
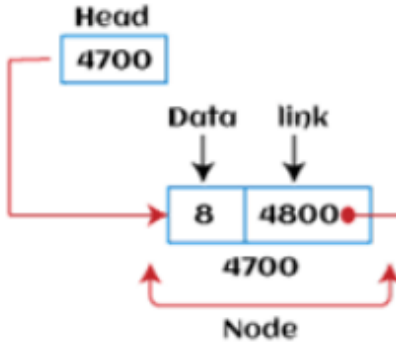


| | | | | | | |
|--|---|-------------|---------|--------------------------|-----------------|-----------------------|
| St. Peter's Engineering College (Autonomous) Dullapally (P), Medchal, Hyderabad – 500100. QUESTION BANK | | | | Dept. | : | CSE, CSM, CSD, CSC |
| | | | | Academic Year 2023-24 | | |
| Subject Code | : | AS22-05ES07 | Subject | : | Data Structures | |
| Class/Section | : | B.Tech. | Year | : | I | Semester : II |

| BLOOMS LEVEL | | | | | |
|--------------|----|------------|----|--------|----|
| Remember | L1 | Understand | L2 | Apply | L3 |
| Analyze | L4 | Evaluate | L5 | Create | L6 |

| Q. No | Question (s) | Marks | BL | CO |
|-------------------|--|-------|----|--------|
| UNIT - III | | | | |
| 1 | a) What is the definition of a Linked List? | 1M | L1 | C123.2 |
| | A linked list is a linear data structure which can store a collection of "nodes" connected together via links i.e. pointers. | | | |
| | b) Define Dynamic memory allocation. | 1M | L1 | C123.2 |
| | The mechanism by which memory can be allocated to variables during the run time is called Dynamic Memory Allocation | | | |
| | c) How can we determine if a Singly Linked List is empty? | 1M | L1 | C123.2 |
| | Check the condition for the head pointer. If(head == NULL), if it is true then the list is empty | | | |
| | d) Describe the function used to release memory in a Linked List and explain its purpose? | 1M | L2 | C123.2 |
| | The free() function is used to release memory that was allocated by malloc(). In the case of a linked list, each node of the linked list is allocated memory with a call to malloc(), therefore, when that node is no longer needed, the allocated memory of that node will need to be released through a call to free(). | | | |
| | e) Explain the difference between a Linked List and a Circular Linked List? | 1M | L2 | C123.2 |
| | In linear linked lists, where the last node points to null, in a circular linked list, it loops back to the first node, creating a continuous, circular chain of nodes. There is no NULL at the end. | | | |
| 2 | a) Define a Linked List and explain the structure of a node within it? | 3M | L2 | C123.2 |
| | <p>A linked list is a linear data structure which can store a collection of "nodes" connected together via links i.e. pointers. Linked lists nodes are not stored at a contiguous location, rather they are linked using pointers to the different memory locations.</p> <p>A linked list is a chain of nodes, and each node has the following parts:</p> <p style="padding-left: 40px;">Data – Stores the information</p> <p style="padding-left: 40px;">Next – Stores the address to next node</p> | | | |

| | | | | |
|---|--|----|--------|--|
| |  | | | |
| b) Define Dynamic memory allocation and provide the syntax for its usage? | 3M | L2 | C123.2 | |
| <p>The mechanism by which memory can be allocated to variables during the run time is called Dynamic Memory Allocation.</p>  <p>Syntax: Struct node *temp; temp=(struct node *)malloc(sizeof(struct node));</p> | | | | |
| c) Write the syntax for creating a node in a Linked List | 3M | L3 | C123.2 | |
| <p>Syntax: struct node { int data; struct node *next; };</p> | | | | |
| d) Write the syntax for creating a node in a Doubly Linked List | 3M | L3 | C123.2 | |
| <p>Syntax: struct node { int data; struct node *next; struct node *previous; };</p> | | | | |
| e) Discuss the differences between a Singly Linked List and a Doubly Linked List | 3M | L4 | C123.2 | |
| | | | | |

