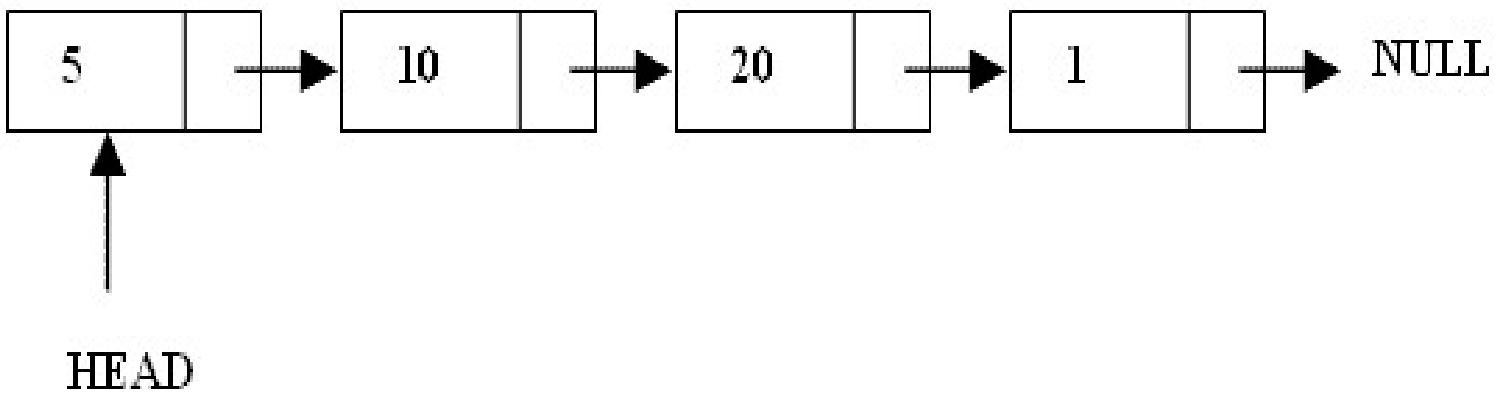
Linked List

Linked List

- Linked List is a data structure used for storing collection of data.
- Linked List has the following properties:
  - Successive elements are connected by pointers.
  - Last elements points to NULL.
  - It can grow or shrink size during execution of a program.
  - It can be made just as long as required.
  - It does not waste memory space.
  - Insertions and deletions are easier and efficient.

Linked list

- Nodes make up linked lists.
- Nodes are structures made up of data and a pointer to another node.
- Usually the pointer is called next.



