03

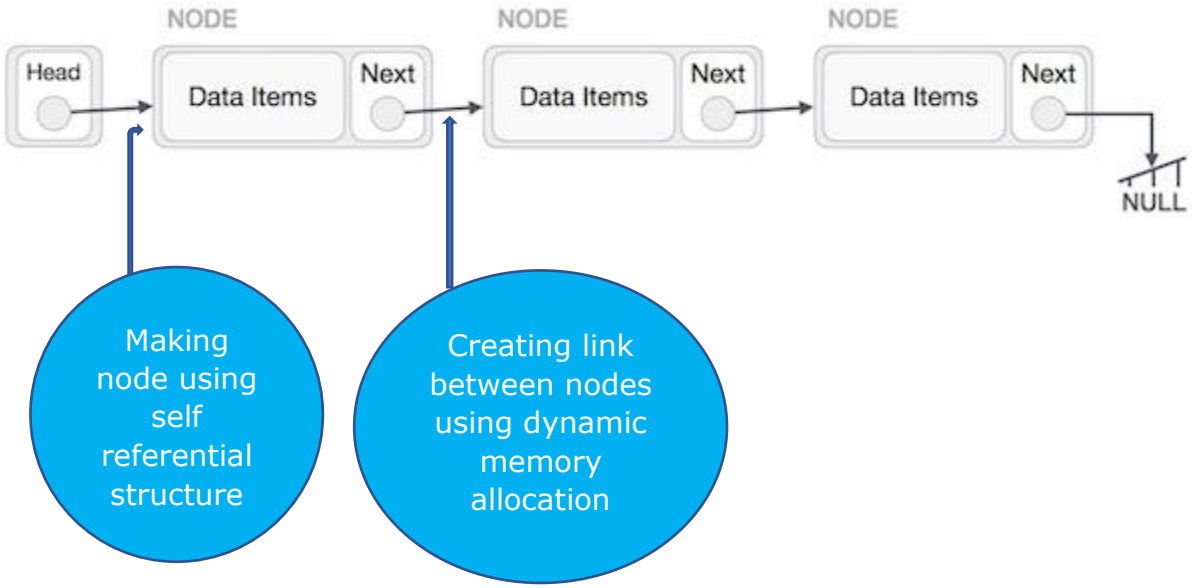
Linked list and operations

DATA STRUCTURE LAB

SESSION - 03

LINK LIST

- Linked List is a *linear data structure* which is a sequence of data structures, that are connected together via links and it is very common data structure which consists of group of nodes in a sequence that is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain.



- Linked List contains a link element called head.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

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.

A simple C program to introduce a linked list

```
#include<stdio.h>
#include<stdlib.h>

struct node
{
    int data;
    struct node *next;
};

// Program to create a simple linked list with 3 nodes

int main()
{
    struct node* head = NULL;
    struct node* second = NULL;
    struct node* third = NULL;

    // allocate 3 nodes in the heap

    head = (struct node*)malloc(sizeof(struct node));
    second = (struct node*)malloc(sizeof(struct node));
    third = (struct node*)malloc(sizeof(struct node));

    /* Three blocks have been allocated  dynamically.
       We have pointers to these three blocks as first, second and third

         head           second          third
          |               |              |
```

```
      |               |              |
  +---+-----+     +----+----+     +----+----+
  | #  | #  |     | #   | # |     |  # |  # |
  +---+-----+     +----+----+     +----+----+

 '#' represents any random value.
 Data is random because we haven't assigned anything yet  */

head->data = 1;                          //assign data in first node
head->next = second;                 // Link first node with the second node

/* data has been assigned to data part of first block (block
  pointed by head).  And next pointer of first block points to
  second.  So they both are linked.

     head          second         third
      |              |              |
      |              |              |
  +---+---+     +----+----+     +-----+----+
  | 1 | o----->| #   | #  |     |  #  | #  |
  +---+---+     +----+----+     +-----+----+
*/

second->data = 2;                  //assign data to second node
second->next = third;             // Link second node with the third node

/* data has been assigned to data part of second block (block pointed by
   second). And next pointer of the second block points to third block.
  So all three blocks are linked.

     head         second         third
      |             |             |
```

```
        |             |             |
    +---+---+     +---+---+     +----+----+
    | 1 | o-----> | 2 | o-----> |  # |  # |
    +---+---+     +---+---+     +----+----+      */

   third->data = 3;                          //assign data to third node
   third->next = NULL;

   /* data has been assigned to data part of third block (block pointed
     by third). And next pointer of the third block is made NULL to indicate
     that the linked list is terminated here.

      We have the linked list ready.

            head
              |
              |
         +---+---+     +---+---+       +----+------+
         | 1 | o-----> |  2  |  o-----> |  3 | NULL |
         +---+---+     +---+---+       +----+------+


     Note that only head is sufficient to represent the whole list.  We can
     traverse the complete list by following next pointers.    */

   return 0;

}
```

