Linked List

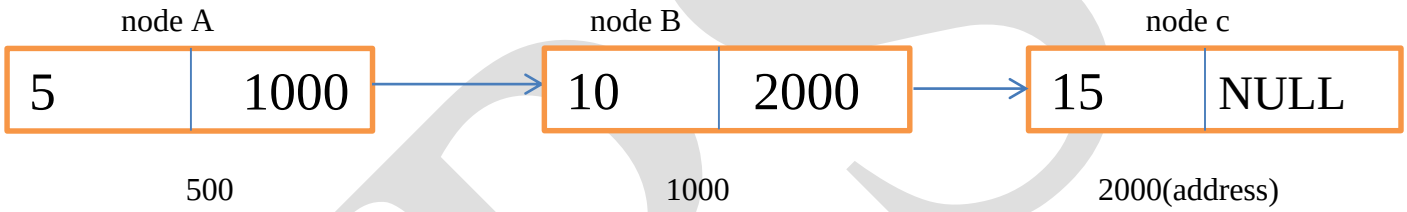
A linked list is a series of connected nodes.

Each node is consisting of a data field and address field (pointer or reference) to some other node. The last node contains NULL link .The list may or may not contain Header. In linked list insertion and deletion operation is easy to implement.

Representation of node:

Data	Address

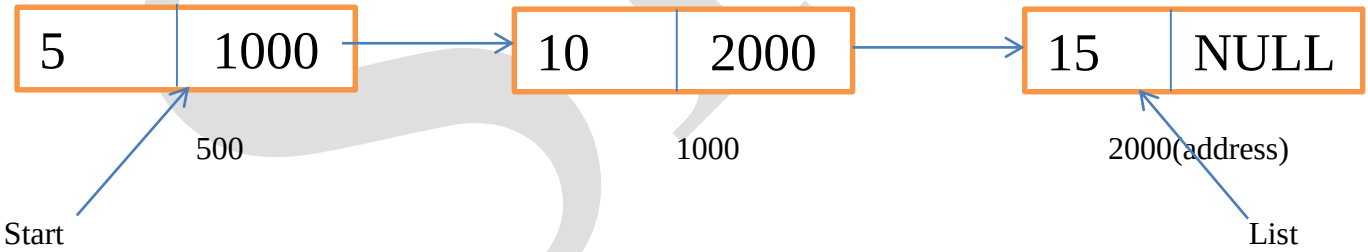
Linked list:



Linked list have been divided into three types.

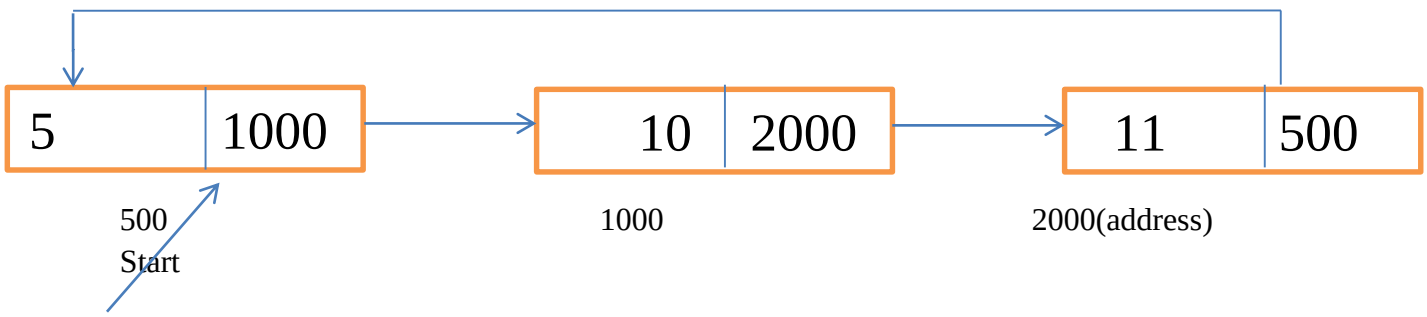
1. **Singly linked list**:

Two successive nodes of the linked list are linked with each other in sequential linear manner.



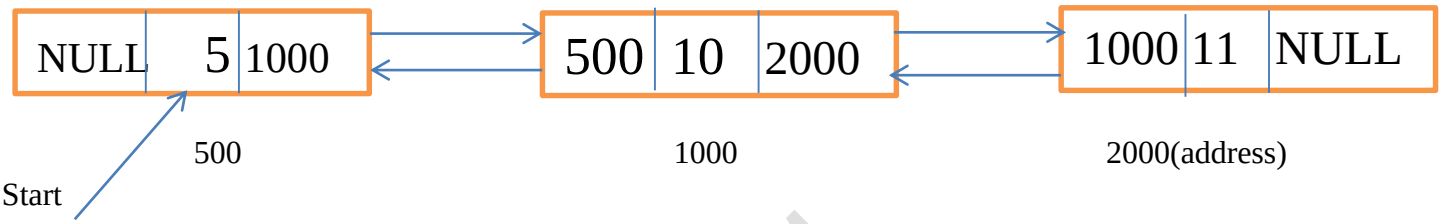
2. **Circular linked list**:

In a circular list the first and the last elements are adjacent. A linked lit can be made circular by storing the address of the first node in the next field of the last node.



3. Doubly linked list:

In this type each node holds two pointer fields .In doubly linked list address of next as well as preceding element are linked with current node.



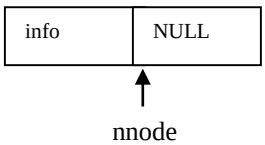
Difference between Array and Linked list

Array	Linked List
1. An array is represented in memory using sequential mapping i.e; element has fixed distance apart.	1. In linked representation, it is not necessary that the element be at fixed distance apart.
2. In array insertion and deletion operation is complicated to implement.	2. In link list insertion and deletion operation is easy to implement.
3. Memory should be allocated at the time when programmer is writing a program.	3. Memory should be allocated at run time. i.e; after executing program.
4. Array has homogeneous value i.e; each element is independent of other position.	4. Each element in link list is connected with previous node which is pointer to the node.
5. Array is static data structure.	5. Link list is dynamic type data structure.
6. Memory storage space required is more.	6. Less space is required for memory storage
7. The size of array is fixed	7. Size of link list is not fixed.

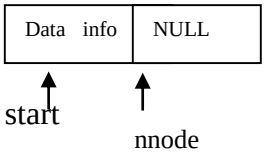
Singly Linked List

1. Create new Linked list

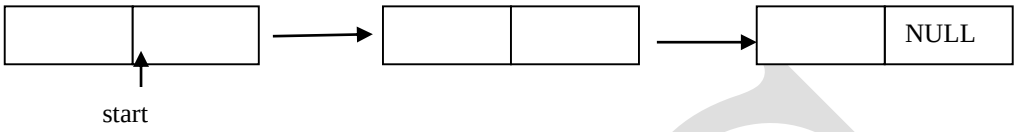
Step 1: start = NULL
Step 2: Create new node (nnode)
Step 3: nnode->next= NULL
Step4: Assign value (nnode->info)



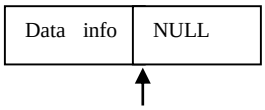
Step 5: start = nnode



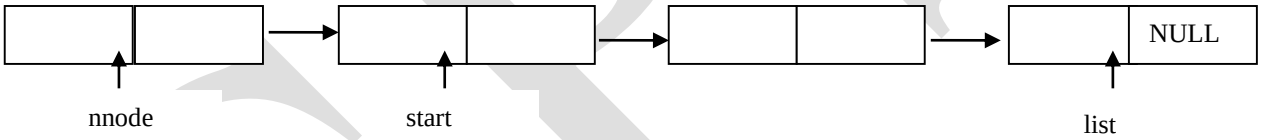
2. Insert at the beginning



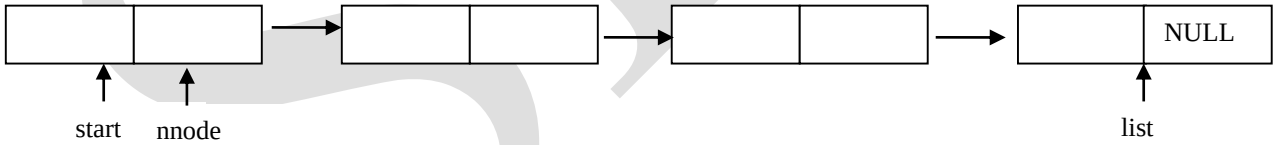
Step1: Create new node (nnode)
Step 2: nnode->next= NULL
Step3: Assign value (nnode->info)



Step 4: nnode ->next =start

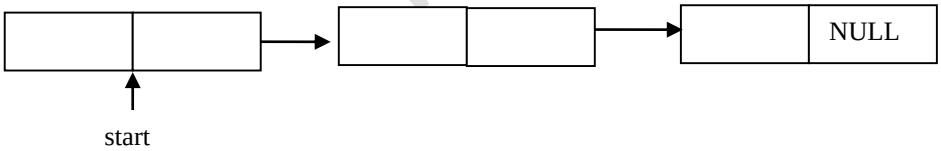


Step 5 :start= nnode



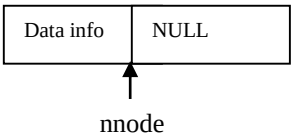
---

3. Insert at the end :



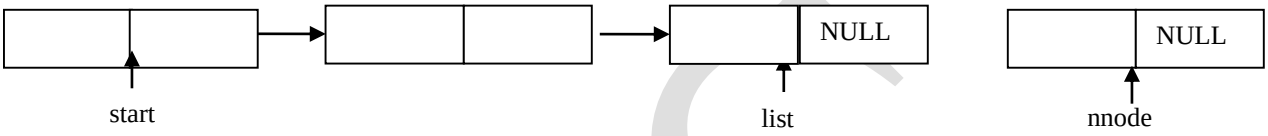
Step 1: create new node
Step 2: nnode ->next =NULL

Step 3: Assign value (nnode->info)

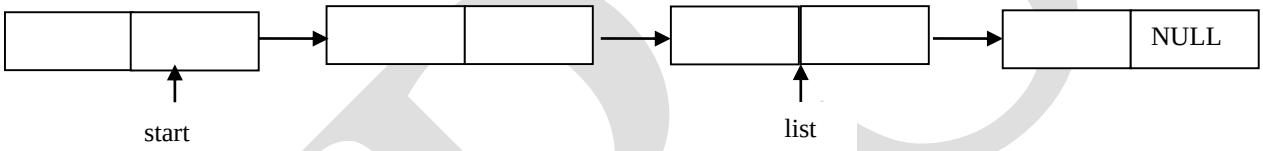


Step 4: travel up to the last node

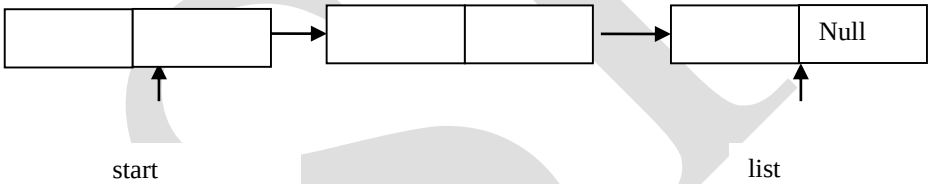
```
list = start
  while(list ->next! =NULL)
 {
       List=list->next
 }
```



Step 5:list->next=nnode



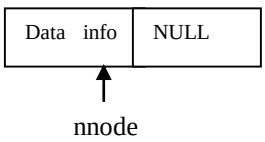
4. Insert at the specific position:



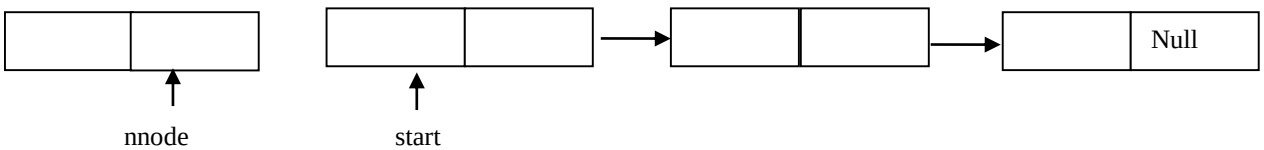
Step1: Create new node(nnode)

Step 2: nnode->next= NULL

Step 3: Assign value (nnode->info)

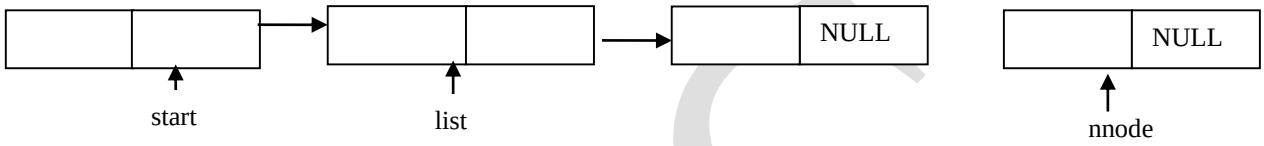


Step 4:Take a position from user at p.