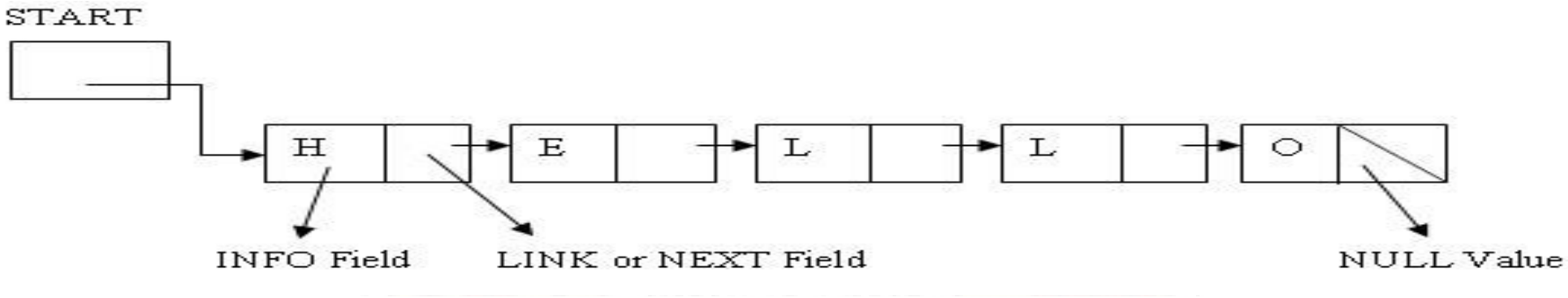
Arrays Vs Linked Lists

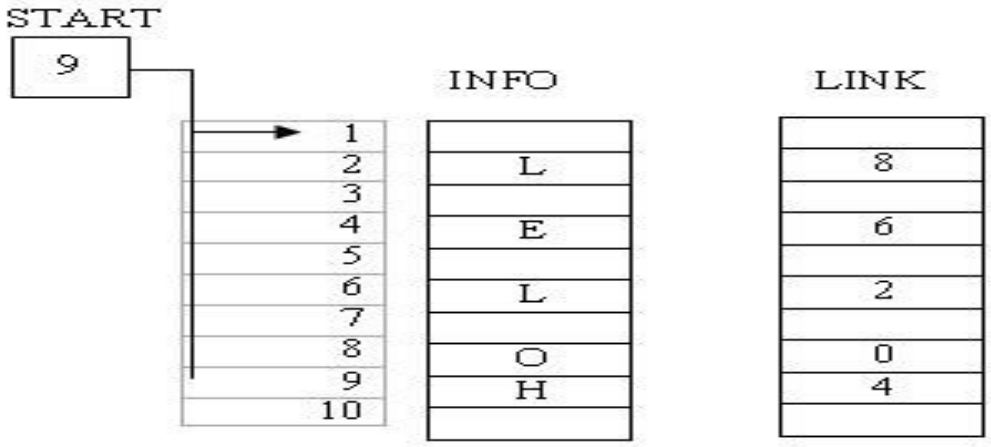
	Array	Linked List
Define	Array is a collection of elements having same data type with common name.	Linked list is an ordered collection of elements which are connected by links/pointers.
Access	In array, elements can be accessed using index/subscript value, i.e. elements can be randomly accessed like arr[0], arr[3], etc. So array provides fast and **random access.**	In linked list, elements can't be accessed randomly but can be accessed only **sequentially** and accessing element takes **0(n) time**.
Memory Structure	In array, elements are stored in **consecutive** manner in memory.	In linked list, elements can be stored at any available place as address of node is stored in previous node.
Insertion & Deletion	Insertion & deletion takes more time in array as elements are stored in consecutive memory locations.	Insertion & deletion are fast & easy in linked list as only value of pointer is needed to change.

Arrays Vs Linked Lists

	Array	Linked List
Memory Allocation	In array, memory is allocated at compile time i.e. **Static Memory Allocation**.	In linked list, memory is allocated at run time i.e. **Dynamic Memory Allocation.**
Types	Array can be **single dimensional**, two dimension or **multidimensional**.	Linked list can be **singly**, **doubly** or **circular** linked list.
Dependency	In array, each element is independent, no connection with previous element or with its location.	In Linked list, location or address of elements is stored in the link part of previous element/node.
Extra Space	In array, no pointers are used like linked list so no need of extra space in memory for pointer.	In linked list, adjacency between the elements are maintained using pointers or links, so pointers are used and for that extra memory space is needed.

Representation of Linked list in memory





Here

START = 9	=>	INFO[9] = H is the first character.
LINK[9] = 4	=>	INFO[4] = E is the second character.
LINK[4] = 6	=>	INFO[6] = L is the third character.
LINK[6] = 2	=>	INFO[2] = L is the fourth character.
LINK[2] = 8	=>	INFO[8] = O is the fifth character.
LINK[8] = 0	=>	The NULL value, so the LIST ends here.

Types of lists

There are four types of linked list:

- Singly Linked list
- Doubly linked list
- Circular Linked List
- Circular Doubly Linked List

Operation on linked list

The basic operations to be performed on the linked lists are as follows:

1. Creation
2. Insertion
3. Deletion
4. Traversing
5. Searching
6. Concatenation
7. Display

Singly Linked List

- All nodes are linked together in some sequential manner. Hence, it is also called Linear Linked List.
- Each node has only one link part
- Each link part contains the address of the next node in the list
- Link part of the last node contains NULL value which signifies the end of the node.

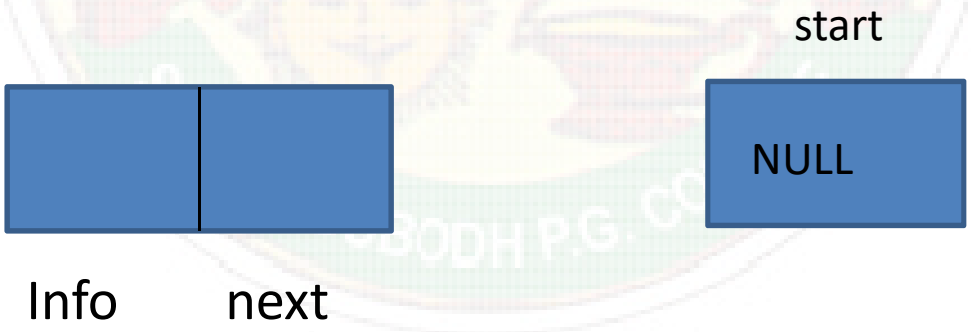
Singly Linked List

- Steps to store data:

1. Create Node
2. Store data in it.
3. Link this with previous stored data

Creating a node

```
typedef struct link
{
   int info;
   struct link *next                    //pointer to node i.e. node stores
                                          the address of type link
 }node;

node *start=NULL;                  // List Empty
```



```
void create()
{
        node *p, *l;
        int i,n;
        l=start;
        printf("Enter number of nodes");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                p = (node *) malloc(sizeof(node));
                if(p == NULL)
                        printf("Overflow problem");          // No memory in RAM
                else
                {
                        printf("enter node data");                  //store data
                        scanf("%d",p->info);
                        p->next=NULL;
                        if(start==NULL)                             // p is first node
                                start=p;
                        else
                                l->next=p;     //link with previous stored data
                         l=p;                  // new node is last node
                }
        }
}
```

Traversal

```
void traverse()
{
        node *p;
        p=start;
        while(p!=NULL)
        {
                printf("%d",p->info);
                p=p->next;
        }
}
```

Searching

```
void search(int ele)
{
        node *p;
        p=start;
        while((p!=NULL) && (p->info!=ele)
                p=p->next;
        if(p->info == ele)
                printf(Element found”);
        else
                printf(“element not found”);
}
```

Insertion

To insert an element into the Linked List, following three things should be done:

1. Allocating a node
2. Assigning the data
3. Adjusting the pointers

Insertion in Linked List can be done at:

1. The beginning
2. The end
3. Any position

Insertion at beginning

ALGORITHM

Step 1. Create a new node and assign the address to any node say ptr.

Step 2. ASSIGN INFO[PTR] := ITEM

Step 3. IF(START = NULL)

ASSIGN NEXT[PTR] := NULL

ELSE

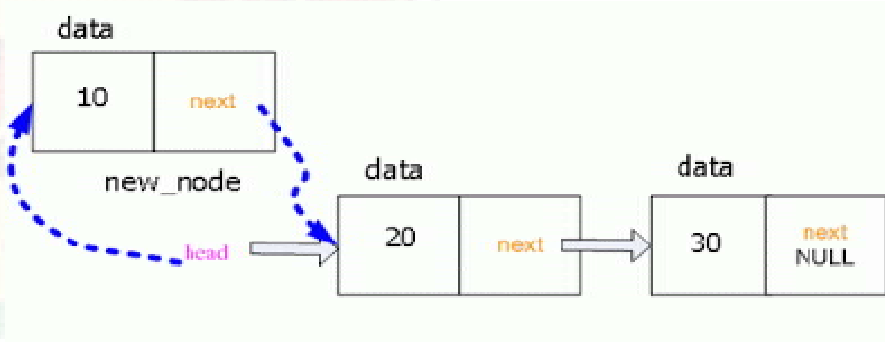
ASSIGN NEXT[PTR] := START

Step 4. ASSIGN START := PTR

Step 5. EXIT

Insertion at beginning

```
Void insertS(int item)
{
        node *ptr,*p;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
        {       printf("Overflow");
        }
        else
        {
                ptr->info=item;
                if(start==NULL)
                        ptr->next=NULL;
                else
                        ptr->next=start;
                start=ptr;
        }
}
```



Insertion at End

ALGORITHM

Step 1. Create a new node and assign the address to any node say ptr.

Step 2. ASSIGN INFO[PTR] := ITEM
NEXT[PTR] :=NULL

Step 3. IF(START = NULL)
ASSIGN START=PTR.
ELSE
{
ASSIGN P := START
WHILE(NEXT[P] != NULL)
ASSIGN P := NEXT[P]
ASSIGN NEXT[P] = PTR
}

Step 4. EXIT

Insertion at End

```
Void insertL(int item)
{
        node *ptr,*p;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
                printf("OVERFLOW");
        else
        {
                ptr->info=item;
                if(start == NULL)
                        start = ptr;
                else
                {
                        p=start;
                        while(p->next !=NULL)
                        {
                           p = p->next;
                        }
                        p->next=ptr;
                }
                ptr->next=NULL;
        }
}
```

