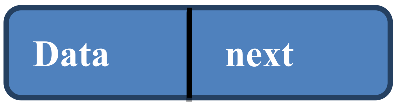
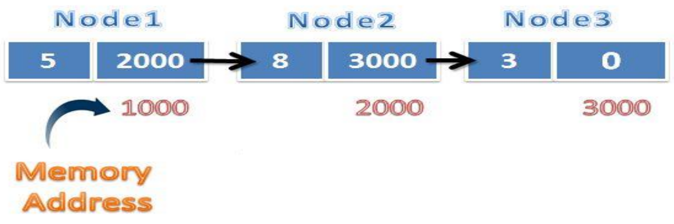
Linked list

- Linked List is **Series of Nodes.**
- Each node Consist of two Parts viz **Data Part** & **Pointer Part**.
- Pointer Part stores the **address of the next node.**



- It is a data Structure which consists of group of nodes that forms a sequence.
- Linked list comprise of group or list of nodes in which each node have link to next node to form a chain



What is linked list? State its advantage, Explain its types [3M DEC-18]

Advantages of linked list

- Linked List is **Dynamic data Structure**.
- Linked List **can grow and shrink during run time**.
- **Insertion and Deletion** Operations are Easier.
- **Efficient Memory Utilization**, i.e no need to pre-allocate memory
- Faster Access time, can be expanded in **constant time without memory overhead**
- Linear Data Structures such as Stack, Queue can be **easily implemeted** using Linked list

Disadvantages of Linked List

1. **Wastage of Memory**

   Pointer Requires **extra memory for storage**.

2. **No Random Access**

   In array we can access **nth element easily just by using a[n]**.
   In Linked list no random access is given to user, we have to **access each node sequentially**.

3. **Time Complexity**

   Array can be randomly accessed , while the Linked list **cannot be accessed Randomly.**

4. **Reverse Traversing is difficult**

   In case if we are using singly linked list then it is **very difficult to traverse linked list from end**.

5. **Heap Space Restriction**

   If there is **insufficient space in heap then it won't create any memory**.

Linked list as an ADT

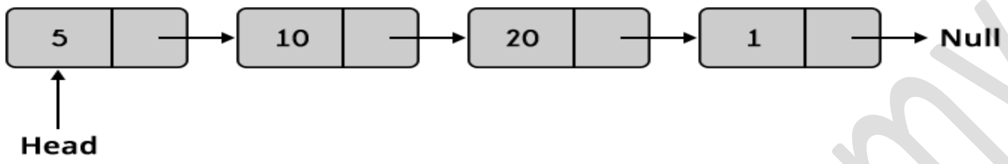
Structure in "c" can be used to define a node.
Address of the successor node can be stored in pointer variable.

Linked list is a self referential structure in "c". each node has variable to store data and next field store the address of next field.

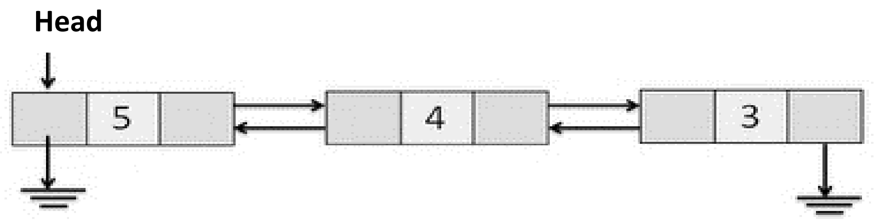
```
typedef struct node
{
        int data;
        struct node *next;
}node;
```

Types of linked list

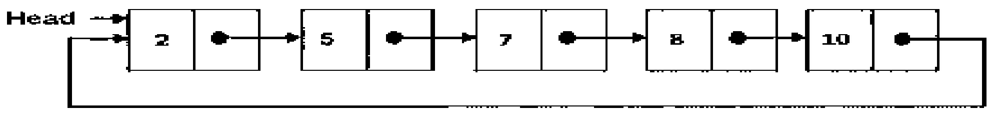
1. **Singly Linked List**
   - In this type of Linked List two **successive nodes** are linked together in **linear fashion** .
   - Each Node contain **address of the next node** to be followed.
   - In Singly Linked List only Linear or Forward **Sequential movement is possible**



2. **Doubly linked list**
   - In this type of linked list each node holds two pointer field.
   - In doubly linked list address of next as well as preceding element are linked with current node.



3. **Circular linked list**
   - In a circular list the first and last element are adjacent.
   - A linked list can be made circular by storing the address of first node in next field of last node.



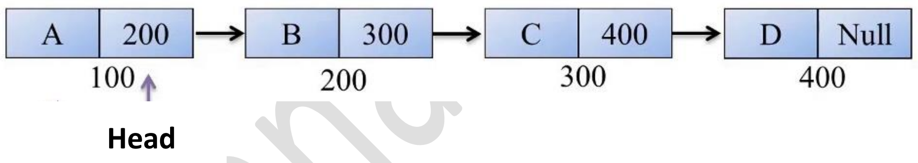
What is the difference between Singly Linked List and Doubly Linked List?

- Each element in the singly linked list contains a reference to the next element in the list, while each element in the doubly linked list contains references to the next element as well as the previous element in the list.

- Doubly linked lists require more space for each element in the list and elementary operations such as insertion and deletion is more complex since they have to deal with two references.
- But doubly link lists allow easier manipulation since it allows traversing the list in forward and backward directions.

1. Singly Linked List

- In this type of Linked List two **successive nodes** are linked together in **linear fashion**.
- Each Node will store the data and **address of the next node** to be followed.
- In Singly Linked List only Linear or Forward **Sequential movement is possible**.
- Elements are accessed sequentially , **no direct access** is allowed.
- Last Node have successor reference as "**NULL**".