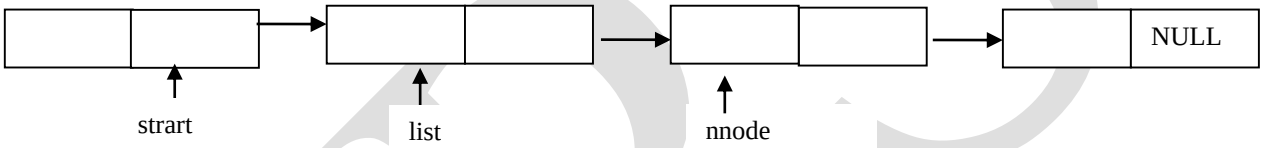


Step 5: Travel up to p-1 position

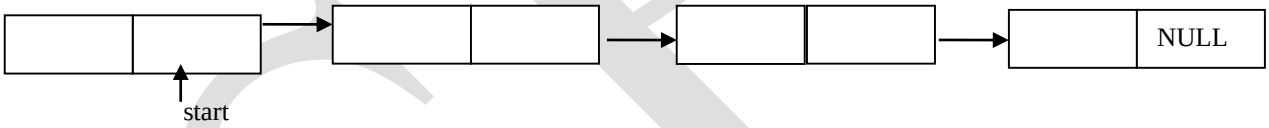
```
list=start
c=1
while(c<(p-1))
{
    list=list->next
    c++
}
```



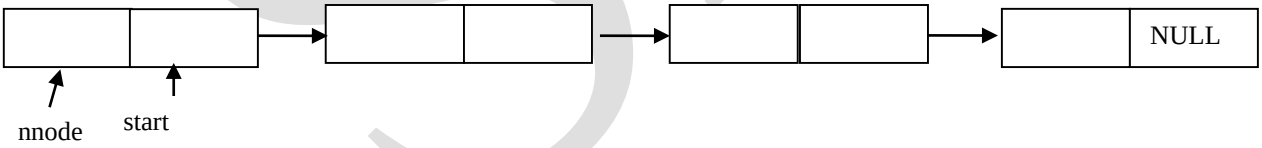
Step 6: nnode->next=list->next
list->next=nnode



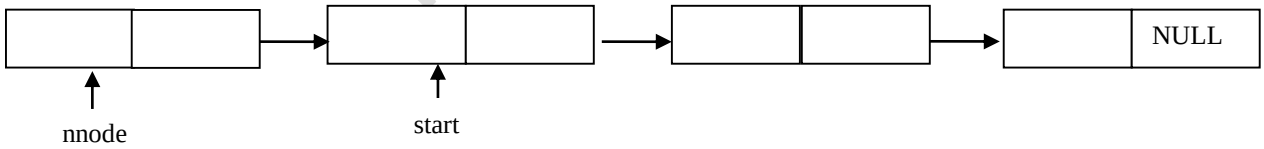
5. Delete from the beginning



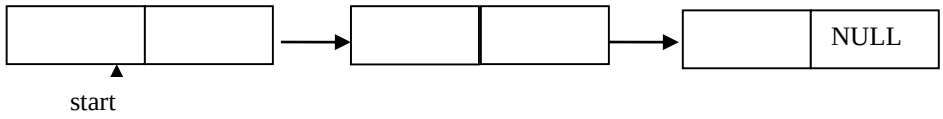
Step 1: nnode=start



Step 2: start=nnode->next

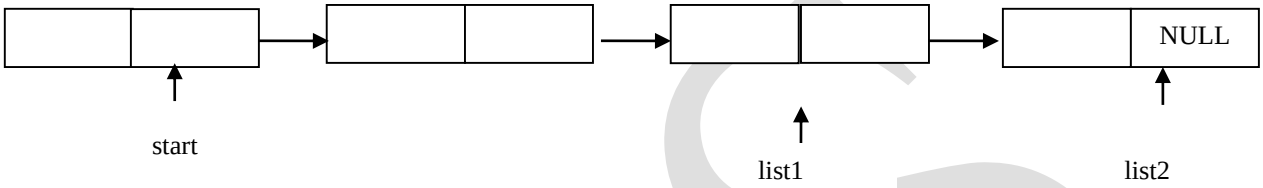


Step 3: free nnode

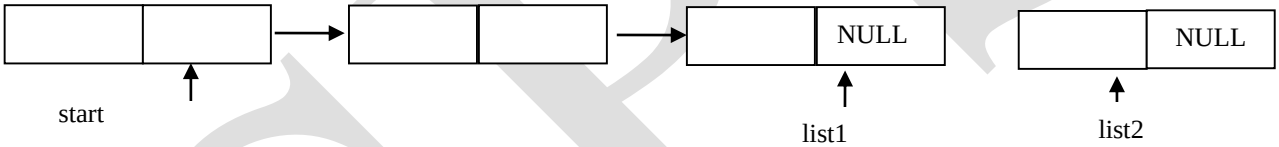


6. Delete from the end:

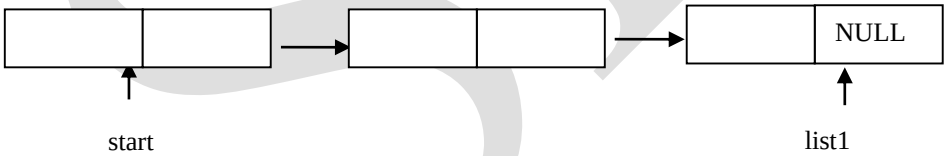
Step 1: list1=start
Step 2: list2=list1->next
Step 4: while(list2->next!=NULL)
{
list1=list2
list2=list2->next
}



Step 5:list1->next=NULL

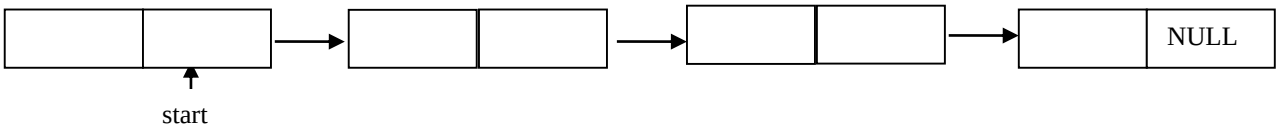


Step 5: free(list2)

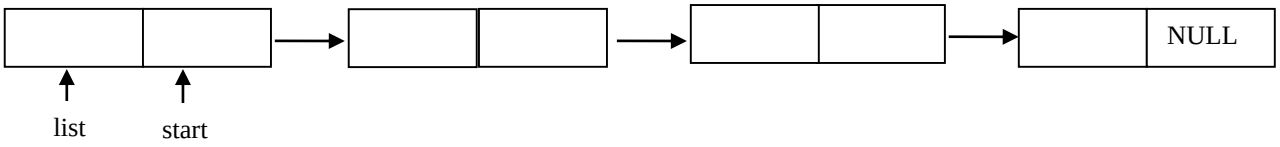


---

7. **Delete from specific position**:

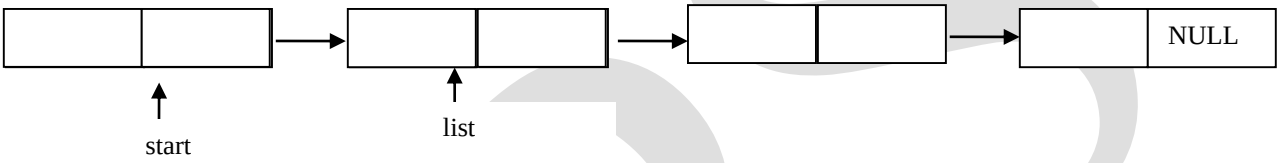


Step 1: list=start

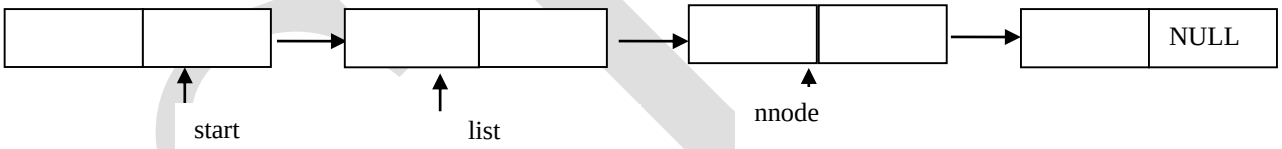


Step 2: take position to delete node (p)
Step 3: travel up to (position-1)node

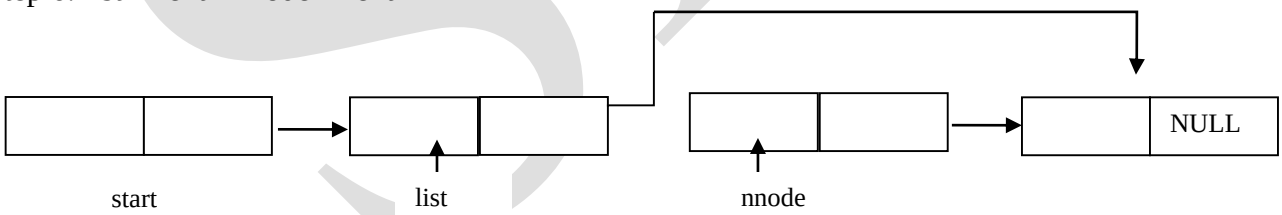
```
While(c<(p-1))
{
        List=list->next;
    c++;

 }
```



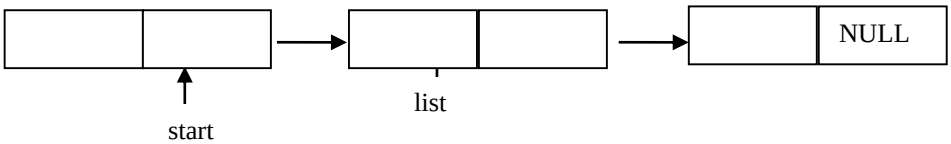
Step 4: nnode =list->next



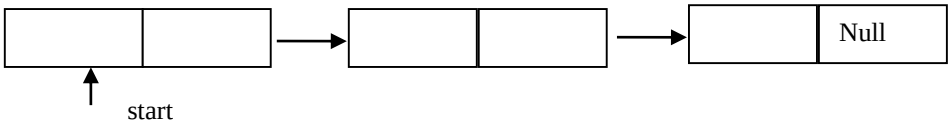
Step 6: list->next= nnode->next



Step 7:free(nnode)