| | <table><tr><th>Singly Linked List</th><th colspan="3">Doubly Linked List</th></tr><tr><td>SLL nodes contains 2 field -data field and next link field.</td><td colspan="3">DLL nodes contains 3 fields -data field, a previous link field and a next link field.</td></tr><tr><td>In SLL, the traversal can be done using the next node link only. Thus traversal is possible in one direction only.</td><td colspan="3">In DLL, the traversal can be done using the previous node link or the next node link. Thus traversal is possible in both directions (forward and backward).</td></tr><tr><td>The SLL occupies less memory than DLL as it has only 2 fields.</td><td colspan="3">The DLL occupies more memory than SLL as it has 3 fields.</td></tr><tr><td>Singly linked list is less efficient.</td><td colspan="3">Doubly linked list is more efficient</td></tr></table> | | | | Singly Linked List | Doubly Linked List | | | SLL nodes contains 2 field -data field and next link field. | DLL nodes contains 3 fields -data field, a previous link field and a next link field. | | | In SLL, the traversal can be done using the next node link only. Thus traversal is possible in one direction only. | In DLL, the traversal can be done using the previous node link or the next node link. Thus traversal is possible in both directions (forward and backward). | | | The SLL occupies less memory than DLL as it has only 2 fields. | The DLL occupies more memory than SLL as it has 3 fields. | | | Singly linked list is less efficient. | Doubly linked list is more efficient | | |
|--|--|----|----|--------|--------------------|--------------------|--|--|---|---|--|--|--|---|--|--|--|---|--|--|---------------------------------------|--------------------------------------|--|--|
| Singly Linked List | Doubly Linked List | | | | | | | | | | | | | | | | | | | | | | | |
| SLL nodes contains 2 field -data field and next link field. | DLL nodes contains 3 fields -data field, a previous link field and a next link field. | | | | | | | | | | | | | | | | | | | | | | | |
| In SLL, the traversal can be done using the next node link only. Thus traversal is possible in one direction only. | In DLL, the traversal can be done using the previous node link or the next node link. Thus traversal is possible in both directions (forward and backward). | | | | | | | | | | | | | | | | | | | | | | | |
| The SLL occupies less memory than DLL as it has only 2 fields. | The DLL occupies more memory than SLL as it has 3 fields. | | | | | | | | | | | | | | | | | | | | | | | |
| Singly linked list is less efficient. | Doubly linked list is more efficient | | | | | | | | | | | | | | | | | | | | | | | |
| 3 | a) What are the advantages and disadvantages of using a Linked List? | 5M | L4 | C123.2 | | | | | | | | | | | | | | | | | | | | |
| | <p>Advantages of Linked List</p> <ul style="list-style-type: none">• Dynamic data structure - The size of the linked list may vary according to the requirements. Linked list does not have a fixed size.• Insertion and deletion - Insertion and deletion operations are quite easier in the linked list. There is no need to shift elements after the insertion or deletion of an element only the address present in the next pointer needs to be updated.• Efficient for large data- When working with large datasets linked lists play a crucial role as it can grow and shrink dynamically.• Flexible: This is because the elements in Linked List are not stored in contiguous memory locations unlike the array.• Implementation - We can implement both stacks and queues using linked list. <p>Disadvantages Of Linked List:</p> <ul style="list-style-type: none">• Memory usage: More memory is required in the linked list as compared to an array. Because in a linked list, a pointer is also required to store the address of the next element and it requires extra memory for itself.• Traversal: In a Linked list traversal is more time-consuming as compared to an array. Direct access to an element is not possible in a linked list as in an array by index.• Lower efficiency at times: For certain operations, such as searching for an element or iterating through the list, can be slower in a linked list.• Complex implementation: The linked list implementation is more complex when compared to array. It requires a complex programming understanding.• Not suited for small dataset: Cannot provide any significant benefits on small dataset compare to that of an array. | | | | | | | | | | | | | | | | | | | | | | | |
| | b) Differentiate between arrays and Linked Lists, highlighting their respective advantages and disadvantages | 5M | L4 | C123.2 | | | | | | | | | | | | | | | | | | | | |