Operation on linked list

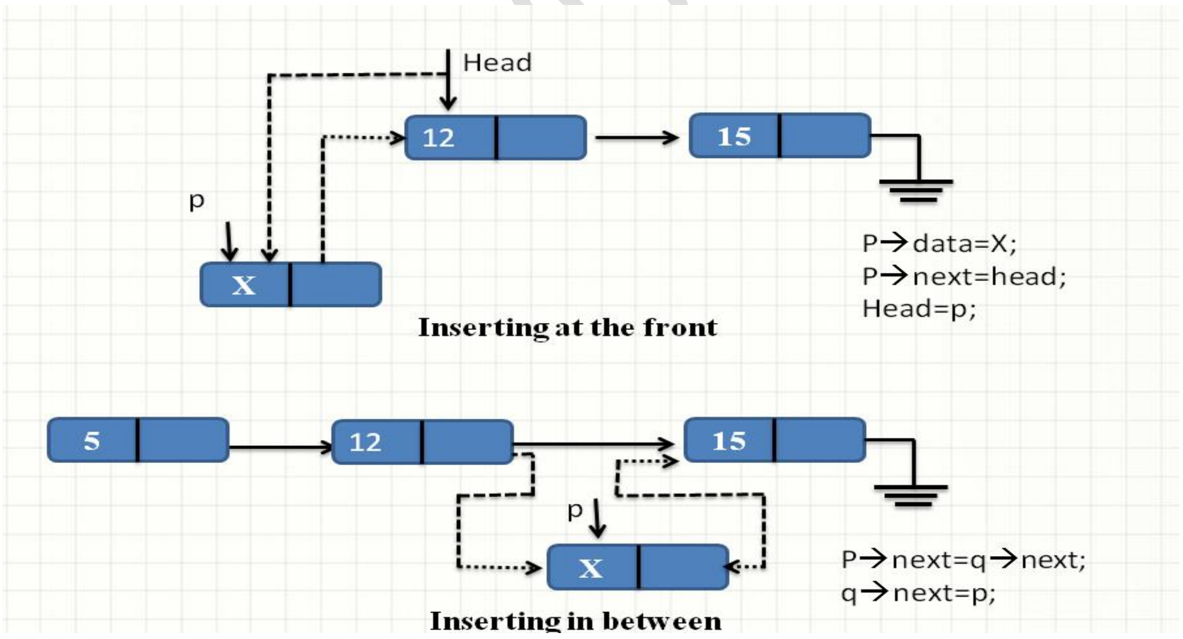
1. **Traversing**
2. **Creating**
3. **Inserting**
4. **Deleting**
5. **Merging**
6. **Sorting**
7. **Searching**

Insertion of node in Singly Linked list

Inserting a new Item, say x has three Situations.

1. Insertion at the front of the list.
2. Insertion at the middle of the list.
3. Insertion at the end of the list.

1. **Inserting a node at beginning of a linked list**



Algorithm

1. Obtain space for new node.
2. Assign data to the data field of the new node.
3. Set the next field of the new node to the beginning of the linked list.

4. Change the reference pointer of the linked list to point to the new node.

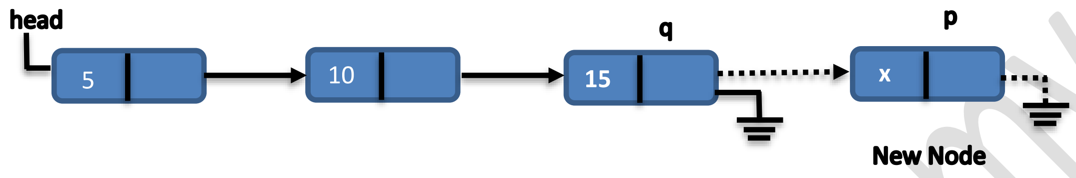
Inserting the new node after node N1

Algorithm

1. Obtain space for new node.
2. Assign value to its data field.
3. Search for the node n1.
4. Set the next field of new node to point to n1→next.
5. Set the next field of n1 to point to the new node.

```
node *insert(node *head, int x, int key)
{
        /* if key is -1 then x is to be inserted at front
        node *p,*q;
        /* space for a new node
        p=(node*) malloc(sizeof(node));
        p->data=x;
        if(head→data==key)
        {
                /* insert the node at the front of the list.
                p->next=head;
                head=p;
        }
        else
        {
                /* search key in the linked list
                q=head;
                while(key!=q->data && q!=null )
                {
                        q=q->next;
                }
                if(q!=null)
                {
                        /*  if key is found
                        p->next=q->next;
                        q->next=p;
                }
        }
}
```

Inserting an item at the end of linked list:



Algorithm:

1. Allocate memory for new node.
2. Assign value to data field of new node.
3. If (head==NULL) then head=p; got step 6.
4. Position a pointer q on last node by traversing the linked list.
5. Store the address of new node in next field of node q.
6. Stop.

```
node *insert_end (node *head, int x)
{
    node *p,*q;
    /* space for a new node
    p=(node*) malloc(sizeof(node));
    p→data=x;
    p→next=NULL;
    if(head==NULL)
    {
            return(p);
    }
    else
    {
            q=head;
            while(q→next!=NULL)
            {
                    q=q→next;
            }
            q→next=p;
            return(head);
    }
```

__Inserting an element x at location ‘LOC’__

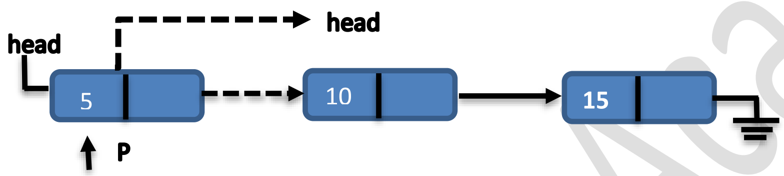
```
node *insert_loc(nod *head, int x, int loc)
{
        node *p,*q;
        inti;
        p=(node*) malloc(sizeof(node));
        p→data=x;
        p→next=NULL;
        if(loc==1)
        {
                p→next=head;
                return(p);
        }
        else
        {
                q=head;
                for(i=1;i<=loc-1;i++)
                {
                        if(q!=NULL)
                        {
                                q=q→next;
                        }
                        else
                        {
                                printf(“\n Overflow);
                                return(head);
                        }
                        p→next=q→next;
                        q→next=p;
                        return(head);
        }
}
```

Delete a node of a linked list

Deleting a node has three cases

1. **Deleting first node**
2. **Deleting last node**
3. **Deleting from middle of list**

1. Deleting the first node



Algorithm

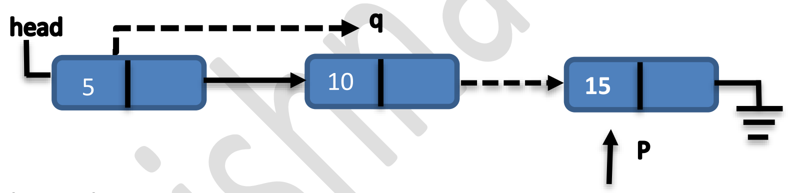
- Store the address of the first node in a pointer variable, say p.
- Move the head to the next node.
- Free the node whose address is stored in the pointer variable P.