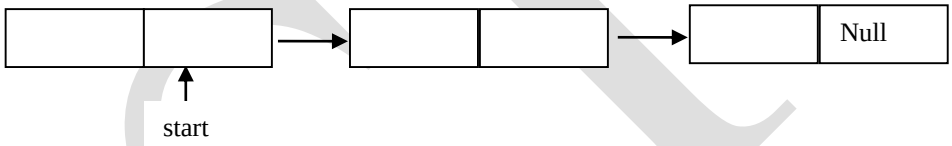
8. Search node:



Step1:Enter value to search node(sno)
Step 2:list=start
Step 3:while(list!=NULL)
{
if(list->info=sno)
{
Printf("Record is found");
Break;
}
list=list->next
}
Step 4:if(list==NULL)
Record is not found

---

9. Modify:



Step 1: take value to modify node(mno)
Step 2: list=start
Step 3: while(list!=NULL)
{
if(list->info==mno)
{
Printf("record is found");
Enter new info
Update list->info;
break;
}
list=list->next
}
Step 4: if(list==NULL)
Record is not found

C Program

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

struct node
{
        int info;
        struct node *next;
};
struct node *start,*list,*list1,*nnode;

void create();
void append();
void display();
void insertP();
void insertN();
void deletP();
void deletN();
void search();
void modify();

void main()
{
        int ch=0;
        while(ch!=10)
        {
                printf("\n****MAIN MENU****");
                printf("\n1.CREATE");
                printf("\n2.DISPLAY");
                printf("\n3.APPEND NODE");
                printf("\n4.Insert node at Position");
                printf("\n5.Insert after Node");
                printf("\n6. Delete node at position");
                printf("\n7. Delete Node");
                printf("\n8.search");
                printf("\n9.Modify");
                printf("\n10.Exit");

                printf("\nEnter Your Choice: ");
                scanf("%d",&ch);
                switch(ch)
```

```
		{
			case 1: create();
				break;
			case 2: display();
				break;
			case 3: append();
				break;
			case 4: insertP();
				break;
			case 5: insertN();
				break;
			case 6: deletP();
				break;
			case 7: deletN();
				break;
			case 8: search();
				break;
			case 9: modify();
				break;
			case 10: break;
			default:printf("\nEnter Proper Choice..");
		}
	}
	printf("\nprogram exit here..");
}

void create()
{
	char ch='y';
	start=NULL;
	list=NULL;

	while(ch=='y')
	{
		nnode=(struct node*)malloc(sizeof(struct node));
		nnode->next=NULL;
		printf("\nEnter value: ");
		scanf("%d",&nnode->info);
		if(start==NULL)
		{
			start=nnode;
			list=nnode;
```

```
                }
                else
                {
                        list->next=nnode;
                        list=list->next;
                }
                printf("\nDo u want to continue(if yes press y): ");
                fflush(stdin);
                ch=getchar();
        }
}

void display()
{
        if(start==NULL)
        {
                printf("\nList is empty..");
        }
        else
        {
                list=start;
                while(list!=NULL)
                {
                        printf("\nValue :%d",list->info);
                        list=list->next;
                }
        }
}


void append()
{
        nnode=(struct node*)malloc(sizeof(struct node));
        nnode->next=NULL;
        printf("\nEnter value: ");
        scanf("%d",&nnode->info);
        list=start;
        while(list->next!=NULL)
        {
                list=list->next;
        }
        list->next=nnode;
```

```
}

void insertP()
{
        int c,p;
        nnode=(struct node*)malloc(sizeof(struct node));
        nnode->next=NULL;
        printf("\nEnter value: ");
        scanf("%d",&nnode->info);
        printf("\nEnter position where you want to insert node: ");
        scanf("%d",&p);
        c=1;
        list=start;
        if(p==1)
        {
          nnode->next=start;
          start=nnode;
        }
        else
        {
          while(list->next!=NULL)
          {
                if(c<(p-1))
                {
                        list=list->next;
                        c++;
                }
                else
                {
                        break;
                }
          }
          nnode->next=list->next;
          list->next=nnode;
        }
        printf("\n Node inserted...");
}

void insertN()
{
        int n;
        nnode=(struct node*)malloc(sizeof(struct node));
```

```
        nnode->next=NULL;
        printf("\nEnter value: ");
        scanf("%d",&nnode->info);
        printf("\nEnter value of node after that you want to insert new node: ");
        scanf("%d",&n);
        list=start;
        while(list->next!=NULL)
          {
              if(list->info!=n)
                 list=list->next;
              else
                 break;
          }
          nnode->next=list->next;
          list->next=nnode;
          printf("\n Node inserted...");
}

void deletP()
{
 int c,p;
 printf("\nEnter position to delete node:");
 scanf("%d",&p);
 c=1;
 list=start;
 if(p==1)
 {
    nnode=start;
    start=nnode->next;
 }
 else
 {
    while(c<(p-1))
    {
        list=list->next;
        c++;
    }
    nnode=list->next;
    list->next=nnode->next;
 }
 free(nnode);
}
```

```
void deletN()
{
 int n;
 printf("\nEnter node that you want to delete:");
 scanf("%d",&n);
 list=start;
 list1=list->next;
 if(start->info==n)
 {
    nnode=start;
    start=nnode->next;
    printf("Node %d is deleted",n);
 }
 else
 {
    while(list1!=NULL)
    {
          if(list1->info!=n)
          {
            list=list1;
            list1=list1->next;
          }
          else
          {
           nnode=list->next;
           list->next=nnode->next;
           printf("Node %d is deleted",n);
           break;
          }
    }
    if(list1==NULL)
          printf("Node %d is not in the list",n);
 }
 free(nnode);
}

void search()
{
  int sno;
  printf("\nEnter value to search record:");
  scanf("%d",&sno);
  list=start;
```

```
  while(list!=NULL)
  {
   if(list->info==sno)
   {
          printf("\nRecord found");
          printf("\nValue is %d",list->info);
          break;
   }
   list=list->next;
   if(list==NULL)
   {
          printf("\nRecord not found");
   }
  }
}

void modify()
{
  int mno;
  printf("\nEnter value to modify record:");
  scanf("%d",&mno);
  list=start;
  while(list!=NULL)
  {
   if(list->info==mno)
   {
          printf("\nRecord found");
          printf("\n Info is %d \n",list->info);
          printf("\nEnter new info:");
          scanf("%d",&list->info);
          printf("\nOne record is modified");
          break;
   }
   list=list->next;
   if(list==NULL)
   {
          printf("\nRecord not found");
   }
  }
}
```

Linked implementation of Stack

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

struct node
{
  int info;
  struct node *next;
};
struct node *tos, *list, *nnode;

void push();
void pop();
void display();
void main()
{
   int ch;
   clrscr();
   tos=NULL;
   printf("Stack operation");
   while(ch!=4)
   {
      printf("\n1. PUSH \t 2. POP \t 3. DISPLAY 4. EXIT  ");
      scanf("%d",&ch);
      switch(ch)
      {
           case 1: push();
                   break;
           case 2: pop();
                   break;
           case 3: display();
                   break;
           case 4: exit(0);
           default: printf("Enter Correct choice");
      }
   }
   getch();
}

void push()
{
 nnode= (struct node *)malloc(sizeof (struct node));
```

```
 nnode->next=NULL;
 printf("Enter value");
 scanf("%d",&nnode->info);
 if(tos==NULL)
 {
  tos=nnode;
 }
 else
 {
  nnode->next=tos;
  tos=nnode;
 }
 printf("Node Inserted");
}
void pop()
{
 if(tos==NULL)
   printf("Stack underflow");
 else
 {
   nnode=tos;
   tos=tos->next;
   printf("\n Element %d is pop",nnode->info);
   free(nnode);
 }
}

void display()
{
 if(tos==NULL)
   printf("Stack is empty");
 else
 {
   list=tos;
   while(list!=NULL)
   {
     printf("Element =%d",list->info);
     list=list->next;
   }
 }
}
```

Linked implementation of Queue

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int info;
 struct node *next;
};
struct node *front,*rear,*nnode,*list;
void insert();
void delet();
void display();

void main()
{
        int ch=0;
    front=NULL;
        rear=NULL;
        while(ch!=4)
        {
                printf("\n****MAIN MENU****");
                printf("\n1.INSERT");
                printf("\n2.DELETE");
                printf("\n3.DISPLAY");
                printf("\n4.EXIT");
                printf("\nEnter Your Choice: ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: insert();
                                break;
                        case 2: delet();
                                break;
                        case 3: display();
                                break;
                        case 4: exit(0);
                        default:printf("\nEnter Proper Choice..");
                }
        }
        printf("\nprogram exits here..");
}
```

```
void insert()
{
        nnode=(struct node*)malloc(sizeof(struct node));
        nnode->next=NULL;
        printf("\nEnter values:");
        scanf("%d",&nnode->info);
        if(front==NULL)
        {
                front=nnode;
                rear=nnode;
        }
        else
        {
                rear->next=nnode;
                rear=rear->next;
        }

}

void delet()
{
        if(front!=NULL)
        {
                nnode=front;
                if(front==rear)
                {
                 front=NULL;
                 rear=NULL;
                }
                else
                {
                 front=front->next;
                }
                printf("Element %d is deleted",nnode->info);
                free(nnode);
        }
        else
        {
                printf("\nQUEUE UNDERFLOW...");
        }
}
```

```
void display()
{
        if(front==NULL)
        {
                printf("\nQUEUE IS EMPTY...");
        }
        else
        {
                list=front;
                printf("\nQueue Elements Are: ");
          while(list!=NULL)
                {
                        printf(" %d",list->info);
                        list=list->next;

                }
        }
}
```

Linked implementation of Priority Queue

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int info;
 struct node *next;
};

struct node *front,*rear,*nnode,*list,*list1;
void insert();
void delet();
void display();

void main()
{
        int ch=0;
     front=NULL;
        rear=NULL;
        while(ch!=4)
        {
                printf("\n****MAIN MENU****");
```

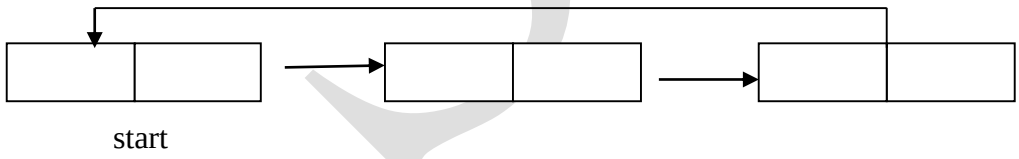
```
                printf("\n1.INSERT");
                printf("\n2.DELETE");
                printf("\n3.DISPLAY");
                printf("\n4.EXIT");
                printf("\nEnter Your Choice: ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: insert();
                                break;
                        case 2: delet();
                                break;
                        case 3: display();
                                break;
                        case 4: exit(0);
                        default:printf("\nEnter Proper Choice..");
                }
        }
        printf("\nprogram exits here..");
}

void insert()
{
        nnode=(struct node*)malloc(sizeof(struct node));
        nnode->next=NULL;
        printf("\nEnter values:");
        scanf("%d",&nnode->info);
        if(front==NULL)
        {
                front=nnode;
                rear=nnode;
        }
        else
        {
             if(nnode->info<front->info)
          {
                nnode->next=front;
                front=nnode;
          }
          else if(nnode->info>rear->info)
          {
                rear->next=nnode;
```

```
            rear=nnode;
        }
        else
        {
            list1 = front;
            list=list1->next;
            while(list1!=rear)
            {
                if(list->info<nnode->info)
                {
                    printf("\nlist=%d\tnnode=%d",list->info,nnode->info);
                    list1=list1->next;
                    list=list->next;
                }
                else
                {
                    nnode->next=list;
                    list1->next=nnode;
                    break;
                }
            }
        }
    }
}

void delet()
{
    if(front!=NULL)
    {
        nnode=front;
        if(front==rear)
        {
          front=NULL;
          rear=NULL;
        }
        else
        {
          front=front->next;
        }
        printf("Element %d is deleted",nnode->info);
        free(nnode);
    }
```