Insertion at any Position

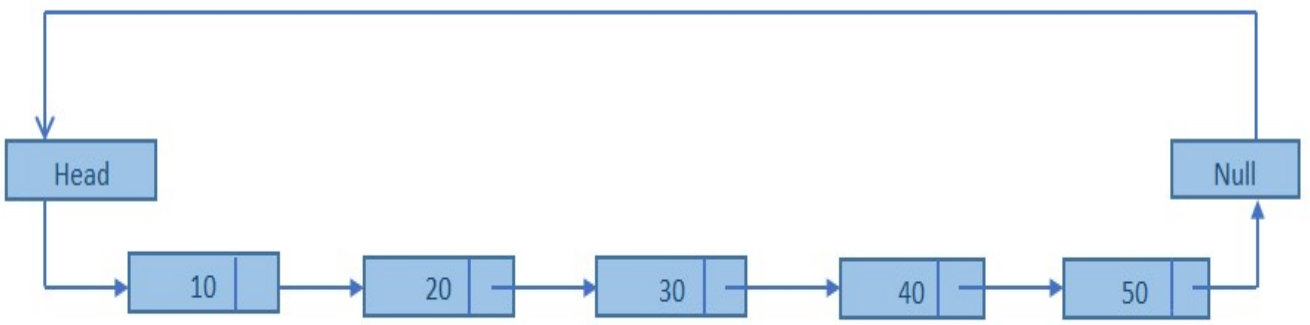
```
Void insertP(int item, int pos)
{
        node *ptr,*p,*q;
        int i;
        p=start;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
                printf("OVERFLOW");
        else
        {
                ptr->info=item;
                for(i=1; i<loc; i++)
                {
                        q=p;
                        p = p->next;
                }
                q->next = ptr;
                ptr->next = p;
        }

}
```

Deletion

```
Void delete(int pos)
{
        node *p,*q;
        q=p=start;                          //p is node to be deleted & q is node previous to p
        for(i=1; (i<pos) && (p != NULL); i++)
        {
                q=p;
                p = p->next;
        }
        if(p == NULL)                                   // now check for p is start/last/any/not present
                printf("Element not found");
        else                                            // Element in List
        {
                if(p == start)
                        start=start->next;
                else
                        q->next=p->next;
        p->next=NULL;
        free(p);
        }
}
```

Circular Linked List

Circular list is a list in which the link field of the last node is made to point to the start/first node of the list.



```
void create()
{
        node *p, *l;
        int i,n;
        l=start;
        printf("Enter number of nodes");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                p = (node *) malloc(sizeof(node));
                if(p == NULL)
                        printf("OVERFLOW");
                else
                {
                        printf("enter node data");                    //store data
                        scanf("%d",p->info);
                        if(start==NULL)                               // p is first node
                                start=p;
                        else
                                l->next=p;     //link with previous stored data
                        l=p;                  // new node is last node
                        p->next=start;
                }
        }
}
```

Insertion

- Insertion at any position is similar to insertion in Linear Linked List.

- But when we are inserting at last or first then it is different.

Insertion at beginning(CLL)

ALGORITHM

Step 1. Create a new node and assign the address to any node say ptr.

Step 2. ASSIGN INFO[PTR] := ITEM

Step 3. ASSIGN P := START

Step 4. REPEAT WHILE(NEXT[P] != START)

P := NEXT[P]

Step 5. ASSIGN NEXT[P] := PTR

NEXT[PTR] := START

START := PTR

Step 6. EXIT

Insertion at beginning (CLL)

```
void insertS(int item)
{
        node *ptr,*p;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
                printf("OVERFLOW");
        else
        {
                ptr->info=item;
                p = start;
                while(p->next != start)
                        p = p->next;
                p->next = ptr;
                ptr->next=start;
                start=ptr;
        }
}
```

Insertion at END (CLL)

```
Void insertL(int item)
{
        node *ptr,*p;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
                printf("OVERFLOW");
        else
        {
                ptr->info=item;
                p = start;
                while(p->next != start)
                        p = p->next;
                p->next = ptr;                  //used for insertion at last
                ptr->next=start;
        }
}
```

Deletion at start

- Deletion at last and any position is similar to insertion in Linear Linked List.
- But when we are deleting at first then it is different. In this case make next node as start. Make last node point to new start.