| ARRAY | LINKED LIST |
|--|---|
| Size of an array is fixed | Size of a list is not fixed |
| Memory is allocated from stack | Memory is allocated from heap |
| It is necessary to specify the number of elements during declaration (i.e., during compile time). | It is not necessary to specify the number of elements during declaration (i.e., memory is allocated during run time). |
| It occupies less memory than a linked list for the same number of elements. | It occupies more memory. |
| Inserting new elements at the front is potentially expensive because existing elements need to be shifted over to make room. | Inserting a new element at any position can be carried out easily. |
| Deleting an element from an array is not possible. | Deleting an element is possible. |

| | | | |
|---|----|----|--------|
| c) Provide an algorithm for inserting a node at the beginning of a Doubly Linked List | 5M | L3 | C123.2 |
|---|----|----|--------|

The diagram illustrates a doubly linked list with nodes A, B, C, and D. Node A is the head. A new node E is being inserted at the beginning. The 'Next' pointer of node E points to node A, and the 'Prev' pointer of node A points to node E. The 'Next' pointer of node A points to node B, and the 'Prev' pointer of node B points to node A. The 'Next' pointer of node B points to node C, and the 'Prev' pointer of node C points to node B. The 'Next' pointer of node C points to node D, and the 'Prev' pointer of node D points to node C. The 'Next' pointer of node D points to NULL.

Algorithm for inserting a node at Beginning of the list

We can use the following steps to insert a new node at beginning of the double linked list...

Step 1 - Create a temp Node with given value and temp → previous as NULL.

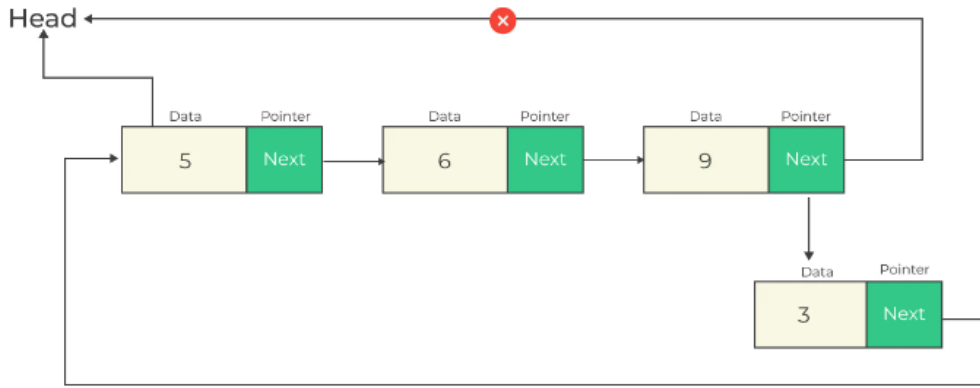
Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, assign NULL to temp → next and temp to head.

Step 4 - If it is not Empty then, assign head to temp → next and temp to head.

| | | | |
|---|----|----|--------|
| d) Provide an algorithm for searching a node in a Singly Linked List? | 5M | L3 | C123.5 |
|---|----|----|--------|

The diagram shows a singly linked list with four nodes: Node1 (value 15, address 1000), Node2 (value 20, address 2000), Node3 (value 30, address 4000), and Node4 (value 40, address 5000). A pointer 'ptr' starts at Node1. The search process is shown: 15 ≠ 30, 20 ≠ 30, 30 = 30 (Found). The value 30 is found in Node3.

