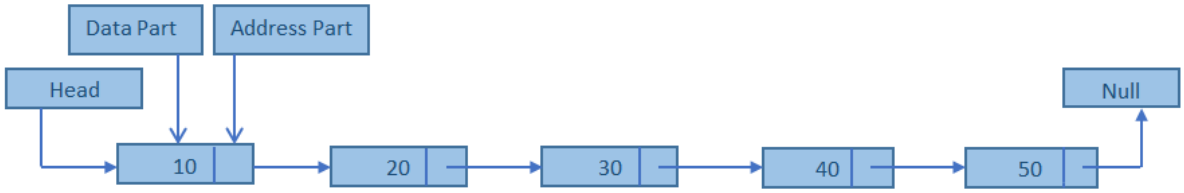
## What is Linked List?

When we want to work with unknown number of data values, we use a linked list data structure to organize that data. Linked list is a linear data structure that contains sequence of elements such that each element links to its next element in the sequence. Each element in a linked list is called as "Node". Link list used for the dynamic memory allocation.

## Representation of link list

Link list consists a series of structure. Each structure consists of a data field and address field. Data field consists data part and the address field contains the address of the successors. The last node signifies the end of the list that means NULL.




**Note:** Head is not a separate node but it is a reference to the first node. If the list is empty, the head is a null reference.

Linked list is a dynamic data structure. While accessing a particular item, start at the head and follow the references until you get that data item.

**The real life example** of Linked List is that of Railway Carriage. It starts from engine and then the coaches follow. Coaches can traverse from one coach to other, if they connected to each other.

**Advantages of Linked List**

- Linked list is dynamic in nature which allocates the memory when required.
- In linked list, stack and queue can be easily executed.
- It reduces the access time.
- Insert and delete operation can be easily implemented in linked list.

**Disadvantages of Linked List**

- Reverse traversing is difficult in linked list.
- Linked list has to access each node sequentially; no element can be accessed randomly.
- In linked list, the memory is wasted as pointer requires extra memory for storage.

## Structure of node

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

The above code identifies the basic structure of declaring a node. Struct node contains an int data field and a pointer to another node can be defined as struct node* next.

## Following are the operations that can be performed on a Linked List:

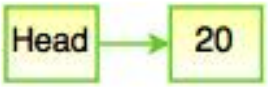
1. Create
2. Insert
3. Delete
4. Traverse
5. Search
6. Concatenation
7. Display

## 1. Create:

Create operation is used to create constituent node when required. In create operation, memory must be allocated for one node and assigned to head as follows.

**Creating first node**

```
head = (node*) malloc (sizeof(node));
head -> data = 20;
head -> next = NULL;
```



## 2. Insert:

Insert operation is used to insert a new node in the linked list. Suppose, we insert a node B (New Node), between A (Left Node) and C (Right Node), it is represented as: point B. next to B.

```
NewNode.next -> RightNode;
```

### We can insert an element using three cases:

i. At the beginning of the list

ii. At a certain position (Middle)

iii. At the end

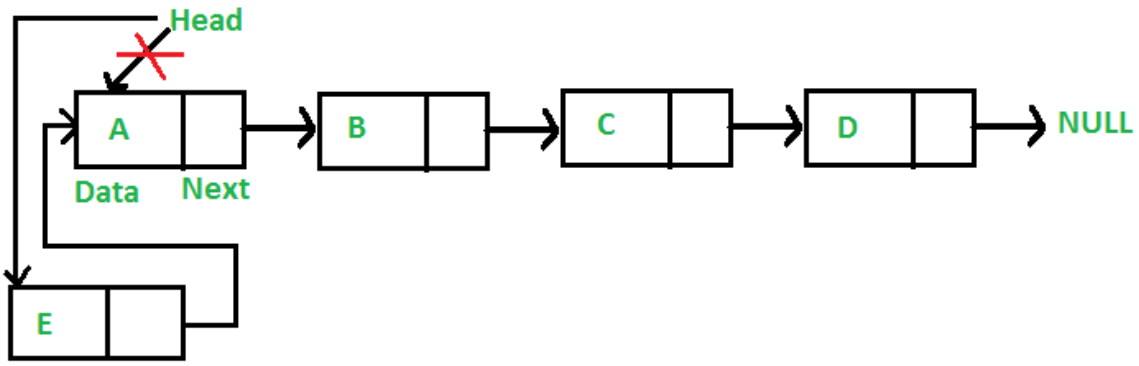
### Inserting an element

```
node* nextnode = malloc(sizeof(node));

nextnode -> data = 22;

nextnode -> next = NULL;

head -> next = nextnode;
```