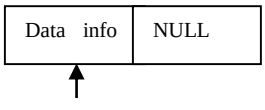
```
        else
        {
                printf("\nQUEUE UNDERFLOW...");
        }
}

void display()
{
        if(front==NULL)
        {
                printf("\nQUEUE IS EMPTY...");
        }
        else
        {
                list=front;
                printf("\nQueue Elements Are: ");
          while(list!=NULL)
                {
                        printf(" %d",list->info);
                        list=list->next;

                }
        }
}
```

Circular Linked List

1) Insert at the beginning

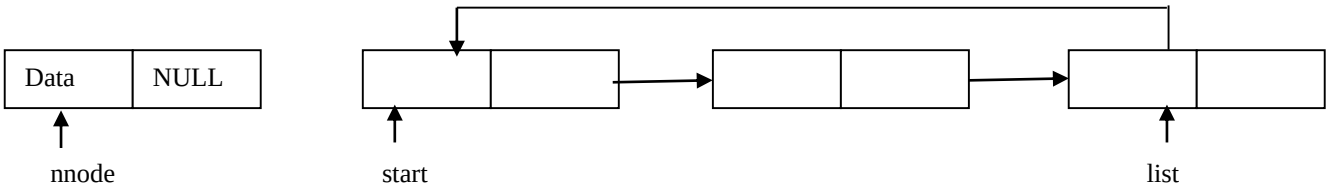


Step1: Create new node (nnode)
Step 2: nnode->next= NULL
Step 3: Assign value (nnode->info)

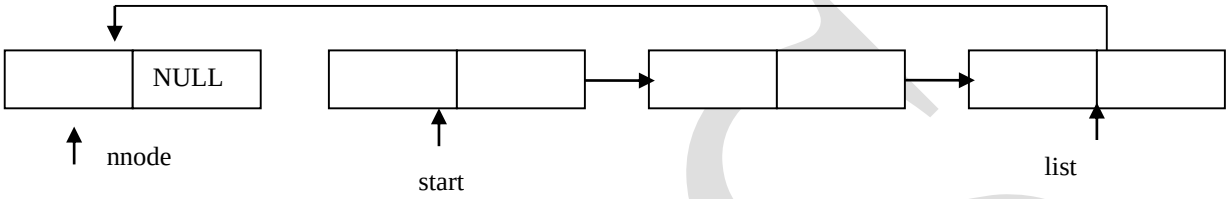


Step 4: travel up to the last node
list = start

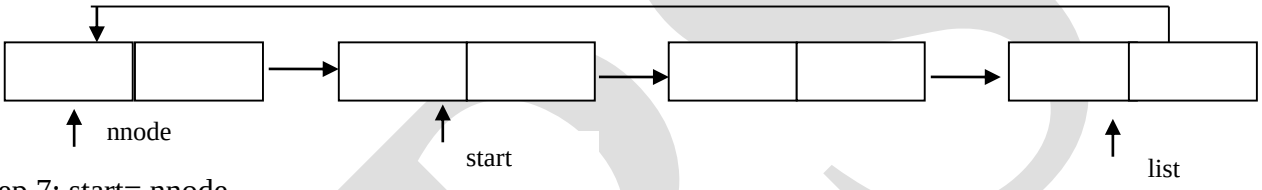
While(list->next!=start)
list =list->next



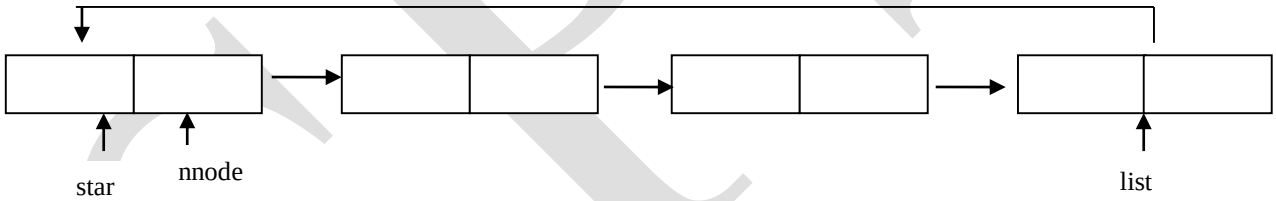
Step 5:list- >next =nnode



Step 6: nnode ->next =start

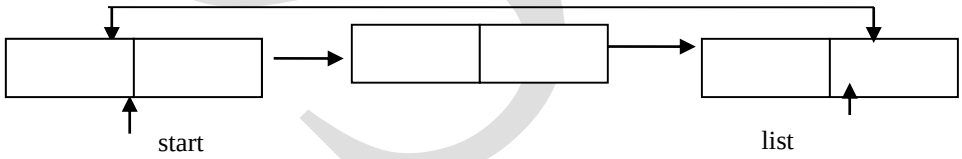


Step 7: start= nnode



---

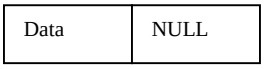
2) Insert at the end :



step1: create new node

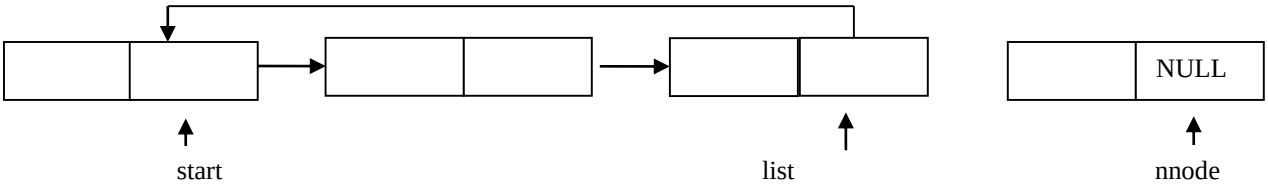
step 2:nnode ->next =NULL

step 3:Assign value (nnode->info)

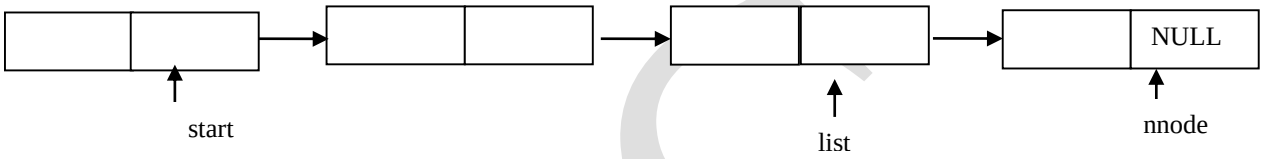


step 4: travel up to the last node
list = start

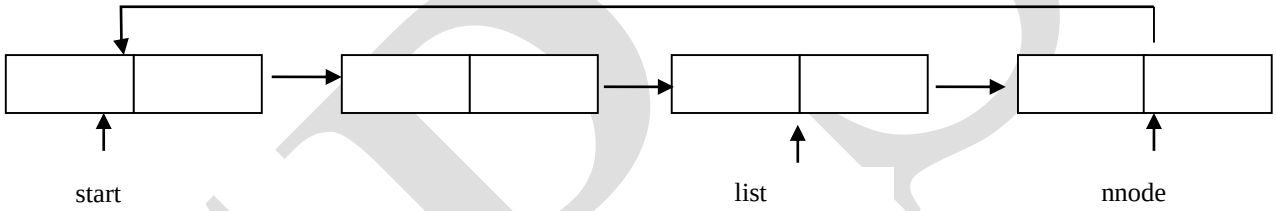
```
while(list ->next! =start)
{
        List=list->next
}
```



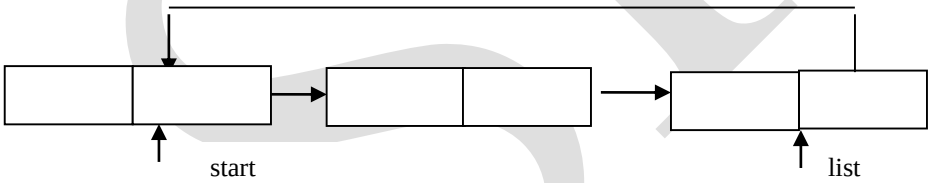
Step 5: list->next=nnode



Step 6: nnode->next=start



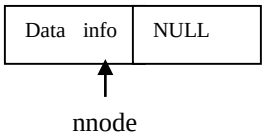
3) Insert at the specific position:



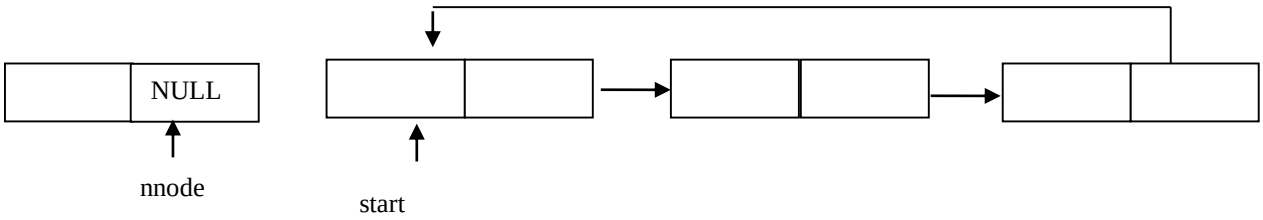
Step1: Create new node (nnode)

Step 2: nnode->next= NULL

Step 3: Assign value (nnode->info)

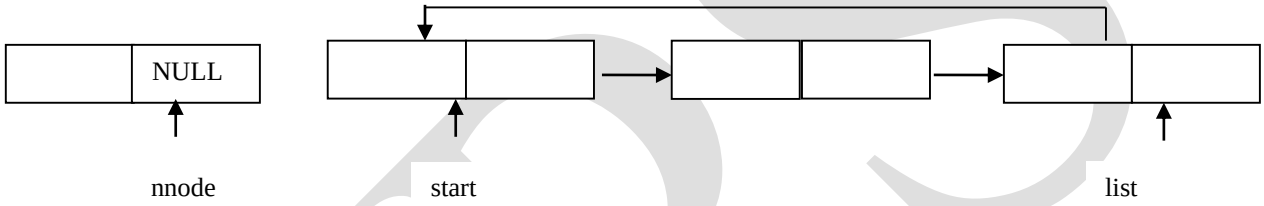


Step 4:Take a position from user at p.

