
# Data Structure & Applications

# Module III Notes

Dr. Ganga Holi,
Professor & Head,

**Department of Information Science & Engineering**

**GLOBAL ACADEMY OF TECHNOLOGY**

**Rajarajeshwari Nagar, Bangalore-560 098.**

# Table of Contents

# 1 Linked List Definition

List is a linear collection of data items. List can be implemented using arrays and linked lists. In arrays there is linear relationship between the data elements which is evident from the physical relationship of data in the memory. The address of any element in the array can easily be computed but, it is very difficult to insert and delete any element in an array. Usually, a large block of memory is occupied by an array which may not be in use and it is difficult to increase the size of an array, if required.



Another way of storing a list is to have each element in a list contain a field called a link or pointer, which contains the address of the address of the next element in the list. The successive elements in the list need not occupy adjacent space in memory. This type of data structure is called linked list.

Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.


### *1.1.1.1 Advantages of Linked Lists*

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

### *1.1.1.2 Disadvantages of Linked Lists*

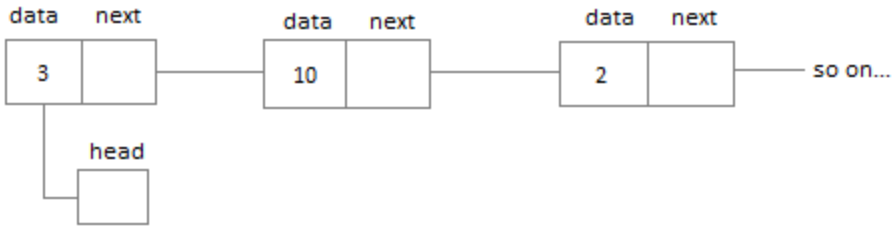
- The memory is wasted as pointers require extra memory for storage.
- No element can be accessed randomly; it has to access each node sequentially.
- Reverse Traversing is difficult in single linked list.

### *1.1.1.3 Applications of Linked Lists*

- Linked lists are used to implement stacks, queues, graphs, etc.
- Linked lists let you insert elements at the beginning and end of the list.
- In Linked Lists we don't need to know the size in advance.

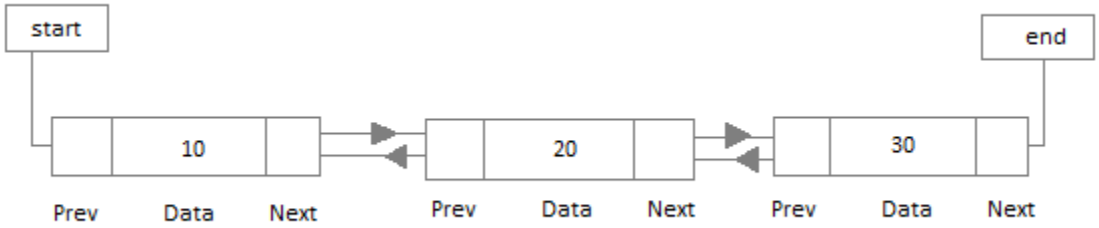
### *1.1.1.4 Types of Linked Lists*

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

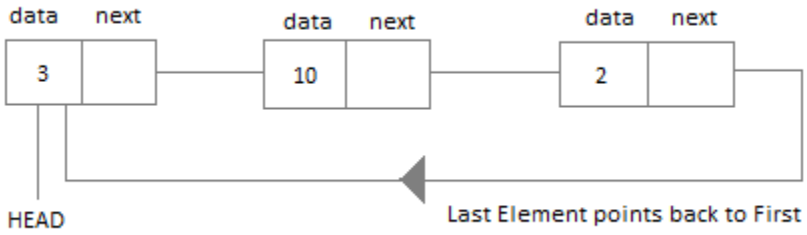





- **Doubly Linked List :** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.




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




## 2 Representation of Linked List in Memory

List can be represented using arrays and linked list.

**Array Representation:**

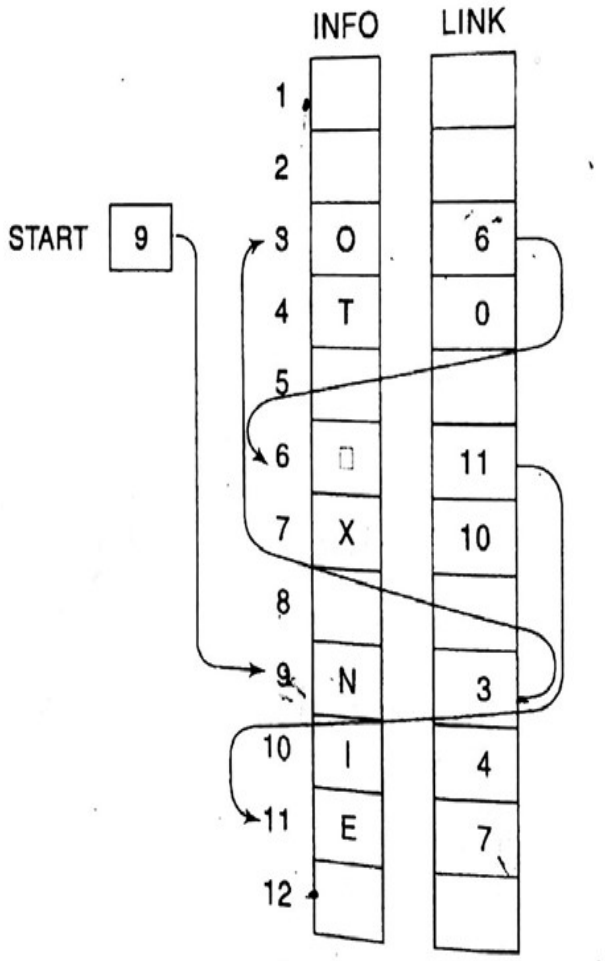
Linked list can be implemented using arrays. One array will be used for storing the information field and another array is used for storing the links. START variable will hold the beginning node address and INFO[9] has value and LINK[9] has the address of another node. Here