Advantages of Linked Lists:

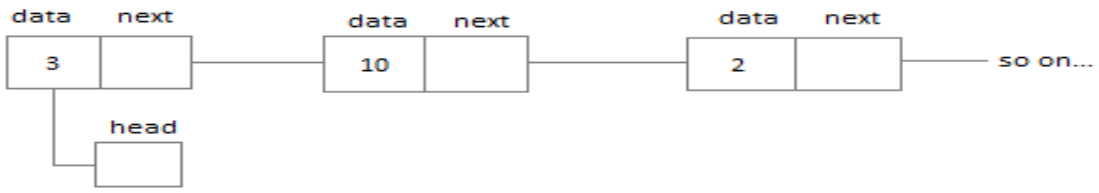
- They are a dynamic in nature which allocates the memory when required.
- Insertion and deletion operations can be easily implemented.
- Linked List reduces the access time.

Disadvantages of Linked Lists:

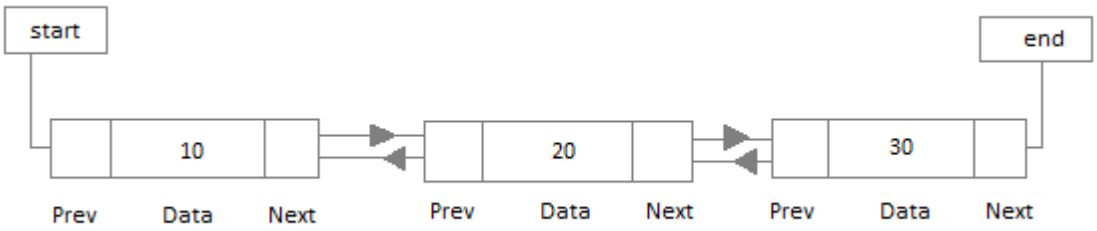
- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in linked list.

Types of Linked Lists

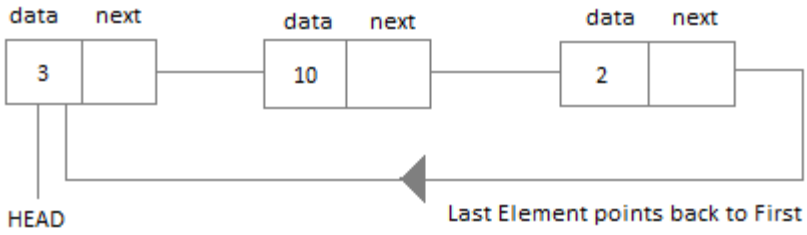
- **Singly Linked List :** Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.



- **Doubly Linked List :** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.

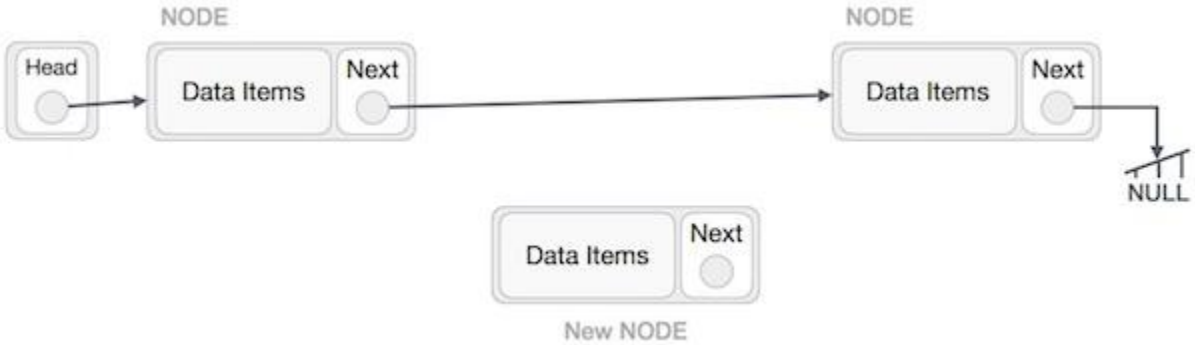


- **Circular Linked List :** In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.