Algorithm Deleting a node from the middle

- Store the address of preceding node in a pointer variable, say p.
- Node to be deleted is marked as key node.
- Store the address of the key node in a pointer variable q, so that it can be free.
- Make the successor of key node as successor of the node pointed by p.
- Free the node whose address is stored in the pointer variable q.

```
node * delete(node *head, int x)
{
        node *p,*q;
        if(x==head→data)
        {
                p=head;
                head=head-->next;
                free(p);
        }
        else
        {
                while(x!=(p→next)→data && p→next!=null)
                {
                        p=p→next;
                }
                if(p→next!=null)
                {
                        q=p→next;
                        p→next=(p→next)→next;
                        free(q);
                }
        }
        return(head);
}
```

Deleting of last node in linked list



Algorithm

2. If the first node itself is the last node then [make the linked list empty]
3. Position a pointer q on first node
4. Traverse in the list till q reach to previous node of last node
5. Position a pointer ‘p’ on last node
6. Free node whose address is store in pointer ‘p’
7. Set q→ next to null.

```
node  *delete_last(node  *head)
{
        node *p,  *q;
        If(head→next==NULL)
        {
                free(head);
                return(head);
        }
        q=head;
        While(q→next→next !=NULL)
        {
                q=q→next;
        }
        p=q→next;
        free(p);
        q→next=NULL;
        return(head);
}
```

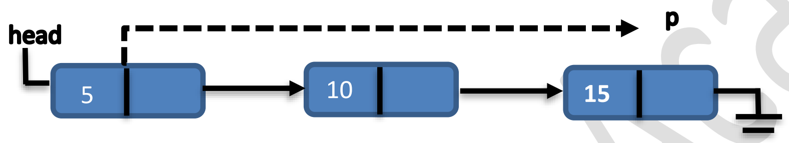
Searching a data in Linked list

Algorithm

```
1. p=head
2. If the data in node p is x then
   End search with success i.e. return(1)
3.Continue searching p=p→next.
4.If linked list end then end with failure i.e. return(0)
5.go to step 2
6.stop.
```

```
        int search(node *head, int x)
        {
                node *p;
                p=head;
                if(head→data==x)
                {
                        return(1);
                }
                else
                {
                        p=p→next;
                        while(p!=null)
                        {
                                if(p-->data=x)
                                        return(1);
                                p=p-->next;
                        }
                        return(0);
                }
        }
```

Counting number of nodes in linked list



```
int count(node *p)
{
        inti;
        i=0;
        while(p!=NULL)
        {
                i=i+1;
                p=p->next;
        }
        return(i);
}
```

Write a program to create linked list

```
#include<stdio.h>
#include<conio.h>
typedef struct node
{
        int data;
        struct node *next;
}node;
void main()
{
        node *Head, *p;
        int n, x, i;
        printf(" Enter no of items");
        scanf("%d", &n);
        Head=(node*)malloc(sizeof(node));
        //read the data in 1st node
        scanf("%d", &(Head->data));
        Head->next=Null;
        p=Head;
        for(i=1;i<n;i++)
        {
                p->next=(node*)malloc(sizeof(node));
                p=p->next;
                p->next=NULL;
                scanf("%d", &(p->data));
        }
        getch();
}
```

Write a ‘C’ program to create a single linked list which support following operations. [10M May-18, Dec-18,

1) creating a linked list
2) inserting a node in the beginning
3) Insert a node at the end
4) delete a specific node
5) Display list

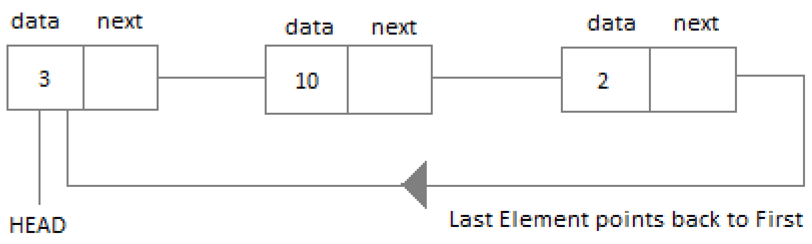
```
#include<stdio.h>
#include<conio.h>
typedef struct node
{
        Int data;
        Struct node *next;
```

```
}node;

node* create();
node* insert_b(node * head, intX);
node* insert_e(node * head, intx);
node* delete(node * head);
void print(node* head);
void main()
{
        int op, x;
        node * head=null;
        do
        {
                printf("1-create\n 2-insert in beginning \n 3-Insert at
                end\n 4-delete a specific\n 5-display\n 6-quit\n");
                printf("enter your choice\n");
                scanf("%d",&top);
                switch(op)
                {

                        case1:
                                head=create();
                                break;
                        case2:
                                printf("enter the data to be inserted\n);
                                scanf("%d",&x);
                                head=insert_b(head, x);
                                break;
                        case3:
                                printf("enter data to be inserted\n");
                                scanf("%d",&x);
                                head=insert_e(head, x);
                                break;
                        case4:
                                printf("enter the data to be deleted\n");
                                scanf("%d",&x);
                                head=delete(head,x);
                                break;
                        case5:
                                print(head);
                        break;
                default:

        }
}while(op!=6);
getch();
}
void create ()
{
                node *head, *p;
                int n, x, i;
                printf(" Enter no of items");
                scanf("%d", &n);
                head=(node*)malloc(sizeof(node));
                //read the data in 1st node
                scanf("%d", &(head->data));
                head->next=null;
                p=head;
                for(i=1;i<n;i++)
                {
                        p->next=(node*)malloc(sizeof(node));
                        p=p->next;
                        p->next=NULL;
                        scanf("%d", &(p->data));
                }
}

node *insert_b(node *head, int x)
{
        node *p;
        /* space for a new node
        p=(node*) malloc(sizeof(node));
        p→data=x;
        /* insert the node at the front of the list.
        p→next=head;
        head=p;
}
```

```
node *insert_e (node *head, int x)
{
        node *p,*q;
        /* space for a new node
        p=(node*) malloc(sizeof(node));
        p→data=x;
        p→next=NULL;
        if(head==NULL)
        {
                return(p);
        }
        else
        {
                q=head;
                while(q→next !=NULL)
                {
                        q=q→next;
                }
                q→next=p;
                return(head);
        }
}

node * delete(node *head, int x)
{
        node *p,*q;
        if(x==head→data)
        {
                p=head;
                head=head→next;
                free(p);
        }
        else
        {
                while(x!=(p→next)→ data && p→next!=null)
                {
                        p=p→next;
                }
                if(p→next!=null)
                {
                        q=p→next;
                        p→next=(p→next)→next;
                        free(q);
                }
        }
        return(head);
}

void print(node * head)
{
        node *p;
        for(p=head; p!=null; p=p→next)
        {
                printf("%d", p→data);
        }
}
```