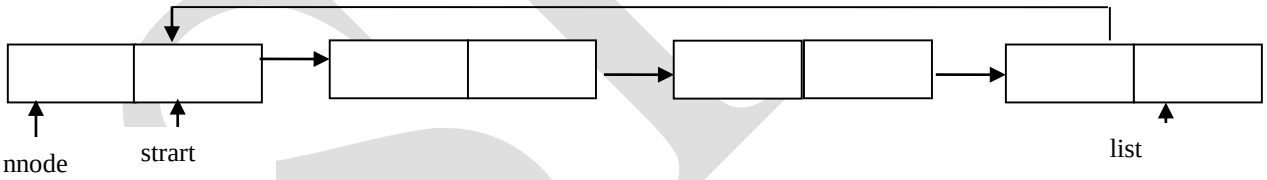


Step 5: Traval up to p-1 position

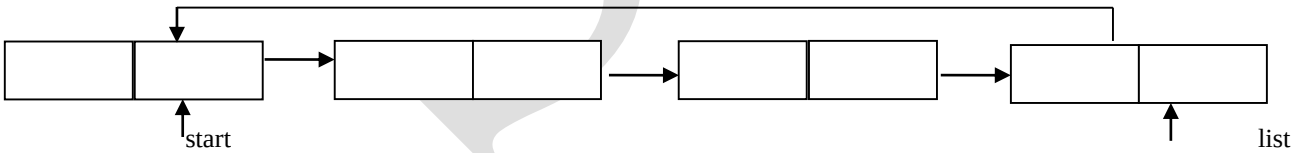
```
list=start
c=1
while(c<(p-1))
{
    list=list->next
    c++
}
```



Step 6:nnode->next=list->next
Step 7: list->next=nnode
Step 8: start=nnode

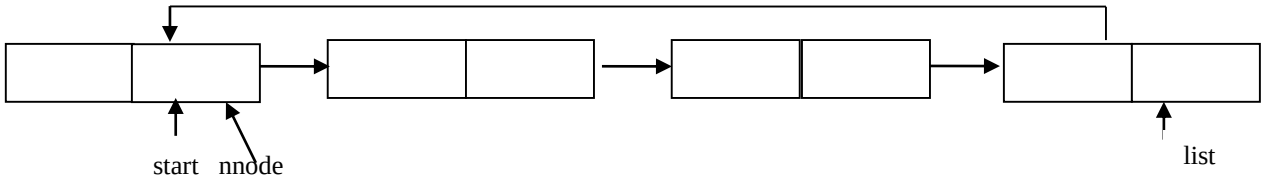


4) Delete from the beginning:

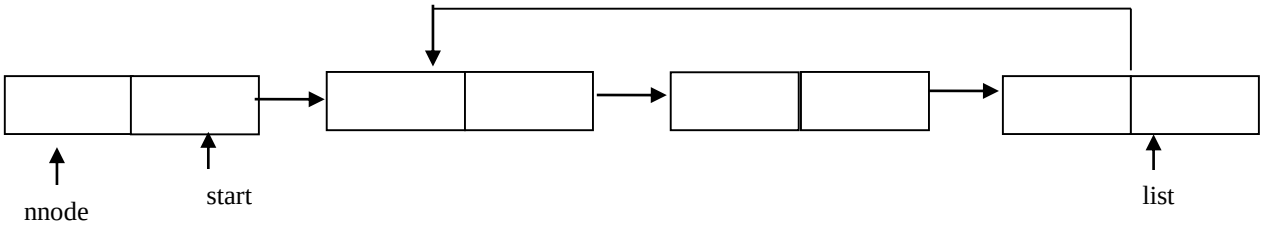


Step 1: list=start
Step 2: nnode=start
Step 3: Travel upto last node

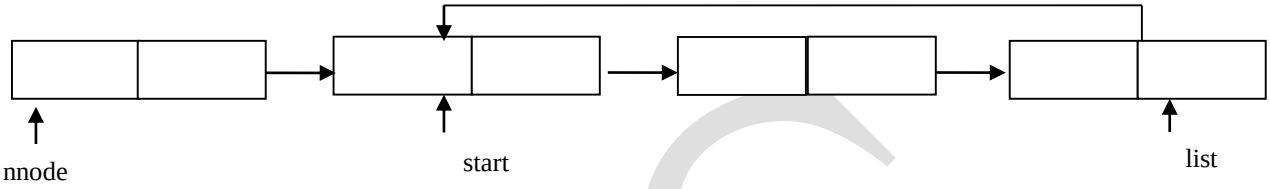
```
while(list->next!=start)
list =list->next
```



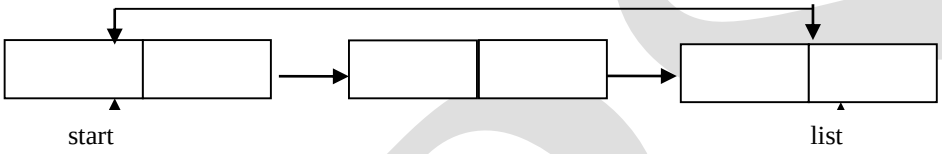
Step 4:list->next=nnode->next



Step 5: start=nnode->next



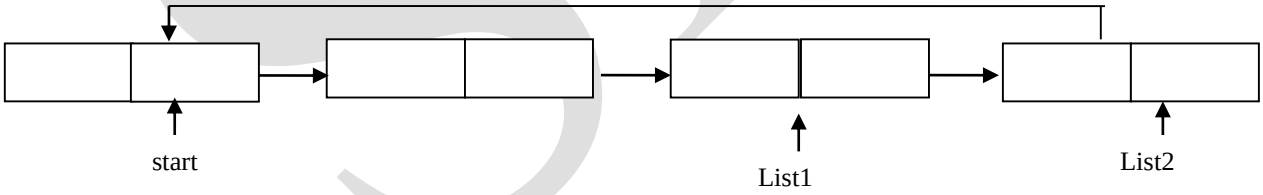
Step 6: free nnode



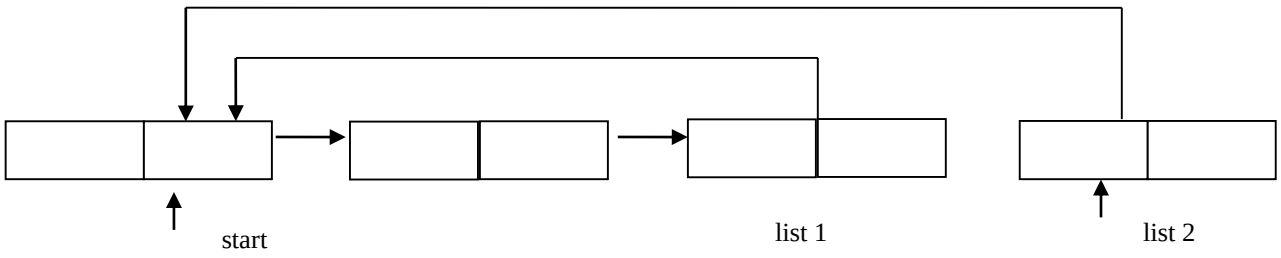
5) Delete from the end:

Step 1: list1=start
Step 2: list2=list1->next
Step 4: while (list2->next!=start)

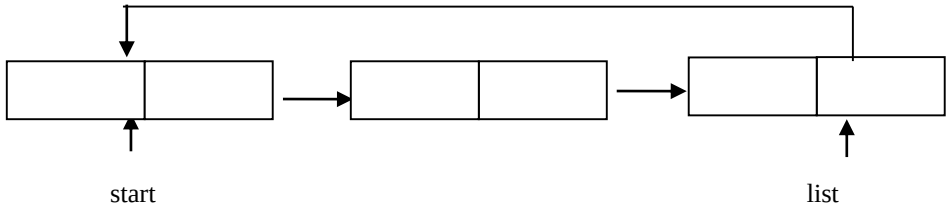
```
{
    list1=list2
    list2=list2->next
}
```



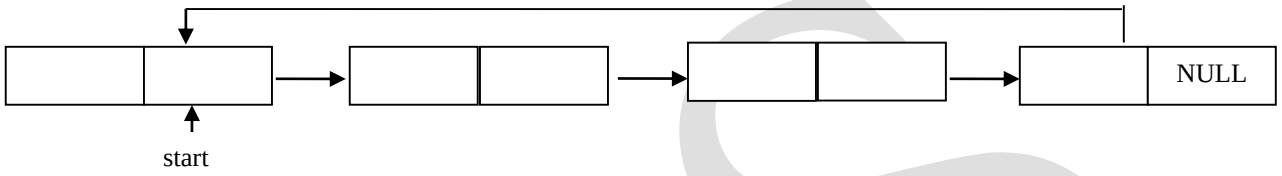
Step 5: list1->next=start



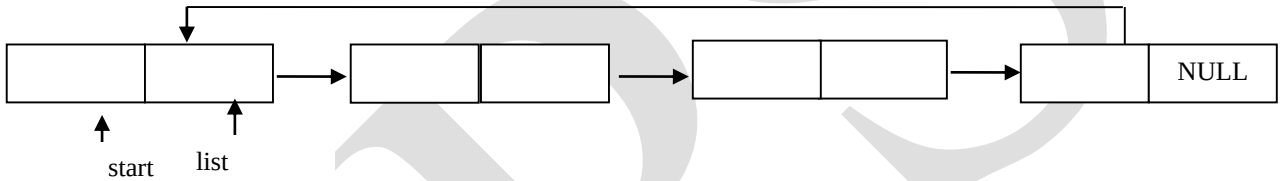
Step 5: free(list2)



6)Delete from specific position:

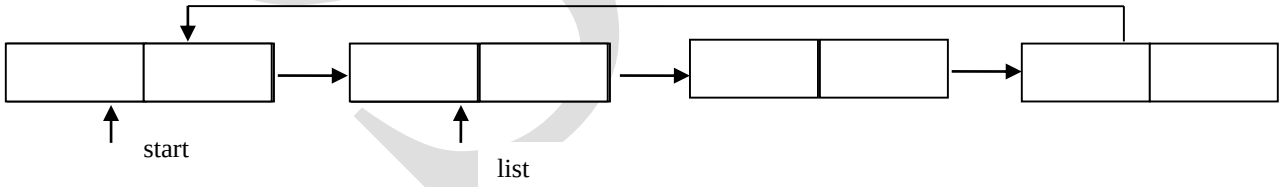


Step 1:list=start

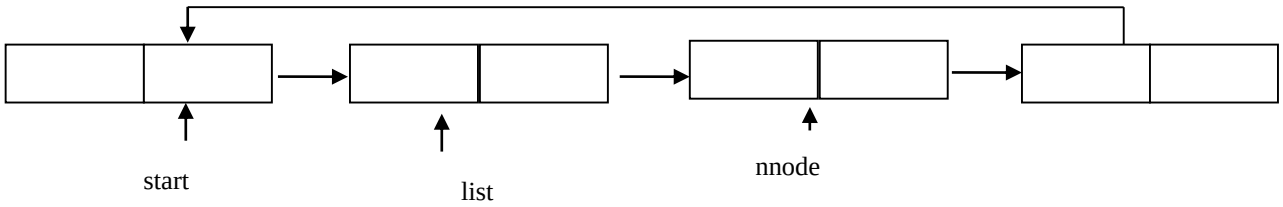


Step 2:take position to delete node (p)
Step 3:travel up to (position-1)node

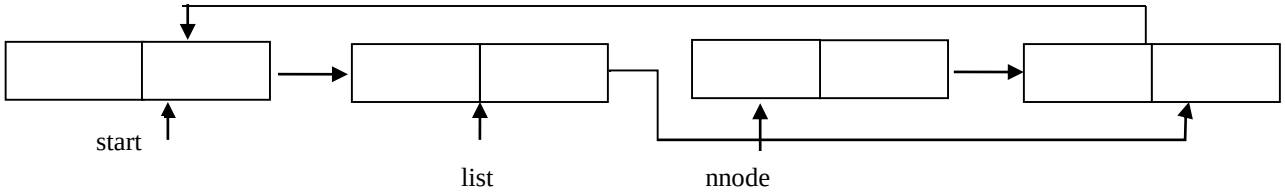
```
c=1
While(c<(p-1))
{
    List=list->next;
    c++;
}
```