The above figure represents the example of create operation, where the next element (i.e 22) is added to the next node by using insert operation.

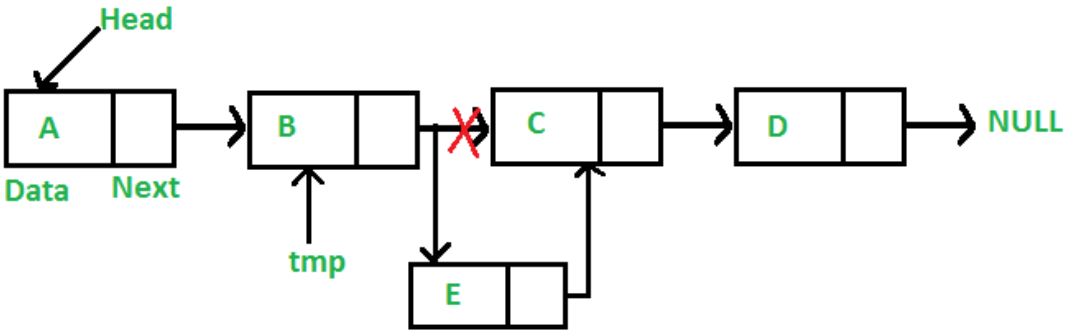
#### i. At the beginning of the list

New node becomes the new head of the linked list because it is always added before the head of the given linked list.



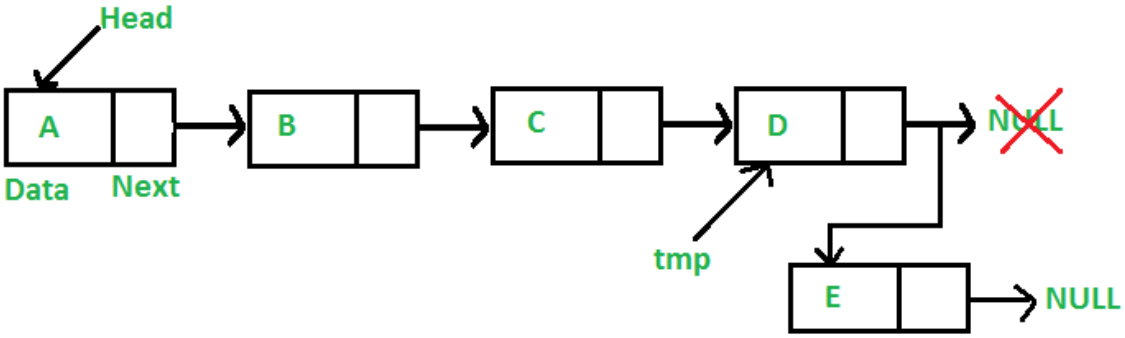

**ii. At certain position (Middle)**

We are given pointer to a node, and the new node is inserted after the given node.




**iii. At the end**

The new node is always added after the last node of the given Linked List.