<u>Circular Linked list</u>

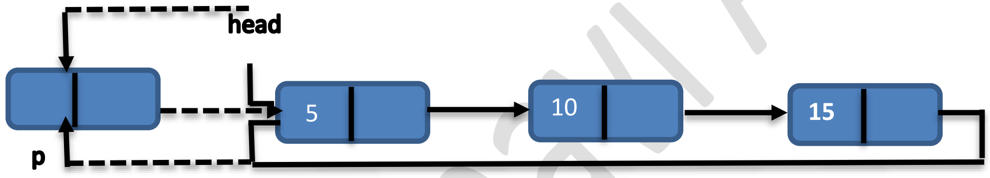
- **In circular linked list, last node is connected back to the first node.**
- **Queue data structure can be implemented using a circular linked list.**
- **Front node can be accessed through the root node.**



Operation on circular linked list:

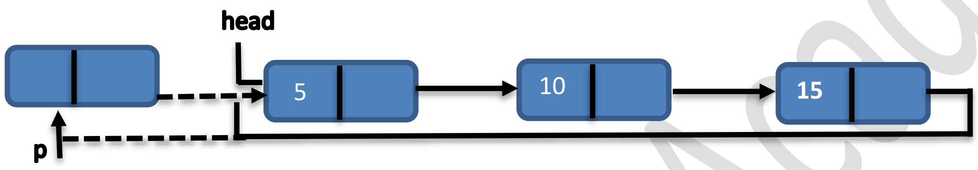
1) Insertion
2) Deletion
3) Display

<u>Insertion of a node at the front of circular linked list</u>



```
node * Insert_b(node *head, int x)
{
        node*p,*q;
        p=(node*)malloc(size of (node);
        p→data=x;
        if(head==NULL)
        {
                head= p;
                head→next=head;
        }
        else
        {
                q=head;
                while(q→next != head)
                {
                        q=q→next;
                }
                q→next=p;
                p→next=head;
                p=head;
                return(head);
        }
}
```

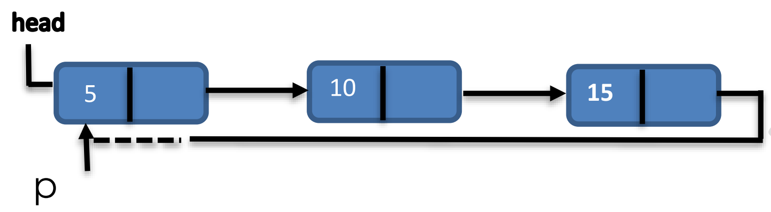
Insertion of a node at the end of circular linked



```
node * insert_e(node *head, int x)
{
        node*p,*q;
        p=(node*)malloc(size of (node);
        p→data=x;
        if(head==NULL)
        {
                head= p;
                p→next=p;
        }
        else
        {
                q=head;
                while(q→next != head)
                {
                        q=q→next;
                }
                q→next=p;
                p→next=head;
                return(head);
        }
}
```

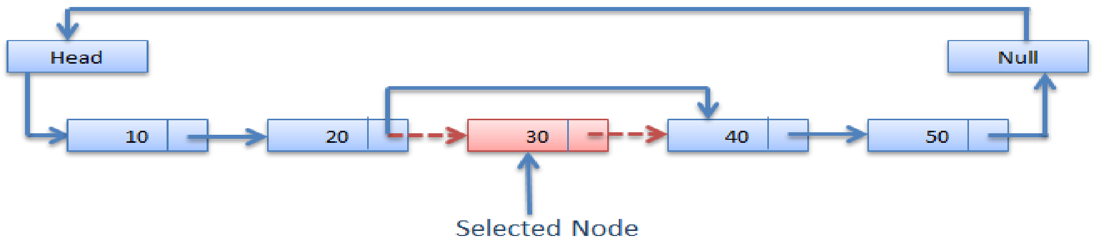
Deletion of node in circular linked list

Delete a beginning node



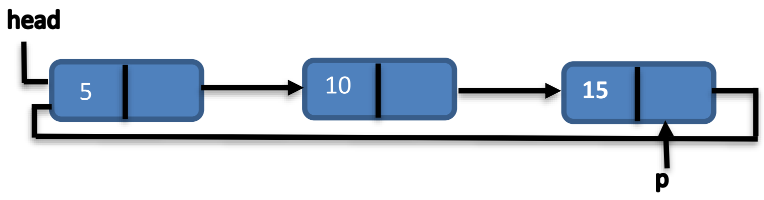
```
void delete_b(node * head)
{
        node *p;
        if(head→data==head);
        {
                free(p);
                head=null
                  return (head);
        }
        else
        {
                p=head;
                head =head→next;
                free(p);
                return (head);
        }
}
```

8. Delete a specific node



```
void delete(node * head, int key)
{
        node *p,*q;
        if(head→data==key);
        {
                free(p);
                head=null
                return (head);
        }
        else
        {
                q=head;
                while (q→next→data !=key)
                {
                        q=q→next;
                }
                p=q→next;
                q→next=p→next;
                free(p);
                return (head);
        }
}
```

3. Delete a end node



```
void delete_e(node * head)
{
        node *p;
        if(head→data==head);
        {
                free(p);
                head=null
                return (head);
        }
        else
        {
                q=head;
                while ((q→next)→next!=head)
                {
                        q=q→next;
                }
                p=q→next;
                q→next=null;
                free(p);
                return (head);
        }
}
```