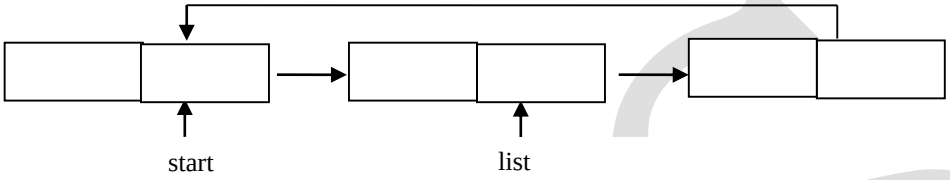
Step 4:nnode=list->next



Step 6:list->next=nnode->next

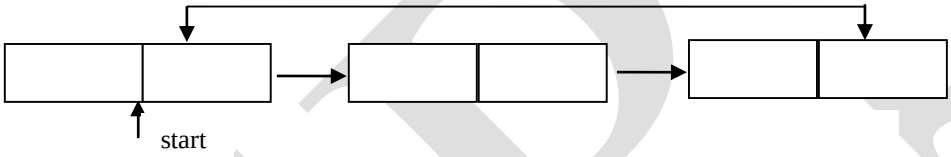


Step 7:free(nnode)



---

7) Search node:



Step1:Enter value to search node(sno)
Step 2:list=start
Step 3:do{

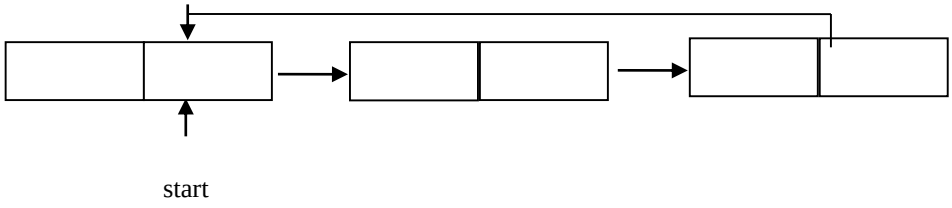
```
            if(list->info==sno)
            {
                Print "Record is found";
                    list=list->next
                Break;
            }
        list=list->next
    } while(list!=start)
```

Step 4:if(list==start)

```
            print "Record is not found"
```

---

8)Modify:



Step 1: take value of modify node(mno)

```
Step 2: list=start
Step 3:do
          {
              if(list->info==mno)
                {
                        Printf(“record is found”);
                        Enter new info
                              Update list->info
                              list=list->next
                        break;
              }
                  list=list->next
          } while(list!=start)
Step 4:     if(list==NULL)
                         Record is not found
```

C Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int info;
 struct node *next;
};
struct node *start,*list,*list1,*nnode;
void create();
void display();
void append();
void insertP();
void insertN();
void deletP();
void deletN();
void search();
void modify();

void main()
{
   int ch=0;
   while(ch!=10)
   {
         printf("\n1:create");
         printf("\n2:display");
         printf("\n3:append");
         printf("\n4:insert at position");
```

```
        printf("\n5:insert after node");
        printf("\n6:delete at position");
        printf("\n7:delete node");
        printf("\n8:search");
        printf("\n9:modify");
        printf("\n10:exit");
        printf("\n Enter choice:");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:create();
                     break;
                case 2:display();
                     break;
                case 3:append();
                     break;
                case 4:insertP();
                     break;
                case 5:insertN();
                        break;
                case 6:deletP();
                     break;
                case 7:deletN();
                        break;
                case 8:search();
                     break;
                case 9:modify();
                     break;
                case 10:exit(0);
                     break;
                default:printf("\nEnter proper choice:");
        }
    }
}

void create()
{
        char ch='y';
        start=NULL;
        list=NULL;
        while(ch=='y')
        {
```

```
                nnode=(struct node*)malloc(sizeof(struct node));
                nnode->next=NULL;
                printf("\nEnter values:");
                scanf("%d",&nnode->info);
                if(start==NULL)
                {
                        start=nnode;
                        list=nnode;
                        nnode->next=start;
                }
                else
                {
                        list->next=nnode;
                        list=list->next;
                        nnode->next=start;
                }
                printf("Do you want to continue (if 'yes' then press 'y')");
                fflush(stdin);
                ch=getchar();
        }
}
void display()
{
        if(start==NULL)
        {
                printf("List is empty");
        }
        else
        {
                list=start;
                do
                {
                        printf("value=%d",list->info);
                        list=list->next;
                }while(list!=start);
        }
}

void append()
{
        nnode=(struct node*)malloc(sizeof(struct node));
        nnode->next=NULL;
```

```
        printf("\nEnter values:");
        scanf("%d",&nnode->info);
        list=start;
        while(list->next!=start)
                list=list->next;
        list->next=nnode;
        nnode->next=start;
        printf("\nNode inserted..");
}

void insertP()
{
        int c,p;
        nnode=(struct node*)malloc(sizeof(struct node));
        nnode->next=NULL;
        printf("\nEnter values:");
        scanf("%d",&nnode->info);
        printf("\nInsert position where you want to insert:");
        scanf("%d",&p);
        c=1;
        list=start;
        if(p==1)
        {
                while(list->next!=start)
                {
                        list=list->next;
                }
                nnode->next=start;
                list->next=nnode;
                start=nnode;
        }
        else
        {
                while(list->next!=start)
                {
                        if(c<(p-1))
                        {
                                list=list->next;
                                c++;
                        }
                        else
                        {
```

```
                        break;
                        }
                }
                nnode->next=list->next;
                list->next=nnode;
                printf("\nNode inserted");
        }
}
void insertN()
{
        int n;
        nnode=(struct node*)malloc(sizeof(struct node));
        nnode->next=NULL;
        printf("\nEnter values:");
        scanf("%d",&nnode->info);
        printf("\nInsert node after which you want to insert:");
        scanf("%d",&n);
        list=start;
        if(start->info==n)
        {
                while(list->next!=start)
                {
                        list=list->next;
                }
                nnode->next=start;
                list->next=nnode;
                start=nnode;
        }
        else
        {
                while(list->next!=start)
                {
                        if(list->info!=n)
                          list=list->next;
                        else
                        {
                                break;
                        }
                }
                nnode->next=list->next;
                list->next=nnode;
                printf("\nNode inserted");
```

```
        }
}

void deletP()
{
        int c,p;
        printf("\nEnter the position to delete:");
        scanf("%d",&p);
        c=1;
        list=start;
        if(p==1)
        {
          while(list->next!=start)
                list=list->next;
          nnode=start;
          start=nnode->next;
          list->next=start;
        }
        else
        {
          while(c<(p-1))
          {
                list=list->next;
                c++;
          }
          nnode=list->next;
          list->next=nnode->next;
        }
        free(nnode);
}

void deletN()
{
        int n;
        printf("\nEnter the node value to delete:");
        scanf("%d",&n);
        list=start;
        list1=list->next;
        if(start->info==n)
        {
          while(list->next!=start)
                list=list->next;
```

```
            nnode=start;
            start=nnode->next;
            list->next=start;
            list1=list1->next;
            printf("Node %d is deleted",n);
          }
          else
          {
            do
            {
             if(list1->info!=n)
             {
                    list=list1;
                    list1=list1->next;
             }
             else
             {
               nnode=list->next;
               list->next=nnode->next;
                printf("Node %d is deleted",n);
                break;
             }
            }while(list1!=start);
          }
         if(list1==start)
             printf("Node %d is not found",n);
          free(nnode);
}

void search()
{
          int sno;
          printf("\nEnter value of search record:");
          scanf("%d",&sno);
          list=start;
          do{

                    if(list->info==sno)
                    {
                              printf("\nRecord found:");
                              printf("\nValue is: %d",list->info);
                              break;
```

```
                }
                list=list->next;
                if(list==start)
                {
                        printf("\nRecord not found.");
                }
        }while(list!=start);

}
void modify()
{
        int mno;
        printf("\nEnter value of search record:");
        scanf("%d",&mno);
        list=start;
        do
        {
                if(list->info==mno)
                {
                        printf("\nRecord found");
                        printf("\nInfo is:%d",list->info);
                        printf("\nEnter new info:");
                        scanf("%d",&list->info);
                        printf("\nOne record is modified");
                        break;
                }
                list=list->next;
                if(list==start)
                {
                        printf("\nRecord not found.....");
                }
        } while(list!=start);
}
```

CIRCULAR QUEUE USING LINKED LIST:-

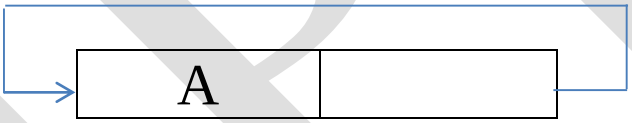
In circular queue, rear node points to the front node.
In circular queue, two pointers must need to be maintained:

(a) Address of the front node (for deletion)
(b) Address of the rare node (for insertion)

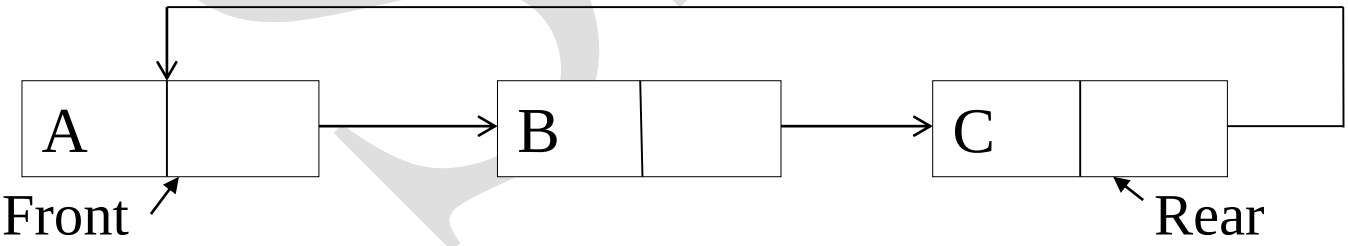
STEPS FOR INSERTION:

Step 1: START
Step 2: Create a node (nnode)
Step 4: nnode->next=NULL
Step 3:Assigne a value

Step 4: If Queue is empty
Then Front=nnode
Rear=nnode
Rear->next=Front



Else
Rear->next=nnode
Rear=Rear->next
Rear->next=Front



Step 5: Stop

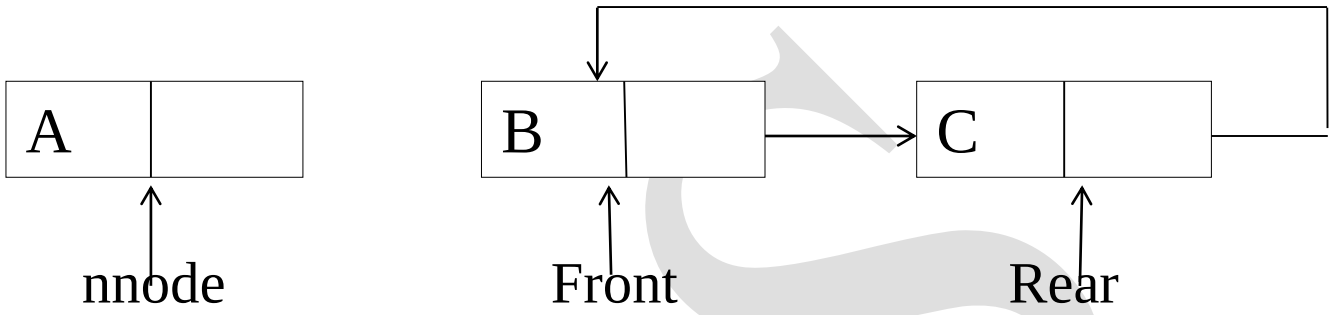
STEPS FOR DELETION:

Step 1: START

Step 2: If Front!=NULL
If Front==Rear
Front=NULL
Rear=NULL