## 3. Delete:

Delete operation is used to delete node from the list. Deletion operation can be performed in three ways. They are as follows...

i. Deleting from Beginning of the list

ii. Deleting from End of the list

iii. Deleting a Specific Node

## Example:

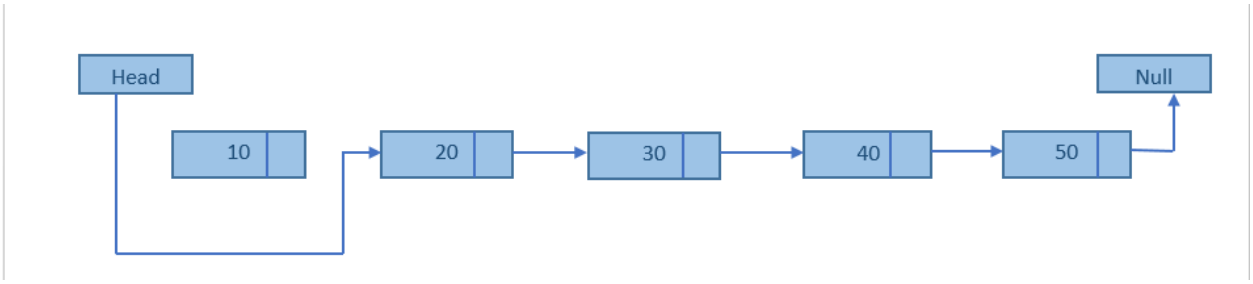
To delete a node from linked list, we need to do following steps.

1) Find previous node of the node to be deleted.

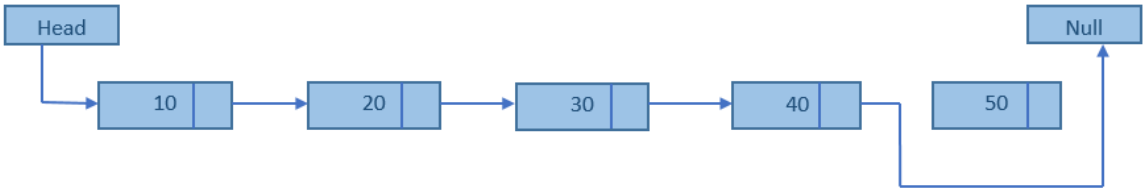
2) Change the next of previous node.

3) Free memory for the node to be deleted.

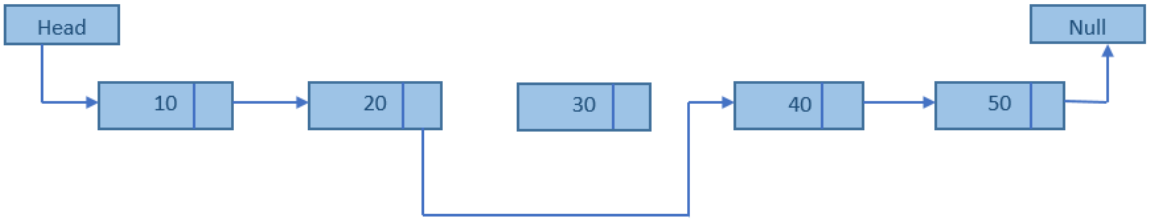
**i. Deleting from Beginning of the list**




**ii. Deleting from End of the list**




**iii. Deleting a Specific Node**



## 4. Traverse:

Traverse operations is a process of examining all the nodes of linked list from the one end to the other end.

## 5. Search:

Search operation is used for finding a particular element in a linked list. Sequential search is the most common search used on linked list structure. Search operation ends with a success if the element is found. If the element is not found, search ends in a failure.

## 6. Concatenation:

Concatenation is the process of appending a second list to the end of the first list.

## 7. Display:

Display operation is used to print each and every node's information. This operation displays the complete list.

## Difference between Array and Linked List:

| Array | Linked List |
|---|---|
| Array is a collection of elements having same data type with common name. | Linked list is an ordered collection of elements which are connected by links. |
| Elements can be accessed randomly. | Elements cannot be accessed randomly. It can be accessed only sequentially. |
| Array elements can be stored in consecutive manner in memory. | Linked list elements can be stored at any available place as address of node is stored in previous node. |
| Insert and delete operation takes more time in array. | Insert and delete operation cannot take more time. It performs operation in fast and in easy way. |
| Memory is allocated at compile time. | Memory is allocated at run time. |
| It can be single dimensional, two dimensional or multidimensional. | It can be singly, doubly or circular linked list. |
| Each array element is independent and does not have a connection with previous element or with its location. | Location or address of element is stored in the link part of previous element or node. |

| | |
|---|---|
| Array elements cannot be added, deleted once it is declared. | The nodes in the linked list can be added and deleted from the list. |
| In array, elements can be modified easily by identifying the index value. | In linked list, modifying the node is a complex process. |
| Pointer cannot be used in array. So, it does not require extra space in memory for pointer. | Pointers are used in linked list. Elements are maintained using pointers or links. So, it requires extra memory space for pointers. |