Display data in a circular linked list

```
void print(node * head)
{
        node *p;
        If(head!=null)
        {
                p=head;
                while(p→next!=head);
                {
                        printf("%d",p→data);
                        p=p→next;
                }
                printf("%d",p→data);
        }
}
```

What is circular queue? Write a program in c to implement it. [M-10 May-2015]

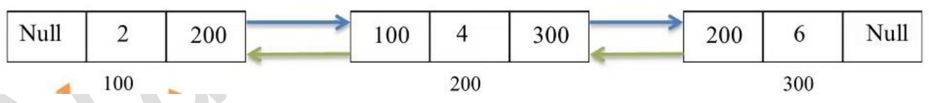
```
#include<stdio.h>
#include<conio.h>
typedef struct node
{
        Int data;
        Struct node*next;
}node;

node* create();
node* insert_b(node * head, intX);
node* insert_e(node * head, intx);
node* delete(node * head);
void print(node* head);
void main()
{
        int op, x;
        node * head=null;
        do
        {
                printf("1-create\n 2-insert in beginning \n 3-Insert at
                end\n 4-delete a specific\n 5-display\n 6-quit\n");
                printf("enter your choice\n");
                scanf("%d",&top);
                switch(op)
                {

                        case1:
                                head=create();
                                break;
                        case2:
                                printf("enter the data to be inserted\n);
                                scanf("%d",&x);
                                Head=insert_b(head, x);
                                break;
                        case3:
                                printf("enter data to be inserted\n");
                                scanf("%d",&x);
                                head=insert_e(head, x);
                                break;
                        case4:
                                printf("enter the data to be deleted\n");
                                scanf("%d",&x);
                                head=delete(head,x);
                                break;
                        case5:
                                print(head);
                                break;
                default:

        }
}while(op!=6);
getch();
}
void create ()
{
                node *head, *p;
                int n, x, i;
                printf(" Enter no of items");
                scanf("%d", &n);
                head=(node*)malloc(sizeof(node));
                //read the data in 1st node
                scanf("%d", &(head->data));
                head->next=head;
                p=head;
                for(i=1;i<n;i++)
                {
                        p->next=(node*)malloc(sizeof(node));
                        p=p->next;
                        p->next=head;
                        scanf("%d", &(p->data));
                }
}
node *insert_b(node *head, int x)
{
                node*p,*q;
        p=(node*)malloc(size of (node);
        p→data=x;
        if(head==NULL)
        {
                head= p;
                p→next=p;
        }
        else
        {
                q=head;
                while(q→next != head)
                {
                        q=q→next;
                }
                q→next=p;
                p→next=head;
                p=head;
                return(head);
        }
}
node *insert_e (node *head, int x)
{
                node*p,*q;
        p=(node*)malloc(size of (node);
        p→data=x;
        if(head==NULL)
        {
                head= p;
                p→next=p;
        }
        else
        {
                q=head;
                while(q→next != head)
                {
                        q=q→next;
                }
                q→next=p;
                p→next=head;
                return(head);
        }
}

void delete(node * head, int key)
{
        node *p,*q;
        if(head→data==key);
        {
                free(p);
                head=null
                return (head);
        }
        else
        {
                q=head;
                while (q→next→data !=key)
                {
                        q=q→next;
                }
                p=q→next;
                q→next=p→next;
                free(p);
                return (head);
        }
}
```

```
void print(node * head)
{
        node *p;
        If(head!=null)
        {
                p=head;
                while(p→next!=head);
                {
                        printf("%d",p→data);
                        p=p→next;
                }
                printf("%d",p→data);
        }
}
```

Doubly Linked List :

In Doubly Linked List , each node contain three fields .

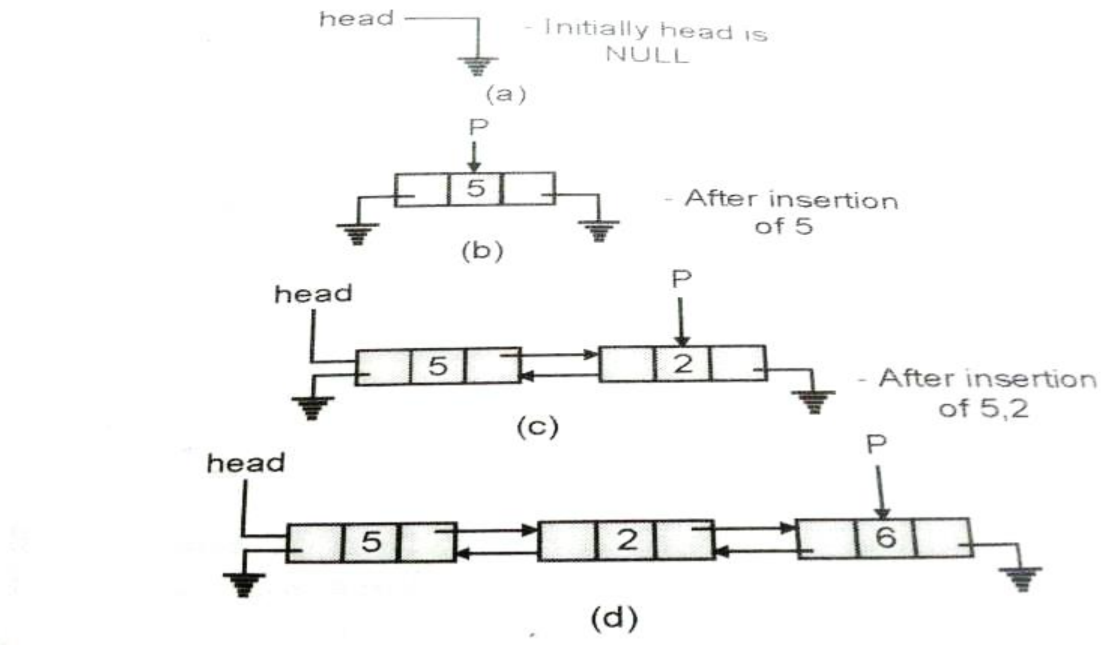
- One for storing the data.
- Another field for storing address of next node to be followed
- Third field contain address of previous node linked to it.
- In DLL we can access both next as well as previous data using "next link" and "previous link".
- Traversing in both direction is allowed.