Insertion Operation

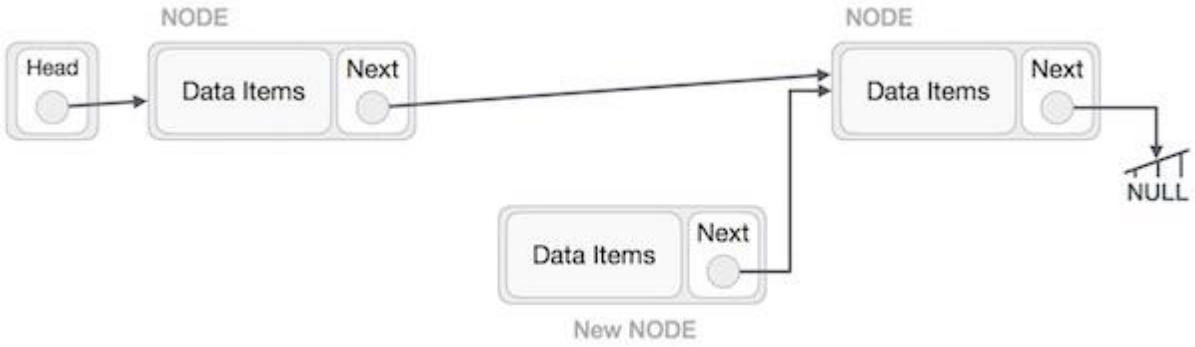
Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



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

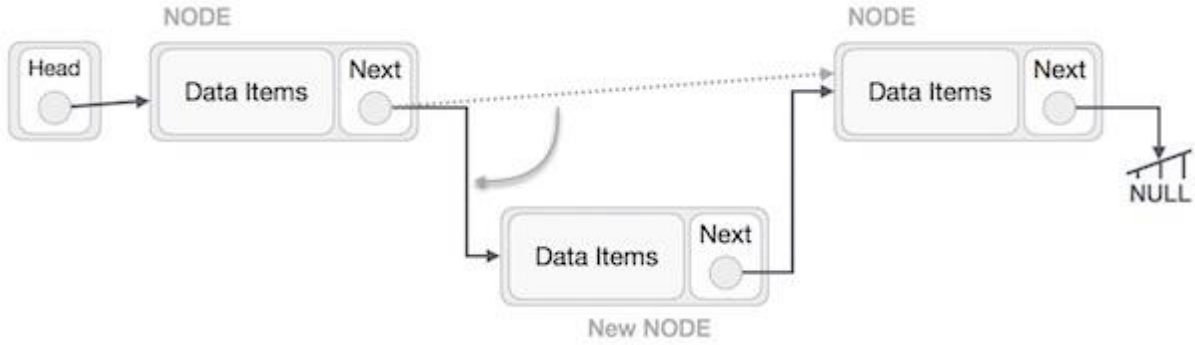
```
NewNode.next –> RightNode;
```

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.

It’s highly recommended that before seeing code assure that you have tried enough to implement in code.

Insert at the front of linklist:

```
void insert_front(node *list,int data)
{
    node *temp;
    temp=(node*)malloc(n*sizeof(node));
    temp->data=data;
    temp-next=list;
    list=temp;
}
```

Insert at after of a node of linklist:

```
void insert_after(node *list,int data)
{
    if(list==NULL)
    {
        printf("previous node cann't be NULL!");
    }
    node *temp= (node*)malloc(1*sizeof(node));
    temp->data=data;
    temp->next=list->next;
    list->next=temp;
}
```

Insert at the end of linklist:

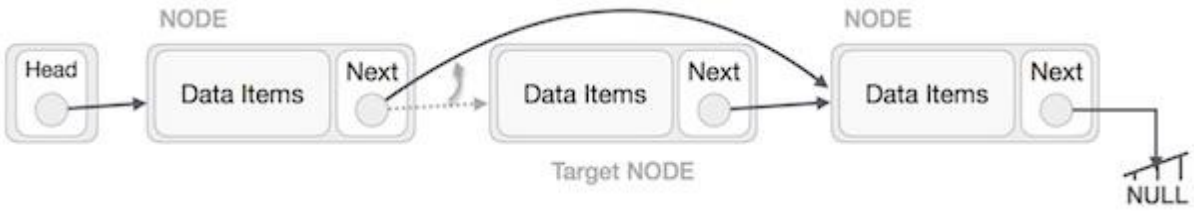
```
void insert_end(node *list, int data)
{
    node *temp=(node*)malloc(1*sizeof(node));
    temp->data=data;
    temp->next=NULL;
    while(list->next!=NULL)
        list=list->next;
    list->next=temp;
    return;
}
```

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.