## Types of Linked List:

Following are the types of Linked List

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. Doubly Circular Linked List

# Singly Linked List

## What is Single Linked List?

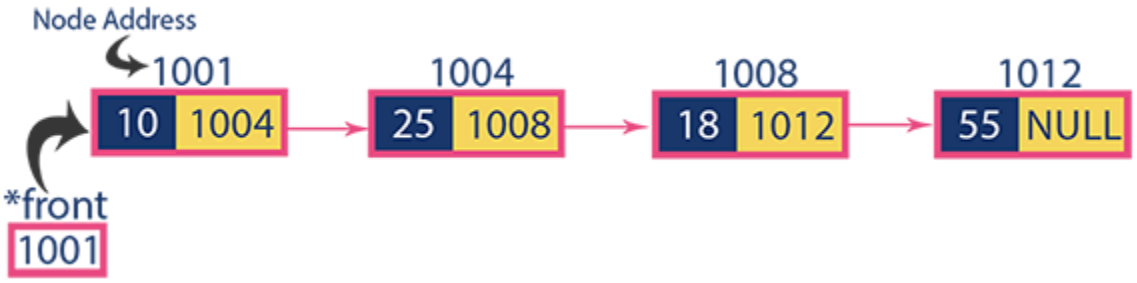
Simply a list is a sequence of data, and linked list is a sequence of data linked with each other. The formal definition of a single linked list is as follows...

**Single linked list is a sequence of elements in which every element has link to its next element in the sequence.**

In any single linked list, the individual element is called as "Node". Every "Node" contains two fields, data field and next field. The data field is used to store actual value of the node and next field is used to store the address of next node in the sequence.

The graphical representation of node in a single linked list is as follows...

## Example:




## Insertion:

In a single linked list, the insertion operation can be performed in four ways. They are as follows...

1. The new node is inserted at the beginning.
2. The new node is inserted at the end.
3. The new node is inserted after a given node.
4. The new node is inserted before a given node.

Before we describe the algorithms to perform insertions in all these four cases, let us first discuss an important term called OVERFLOW. Overflow is a condition that occurs when AVAIL = NULL or no free memory cell is present in the system. When this condition occurs, the program must give an appropriate message.

### 1. Algorithm to Insert a NewNode at the Beginning of a Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

## 2. Algorithm to Insert a NewNode at the End of a Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET NEW_NODE->NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR->NEXT != NULL
Step 8:     SET PTR = PTR->NEXT
        [END OF LOOP]
Step 9: SET PTR->NEXT = NEW_NODE
Step 10: EXIT
```

## 3. Algorithm to Insert a NewNode After a Given Node in a Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR->DATA
         != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR->NEXT
          [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

## 4. Algorithm to Insert a NewNode Before a Given Node in a Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR -> DATA != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 10: PREPTR -> NEXT = NEW_NODE
Step 11: SET NEW_NODE -> NEXT = PTR
Step 12: EXIT
```

## Deletion:

In a single linked list, the deletion operation can be performed in three ways. They are as follows...