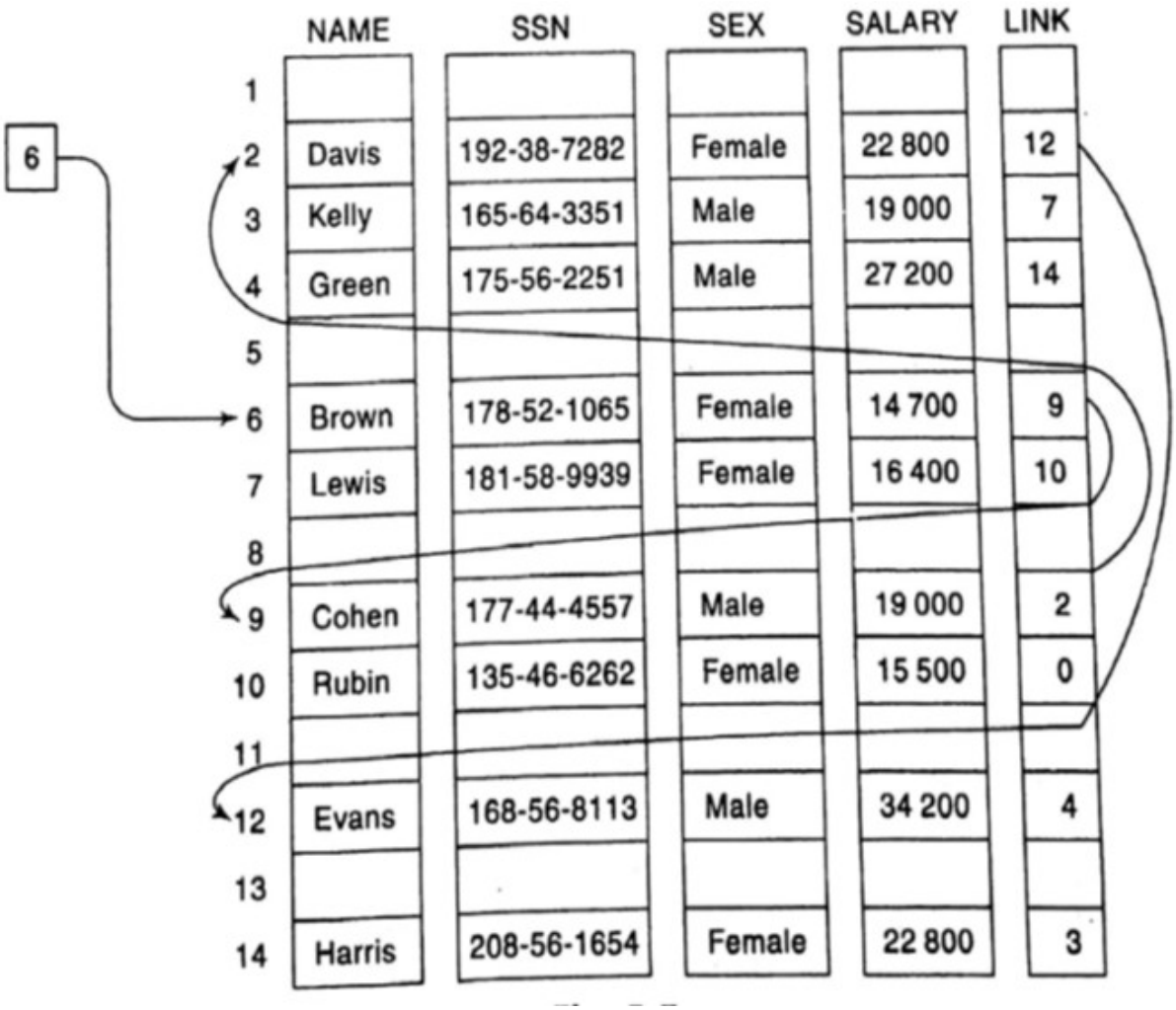
LINK[9]=3 LINK[3]=6

LINK[6]=11 LINk[11]=7

LINK[7]]=10 LINK[10]=4

LINKJ[4]=0 which is the last node in the linked list

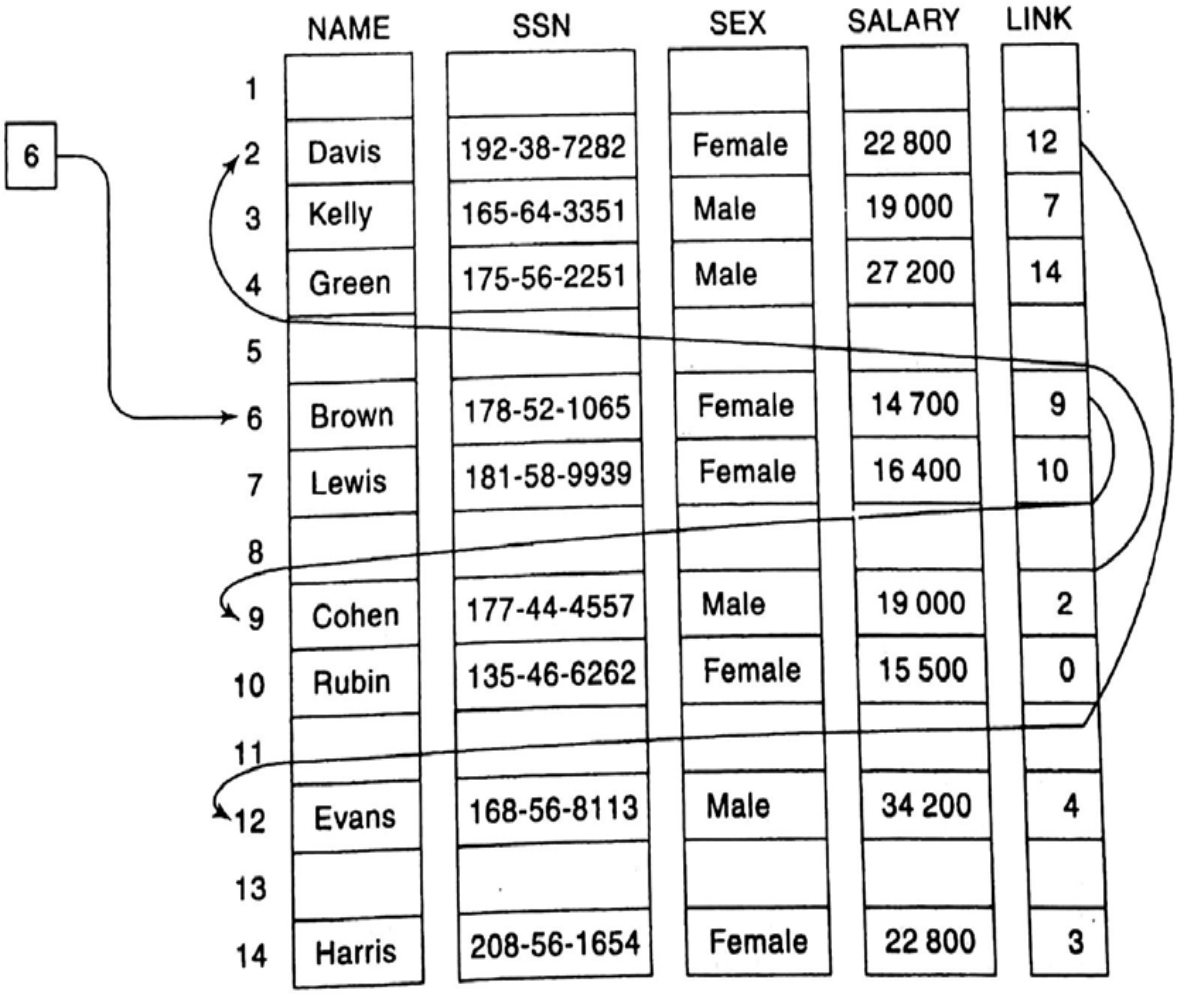



NAME
SSN
SEX
SALARY
LINK
6
1
2 Davis 192-38-7282 Female 22 800 12
3 Kelly 165-64-3351 Male 19 000 7
4 Green 175-56-2251 Male 27 200 14
5
6 Brown 178-52-1065 Female 14 700 9
7 Lewis 181-58-9939 Female 16 400 10
8
9 Cohen 177-44-4557 Male 19 000 2
10 Rubin 135-46-6262 Female 15 500 0
11
12 Evans 168-56-8113 Male 34 200 4
13
14 Harris 208-56-1654 Female 22 800 3