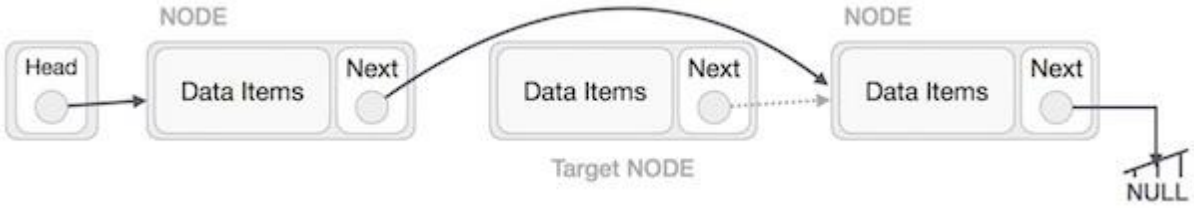
The left (previous) node of the target node now should point to the next node of the target node −

```
LeftNode.next −> TargetNode.next;
```

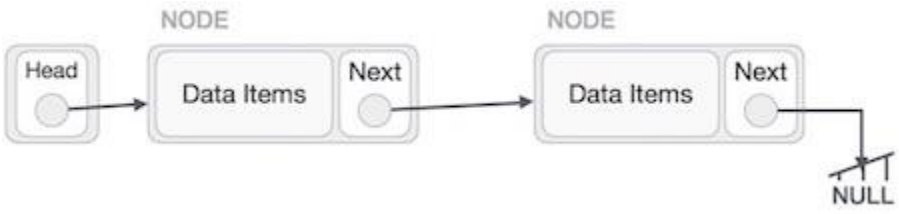


This will remove the link that was pointing to the target node. Now, using the following code, we will remove what the target node is pointing at.

```
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.



Delete Function implementation:

It's highly recommended that before seeing code assure that you have tried enough to implement this function in code.

Delete function implementation:

```
void delete(node *list,int data)
{
    node *temp;
    while(list->next->data != data)
        list=list->next;
    temp=list->next;
    list->next=list->next->next;
```

```
    free(temp);
    printf("%d is deleted",data);
}
```

Update function implementation:

```
void update(node *list,int data1,int data2)
{
    while(list -> data!=data)
        list=list->next;
    list->data=data2;
    printf("%d updated!\n",data1);
}
```

Count function implementation:

```
void count(node *list)
{
    int cnt=0;
    while(list->next != NULL)
    {
        cnt++;
        list=list->next;
    }
    return cnt;
}
```

Exercise:

**write a function “sum” that returns summation of all data in the linklist.

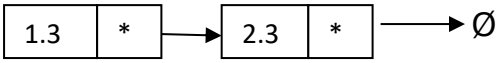
**using the given functions implement a linklist. Where you will insert 12,15,18. Then print them. Then delete the data 15 and print. Now you update the data “18” to “25”. Now print all data again.

*****

EXERCISE part:

LINK LIST:

1.Write C code to create the following Node with the shown data.



2. Draw the output of the code shown below:

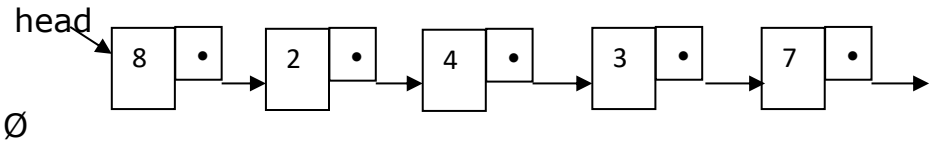
```
struct Node {
    int p, q, r;
    struct Node *p;
};
struct Node *data = (struct Node*) malloc(sizeof(struct Node));
data-> p=1;

data-> q=3;

data->r=4;

data-> p=NULL;
```

3. Consider the following link list:

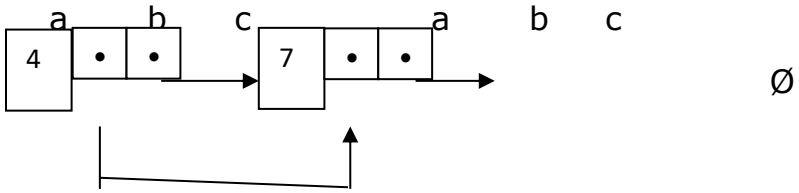


Using the given diagram implement c code. Then insert 9 in between 4&3. Then delete 7 from the linklist.

5. Draw the diagram(node&pointer) by following code.

```
struct Node {
        int a; float b;
        struct Node *next;
  };
typedef struct Node node;
node *x, *y, *z;
x = (node *) malloc( sizeof(node)); x->a=5;  x->b=1.5;
z = (node *) malloc( sizeof(node)); z->a=7; z->b = 2.3;
y = x; x->next = z;  z->next = NULL;
```

6. Write code to create node and assign as per the following diagram.



7. Given a word, your task is to check is it is palindrome or not using link list.