1. Deleting from Beginning of the list
2. Deleting from End of the list
3. Deleting a Specific Node

## 1. Algorithm to Delete the First Node from a Linked List:

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: FREE PTR
Step 5: EXIT
```

## 2. Algorithm to Delete the Last Node from a Linked List:

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != NULL
Step 4:     SET PREPTR = PTR
Step 5:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 6: SET PREPTR -> NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT
```

## 3. Algorithm to Delete the Node after a given node from a Linked List:

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR -> NEXT = PTR -> NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

## Traversing a Linked List:

### 1. Algorithm for traversing a linked list:

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:          Apply Process to PTR->DATA
Step 4:          SET PTR = PTR->NEXT
      [END OF LOOP]
Step 5: EXIT
```

## Searching for a Value in a Linked List:

### 1. Algorithm to search a linked list:

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:     IF VAL = PTR->DATA
                SET POS = PTR
                Go To Step 5
            ELSE
                SET PTR = PTR->NEXT
            [END OF IF]
        [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

**Assignment** Q1.) Write a program to create a linked list and perform insertions and deletions of all cases. Write functions to sort and finally delete the entire list at once. (Implement yourself).

# CIRCULAR LINKED LISTS

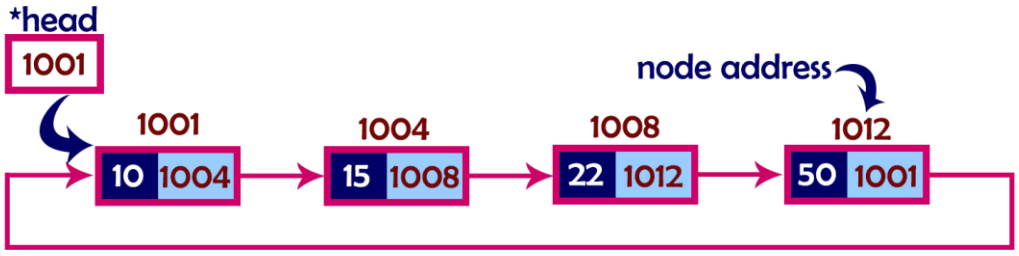
## What is Circular Linked List?

In single linked list, every node points to its next node in the sequence and the last node points NULL. But in circular linked list, every node points to its next node in the sequence but the last node points to the first node in the list.

**Circular linked list is a sequence of elements in which every element has link to its next element in the sequence and the last element has a link to the first element in the sequence.**

That means circular linked list is similar to the single linked list except that the last node points to the first node in the list.

**Example:**




## Inserting a New Node in a Circular Linked List

We will see how a new node is added into an already existing linked list. We will take two cases:

a. The new node is inserted at the beginning of the circular linked list.
b. The new node is inserted at the end of the circular linked list.

### a. Inserting a Node at the Beginning of a Circular Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:     PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = START
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET START = NEW_NODE
Step 11: EXIT
```

b. Inserting a Node at the End of a Circular Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

## Deleting a Node from a Circular Linked List

We will discuss how a node is deleted from an already existing circular linked list. We will take two cases and then see how deletion is done in each case. Rest of the cases of deletion are same as that given for singly linked lists.

a. The first node is deleted.
b. The last node is deleted.

a. Deleting the First Node from a Circular Linked List

```
Step 1: IF START = NULL
              Write UNDERFLOW
              Go to Step 8
         [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4:       SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> NEXT = START -> NEXT
Step 6: FREE START
Step 7: SET START = PTR -> NEXT
Step 8: EXIT
```

b. Deleting the Last Node from a Circular Linked List

```
Step 1: IF START = NULL
              Write UNDERFLOW
              Go to Step 8
         [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != START
Step 4:         SET PREPTR = PTR
Step 5:         SET PTR = PTR -> NEXT
         [END OF LOOP]
Step 6: SET PREPTR -> NEXT = START
Step 7: FREE PTR
Step 8: EXIT
```

**Assignment** Q2.) Write a program to create a circular linked list. Perform insertion and deletion at the beginning and end of the list.

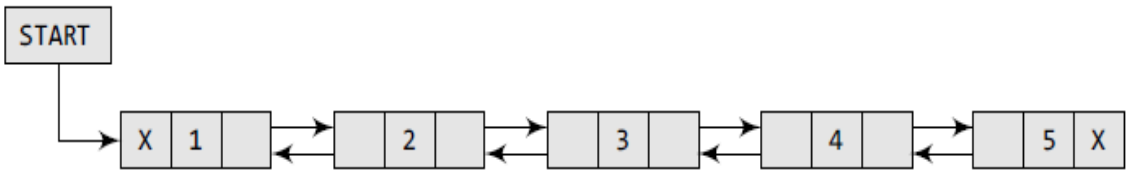
# DOUBLY LINKED LISTS

## What is Double Linked List?

A doubly linked list or a two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence. Therefore, it consists of three parts—data, a pointer to the next node, and a pointer to the previous node. Double linked list can be defined as follows...