Doubly linked list

```
typedef struct dnode
{
        Int data;
        struct dnode *next, *prev;
}dnode;
```

Creation of doubly linked list.



Algorithm for insertion of x.

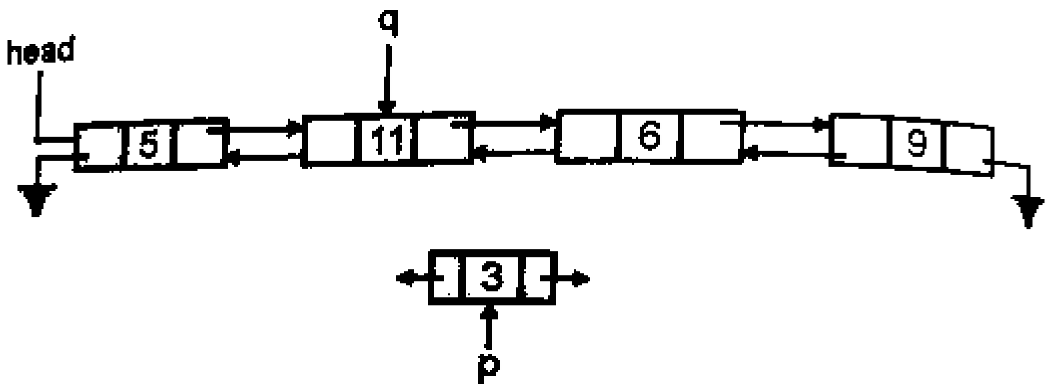
1. dnode *p, *q;
2. Acquire space for the new node.
3. p=(dnode*) malloc(sizeof(dnode));
4. Store x in the newly acquired node p→data=x
5. If head == null then go to step 6 else step 8
6. Change reference of head to new node [Head=p;]
7. Make previous and next field as NULL.
8. [head→prev=head→next=NULL]
9. Set new node next field to head, [p→next=head]
10. Set previous filed to new node p[head→prev=p]
11. Set previous field of new node to NULL, [p→prev=NULL;]
12. Change reference of head to new node [Head=p;]

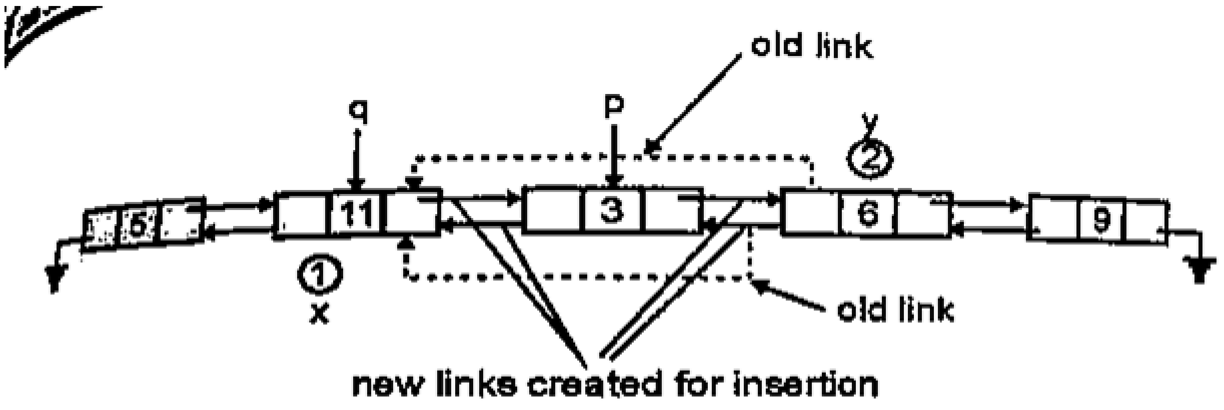
1. Inserting node at the begining

```
dnode * insert_b(dnode *head,int x)

{

    dnode *p;

    p=(dnode*)malloc(sizeof(dnode));

    p→data=x;

    If(head ==NULL)
```

```
        {
                head =p;
                head→next=head→prev=NULL;
        }
        else
        {
                p→next=head;
                head→prev=p;
                p→prev=NULL;
                head=p;
        }
        return (head);
    }
```

Insert In between of DLL





Algorithm for insertion of x after key node

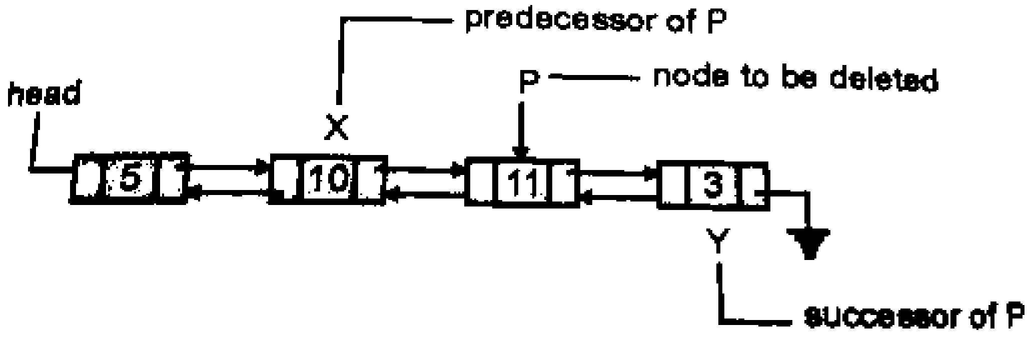
1. dnode *p, *q;
2. Acquire space for the new node.
3. p=(dnode*) malloc(sizeof(dnode));
4. Store x in the newly acquired node p→data=x
5. Assign pointer q to head node [q→head]
6. Traverse till q→data ! key and q!=null
7. Set next filed of new node as key node next [p→next=q→next;
8. Set previous field of new node as key node [p→prev=q;]
9. (p→next)→prev=p;
10. (p→prev)→next=p;