| | | | | |
|---|---|-----|----|--------|
| | Algorithm for searching a node in a Singly Linked List 1 START 2. Check if the list is empty (head == NULL) 3 If the list is empty, it displays “No Need to check the key”. 4. If the list is not empty, iteratively check if the list contains key. 5. If the key is found, it displays the “Element is found at Location X”. 6 If the key element is not present in the list, Element is not Found. 7 END | | | |
| | e) Provide an algorithm for inserting an element at the end of a Circular Linked List | 5M | L3 | C123.2 |
| |  <p>Algorithm for Deleting a node from End of the list We can use the following steps to delete a node from end of the circular linked list...</p> <p>Step 1 - Check whether list is Empty (head == NULL)</p> <p>Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.</p> <p>Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.</p> <p>Step 4 - Check whether list has only one Node (temp1 → next == head)</p> <p>Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate from the function. (Setting Empty list condition)</p> <p>Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until temp1 reaches to the last node in the list. (until temp1 → next == head)</p> <p>Step 7 - Set temp2 → next = head and delete temp1.</p> | | | |
| 4 | a) What is a Linked List, and how is it represented in memory? Explain different types of Linked Lists with neat diagrams. | 10M | L2 | C123.2 |
| | A linked list is a linear data structure which can store a collection of "nodes" connected together via links i.e. pointers. Linked lists nodes are not stored at a contiguous location, rather they are linked using pointers to the different memory locations. | | | |

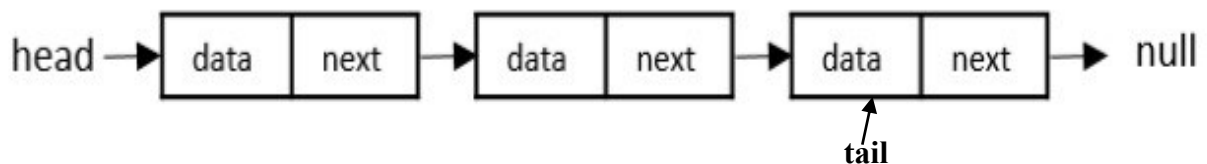
A linked list is a chain of nodes, and each node has the following parts:

Data – Stores the information

Next – Stores the address to next node



A linked list is a dynamic linear data structure whose memory size can be allocated or de-allocated at run time based on the operation insertion or deletion, this helps in using system memory efficiently. Linked lists can be used to implement various data structures like a stack, queue, graph, hash maps, etc.



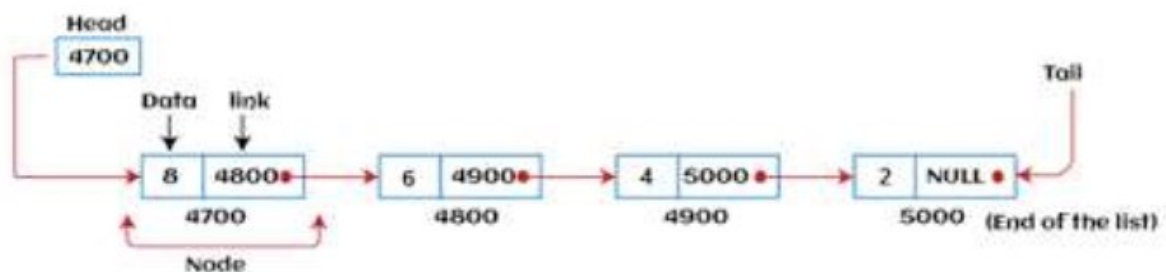
A linked list starts with a **head** node which points to the first node. Every node consists of data which holds the actual data (value) associated with the node and a next pointer which holds the memory address of the next node in the linked list. The last node is called the tail node in the list which points to **null** indicating the end of the list.