**Double linked list is a sequence of elements in which every element has links to its previous element and next element in the sequence.**

Example




In C, the structure of a doubly linked list can be given as,

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

The PREV field of the first node and the NEXT field of the last node will contain NULL. The PREV field is used to store the address of the preceding node, which enables us to traverse the list in the backward direction.

The main advantage of using a doubly linked list is that it makes searching twice as efficient.

## Memory representation of a doubly linked list:

START 1

| | DATA | PREV | NEXT |
|---|---|---|---|
| 1 | H | -1 | 3 |
| 2 | | | |
| 3 | E | 1 | 6 |
| 4 | | | |
| 5 | | | |
| 6 | L | 3 | 7 |
| 7 | L | 6 | 9 |
| 8 | | | |
| 9 | O | 7 | -1 |

## Inserting a New Node in a Doubly Linked List:

We will discuss how a new node is added into an already existing doubly linked list. We will take four cases and then see how insertion is done in each case.

a. The new node is inserted at the beginning.
b. The new node is inserted at the end.
c. The new node is inserted after a given node.
d. The new node is inserted before a given node.

### Inserting a Node at the Beginning of a Doubly Linked List:

```
Step 1: IF AVAIL = NULL
                Write OVERFLOW
                Go to Step 9
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET START -> PREV = NEW_NODE
Step 8: SET START = NEW_NODE
Step 9: EXIT
```

## Inserting a Node at the End of a Doubly Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: EXIT
```

## Inserting a Node After a Given Node in a Doubly Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR -> NEXT
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET PTR -> NEXT = NEW_NODE
Step 11: SET PTR -> NEXT -> PREV = NEW_NODE
Step 12: EXIT
```

## Inserting a Node Before a Given Node in a Doubly Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR
Step 9: SET NEW_NODE -> PREV = PTR -> PREV
Step 10: SET PTR -> PREV = NEW_NODE
Step 11: SET PTR  -> PREV -> NEXT = NEW_NODE
Step 12: EXIT
```

## Deleting a Node from a Doubly Linked List:

we will see how a node is deleted from an already existing doubly linked list. We will take four cases:

a. The first node is deleted.
b. The last node is deleted.
c. The node after a given node is deleted.
d. The node before a given node is deleted.

## Deleting the First Node from a Doubly Linked List:

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 6
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: SET START -> PREV = NULL
Step 5: FREE PTR
Step 6: EXIT
```

## Deleting the Last Node from a Doubly Linked List:

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != NULL
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> PREV -> NEXT = NULL
Step 6: FREE PTR
Step 7: EXIT
```

## Deleting the Node After a Given Node in a Doubly Linked List:

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:      SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> NEXT
Step 6: SET PTR -> NEXT = TEMP -> NEXT
Step 7: SET TEMP -> NEXT -> PREV = PTR
Step 8: FREE TEMP
Step 9: EXIT
```

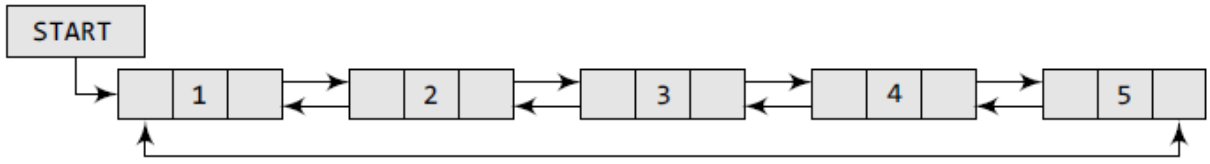
## Deleting the Node Before a Given Node in a Doubly Linked List:

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> PREV
Step 6: SET TEMP -> PREV -> NEXT = PTR
Step 7: SET PTR -> PREV = TEMP -> PREV
Step 8: FREE TEMP
Step 9: EXIT
```

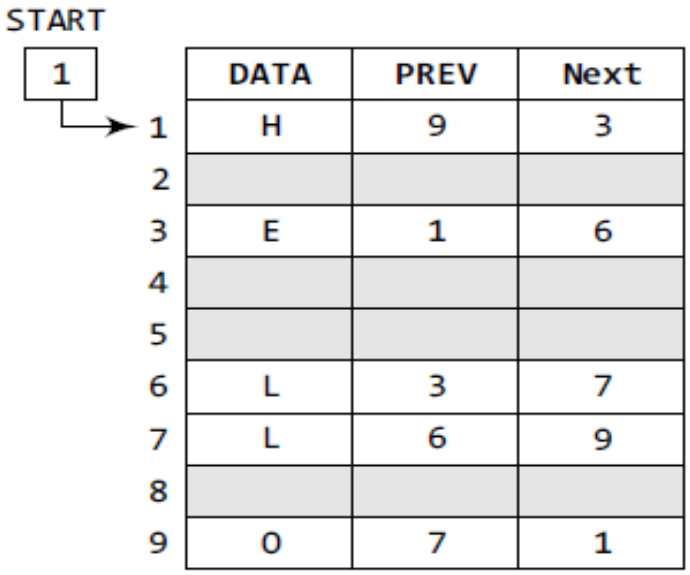
Assignment Q3.) Write a program to create a doubly linked list and perform insertions and deletions in all cases.

## CIRCULAR DOUBLY LINKED LISTS

A circular doubly linked list or a circular two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in the sequence. The difference between a doubly linked and a circular doubly linked list is same as that exists between a singly linked list and a circular linked list. The circular doubly linked list does not contain NULL in the previous field of the first node and the next field of the last node. Rather, the next field of the last node stores the address of the first node of the list, i.e., START. Similarly, the previous field of the first field stores the address of the last node.




The main advantage of using a circular doubly linked list is that it makes search operation twice as efficient.




| | DATA | PREV | Next |
|---|---|---|---|
| 1 | H | 9 | 3 |
| 2 | | | |
| 3 | E | 1 | 6 |
| 4 | | | |
| 5 | | | |
| 6 | L | 3 | 7 |
| 7 | L | 6 | 9 |
| 8 | | | |
| 9 | O | 7 | 1 |

## Inserting a New Node in a Circular Doubly Linked List

We will see how a new node is added into an already existing circular doubly linked list. We will take two cases:

a. The new node is inserted at the beginning.
b. The new node is inserted at the end.

### Inserting a Node at the Beginning of a Circular Doubly Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 13
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET PTR -> NEXT = NEW_NODE
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET NEW_NODE -> NEXT = START
Step 11: SET START -> PREV = NEW_NODE
Step 12: SET START = NEW_NODE
Step 13: EXIT
```

## Inserting a Node at the End of a Circular Doubly Linked List:

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: SET START -> PREV = NEW_NODE
Step 12: EXIT
```

## Deleting a Node from a Circular Doubly Linked List:

we will see how a node is deleted from an already existing circular doubly linked list. We will take two cases:

a. The first node is deleted.
b. The last node is deleted.

## Deleting the First Node from a Circular Doubly Linked List:

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> NEXT = START -> NEXT
Step 6: SET START -> NEXT -> PREV = PTR
Step 7: FREE START
Step 8: SET START = PTR -> NEXT
```

## Deleting the Last Node from a Circular Doubly Linked List:

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4  while PTR -> NEXT != START
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> PREV -> NEXT = START
Step 6: SET START -> PREV = PTR -> PREV
Step 7: FREE PTR
Step 8: EXIT
```

Assignment Q4.) Write a program to create a circular doubly linked list and perform insertions and deletions at the beginning and end of the list.

**********THE END**********