|  |
| --- |
| **NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**  **CS 201–DATA STRUCTURES LAB**  **Lab Session 06** |
| **Instructors:** Mr. Faizan Yousuf, Ms. Safia, Ms. Maham Mobin |

# Outline

* Linked List Basics
* Simply Linked List
* Exercise

**Linked List Basics**

A linked list is a sequence of data structures, which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

* **Link** − each link of a linked list can store a data called an element.
* **Next** − each link of a linked list contains a link to the next link called Next.
* **Linked-List** − A Linked List contains the connection link to the first link called First.

## **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.

## **Single Linked List:**

## **Representation**

Linked list can be visualized as a chain of nodes, where every node points to the next node.



## **Insertion Operation**

Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



Imagine that we are inserting a node B (NewNode), between A (LeftNode) and C (RightNode). Then point B.next to C −

NewNode.next −> RightNode;

It should look like this −



Now, the next node at the left should point to the new node.

LeftNode.next −> NewNode;



This will put the new node in the middle of the two. The new list should look like this −



Similar steps should be taken if the node is being inserted at the beginning of the list. While inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.

## **Deletion Operation**

Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.



The left (previous) node of the target node now should point to the next node of the target node −

LeftNode.next −> TargetNode.next;



This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

TargetNode.next −> NULL;



We need to use the deleted node. We can keep that in memory otherwise we can simply deallocate memory and wipe off the target node completely.



**Exercise:**

**Question No. 1:**

Write a Count () function that counts the number of times a given int occurs in a list. The code for this has the classic list raversal structure as demonstrated in Length ().

void CountTest() {

List myList = BuildOneTwoThree(); // build {1, 2, 3}

int count = Count(myList, 2); // returns 1 since there's 1 '2' in the list

}

/\*

Given a list and an int, return the number of times that int occurs

in the list.

\*/

int Count(struct node\* head, int searchFor) {

// Your code