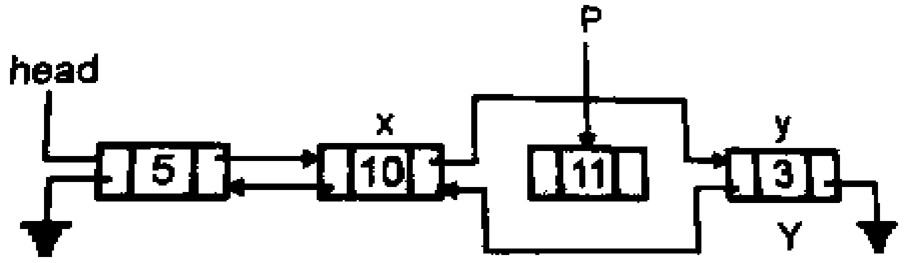
```
dnode * insert_in(dnode *head,int x,int key)
{
        dnode *p,*q;
        p=(dnode*)malloc(sizeof(dnode));
        p→data=x;
        q=head;
        while(q→data!=key && q!=NULL)
        {
                q=q→next;
        }
        If(q→data==key)
        {
                p→next=q→next;
                p→prev=q;
                (p→next)→prev=p;
                (p→prev)→next=p;
        }
        return(head);
}
```

Insert at end

```
dnode * insert_e(dnode *head, int x)
{
        dnode *p,*q;
        P=(dnode*)malloc(sizeof(dnode));
        P→data=x;
        If(head==NULL)
        {
                head=p;
                head→next=head→prev=NULL;
        }
        else
        {
                q=head;
                While(q→next!=NULL)
                {
                        q=q→next;
                }
                q→next=p;
                p→prev=q;
                p→next=NULL;
        }
        return(head);
}
```

Deletion of a node





Deletion of a node pointed by p

```
dnode * delete(dnode *head, dnode *p)
{
        dnode *p,*q;
        If(p==head)
        {
                p→next→prev=NULL;
                Head=p→next;
                Free(p);
                return(p);
        }
        P→prev→next=p→next;
        if(q→next!=NULL)
        {
                p→next→prev=p→prev;
        }
        free(p);
        return(head);
}
```

Write a c program in ‘C’ to implement doubly link-list with methods insert, delete and search. [M-10 May-14]

Write a c program in ‘C’ to implement doubly link-list, perform following operations.

1. Insert node in the beginning
2. Insert node at end
3. Delete a node from end
4. Display the list.

```
#include<stdio.h>
#include<conio.h>
typedef struct dnode
{
        Int data;
        struct dnode *next, *prev;
}dnode;

dnode* create();
dnode* insert_b(dnode * head, intX);
dnode* insert_e(dnode * head, intx);
dnode* delete(dnode * head);
void print(dnode* head);
void main()
{
        int op, x;
        dnode * head=null;
        do
        {
                printf(“1-create\n 2-insert in beginning \n 3-Insert at
                end\n 4-delete a specific\n 5-display\n 6-quit\n”);
                printf(“enter your choice\n”);
                scanf(“%d”,&top);
                switch(op)
                {

                        Case 1:
                                head=create();
                                break;
                        case 2:
                                printf("enter the data to be inserted\n);
                                scanf("%d",&x);
                                Head=insert_b(head, x);
                                break;
                        case 3:
                                printf("enter data to be inserted\n");
                                scanf("%d",&x);
                                head=insert_e(head, x);
                                break;
                        case 4:
                                printf("enter the data to be deleted\n");
                                scanf("%d",&x);
                                head=delete(head,x);
                                break;
                        case 5:
                                print(head);
                                break;
                default:

        }
}while(op!=6);
getch();
}
void create ()
{
                dnode *head, *p;
                int n, x, i;
                printf(" Enter no of items");
                scanf("%d", &n);
                head=(dnode*)malloc(sizeof(dnode));
                //read the data in 1st node
                scanf("%d", &(head->data));
                head→next=head→prev=NULL;
                p=head;
                for(i=1;i<n;i++)
                {
                        q=p;
```

```
                    p→next=(dnode*)malloc(sizeof(dnode));
                    p=p→next;
                    p→next=NULL;
                    p→prev=q;
                    scanf("%d", &(p->data));
            }
}
dnode * insert_b(dnode *head,int x)
{
        dnode *p;
        p=(dnode*)malloc(sizeof(dnode));
        p→data=x;
        If(head ==NULL)
        {
                head =p;
                head→next=head→prev=NULL;
        }
        else
        {
                p→next=head;
                head→prev=p;
                p→prev=NULL;
                head=p;
        }
        return (head);
    }
```

```
dnode * insert_e(dnode *head, int x)
{
        dnode *p,*q;
        P=(dnode*)malloc(sizeof(dnode));
        P→data=x;
        If(head==NULL)
        {
                head=p;
                head→next=head→prev=NULL;
        }
        else
        {
                q=head;
                While(q→next!=NULL)
                {
                        q=q→next;
                }
                q→next=p;
                p→prev=q;
                p→next=NULL;
        }
        return(head);
}
dnode * delete(dnode *head, dnode *p)
{
        dnode *p,*q;
        If(p==head)
        {
                p→next→prev=NULL;
                Head=p→next;
                Free(p);
                return(p);
        }
        P→prev→next=p→next;
        if(q→next!=NULL)
        {
                p→next→prev=p→prev;
        }
        free(p);
        return(head);
}
void print(node * head)
{
        node *p;
        If(head!=null)
        {
                p=head;
                while(p→next!=head);
                {
                        printf("%d",p→data);
                        p=p→next;
                }
                printf("%d",p→data);
        }
}
```

LINKED REPRESENTATION OF STACK

Stack can be represented through singly connected linked list.