Suppose the personnel file of a small company contains the following data on its nine employees:

Name, Social Security Number, Sex, Monthly Salary

Normally, four parallel arrays, say NAME, SSN, SEX, SALARY, are required to store the data as discussed in Sec. 4.12. Figure 5.7 shows how the data may be stored as a sorted (alphabetically) linked list using only an additional array LINK for the nextpointer field of the list and the variable START to point to the first record in the list. Observe that 0 is used as the null pointer.



| | NAME | SSN | SEX | SALARY | LINK |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | Davis | 192-38-7282 | Female | 22 800 | 12 |
| 3 | Kelly | 165-64-3351 | Male | 19 000 | 7 |
| 4 | Green | 175-56-2251 | Male | 27 200 | 14 |
| 5 | | | | | |
| 6 | Brown | 178-52-1065 | Female | 14 700 | 9 |
| 7 | Lewis | 181-58-9939 | Female | 16 400 | 10 |
| 8 | | | | | |
| 9 | Cohen | 177-44-4557 | Male | 19 000 | 2 |
| 10 | Rubin | 135-46-6262 | Female | 15 500 | 0 |
| 11 | | | | | |
| 12 | Evans | 168-56-8113 | Male | 34 200 | 4 |
| 13 | | | | | |
| 14 | Harris | 208-56-1654 | Female | 22 800 | 3 |

6

Fig. 5.7

## 3 Memory Allocation and Deallocation

Memory cane be allocated on heap if linked list is used and free function is used to deallocate the memory.

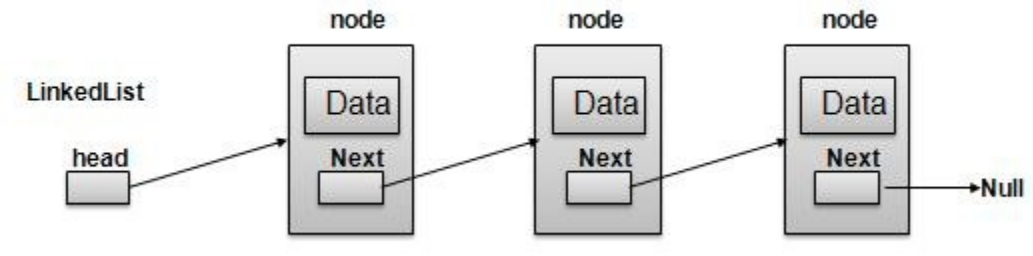
# 4 Linked List Operations

## 1. Node creation

A linked-list is a sequence of nodes which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list the second most used data structure after array. Following are important terms to understand the concepts of Linked List.



- **INFO/Link** − Each Link of a linked list can store a data called an element.
- **Next/Ptr** − Each Link of a linked list contain a link to next link called Next.
- **LinkedList** − A LinkedList contains the connection link to the first Link called First.




As per above shown illustration, following are the important points to be considered.

- Linked List contains a pointer variable called first.
- Each node carries a data field(s) and a pointer Field called next.
- Each node is linked with its next pointer using its next pointer.
- Last node carries a pointer as null to mark the end of the list.

```
Node can be created by using malloc function
sturct node {
int data;
struct node *next;
};
typedef struct node NODE;
NODE    *first=NULL;
NODE *temp;
Temp=(NODE*)malloc(siezof(NODE));
```

## 2. Insertion:

Insertion of node can be done in various ways. 1. Insertion at front. 2. Insert at rear end 3. Insert into ordered list 4. Insert at specified position.

```
void addrear(int data)
{       NODE *temp,*cur;
        temp=(NODE*)malloc(sizeof(NODE));
        temp->data=data; temp->next=NULL;
        if(first==NULL)
                temp=first; return;
        cur=first;
        while(cur->next != NULL)
        {   cur=cur->next;
        }
        cur->next =temp;
        return ;
}

void addfront(int data)
{           NODE *temp;
          temp=(NODE*)malloc(sizeof(NODE));
             temp->data=data; temp->next=NULL:
                    if (first== NULL)
                first=temp;
             else
             {          temp->next=first;
                        first=temp;
             }
             return ;
}
```



### 3. Deletion

Deletion of node can be done in various ways. 1. Deletion at from. 2. Deletion at rear end. 3. Deletion into ordered list 4. Deletion at specified position.

```
void deletefront()
{
   NODE *temp; int num;
   temp=first;
    if(first==NULL)
   {
        printf("List is Empty"); return;
   }
          printf("deleted element is %d\n", first->data);
          first=first->next;
  free(temp);
  return ;
```

```
}

void deleterear()
{
      NODE *cur, *prev;
       cur=first;
       prev=NULL;
       if(first==NULL)
      {
        printf("List is Empty"); return;
       }
       if(first->next==NULL)
       {        first=NULL;
                free(cur); return;
       }
       while(cur->next!=NULL)
       {       prev=cur;
                       cur=cur->next;
        }
        prev->next=NULL;
        free(cur);
        return;
}
```



### 4. Traversing & Displaying

```
void display()
{       NODE     *r;
        r=first;
        printf("Data \n");
        if(r==NULL)
         return;
        while(r!=NULL)
        {    printf("%d→", r->data);
             r=r->next;
        }
}
```

**What is node? How it is created?**

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

typedef struct node NODE;
NODE *first=NULL;
```

Node is created by calling malloc function and storing the return value in pointer to node variable.



## 5 Write functions to insert the node at front and rear end.

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

typedef struct node NODE;

NODE* insertfront(NODE* first, int data)
{
        NODE* temp;
        temp=(NODE*)malloc(sizeof(NODE));
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return NULL;
        }
        temp→item=data;
        temp→next=NULL;
        if(first==NULL)
                first=temp;
        else
        {
                temp→next=first;
                first=temp;
        }
        return first;
}

NODE* insertrear(NODE* first, int data)
```

```
{
        NODE *temp, *cur;
        temp=(NODE*)malloc(sizeof(NODE));
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return NULL;
        }
        temp→item=data;
        temp→next=NULL;
        if(first==NULL)
                first=temp;
        else
        {
                cur=first;
                while(cur→next!=NULL)
                {
                        cur=cur→next;
                }
                cur→next=temp;
        }
        return first;
}

void display(NODE* first)
{       NODE *cur;
        if(first==NULL)
        {
                printf(" Empty List\n"); return;
        }
        cur=first;
        while(cur!=NULL)
        {
                printf("%d→", cur→item);
                cur=cur→next;
        }
        printf("NULL");
}
```