Dynamic Memory allocation for node

```
struct node *temp;
```

```
temp = (struct node *)malloc(sizeof(struct node *));
```

Representation of Linked List



START →

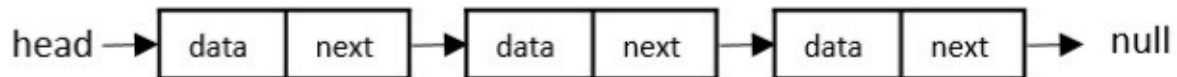
| INFO | LINK |
|------|------|
| 8 | 4800 |
| 6 | 4900 |
| 4 | 5000 |
| 2 | NULL |

Types of Linked List

Following are the various types of linked list.

1. Singly Linked Lists

Generally, a Linked List means a singly linked list. Singly linked lists contain two fields in one node; one field holds the data and the other field holds the address of the next node of the list. Traversals can be done in one direction only as there is only a single link between two nodes of the same list.



2. Doubly Linked Lists

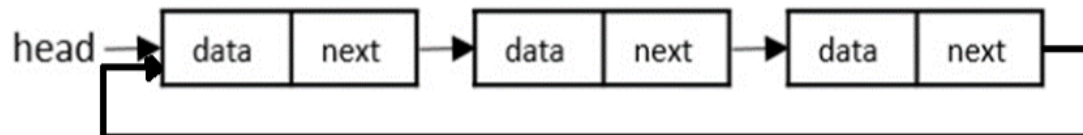
Doubly Linked Lists contain three fields in one node; one field holds the data and the other fields hold the addresses of the previous and next nodes in the list. The list is traversed twice as the nodes in the list are connected to each other from both sides.



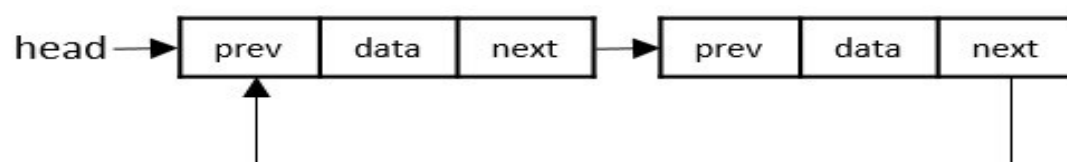
3. Circular Linked Lists

Circular linked lists can exist in both singly linked list and doubly linked list. Since the last node and the first node of the circular linked list are connected. In the circular linked list, you can start from any node and traverse the list either forward or backwards until it broken i.e., no starting or endpoint.

Circular Singly Linked List:



Circular Doubly Linked List:



b) Explain the operations performed in a Singly Linked List and illustrate them with suitable diagrams?

10M

L3

C123.2

The basic operations in the linked lists are insertion, deletion, searching, display, and deleting an element at a given key. These operations are performed on Singly Linked Lists as given below

- **Insertion**
- **Deletion**
- **Traversal**
- **Search**

(i) 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.

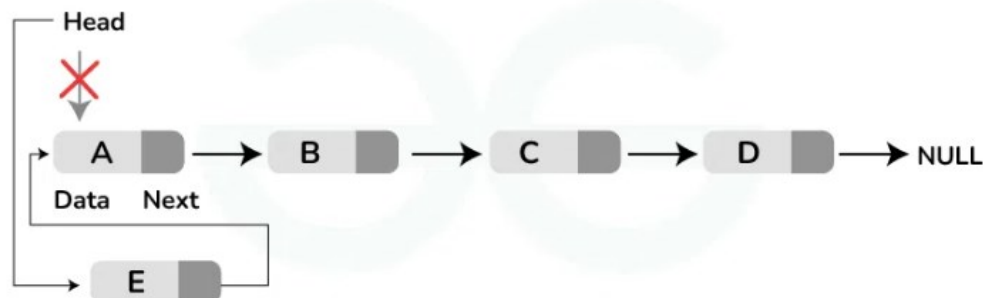
The insertion into a singly linked list can be performed at different positions. Based on the position of the new node being inserted, the insertion is categorized into the following categories.