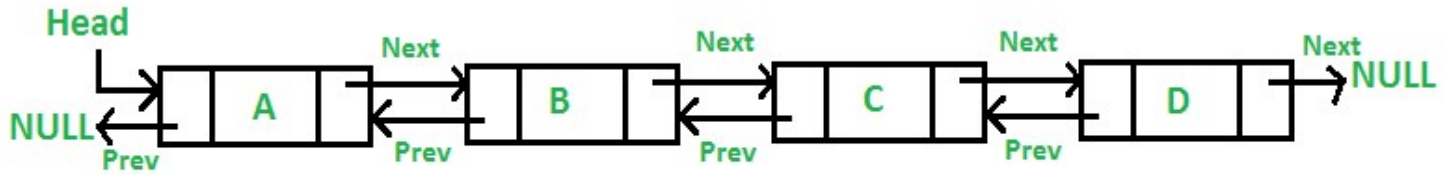
```
void deleteS()
{
        node *l;
        l=start;
        while(l->next != start)
                l = l->next;
        l->next = start->next;
        start = start->next;
        free(l);
}
```

Deletion at End

```
void deleteE()
{
        node *l, *q;
        q=l=start;
        while(l->next != start)
        {
                q=l;
                l = l->next;
        }
        q->next = l->next;
        free(l);
}
```

Doubly Linked list

- A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.
- It is also known as two-way list.



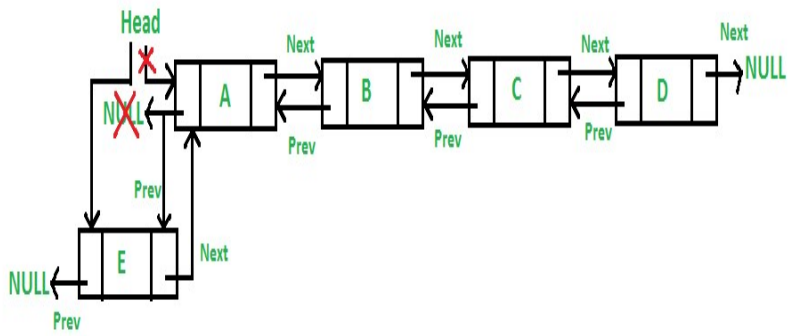
Doubly Linked list

```
typedef struct link
{
    int info;
    struct link *next;
    struct link *prev;
 }node;

Node *start = NULL;
```

```
void create()
{
        node *p, *l;
        int i,n;
        l=start;
        printf("Enter number of nodes");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                p = (node *) malloc(sizeof(node));
                if(p == NULL)
                        printf("OVERFLOW");
                else
                {
                        printf("enter node data");
                        scanf("%d",p->info);
                        p->next = NULL;
                        if(start==NULL)
                        {       start=p;
                                p->prev =NULL;
                        }
                        else
                        {       l->next=p;
                                p->prev=l;
                        }
                        l=p;
                }
        }
}
```

Insertion at beginning (Dll)

```
void insertS(int item)
{
        node *ptr;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
                printf("OVERFLOW");
        else
        {       ptr->info=item;
                if(start == NULL)
                {       start = ptr;
                        ptr->next = NULL;
                        ptr->prev = NULL;
                }
                else
                {       ptr->prev = NULL;
                        ptr->next = start;
                        start->prev = ptr;
                        start = ptr;
                }
        }
}
```



Insertion at END(DLL)

```
void insertS(int item)
{
        node *ptr, *p;
        p = start;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
                printf("OVERFLOW");
        else
        {       ptr->info=item;
                if(start == NULL)
                {       start = ptr;
                        ptr->next = NULL;
                        ptr->prev = NULL;
                }
                else
                {       while(p->next != NULL)
                                p = p->next;
                        p->next = ptr;
                        ptr->prev = p;
                        ptr->next = NULL;
                }
        }
}
```

Insertion at position(DLL)