## 6 Write functions to delete node from front and rear end.

```
NODE* deletefront(NODE* first)
{
        NODE    *temp;
        temp=first;
        if(first==NULL)
        {
```

```
                printf("List is Empty"); return first;
        }
        printf("Front element deleted =%d\n",first→item);
        first=first→next;
        free(temp);
        return first;
}

NODE*  deleterear(NODE* first)
{
        NODE    *cur, *prev;
        cur=first;
        if(first==NULL)
        {
                printf("List is Empty"); return first;
        }
        if(first→next==NULL)
        {       printf("Deleted *** rear element=%d\n",first→item);
                first=NULL;
                free(cur); return first;
        }
        prev=NULL;
        while(cur→next!=NULL)
        {       prev=cur;
                cur=cur→next;
        }
        prev→next=cur→next;
        printf("Deleted Rear element=%d\n",cur→item);
        free(cur);
        return first;
}
```



## 7 Write functions to insert into ordered linked list

```
NODE* orderedInsert(NODE* first, int ele)
{
        NODE    *nn, *pre, *cur;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn→next=NULL;
        if(first==NULL)
        {
                first=nn; return first;
        }
        if(first→item >ele)
        {
```

```
                nn→next=first;
                first=nn;
                return first;
        }

        pre=first; cur=first;
        while(cur→item <=ele && cur→next !=NULL)
        {   pre=cur;
                cur=cur→next;
        }
        if(cur→item >ele)
        {       pre→next=nn;
                nn→next=cur;
        }
        else
        {
                cur→next=nn;
        }
        return first;
}
```



## 8 Write a function to delete a specified element.

```
NODE* deleteFromordered(NODE* first, int ele)
{
        NODE *pre, *cur;
        if(first==NULL)
        {
                printf("Empty list\n"); return first;
        }
        cur=first;
        if(first→item ==ele)
        {
                first=NULL;
                printf("Deleted Rear element=%d\n",cur→item);
                free(cur);
                return first;
        }
        pre=first; cur=first;
        while(cur→item !=ele && cur→next !=NULL)
        {       pre=cur;
                cur=cur→next;
        }

        if(cur-->item !=ele && cur-->next ==NULL)
```

```
        {
                printf("Element not found\n"); return first;
        }
        pre→next=cur→next;
        }
        return first;
}
```



## 9 Write a function to insert at specified position

```
NODE*  InsertAtPos(NODE* first, int ele, int pos)
{
        NODE    *nn, *pre, *cur; int count=1;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn→next=NULL;
        if(first==NULL)
        {
                first=nn; return first;
        }
        if(pos==1)
        {
                nn→next=first;
                first=nn;
                return first;
        }

        pre=first; cur=first;
        while(count < pos && cur→next !=NULL)
        {       pre=cur; count++;
                cur=cur→next;
        }
        if(count >= pos)
        {       pre→next=nn;
                nn→next=cur;
        }
        else
        {
                cur→next=nn;
        }
        return first;
}
```

## 10 Write a function to delete element from specified position

```
NODE*    DeleteFromPos(NODE* first, int pos)
{
        NODE    *pre, *cur; int count=1;
        pre=NULL; cur=first;
        if(first==NULL)
    {
        printf("List is Empty"); return first;
        }
        if(pos==1)
        {
                printf("Front element deleted =%d\n",first→item);
                first=first→next;
                free(cur);
                return first;
        }
        pre=first; cur=first;
        while(count != pos && cur→next !=NULL)
        {       pre=cur; count++;
                cur=cur→next;
        }
        if(count!=pos && cur-->next==NULL)
        {
                printf(" position is not found\n");
        }
        if(count == pos)
        {       pre→next=cur→next;
                printf("Element deleted %d from pos  %d\n",pos, first→item);
                free(cur);
        }
        return first;
}
```

## 11 Write a function to traverse the linked list and display alternative nodes.

```
void displayalt()
{
        NODE *temp;
        temp=first;
        if(first==NULL)
        {  printf("empty list\n");  return;
        }
        while(temp!=NULL)
        {
                printf("%d→",temp→item);
```

```
                if(temp→next==NULL) break;
                temp=temp→next→next;
        }

        printf("\ndone\n");
}

int main()
{
        int i,ele;
        printf("enter 4 elements=");
        for(i=0;i<5;i++)
        {
                printf("enter ele=");
                scanf("%d",&ele);
                orderedInsert(ele);
                display();
        }
        printf(" Display alternate elements\n");
        displayalt();
}
```



## 12 Write a function to reverse the linked list.

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

typedef struct node NODE;
NODE *first=NULL;
NODE *revFirst=NULL;
NODE *newfirst=NULL;
void orderedInsert(int ele)
{
        NODE *nn, *pre, *cur;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn-->next=NULL;
        if(first==NULL)
        {
                first=nn; return;
        }
        if(first→item >ele)
        {
```

```
                nn→next=first;
                first=nn; return;
        }


        pre=first; cur=first;
        while(cur→item <=ele && cur→next !=NULL)
        {       pre=cur;
                cur=cur→next;
        }
        if(cur→item >ele)
        {       pre→next=nn;
                nn→next=cur;
        }
        else
        {
                cur→next=nn;
        }
}

void display(NODE *first)
{
        NODE *temp;
        temp=first;
        if(first==NULL)
        {  printf("empty list\n");  return;
        }
        while(temp!=NULL)
        {
                printf("%d→",temp→item);  temp=temp→next;
        }
        printf("\ndone\n");
}

NODE* insertFront(NODE *revFirst,int item)
{
        NODE *nn;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=item;     nn→next=NULL;
        if(revFirst==NULL)
                revFirst=nn;
        else
        {
                nn→next=revFirst;
                revFirst=nn;
        }
        return revFirst;
}

void reverse1()
```

```
{
   struct node *nn,*prev   = NULL;
   struct node *current = first;
   while (current!= NULL)
   {   revFirst=insertFront(revFirst,current→item);
       prev = current;
       current = current→next;
   }
}

NODE* reverse2()
{
   struct node* prev   = NULL;
   struct node* current = first;
   struct node* next;
   while (current != NULL)
   {
      next  = current→next;
      current→next = prev;
      prev = current;
      current = next;
   }
   newfirst = prev;
}


int main()
{
       int i,ele,n;
       printf("enter n="); scanf("%d",&n);
       printf("enter %d elements=",n);
       for(i=0;i<n;i++)
       {
              printf("enter ele=");
              scanf("%d",&ele);
              orderedInsert(ele);
              display(first);
       }
       printf("\n\n----------\n  Display LL elements\n");
       display(first);
       printf("Display Reverse elements\n");
       reverse1();
       display(revFirst);
       printf("Display Reverse using II ,method elements\n");
       reverse2();
       display(newfirst);