| SN _o | Operation | Description |
|-----------------|--------------------------------|---|
| 1 | Insertion at beginning | It involves inserting any element at the front of the list. We just need to a few link adjustments to make the new node as the head of the list. |
| 2 | Insertion at end of the list | It involves insertion at the last of the linked list. The new node can be inserted as the only node in the list or it can be inserted as the last one. Different logics are implemented in each scenario. |
| 3 | Insertion after specified node | It involves insertion after the specified node of the linked list. We need to skip the desired number of nodes in order to reach the node after which the new node will be inserted. . |

1) Insert a Node at the Front/Beginning of Linked List

To insert a node at the start/beginning/front of a Linked List, we need to:

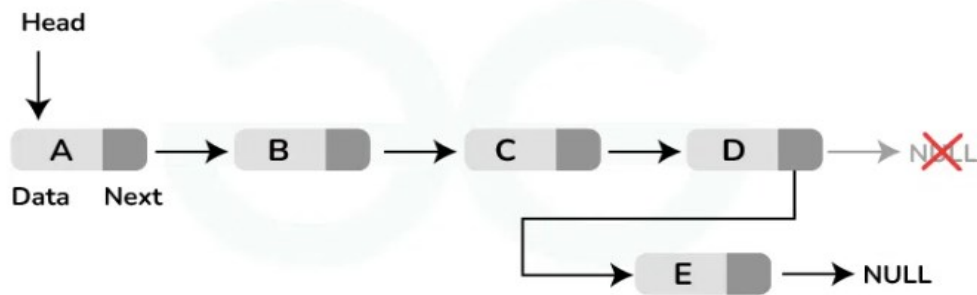
- Make the first node of Linked List linked to the new node or temp node
- Remove the head from the original first node of Linked List
- Make the new node as the Head of the Linked List.



2) Insert a Node at the End of Linked List

To insert a node at the end of a Linked List, we need to:

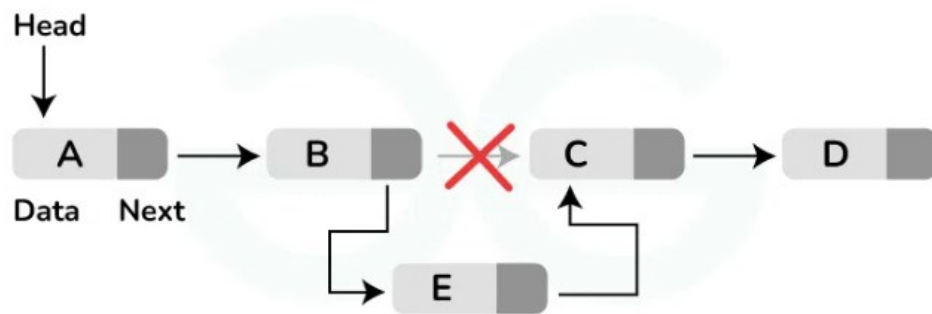
- Go to the last node of the Linked List
- Change the next pointer of last node from NULL to the new node
- Make the next pointer of new node as NULL to show the end of Linked List.



3) Insert a Node after a Given Node in Linked List

To insert a node after a given node in a Linked List, we need to:

- Check if the given node exists or not.
- If it does not exist, terminate the process.
- If the given node exists,
 - Make the element to be inserted as a new node
 - Change the next pointer of given node to the new node
 - Now shift the original next pointer of given node to the next pointer of new node



(ii) Deletion Operation

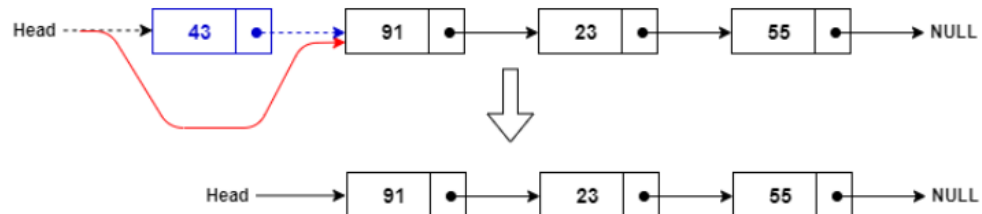
The Deletion of a node from a singly linked list can be performed at different positions. Based on the position of the node being deleted, the operation is categorized into the following categories.