```
        Else
                nnode = Front
                Front=Front->next;
                Rear->next=Front;
```

Step 3: Free (nnode)



Step 4: If Front==NULL
Then PRINT "Queue underflow"
Step 5: STOP

C Program (using Linked List)

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
 int info;
 struct node *next;
};
struct node *front,*rear,*nnode,*list;
void insert();
void delet();
void display();

void main()
{
        int ch=0;
        clrscr();
        while(ch!=4)
        {
                printf("\n****MAIN MENU****");
                printf("\n1.INSERT");
```

```
                printf("\n2.DELETE");
                printf("\n3.DISPLAY");
                printf("\n4.EXIT");
                printf("\nEnter Your Choice: ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: insert();
                                break;
                        case 2: delet();
                                break;
                        case 3: display();
                                break;
                        case 4: exit(0);
                        default:printf("\nEnter Proper Choice..");
                }
        }
        printf("\nprogram exits here..");
}

void insert()
{
        nnode=(struct node*)malloc(sizeof(struct node));
        nnode->next=NULL;
        printf("\nEnter values:");
        scanf("%d",&nnode->info);
        if(front==NULL)
        {
                front=nnode;
                rear=nnode;
                nnode->next=front;
        }
        else
        {
                rear->next=nnode;
                nnode->next=front;
                rear=rear->next;
        }
}

void delet()
{
```

```
        if(front!=NULL)
        {
                if(front==rear)
                {
                 front=NULL;
                 rear=NULL;
                }
                else
                {
                 nnode=front;
                 front=front->next;
                 rear->next=front;
                }
                free(nnode);
        }
        else
        {
                printf("\nQUEUE UNDERFLOW...");
        }
}

void display()
{
        if(front==NULL)
        {
                printf("\nQUEUE IS EMPTY...");
        }
        else
        {
                list=front;
                printf("\nQueue Elements Are: ");
                do
                {
                        printf(" %d",list->info);
                        list=list->next;

                }while(list!=front);
        }
}
```

Singly linked list	Doubly linked list
1. Singly linked list allow you to go one way direction. 2. Singly linked list uses less memory per node. (one pointer) 3. There is a little known trick that lets you delete from a singly linked list in 0(1) but the list must be circular for it to work. 4. Complexity of insertion and deletion at known position is 0(1) 5. If we need to save memory in need to update node values frequently and searching is not required, we can use singly linked list. 6. In single list each node contains at least two parts. a) Info b) Link	1. Doubly linked list has two way directions next and previous. 2. Doubly linked list uses more memory per node than singly linked list. (two pointers) 3. Doubly linked list can be used in places where singly linked list would not work but they require slightly more "housekeeping" and are slightly less Efficient on insertion as the result 4. Complexity of insertion and deletion at known position is 0(n) 5. If we need faster performance in searching is not a limitation we use doubly linked list. 6. In doubly linked list each node contains at least three parts a) Info b) Link to next node c) Link to previous node

Applications of doubly linked list

1. Applications that have an MRU list (a linked list of file names)
2. The cache in your browser that allows you to hit the BACK button (a linked list of URLs)
3. Undo functionality in Photoshop or Word (a linked list of state)
4. A stack, hash table, and binary tree can be implemented using a doubly linked list.
5. A great way to represent a deck of cards in a game.

Doubly Linked List

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

struct node
{
        int info;
        struct node *prev;
        struct node *next;
};

struct node *nnode,*start,*list,*ppt,*npt;
void create();
```

```
void ftrav();
void btrav();
void append();
void insertP();
void insertN();
void deletP();
void deletN();
void search();
void modify();
void main()
{
  int ch=0;
  while(ch!=11)
  {
        printf("\n1.Create");
        printf("\n2.ftrav");
        printf("\n3.btrav");
        printf("\n4.Append");
        printf("\n5.Insert at Position");
        printf("\n6.Insert after node");
        printf("\n7.Delete from position");
        printf("\n8.Delete Node");
        printf("\n9.Search");
        printf("\n10.Modify");
        printf("\n11.Exit");
        printf("\nEnter your choice");
        scanf("%d",&ch);
        switch(ch)
        {
              case 1:create();
                    break;
              case 2:ftrav();
                    break;
              case 3:btrav();
                    break;
              case 4:append();
                    break;
              case 5:insertP();
                    break;
              case 6:insertN();
                    break;
              case 7:deletP();
                    break;
              case 8:deletN();
                    break;
              case 9:search();
                    break;
              case 10:modify();
                    break;
```

```
                case 11:exit(0);
                default:printf("\nEnter correct choice");
        }
   }
}

void create()
{
        char ch='y';
        start=NULL;
        list=NULL;
        while(ch=='y')
        {
                nnode=(struct node *)malloc(sizeof(struct node));
                printf("\nEnter value:");
                scanf("%d",&nnode->info);
                nnode->prev=NULL;
                nnode->next=NULL;
                if(start==NULL)
                {
                        start=nnode;
                        list=nnode;
                }
                else
                {
                        list->next=nnode;
                        nnode->prev=list;
                        list =nnode;
                }
                printf("\nDo you want to continue?(If yes ,press'y')");
                fflush(stdin);
                ch=getch();
        }
}

void ftrav()
{
        if(start==NULL)
        {
                printf("\nList is empty");
        }
        else
        {
                list=start;
                while(list!=NULL)
                {
                        printf("\ninfo=%d",list->info);
                        list=list->next;
                }
```

```
        }
}
void btrav()
{
        if(start==NULL)
        {
                printf("\nList is empty");
        }
        else
        {
                list=start;
                while(list->next!=NULL)
                        list=list->next;
                while(list!=NULL)
                {
                        printf("\ninfo=%d",list->info);
                        list=list->prev;
                }
        }
}

void append()
{
        nnode=(struct node *)malloc(sizeof(struct node));
        printf("\nEnter value:");
        scanf("%d",&nnode->info);
        nnode->prev=NULL;
        nnode->next=NULL;
        list=start;

        while(list->next!=NULL)
                        list=list->next;
        nnode->prev=list;
        list->next=nnode;
}

void insertP()
{
        int p,c;
        nnode=(struct node *)malloc(sizeof(struct node));
        printf("\nEnter value:");
        scanf("%d",&nnode->info);
        nnode->prev=NULL;
        nnode->next=NULL;
        printf("\nEnter the position where you want to insert a node:");
        scanf("%d",&p);
        list=start;
        c=1;
        if(p==1)
```

```
        {
                nnode->next=start;
                start->prev=nnode;
                start=nnode;
        }
        else
        {
                while(list->next!=NULL)
                {
                        if(c<(p-1))
                        {
                                list=list->next;
                                c++;
                        }
                        else
                        {
                                break;
                        }
                }
                npt=list->next;
                nnode->prev=list;
                nnode->next=npt;
                list->next=nnode;
                npt->prev=nnode;

        }
}

void insertN()
{
        int sno;
        nnode=(struct node *)malloc(sizeof(struct node));
        nnode->next=NULL;
        nnode->prev=NULL;
        printf("\nEnter value:");
        scanf("%d",&nnode->info);
        printf("\nEnter the Info after which you want to inser node:");
        scanf("%d",&sno);
        list=start;
        while(list->next!=NULL)
        {
           if(list->info!=sno)
           {
                list=list->next;
           }
           else
           {
                break;
           }
```

```
        }
        npt=list->next;
        nnode->prev=list;
        nnode->next=npt;
        list->next=nnode;
        npt->prev=nnode;
}

void deletP()
{
        int p,c;
        printf("\nEnter the position of node  which you want to delete:");
        scanf("%d",&p);
        list=start;
        c=1;
        if(p==1)
        {
                npt=list->next;
                npt->prev=NULL;
                start=npt;
                free(list);
        }
        else
        {
                while(list->next!=NULL)
                {
                        if(c<(p-1))
                        {
                                list=list->next;
                                c++;
                        }
                        else
                        {
                                nnode=list->next;
                                npt=nnode->next;
                                list->next=npt;
                                npt->prev=list;
                                free(nnode);
                                break;
                        }
                }

        }
}

void deletN()
{
        int n;
        printf("\nEnter the info of node  which you want to delete:");
```

```
        scanf("%d",&n);
        list=start;
        if(start->info==n)
        {
                nnode=start;
                npt=list->next;
                npt->prev=NULL;
                start=npt;
        }
        else
        {
           while(list->next!=NULL)
           {
                if(list->info!=n)
                {
                  list=list->next;
                }
                else
                {
                  break;
                }
           }
           nnode=list;
           ppt=nnode->prev;
           npt=nnode->next;
           ppt->next=npt;
           npt->prev=ppt;
        }
        free(nnode);
}

void search()
{
        int sno;
        printf("nEnter a value to search record:");
        scanf("%d",&sno);
        list=start;
        while(list!=NULL)
        {
                if(list->info==sno)
                {
                        printf("\nRecord found");
                        printf("\nValue is %d",list->info);
                        break;
                }
                list=list->next;
        }
        if(list==NULL)
        {
```