}
```

## Single Linked List with Header Node

**13 Write a program to insert at front, rear, in ordered, at specified position, and delete from front, rear, ordered, element and from specified position.**



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

typedef struct node NODE;
NODE *head=NULL;

void insertfront(int data)
{
        NODE* temp;
        temp=(NODE*)malloc(sizeof(NODE));
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return ;
        }
        temp→item=data;
        temp→next=NULL;
        if(head→next==NULL)
        {
                head→next=temp;  head→item++;
        }
        else
        {
                temp→next=head→next;
                head→next=temp; head→item++;
        }
        return;
}



void insertrear(int data)
{
        NODE *temp, *cur;
        temp=(NODE*)malloc(sizeof(NODE));
```

```
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return ;
        }
        temp-->item=data;
        temp→next=NULL;
        if(head→next==NULL)
        {
                head→next=temp;
        }
        else
        {
                cur=head→next;
                while(cur→next!=NULL)
                {
                        cur=cur→next;
                }
                cur→next=temp;
        }
        head→item++;
        return ;
}


void deletefront()
{
        NODE *temp; int num;
        temp=head→next;
        if(head→next==NULL)
        {
                printf("List is Empty"); return;
        }
        printf("Front element deleted =%d\n",temp-->item);
        head→next=temp→next;  head→item--;
        free(temp);
        return ;
}

void  deleterear()
{
        NODE *cur, *prev;
        cur=head→next;
        if(head→next==NULL)
        {       printf("List is Empty"); return;
        }
        if(cur→next==NULL)
        {       printf("Deleted *** rear element=%d\n",cur→item);
                head→next=NULL;   head→item--;
                free(cur); return ;
        }
```

```
        prev=NULL;
        while(cur→next!=NULL)
        {       prev=cur;
                cur=cur→next;
        }
        prev→next=cur→next;   head→item--;
        printf("Deleted Rear element=%d\n",cur→item);
        free(cur);
        return ;
}

void orderedInsert(int ele)
{
        NODE *nn, *pre, *cur, *first;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn→next=NULL;
        first=head→next;
        if(first==NULL)
        {
                head→next=nn;   head→item++; return;
        }
        if(first→item >ele)
        {
                nn→next=first;
                head→next=nn;   head→item++;
                return;
        }
        pre=first; cur=first;
        while(cur→item <=ele && cur→next !=NULL)
        {       pre=cur;
                cur=cur→next;
        }
        if(cur→item >ele)
        {       pre→next=nn;
                nn→next=cur; head→item++;
        }
        else
        {
                cur→next=nn; head→item++;
        }
}

void deleteFromordered(int ele)
{
        NODE *pre, *cur;
        if(head-->next==NULL)
        {
                printf("Empty list\n"); return ;
        }
        cur=head→next;
```

```
        if(cur→item ==ele)
        {
                head→next=NULL;    head→item--;
                printf("Deleted Rear element=%d\n",cur→item);
                free(cur);
                return ;
        }
        pre=NULL;
        while(cur→item !=ele && cur→next !=NULL)
        {       pre=cur;
                cur=cur→next;
        }
        if(cur→item !=ele && cur→next ==NULL)
        {
                printf(" Element not found\n"); return;
        }
        if(cur→item == ele)
        {       pre→next=cur→next;   head→item--;
        }
        return ;
}


void InsertAtPos(int ele, int pos)
{
        NODE  *nn, *pre, *cur; int count=1;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn→next=NULL;
        if(head→next==NULL)
        {
                head→next=nn;  head→item++;         return;
        }
        if(pos==1)
        {
                nn→next=head→next;
                head→next=nn;     head→item++;
                return;
        }

        pre=NULL; cur=head→next;
        while(count < pos && cur→next !=NULL)
        {       pre=cur; count++;
                cur=cur→next;
        }
        if(count >= pos)
        {       pre→next=nn;
                nn→next=cur;
        }
        else
        {
```

```
                cur→next=nn;
        }
        head→item++;
        return ;
}

void DeleteFromPos(int pos)
{
        NODE *pre, *cur; int count=1;
        pre=NULL; cur=head-->next;
        if(head→next==NULL)
        {
                printf("List is Empty"); return ;
        }
        if(pos==1)
        {
                printf("Front element deleted =%d\n",cur-->item);
                head→next=cur→next;  head→item--;
                free(cur);
                return ;
        }
        pre=NULL; cur=head→next;
        while(count != pos && cur→next !=NULL)
        {       pre=cur; count++;
                cur=cur→next;
        }
        if(count != pos && cur→next ==NULL)
        {
                printf("Posotion not valid\n");  return;
        }
        if(count == pos)
        {       pre→next=cur→next;    head-->item--;
                printf("Element deleted %d from pos  %d\n",pos, cur→item);
                free(cur);
        }
        return ;
}


void display()
{
        NODE *temp, *first;
        temp=first=head→next;;
        if(first==NULL)
        {
                printf("empty list\n");  return;
        }
        while(temp!=NULL)
        {
                printf("%d→",temp→item);  temp=temp→next;
```

```
        }
        printf("\ndone total nodes=%d\n", head→item);
}

int main()
{
        int i,ele, val, pos;
        head=(NODE*) malloc(sizeof(NODE));
        head→next=NULL; head-->item=0;
        printf("enter 4 elements=");
        for(i=0;i<4;i++)
        {
                printf("enter ele=");
                scanf("%d",&val);
                insertfront(val);
                //orderedInsert(val);
                display();
        }
        printf("\n item value and position=");
        scanf("%d%d", &val,&pos);
        InsertAtPos(val,pos);
        display();
        printf("\n Enter value to delete =");
        scanf("%d", &val);
        deleteFromordered(val);
        display();
        printf("\n Enter posotion from which ele to delete =");
        scanf("%d", &pos);
        DeleteFromPos(pos);
        display();
}
```



## 14 Write a program to insert, delete from both ends and insert & delete from Ordered using circular single linked list.