| SN | Operation | Description |
|----|---------------------------------|--|
| 1 | Deletion at beginning | It involves deletion of a node from the beginning of the list. This is the simplest operation among all. It just need a few adjustments in the node pointers. |
| 2 | Deletion at the end of the list | It involves deleting the last node of the list. The list can either be empty or full. Different logic is implemented for the different scenarios. |
| 3 | Deletion after specified node | It involves deleting the node after the specified node in the list. we need to skip the desired number of nodes to reach the node after which the node will be deleted. This requires traversing through the list. |

1) Deletion at Beginning

To delete a node from the start/beginning/front of a Linked List, we need to:

- Remove the head from the original first node of Linked List
- Make the second node as the Head of the Linked List.
- Free memory for the node to be deleted

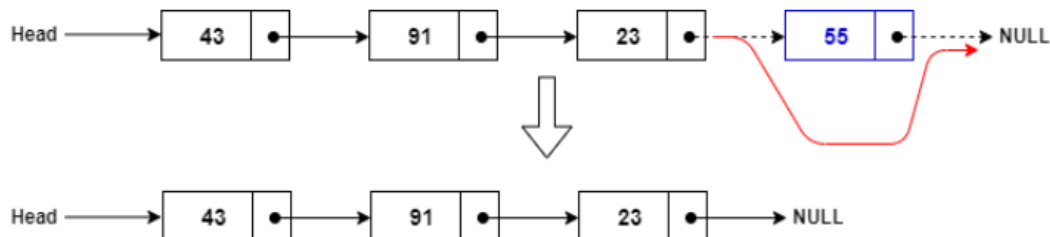


2) Deletion at Ending

To delete a node from the end of a Linked List, we need to:

- Go to the last node of the Linked List
- Change the previous pointer of last node to NULL.
- Free memory for the node to be deleted

In this deletion operation of the linked, we are deleting an element from the ending of the list.

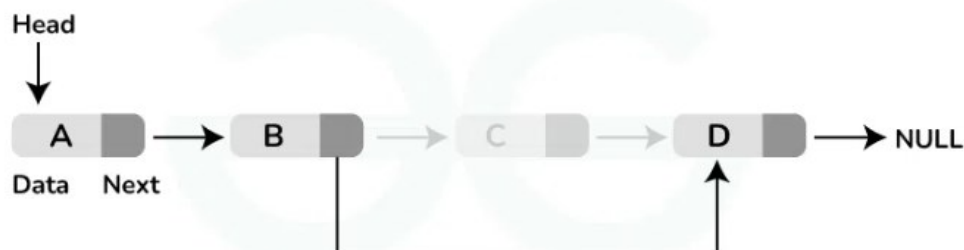


3) Deletion at a Given Position

To insert a node after a given node in a Linked List, we need to:

- Check if the given node exists or not.
- If it does not exist, terminate the process.
- If the given node exists,
 - Change the previous pointer next field to given node next field.
 - Change the given next field to NULL.
 - Free memory for the node to be deleted

In this deletion operation of the linked, we are deleting an element at any position of the list.



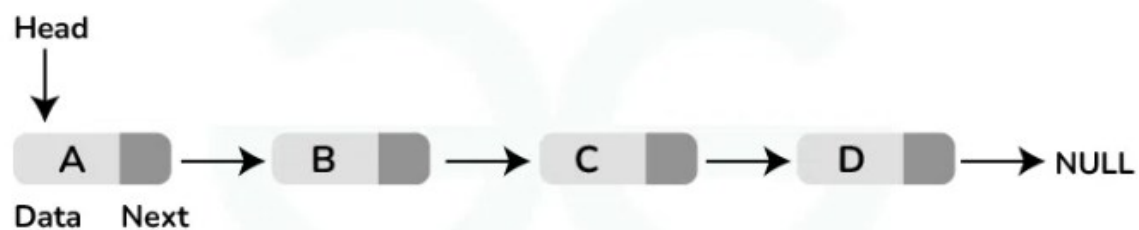
(iii) Traversal Operation

In traversing, we simply visit each node of the list at least once in order to perform some specific operation on it, for example, printing data part of each node present in the list.