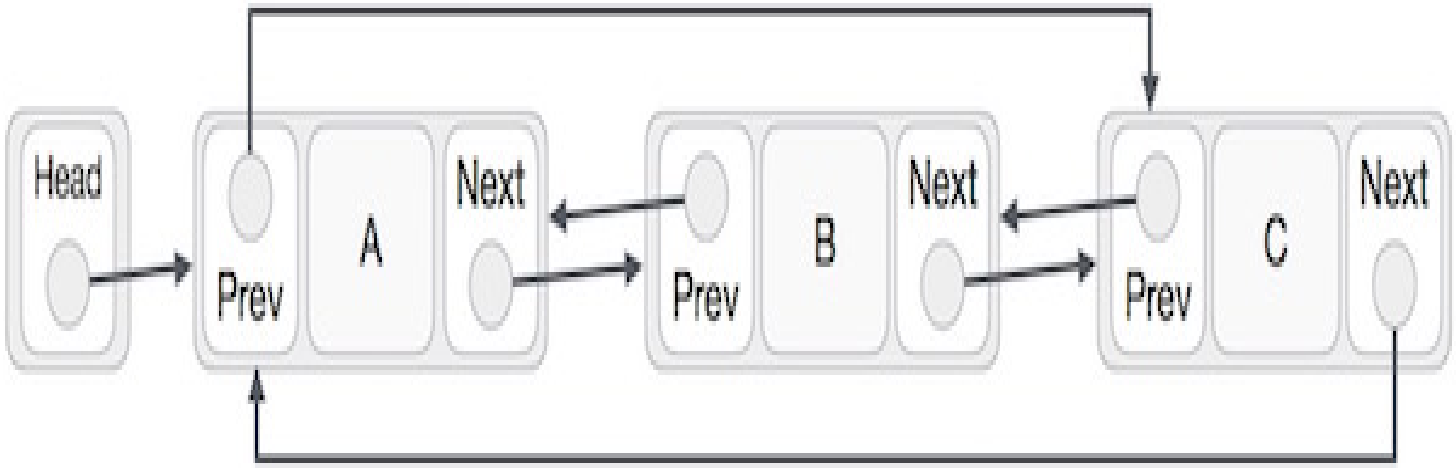
```
void insertS(int item)
{
        node *ptr, *p, *temp;
        p = start;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
                printf("OVERFLOW");
        else
        {       ptr->info=item;
                for(i=1; i<pos; i++)
                        p=p->next;
                ptr->prev = p;
                ptr->next = p->next;
                p->next = ptr;
                temp = ptr->next;
                temp->prev = ptr;
        }
}
```

Deletion in DLL

```
Void delete(int pos)
{
        node *p,*q;
        p=start;                        //p is node to be deleted & q is node previous to p
        for(i=1; (i<pos) && (p != NULL); i++)
        {
                p = p->next;
        }
        if(p == NULL)
                printf("Element not found");
        else
        {
                if(p == start)
                        start=start->next;
                else
                {       p->prev->next = p->next;
                        p->next->prev = p->prev;
                }
        free(p);
        }
}
```

Doubly circular Linked list(dcll)

- In doubly linked list, the next pointer of the last node points to the first node and the previous pointer of the first node points to the last node making the circular in both directions.
- Insertion and deletion at specified position is same as doubly linked list.



```
void create()
{
        node *p, *l;
        int i,n;
        l=start;
        printf("Enter number of nodes");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                p = (node *) malloc(sizeof(node));
                if(p == NULL)
                        printf("OVERFLOW");
                else
                {
                        printf("enter node data");
                        scanf("%d",p->info);
                        p->next = NULL;
                        p->prev = NULL;
```

```
                if(start==NULL)
                {       start=p;
                        p->next = start;
                        p->prev= start;
                }
                else
                {       l->next=p;
                        p->prev=l;
                }
                l=p;
                p->next = start;
        }                                       // end of else
    }                                           // end of for loop
}                                               //end of create()
```

Insertion at beginning (DCLL)

```
void insertS(int item)
{
        node *ptr,*p;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
                printf("OVERFLOW");
        else
        {
                ptr->info=item;
                p = start;
                while(p->next != start)
                        p = p->next;
                p->next = ptr;
                ptr->next=start;
                start=ptr;
                start->prev = start
        }
}
```

Insertion at END (DCLL)

```
Void insertL(int item)
{
        node *ptr,*p;
        ptr=(node *) malloc(sizeof(node));
        if(ptr == NULL)
                printf("OVERFLOW");
        else
        {
                ptr->info=item;
                p = start;
                while(p->next != start)
                        p = p->next;
                p->next = ptr;
                ptr->next=start;
                ptr->prev = p;
        }
}
```

Deletion in DLL

```
Void delete(int pos)
{
        node *p,*q;
        p=start;                        //p is node to be deleted & q is node previous to p
        for(i=1; (i<pos) && (p != NULL); i++)
        {
                p = p->next;
        }
        if(p == NULL)
                printf("Element not found");
        else
        {
                if(p == start)
                {       start=start->next;
                        start->prev = p;
                {
                else
                {       p->prev->next = p->next;
                        p->next->prev = p->prev;
                }
        free(p);
        }
}
```

Thanks