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

Initilize a stack

```
void init (stack **T)

{

        *T= Null;
}
```

Insert data in stack

```
void push(stack **T, intX)
{
        stack *P;
        p=(stack*) malloc (size of (stack));
        P→data= X;
        P→next= *T;
        *T= P;
}
```

Delete the top element of a stack

```
Int top (stack **T)
{
        Int x;
        Stack *p;
        P= *T;
        *T= P→next;
        x= P→data;
        free(p);
        return (X);
}
int empty(stack *top)
{
                if (Top= = Null)
                {
                        return (1);
                }
                else
                {
                        return (0);
                }
}
```

```
dnode * delete(dnode *head, dnode *p)
{
        dnode *p;
        for ( i=p
```

Write a program to reverse a string represented using stack

```
# include<stdio.h>
#include<conio.h>
typedef struct stack
{
                char data;
                struct stack *next;
}

void init ( stack **);

int empty (stack *);

char pop (stack  **);

void push (stack **, char);

void main()

{
        Stack *top;
        char x;
        init(&top);
        printf("enter the spring\n")
        While ((x=getchar())!='\n')
```

```
{
        Push(&top, x);
}
while(!empty (top))
{
        x =pop(&top);
        printf("%c", x);
}
}
void init(stack**T)
{
        *T=NULL;
}
Int empty (stack *top)
{
        If(top= = NULL)
                return(1);
        return(0);
}
void push(stack**T, char x)
{
        stack *p;
        p=(stack*)malloc(sizeof(stack));
        P→data= x;
        P→next= *T;
        *T=P;
}
char pop (stack**T)
{
        char x;
        Stack *P;
        P=*T;
        *T= P→next;
        X= P→data;
        free(p);
```

```
                return(x);
        }
```

<u>Representation of queue using linked list</u>

A Queue can be represented by a linked structure of element.
Each element is stored in node & memory acquired during runtime using malloc() fuction.

Write a program, for various operations on queue using circular linked list.

```
#include<stdio.h>
#include<conio.h>
typedef struct node
{
        int data;
        struct node*next;
}node;
Void int(node **R);
Void enqueuer(node **R,int x);
Int dequeue(node **R);
Int empty(node **R);
Void print(node **R);
Void main()
{
        Int x,ch;
        Int n=0,i;
        node *rear;
        init(&rear);
        do
        {
                Printf("1.Insert\n2.deletion\3.print\n4.Quit");
                Printf("Enter your option\n');
                Scanf("%d",&ch);
                Switch(ch)
                {
                        Case 1:
```

```
                    Printf ("Nuumber of elements to be
                    inserted");
                    Scanf("%d",&n);
                    For (!=0;1<n;!++)
                    {
                        Scanf("%d",&X);
                        Enqueue (&rear,X);
                    }
                    Break;
                Case 2:
                    If(!empty(&rear))
                    {
                        X= dequeuer(&rear);
                        Printf("\element delete=%d",x);
                    }
                    Else
                    {
                        Printf("underflow...");
                    }
                    Break;
                Case 3:
                    Print(&rear);
                    Break;
                Default:

            }
            }while(ch!=4);
    getch();
}
Void enqueuer(node**R, intX)
{
    node*P;
    P→data=X;
    If(empty(*R))
    {
        P→next=p;
        *R=P;
    }
    else
    {
```

```
                P→next=(*R)→next;
                (*R)→next)=P;
                *R=P;
        }
}
Int dequeue (node**R)
{
        Int X;
        node*P;
        P=(*R)→next;
        P→data=x;
        If(P→next==P)
        {
          *R=NULL;
          free(p);
          return(X);
        }
        (*R)→next= P→next;
        Free(P);
        Return(X);
}
Void print(node*rear)
{
        node*P;
        If (!empty(rear))
        {
                P=rear→next;
                While(P!=rear)
                {
                        Printf("%d",P→data);
                        P=P→next;
                }
            Printf("%d",P→data);
          }
}
Int empty(node*P)
{
        If (P→nexrt=== -1)
        {
          return(1);
        }
          return(0);
}
```

Important Questions

- Write a c program to implement singly list which supports the following operation.
    - Insert node in the beginning.
    - Insert a node in the end
    - Insert a node after a specific node.
    - Deleting a specific node.
    - Displaying the list.
- State difference between singly and doubly linked list along with application.
- What is circular queue? Write a program in c to implement it. [M-10 May-2015]
- Write a c program in 'C' to implement doubly link-list with methods insert, delete and search. [M-10 May-14]
- What is linked list and explain its types [3M May-18]
- Insert the following elements in AVL tree 44,17,32,78,50,88,48,62,54 explain different types of rotations that can be used. [10M May-18]
- Explain different types of tree traversal techniques with example, also write recursive function for each traversal technique. [10M May-18]
- Write a function in c to implement binary search [5M Dec-18]
- What are expression trees ? what are its advantages ? Derive the expression tree for following algebraic expression ?
  (a + (b/c))*((d/e)-f) [8M DEC-18]

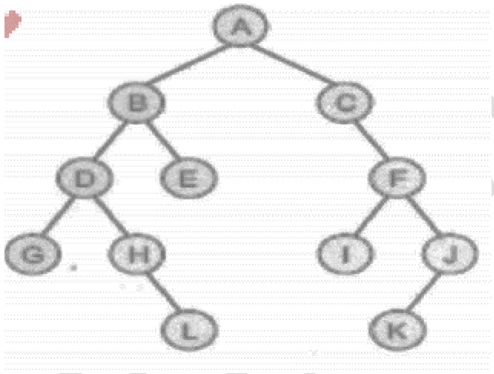
- Give postorder and inorder traversal of a binary tree , construct the tree [10M DEC-18]

Postorder	D	E	F	B	G	L	J	K	H	C	A
Inorder	D	B	F	E	A	G	C	L	J	H	K

- Write a short not on AVL tree and expression tree [10M DEC-17]
- Explain AVL tree, Insert the following elements in AVL search tree 63, 52,49,83, 92, 29, 23, 54, 13, 99 [10M DEC-17]
- Explain Threaded Binary Tree [5M DEC-17]

- Describe expression Tree with example
- Write a program in c to delete a node from a Binary search tree. The program should consider the all the possible cases.

- IT
  What is linked list? State its advantage, Explain its types [3M DEC-18, May-18]
- Define minimum spanning tree. List techniques to compute minimum spanning tree. [3M May-18]
- Define expression tree with example. [3M DEC-18, May-18]
- Write an algorithm to create doubly linked list and display it [10M May-18]
- Define binary search tree ? Explain different operation on binary search tree with example. [10M May-18]
- Define binary search tree ? find inorder, preorder and postorder of following binary tree. [10M DEC-18]



-
- Define binary search tree ? write an algorithm for following operation
  - Insertion
  - Deletion

- What is minimum spanning tree? Draw MST using kruskal's and prims algorithm. [10M May-18]
- What is minimum spanning tree? Explain kruskal's algorithm with an example. [10M May-18]
- Explain binary search tree. Construct binary search tree for following elements 47,12,75, 88, 90, 73, 57, 1, 85, 50, 62 [10M DEC-18]