```
                printf("\nRecord not found");
        }
}

void modify()
{
        int mno;
        printf("\nEnter a value to be modify:");
        scanf("%d",&mno);
        list=start;
        while(list!=NULL)
        {
                if(list->info==mno)
                {
                        printf("\nRecord found");
                        printf("\nValue is %d",list->info);
                        printf("\nEnter a new value:");
                        scanf("%d",&list->info);
                        printf("\nOne record is modified");
                        list=NULL;
                        break;
                }
                list=list->next;
        }
        if(list==NULL)
        {
                printf("\nRecord not found");
        }
}
```

Circular Doubly Linked List

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *prev;
        struct node *next;
};
struct node *nnode,*start,*list,*ppt,*npt;
void create();
void ftrav();
void btrav();
void append();
void insertP();
void insertN();
void deletP();
void deletN();
void search();
void modify();
void main()
{
        int ch=0;
        clrscr();
        while(ch!=11)
        {
        printf("\n1.Create");
        printf("\n2.ftrav");
        printf("\n3.btrav");
        printf("\n4.Append");
        printf("\n5.Insert at Position");
        printf("\n6.Insert after node");
        printf("\n7.Delet from position");
        printf("\n8.Delet Node");
        printf("\n9.Search");
        printf("\n10.Modify");
        printf("\n11.Exit");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1:create();
                        break;
                case 2:ftrav();
                        break;
```

```
                case 3:btrav();
                        break;
                case 4:append();
                        break;
                case 5:insertP();
                        break;
                case 6:insertN();
                        break;
                case 7:deletP();
                        break;
                case 8:deletN();
                        break;
                case 9:search();
                        break;
                case 10:modify();
                        break;
                case 11:exit(0);
                default:printf("\nEnter correct choice");
                        break;
        }
        }
}
void create()
{
        char ch='y';
        start=NULL;
        list=NULL;
        while(ch=='y')
        {
                nnode=(struct node *)malloc(sizeof(struct node));
                printf("\nEnter value:");
                scanf("%d",&nnode->info);
                nnode->prev=NULL;
                nnode->next=NULL;
                if(start==NULL)
                {
                        start=nnode;
                        list=nnode;
                        nnode->next=nnode;
                        nnode->prev=nnode;
                }
                else
                {
                        list->next=nnode;
                        nnode->prev=list;
                        nnode->next=start;
                        start->prev=nnode;
                        list=nnode;
                }
```

```
                printf("\nDo you want to continue?(If yes ,press'y')");
                fflush(stdin);
                ch=getch();
        }
}

void ftrav()
{
        if(start==NULL)
        {
                printf("\nList is empty");
        }
        else
        {
                list=start;
                do
                {
                        printf("\ninfo=%d",list->info);
                        list=list->next;
                }while(list!=start);

        }
}
void btrav()
{
        if(start==NULL)
        {
                printf("\nList is empty");
        }
        else
        {
                list=start;
                ppt=list->prev;
                do
                {
                        printf("\ninfo=%d",ppt->info);
                        ppt=ppt->prev;
                }while(ppt!=list->prev);
        }
}

void append()
{
        nnode=(struct node *)malloc(sizeof(struct node));
        printf("\nEnter value:");
        scanf("%d",&nnode->info);
        nnode->prev=NULL;
        nnode->next=NULL;
        list=start;
```

```
	if(start==NULL)
	{
		start=nnode;
		nnode->prev=start;
		nnode->next=start;
	}
	else
	{
		list=start->prev;
		nnode->prev=list;
		nnode->next=start;
		list->next=nnode;
		start->prev=nnode;
	}
}

void insertP()
{
	int p,c;
	nnode=(struct node *)malloc(sizeof(struct node));
	printf("\nEnter value:");
	scanf("%d",&nnode->info);
	nnode->prev=NULL;
	nnode->next=NULL;
	printf("\nEnter the position where you want to insert a node:");
	scanf("%d",&p);
	list=start;
	c=1;
	if(p==1)
	{
		list=start->prev;
		nnode->next=start;
		nnode->prev=list;
		start->prev=nnode;
		list->next=nnode;
		start=nnode;
	}
	else
	{
		while(list->next!=start)
		{
			if(c<(p-1))
			{
				list=list->next;
				c++;
			}
			else
			{
				break;
```

```
                        }
                }
                npt=list->next;
                nnode->prev=list;
                nnode->next=npt;
                list->next=nnode;
                npt->prev=nnode;
        }
}

void insertN()
{
        int sno;
        nnode=(struct node *)malloc(sizeof(struct node));
        nnode->next=NULL;
        nnode->prev=NULL;
        printf("\nEnter value:");
        scanf("%d",&nnode->info);
        printf("\nEnter the Info after which you want to insert node:");
        scanf("%d",&sno);
        list=start;
        while(list->next!=start)
        {
           if(list->info!=sno)
           {
                list=list->next;
           }
           else
           {
                break;
           }
        }
        npt=list->next;
        nnode->prev=list;
        nnode->next=npt;
        list->next=nnode;
        npt->prev=nnode;
}

void deletP()
{
        int p,c;
        printf("\nEnter the position of node  which you want to delete:");
        scanf("%d",&p);
        list=start;
        c=1;
        if(p==1)
        {
                npt=list->next;
```

```
                ppt=list->prev;
                npt->prev=ppt;
                ppt->next=npt;
                start=npt;
                free(list);
        }
        else
        {
                while(list->next!=start)
                {
                        if(c<(p-1))
                        {
                                list=list->next;
                                c++;
                        }
                        else
                        {
                                nnode=list->next;
                                npt=nnode->next;
                                list->next=npt;
                                npt->prev=list;
                                free(nnode);
                                break;
                        }
                }

        }
}

void deletN()
{
        int sno;
        printf("\nEnter the info of node  which you want to delete:");
        scanf("%d",&sno);
        list=start;
        if(start->info==sno)
        {
                nnode=start;
                ppt=list->prev;
                npt=list->next;
                ppt->next=npt;
                npt->prev=ppt;
                start=npt;
        }
        else
        {
           while(list->next!=start)
           {
                if(list->info!=sno)
```

```
                {
                  list=list->next;
                }
                else
                {
                  break;
                }
            }
            nnode=list;
            ppt=nnode->prev;
            npt=nnode->next;
            ppt->next=npt;
            npt->prev=ppt;
        }
        free(nnode);
}

void search()
{
        int sno;
        printf("nEnter a value to search record:");
        scanf("%d",&sno);
        list=start;
        do
        {
                if(list->info==sno)
                {
                        printf("\nRecord found");
                        printf("\nValue is %d",list->info);
                        list=NULL;
                        break;
                }
                list=list->next;
        }while(list!=start);
        if(list==start)
        {
                printf("\nRecord not found");
        }
}

void modify()
{
        int mno;
        printf("\nEnter a value to be modify:");
        scanf("%d",&mno);
        list=start;
        do
        {
                if(list->info==mno)
```

```
                {
                        printf("\nRecord found");
                        printf("\nValue is %d",list->info);
                        printf("\nEnter a new value:");
                        scanf("%d",&list->info);
                        printf("\nOne record is modified");
                        list=NULL;
                        break;
                }
                list=list->next;
        }while(list!=start);
        if(list==start)
                {
                        printf("\nRecord not found");
                }
}
```

Application of Linked List

1. Representation of Sparse Matrix
2. Polynomial representation

Sparse Matrix

A sparse matrix is that matrix which has very few non zero elements as compared to the size of M X N of the matrix. E.g. if the matrix is of size 50 X 60 and only 9 elements are non-zero, then store these 9 elements it require 50 X 60 X 2 memory, and to access these 9 elements it has to scan 3000 elements.

Therefore concept of sparse matrix came forward to store only non-zero elements of the matrix and still carry out the operations efficiently.

Representation of Sparse matrix

The representation of sparse matrix will be a triplet only. Basically sparse matrix means very few non zero value in it & rest of the matrix is empty.

So far efficient representation will consider only non zero values along with its positions. In the representation of sparse matrix first rows first cell contain pointer to the first rhead (Row head), second contain total number of rows, third contain total number of column, and last cell contain pointer to the first chead (column head).

A pointer to rhead node	Number of rows	Number of column	A pointer to chead node

smat node (Sparse Matrix node)

rhead node contains three cell to store information. In rhead node's first cell contain pointer to first non-zero node in the row list, second contain row number and third contain a pointer to the next rhead node.

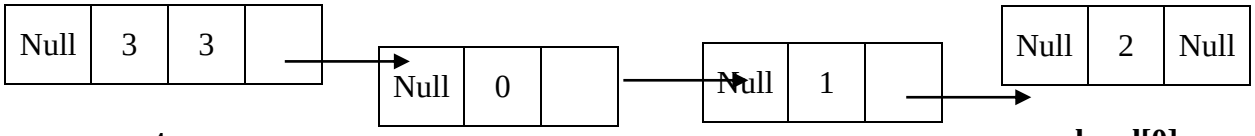
Pointer to next rhead	Row number	A Pointer to first non-zero node in row list

rhead Node

chead node contains three cell to store information. In chead node's first cell contain pointer to first non-zero node in the column list, second contain column number and third contain a pointer to the next chead node.

A Pointer to first non-zero node in column list	Column number	Pointer to next chead

chead Node



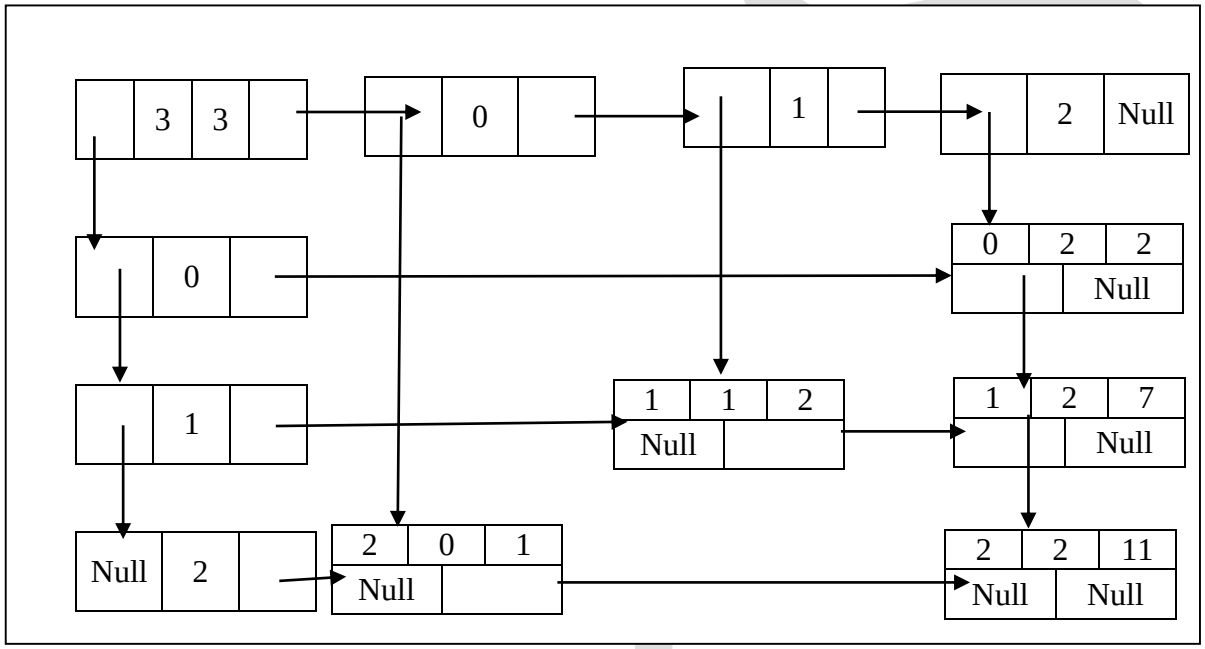
smat **chead[0]**

chead[1] **chead[2]**

Information stored in node containing non-zero data is row number, column number, non-zero value, a pointer to the next node in same column list, a pointer to the next node in the list same row list.

Row number	Column number	Non-zero value
A pointer to next node in same column list		A pointer to next node in same row list

Node



Sparse Representation

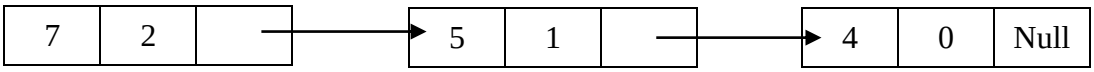
Polynomial Representation:

Polynomial has main field as coefficient and exponent. In linked list representation each node of polynomial will have one more field called link field to point next term in the polynomial.

Representation of node is

coefficient	**exponent**	**next**

To represent $7X^2 + 5X + 4$ the link list will be,



Node structure for a singly linked list for reporting a term of polynomial can be defined as follows

```
struct node
{
        int coef;
        int pow;
        struct node *next;
};
```

Addition of two Polynomial equation

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

struct node
{
  int coe;
  int pow;
  struct node * next;
};
struct node *nnode, *start1, *start2, *start3, *list1, *list2, *list3;

struct node * create();
void display(struct node *);
void add();

void main()
{
 start1=NULL;
 start2=NULL;
 printf("Enter first Polynomial equation\n");
 start1=create();
 printf("Enter Second Polynomial equation\n");
 start2=create();
 printf("\nFirst Polynomial equation");
 display(start1);
 printf("\nSecond Polynomial equation");
 display(start2);

 add(start1,start2);
 getch();
```

```
}

struct node * create()
{
 int i;
 struct node *start;
 start=NULL;
 for(i=2; i>=0; i--)
 {
  nnode= (struct node *) malloc( sizeof (struct node));
  printf("Enter coe of X ^ %d ",i);
  scanf("%d",&nnode->coe);
  nnode->pow=i;
  nnode->next=NULL;
  if(start==NULL)
  {
    start=nnode;
    list1=start;
  }
  else
  {
   list1->next=nnode;
   list1=list1->next;
  }
 }
 return start;
}
void display(struct node *list)
{
 int i;
 for(i=2; i>=0; i--)
 {
  printf("+ %dX^%d ",list->coe,list->pow);
  list=list->next;
 }
}

void add(struct node *list1, struct node * list2)
{
 int i;
 start3=NULL;
 list3=start3;
 for(i=2; i>=0; i--)
 {
  nnode= (struct node *) malloc( sizeof (struct node ));
   nnode->coe=list1->coe+list2->coe;
   nnode->pow=list1->pow;
   list1=list1->next;
   list2=list2->next;
```

```
  if(list3==NULL)
   {
      start3=nnode;
      list3=start3;
   }
   else
   {
     list3->next=nnode;
     list3=nnode;
   }
 }
 printf("\n Addition is ");
 display(start3);
}
```