The process of traversing a singly linked list involves printing the value of each node and then going on to the next node and print that node's value also and so on, till we reach the last node in the singly linked list, whose next node is pointing towards the null

To print the node data from first node to the last node of a Linked List, we need to:

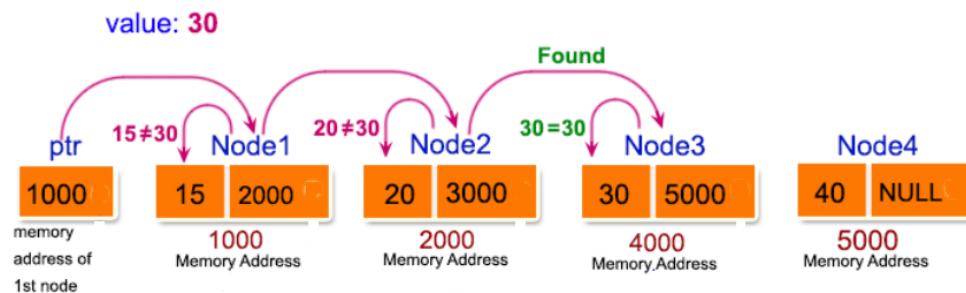
- A pointer is required to travel from first node to the last node.
- While traveling the pointer it prints the node data and move to next node until it reaches to last node.



(iv) Search Operation

In searching, we match each element of the list with the given element. If the element is found on any of the location, then location of that element is returned otherwise null is returned.

Searching for an element in the list using a key element. This operation is done in the same way as array search; comparing every element in the list with the key element given.



c) Write a C Program for deleting a node from a Singly Linked List after a specified given position?

10M

L4

C123.2

Program: The implementations of this operation in C programming language –

```

#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
}*head=NULL,*ptr;

// Display the list

```

```
void traversal()
{
    if (head== NULL)
        printf("List if Empty");
    else
    {
        ptr = head;
        //start from the beginning
        while(ptr != NULL)
        {
            printf(" ->%d ",ptr->data);
            ptr = ptr->next;
        }
        printf("\n");
    }
}

//Insertion at the ending
void insertatend(int data)
{
    //create a link
    struct node *temp = (struct node*) malloc(sizeof(struct node));
    temp->data = data;
    temp->next = NULL;
    if (head== NULL)
        head=temp;
    else
    {
        ptr=head;
        while(ptr->next!=NULL)
        {
            ptr=ptr->next;
        }
        ptr->next=temp;
    }
}

void deleteaftergivennode(int ele)
{
    if (head== NULL)
        printf("List is empty");
    else
    {
        struct node *p;
        ptr=head;
        while(ptr->data != ele)
        {
            p=ptr;
```

```

        ptr=ptr->next;
    }
    p->next=ptr->next;
    free(ptr);
}
}

void main()
{
    int i,n,k,ele;
    printf("Enter the Number of Nodes\t");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("Enter the Data:\t");
        scanf("%d",&k);
        insertatend(k);
    }
    printf("Linked List: ");
    traversal();
    printf("Enter the Node:\t");
    scanf("%d",&ele);
    deleteaftergivennode(ele);
    printf("After deleting Linked List:\t");
    traversal();
}

Output:
Enter the Number of Nodes      4
Enter the Data: 10
Enter the Data: 20
Enter the Data: 30
Enter the Data: 40
Linked List:                    ->10  ->20  ->30  ->40
Enter the Node: 20
After deleting Linked List:     ->10  ->30  ->40

```