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

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

typedef struct node NODE;

NODE* insertfront(NODE* first, int data)
```

```
{
        NODE* temp, *cur;
        temp=(NODE*)malloc(sizeof(NODE));
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return NULL;
        }
        Temp→item=data;
        Temp→next=NULL;
        if(first==NULL)
        {
                first=temp; first→next=first;
        }
        else
        {       cur=first;
                while(cur→next!=first)
                {
                        cur=cur→next;
                }
                temp→next=first;
                first=temp;
                cur→next=first;
        }
        return first;
}

NODE* insertrear(NODE* first, int data)
{
        NODE *temp, *cur;
        temp=(NODE*)malloc(sizeof(NODE));
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return NULL;
        }
        temp→item=data;
        temp→next=NULL;
        if(first==NULL)
        {       first=temp; first→next=first;
        }
        else
        {
                cur=first;
                while(cur→next!=first)
                {
                        cur=cur→next;
                }
                cur→next=temp; temp→next=first;
        }
        return first;
}
```

```
void display(NODE* first)
{       NODE *cur;
        if(first==NULL)
        {
                printf(" Empty List\n"); return;
        }
        cur=first;
        while(cur-->next!=first)
        {
                printf("%d-→", cur→item);
                cur=cur→next;
        }
        printf("%d-→", cur→item);
        printf("NULL\n\n");

}

NODE* deletefront(NODE* first)
{
   NODE *temp, *last; int num;
   temp=first;
   if(first==NULL)
   {
       printf("List is Empty"); return first;
       }
       printf("Front element deleted =%d\n",first→item);
       last=first;
       while(last→next!=first)
       {       last=last→next;
       }
       first=first→next;
       last→next=first;

   free(temp);
   return first;
}

NODE*  deleterear(NODE* first)
{
   NODE *cur, *prev;
   cur=first;
   if(first==NULL)
   {
       printf("List is Empty"); return first;
       }
       if(first→next==NULL)
   {   printf("Deleted *** rear element=%d\n",first→item);
               first=NULL;
               free(cur); return first;
```

```
        }
        prev=NULL;
        while(cur→next!=first)
    {   prev=cur;
            cur=cur→next;
    }
    prev→next=cur→next;
    printf("Deleted Rear element=%d\n",cur→item);
    free(cur);
    return first;
}

NODE* orderedInsert(NODE* first, int ele)
{
    NODE *nn, *pre, *cur, *last;
    nn=(NODE*)malloc(sizeof(NODE));
    nn→item=ele; nn→next=NULL;
    if(first==NULL)
    {
        first=nn; first→next=first;  first;
    }
    if(first→item >ele)
    {
        nn→next=first;
        last=first;
        while(last→next!=first)
        {       last=last→next;
        }
        first=nn;
        last→next=first;
        return first;
    }

    pre=first; cur=first;
    while(cur→item <=ele && cur→next !=NULL)
    {       pre=cur;
        cur=cur→next;
    }
    if(cur→item >ele)
    {       pre→next=nn;
        nn→next=cur;
    }
    else
    {
        cur→next=nn;   nn→next=first;
    }
    return first;
}

NODE* deleteFromordered(NODE* first, int ele)
```

```
{
        NODE *pre, *cur,*last;
        if(first==NULL)
        {
                printf("Empty list\n"); return first;
        }
        cur=first;
        if(first→item ==ele)
        {
                first=first→next;
                printf("Deleted Rear element=%d\n",cur→item);
                last=first;
                while(last→next!=first)
                {       last=last→next;
                }
                last→next=first;
                free(cur);
                return first;
        }
        pre=first; cur=first;
        while(cur→item !=ele && cur→next !=first)
        {       pre=cur;
                cur=cur→next;
        }
        if(cur→item !=ele && cur→next ==first)
        {
                printf("Element not found\n"); return first;
        }
        pre→next=cur→next; free(cur);
        return first;
}


NODE* InsertAtPos(NODE* first, int ele, int pos)
{
        NODE *nn, *pre, *cur, *last; int count=1;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn→next=NULL;
        if(first==NULL)
        {
                first=nn; first→next=first;  return first;
        }
        if(pos==1)
        {
                nn→next=first;
                last=first;
                while(last→next!=first)
                {       last=last→next;
                }
                first=nn;    last→next=first;
```

```
                return first;
        }
        pre=NULL; cur=first;
        while(count < pos && cur→next !=NULL)
        {       pre=cur; count++;
                cur=cur→next;
        }
        if(count >= pos)
        {       pre→next=nn;
                nn→next=cur;
        }
        else
        {
                cur→next=nn;   nn→next=first;
        }
        return first;
}

NODE* DeleteFromPos(NODE* first, int pos)
{
        NODE *nn, *pre, *cur, *last; int count=1;
        pre=NULL; cur=first;
        if(first==NULL)
        {
                printf("List is Empty"); return first;
        }
        if(pos==1)
        {
                printf("Front element deleted =%d\n",first→item);
                last=first;
                while(last→next!=first)
                {       last=last→next;
                }
                first=first→next; last→next=first;
                free(cur);
                return first; ;
        }
        pre=NULL; cur=first;
        while(count != pos && cur→next !=NULL)
        {       pre=cur; count++;
                cur=cur→next;
        }
        if(count!=pos && cur→next==NULL)
        {
                printf(" position is not found\n");
        }
        if(count == pos)
        {       pre→next=cur→next;
                printf("Element deleted %d from pos  %d\n",pos, first→item);
                free(cur);
```

```
        }
        return first;
}


int main()
{   int val, n,i,pos;
        NODE *first=NULL;
        printf(" Enter n of nodes n=");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n item value=");
                scanf("%d", &val);
                first=insertrear(first,val);
                //first=insertfront(first,val);
                //first=orderedInsert(first,val);
                        display(first);
        }

