.png)

Thus the new linked list becomes 1->2->4->6->8->10.

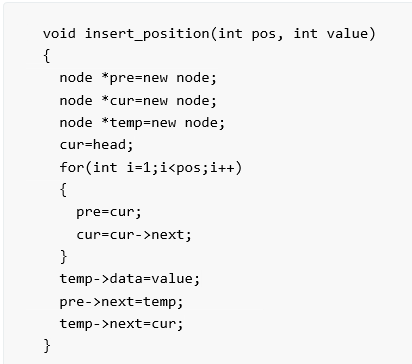


**2) After the given Node**

Here, a node is given and we have to add a new node after the given node. In the below-linked list a->b->c->d ->e, if we want to add a node f after node c then the linked list will look as follows:

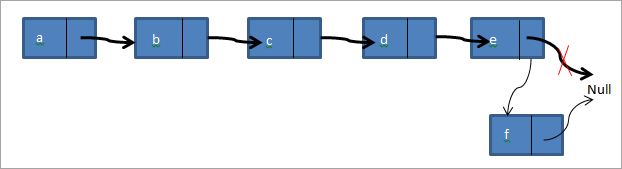
[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/to-add-a-node-f-after-node-c.png)

Thus in the above diagram, we check if the given node is present. If it’s present, we create a new node f. Then we point the next pointer of node c to point to the new node f. The next pointer of the node f now points to node d.



**3) At the end of the Linked List**

In the third case, we add a new node at the end of the linked list. Consider we have the same linked list a->b->c->d->e and we need to add a node f to the end of the list. The linked list will look as shown below after adding the node.

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2019/06/3.At-the-end-of-the-linked-list.png)

Thus we create a new node f. Then the tail pointer pointing to null is pointed to f and the next pointer of node f is pointed to null. We have implemented all three types of insert functions in the below C++ program.

### **2) Deletion**

Like insertion, deleting a node from a linked list also involves various positions from where the node can be deleted. We can delete the first node, last node or a random kth node from the linked list. After deletion, we need to adjust the next pointer and the other pointers in the linked list appropriately so as to keep the linked list intact.

There are also three cases in which a node can be deleted:

1. Deletion at the start
2. Deletion at the end
3. Deletion at a particular position

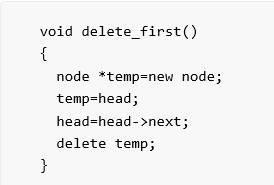
**Deletion at the Start**

In this case, the first node of the linked list is deleted. I know, you remember that the first node is called the head. So, we are going to delete the head node. The process of deletion includes:

Declare a temp pointer and pass the address of the first node, i.e. head to this pointer.

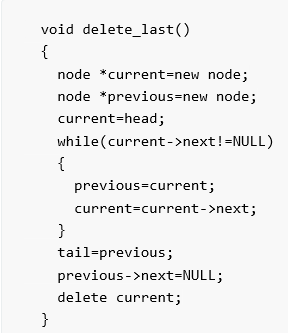
Declare the second node of the list as head as it will be the first node of linked list after deletion.

Delete the temp node.



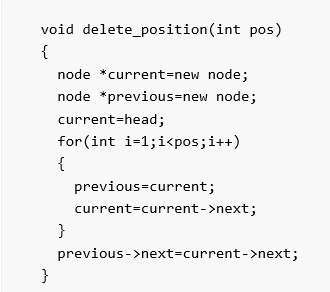
**Deletion at the End**

Deletion of the last node is a bit difficult to understand than the first node. In the case of the first node, you just need access to the head and you can delete it. But in the case of the last node, you also need access to the second to the last node of the linked list as you will delete the last node and make the previous node as the tail of linked list.



**Deletion at a Particular Position**

In linked list, we can delete a specific node. The process of deletion is simple. Here we don’t use the head and tail nodes. We ask the user to input the position of the node to be deleted. After that, we just move two temporary pointers through the linked list until we reach our specific node. Now, we delete our current node and pass the address of the node after it to the previous pointer. This way, the current node is removed from the linked list and the link is established between its previous and next node



# Practice Task

### **Lab Task 1:**

Assume that a singly linked list is implemented with a header node, but no tail node, and that it maintains only a pointer to the header node.

Write a class that includes methods to

* Return the size of the linked list (total number of nodes in list)
* Print the linked list
* Test if a value x is contained in the linked list

Add a value x if it is not already contained in the linked list e. remove a value x if it is contained in the linked list

**Lab Task 2:**

Write a class named Mobile that holds data about a mobile in a retail store. The class should have the following member variables:

• Name/brand. A string that holds name of the brand of phone.

• unitsOnHand. An int that holds the number of units currently in inventory.

• price. A double that holds the retail price.

Write a constructor that accepts arguments for each member variable, appropriate functions that store values in these member variables, and accessor functions that return the values in these member variables.

Once you have written the class,

Write another class Store (link list) having Mobile as a node then provide function to insert new mobile in list and to Delete a mobile and to display all mobiles.

Write a main program that shows all the functionality.

Note: you have to implement all scenarios (insert and delete at start, end, in between).

# 8. Evaluation criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Task No** | **Description** | **Marks** |
|  |  |  |  |
| 1. | 1. | Task 1 | 10 |
| 2. | 2. | Task 2 | 10 |

# 9. Further Reading

# 9.1. Books

The slides and reading material can be accessed from the folder of the class instructor available at vle.

# 10. Out comes

The outcomes of this lab were:

Students will be able to

1. Learn and understand Link list.

2. Learn and implement Insertion and deletion in Link list.