        printf("\n Linked List is \n");
        display(first);
        first=deletefront(first);
        display(first);
        first=deleterear(first);
        display(first);
        printf("\n item value and position=");
        scanf("%d%d", &val,&pos);
        first=InsertAtPos(first,val,pos);
        display(first);
        printf("\n Enter value to delete =");
        scanf("%d", &val);
        first=deleteFromordered(first, val);
        display(first);
        printf("\n Enter posotion from which ele to delete =");
        scanf("%d", &pos);
        first=DeleteFromPos(first, pos);
        display(first);
        return 0;
}
```



## 15 Write a program to insert and delete from circular single linked list with head node.

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

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

typedef struct node NODE;
NODE *head=NULL;

void insertfront(int data)
{
        NODE* temp, *last;
        temp=(NODE*)malloc(sizeof(NODE));
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return ;
        }
        temp→item=data;
        temp→next=NULL;
        if(head→next==NULL)
        {       printf(" First element\n\n");
                head→next=temp;  temp→next=temp; head→item++;
        }
        else
        {
                temp→next=head→next;
                last=head→next;
                while(last→next!=head→next)
                {
                        last=last→next;
                }
                head→next=temp;
                last→next=head→next; head→item++;
        }
        return;
}

void insertrear(int data)
{
        NODE *temp, *cur;
        temp=(NODE*)malloc(sizeof(NODE));
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return ;
        }
        temp→item=data;
        temp→next=NULL;
        if(head→next==NULL)
        {
                head→next=temp;  temp→next=temp;
```

```
        }
        else
        {
                cur=head→next;
                while(cur→next!=head→next)
                {
                        cur=cur→next;
                }
                cur→next=temp;  temp→next=head→next;
        }
        head→item++;
        return ;
}


void deletefront()
{
   NODE *temp, *last; int num;
   temp=head-->next;
   if(head→next==NULL)
   {
       printf("List is Empty"); return;
       }
       printf("Front element deleted =%d\n",temp→item);

       last=head→next;
       while(last→next!=head→next)
       {
                       last=last→next;
       }
       head→next=temp→next;
       last→next=head→next;
       head-->item--;
   free(temp);
   return ;
}

void  deleterear()
{
   NODE *cur, *prev;
   cur=head-->next;
   if(head-->next==NULL)
   {
       printf("List is Empty"); return;
       }
       if(cur→next==NULL)
       {       printf("Deleted *** rear element=%d\n",cur→item);
               head→next=NULL;   head→item--;
               free(cur); return ;
       }
```

```
        prev=NULL;
        while(cur→next!=NULL)
         {      prev=cur;
                cur=cur→next;
        }
        prev→next=cur→next;   head→item--;
        printf("Deleted Rear element=%d\n",cur→item);
        free(cur);
        return ;
}

void orderedInsert(int ele)
{
        NODE *nn, *pre, *cur, *last;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn→next=NULL;
        cur=head→next;
        if(head→next==NULL)
        {
                head→next=nn; nn→next=nn;  head→item++;  return;
        }
        if(cur→item >ele)
        {
                nn→next=cur;

                last=cur;
                while(last→next!=head-->next)
                {
                        last=last→next;
                }
                head→next=nn; last→next=head→next;
                head→item++;
                return;
        }
        pre=NULL; cur=head→next;
        while(cur→item <=ele && cur→next !=head→next)
        {       pre=cur;
                cur=cur→next;
        }
        if(cur→item >ele)
        {       pre→next=nn;
                nn→next=cur; head→item++;
        }
        else
        {
                cur→next=nn;  nn→next=head→next;   head→item++;
        }
}

void deleteFromordered(int ele)
```

```
{
        NODE *pre, *cur, *last;
        if(head→next==NULL)
        {
                printf("Empty list\n"); return ;
        }
        cur=head→next;
        if(cur→item ==ele)
        {   if(cur→next==head→next)
                {       head→next=NULL;
                }
                else
                {
                        last=head→next;
                        while(last→next!=head→next)
                        {
                                last=last→next;
                        }
                        head→next=cur→next;
                        last→next=head→next;
                }
                printf("Deleted Rear element=%d\n",cur→item);
                free(cur);
                head→item--;
                return ;
        }
        pre=NULL;
        while(cur→item !=ele && cur→next !=head→next)
        {       pre=cur;
                cur=cur→next;
        }
        if(cur→item !=ele && cur→next ==NULL)
        {
                printf(" Element not found\n"); return;
        }
        if(cur→item == ele)
        {       pre→next=cur→next;   free(cur); head→item--;
        }
        return ;
}

void InsertAtPos(int ele, int pos)
{
        NODE  *nn, *pre, *cur, *last; int count=1;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn→next=NULL;
        if(head→next==NULL)
        {
                head→next=nn;  nn→next=nn;  head→item++;       return;
        }
```

```
        if(pos==1)
        {
                nn→next=head→next;
                last=head→next;
                while(last→next!=head→next)
                {
                        last=last→next;
                }
                head→next=nn;
                last→next=head→next;   head→item++;
                return;
        }

        pre=NULL; cur=head→next;
        while(count < pos && cur→next !=head→next)
        {       pre=cur; count++;
                cur=cur→next;
        }
        if(count >= pos)
        {       pre→next=nn;
                nn→next=cur;
        }
        else
        {
                cur→next=nn;    nn→next=head→next;
        }
        head→item++;
        return ;
}

void DeleteFromPos(int pos)
{
        NODE *pre, *cur, *last; int count=1;
        pre=NULL; cur=head→next;
        if(head→next==NULL)
   {
        printf("List is Empty"); return ;
        }
        if(pos==1)
        {
                printf("Front element deleted =%d\n",cur→item);
                last=head→next;
                while(last→next!=head→next)
                {
                        last=last→next;
                }
                head→next=cur→next;
                last→next=head→next;   head→item--;
                free(cur);
                return ;
```

```
        }
        pre=NULL; cur=head→next;
        while(count != pos && cur→next !=NULL)
        {       pre=cur; count++;
                cur=cur→next;
        }
        if(count != pos && cur→next ==NULL)
        {
                printf("Posotion not valid\n");  return;
        }
        if(count == pos)
        {       pre→next=cur→next;    head→item--;
                printf("Element deleted %d from pos  %d\n",pos, cur→item);
                free(cur);
        }
        return ;
}


void display()
{
        NODE *temp;
        temp=head→next;
        if(head→next==NULL)
        {       printf("empty list\n");  return;
        }
        while(temp→next!=head→next)
        {
                printf("%d→",temp→item);  temp=temp→next;
        }
        printf("%d→",temp→item);
        printf("\n \nDone total nodes=%d\n", head→item);
}

int main()
{
        int i,ele, val, pos,n;
        head=(NODE*) malloc(sizeof(NODE));
        head→next=NULL; head→item=0;
        printf("enter n=");
        scanf("%d",&n);
        printf("enter %d elements=",n);
        for(i=0;i<n;i++)
        {
                printf("enter ele=");
                scanf("%d",&val);
                //insertrear(val);
                //insertfront(val);
                orderedInsert(val);
                display();
```

```
        }
        printf("\n item value and position=");
        scanf("%d%d", &val,&pos);
        InsertAtPos(val,pos);
        display();
        printf("\n Enter value to delete =");
        scanf("%d", &val);
        deleteFromordered(val);
        display();
        printf("\n Enter posotion from which ele to delete =");
        scanf("%d", &pos);
        DeleteFromPos(pos);
        display();
}
```



## 16 Write a program to insert & delete from both ends with first & last pointer.

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

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

typedef struct node NODE;
void insertfront(NODE** first, NODE **last, int data)
{
        NODE* temp;
        temp=(NODE*)malloc(sizeof(NODE));
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return ;
        }
        Temp→item=data;
        Temp→next=NULL;
        if(*first==NULL)
        {
                *first=temp; *last=temp;
        }
        else
        {
                temp→next=*first;
                *first=temp;
```

```
        }
        return;
}

void insertrear(NODE** first, NODE **last, int data)
{
        NODE *temp, *cur;
        temp=(NODE*)malloc(sizeof(NODE));
        if(temp==NULL)
        {
                printf("insufficient memory\n");  return ;
        }
        temp→item=data;
        temp→next=NULL;
        if(*first==NULL)
        {
                *first=temp; *last=temp;
        }
        else
        {
                cur=*last;
                cur→next=temp;
                *last=temp;

        }
        return ;
}

void display(NODE* first)
{       NODE *cur;
        if(first==NULL)
        {
                printf(" Empty List\n"); return;
        }
        cur=first;
        while(cur!=NULL)
        {
                printf("%d→", cur->item);
                cur=cur→next;
        }
        printf("NULL\n\n");
}

void deletefront(NODE** first, NODE **last)
{
        NODE *temp; int num;
```

```
        temp=*first;
        if(*first==NULL)
        {
                printf("List is Empty"); return ;
        }
        printf("Front element deleted =%d\n",temp→item);
        *first=temp→next;
        free(temp);
        return ;
}

NODE*  deleterear(NODE** first, NODE **last)
{
        NODE *cur, *prev;
        cur=*first;
        if(*first==NULL)
        {
                printf("List is Empty"); return;
        }
        if(cur→:next==NULL)
        {       printf("Deleted *** rear element=%d\n",cur→item);
                *first=NULL;
                free(cur); return ;
        }
        prev=NULL;
        while(cur→next!=NULL)
        {       prev=cur;
                cur=cur→next;
        }
        Prev→next=cur→next;
        printf("Deleted Rear element=%d\n",cur→item);
        free(cur);
        return;
}

NODE* orderedInsert(NODE** first, NODE **last, int ele)
{
        NODE *nn, *pre, *cur;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn→next=NULL;
        cur=*first;
        if(*first==NULL)
        {
                *first=nn; return ;
        }
        if(cur→item >ele)
```

```
        {
                nn→next=*first;
                *first=nn;
                return ;
        }

        pre=NULL; cur=*first;
        while(cur→item <=ele && cur→next !=NULL)
        {       pre=cur;
                cur=cur→next;
        }
        if(cur→item >ele)
        {       pre→next=nn;
                nn→next=cur;
        }
        else
        {
                cur→next=nn;
        }
        return ;
}


int main()
{  int val, n,i,pos;
        NODE *first=NULL, *last=NULL;
        printf(" Enter n of nodes n=");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n item value=");
                scanf("%d", &val);
                //first=insertfront(first,val);
                orderedInsert(&first,&last,val);
                display(first);
        }
        printf("\n Linked List is \n");
        display(first);
        deletefront(&first,&last);
        display(first);
        deleterear(&first,&last);
        display(first);
        return 0;
}
```

## Doubly Linked List

## 17 Write a progrm to insert, delete from both ends and insert into ordered DLL and delete from Ordered DLL.



```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct node
{       int  item;
        struct node *next;
        struct node *prev;
};
typedef struct node NODE;
NODE *first=NULL;

void insertrear(int ele)
{       NODE *temp,*cur, *prev;
        temp= (NODE *)malloc(sizeof(NODE));
        temp→item=ele; temp→next=NULL;
        if(first==NULL)
        {       first=temp; return;
        }
        cur=first; prev=NULL;
        while(cur→next != NULL)
        {       prev=cur;
                cur=cur→next;
        }
        cur→next =temp;
        tem→prev=prev;
        return ;

}


void insertfront(int ele)
{       NODE *temp;
        temp= (NODE *)malloc(sizeof(NODE));
        temp→item=ele;  temp→next=NULL;
        if (first== NULL)
        {
                first=temp;
        }
        else
        {       temp→next=first;
                first→prev=temp;
                first=temp;
        }
        return ;
}
```

```
void deletefront()
{
        NODE *temp; int num;
        temp=first;
        if(first==NULL)
        {
                printf(" List is Empty \n");return ;
        }
        if(first→next==NULL)
                   first=NULL;
        else
        {       first=first→next;
                First→prev=NULL;
        }
        printf(" Element=%d\n", temp→item);
        free(temp);
        return ;
}

void deleterear()
{
        NODE *cur, *prev;
        cur=first;
        prev=NULL;
        if(first==NULL)
        {
                printf(" List is Empty \n");return ;
        }
        if(first→next==NULL)
        {
                first=NULL;
        }
        else
        {
                while(cur→next!=NULL)
                {       prev=cur;
                        cur=cur→next;
                }
                prev→next=NULL;
        }
        printf(" Element=%d\n", cur→item);
        free(cur);
        return;
}

void  display()
{       NODE *r;  int count=0;
        r=first;
```

```
        if(r==NULL)
        {
                return;
        }
        printf("Doubly Linked List\n");
        while(r!=NULL)
        {       printf("%d-→",r->item);
                r=r→next;  count++;
        }
        printf("\n   No. Of Nodes=%d\n",count);
}

void insertOrdered(int ele)
{
        NODE *nn, *pre, *cur;
        nn=(NODE*)malloc(sizeof(NODE));
        nn→item=ele; nn→next=NULL;  nn→prev=NULL;
        if(first==NULL)
        {
                first=nn; return;
        }
        if(first→item >ele)
        {
                nn→next=first;  first→prev=nn;
                first=nn; return;
        }
        pre=NULL; cur=first;
        while(cur→item <=ele && cur→next !=NULL)
        {       pre=cur;
                cur=cur→next;
        }
        if(cur→item >ele)
        {       pre→next=nn; nn→prev=pre;
                nn→next=cur;  cur→prev=nn;
        }
        else
        {
                cur→next=nn; nn→prev=cur;
        }
}

void deleteOrdered(int ele)
{
        NODE *pre, *cur;
        if(first==NULL)
        {
                printf("Empty list\n"); return first;
        }
        cur=first;
        if(first→item ==ele)
```

```
        {
                first=first→next;   if(first!=NULL) first→prev=NULL;
                printf("Deleted Rear element=%d\n",cur→item);
                free(cur);
                return first;
        }
        pre=NULL; cur=first;
        while(cur→item !=ele && cur→next !=NULL)
        {       pre=cur;
                cur=cur→next;
        }
        if(cur→item !=ele && cur→next ==NULL)
        {
                printf("Element not found\n"); return first;
        }
        pre→next=cur→next;
        return ;
}

int  main()
{  int ele;   int i,  ch;
   first=NULL;
   while(1)
   {
       printf("\nList Operations\n");
       printf("===============\n");
       printf("1.Insert at Front end\n");
       printf("2.Deletion from the Front end\n");
       printf("3.Insertion at the Rear end of DLL\n");
       printf("4.Deletion from the rear end End of DLL");
       printf("5.Display DLL\n");
       printf("6.Perform Insertion into Ordered DLL\n");
       printf("7.Deletion from the Ordered DLL\n");
       printf("8.Exit\n");
       printf("Enter your choice : ");
       scanf("%d",&i);
       switch(i)
       {
              case 1       :      printf("Enter Item="); scanf("%d",&ele);
                                  insertfront(ele);  break;
              case 2       :      deletefront(); break;
              case 3       :      printf("Enter Item="); scanf("%d",&ele);
                                  insertrear(ele);  break;
              case 4       :      deleterear(); break;
              case 5       :      if(first==NULL)
                                      printf("List is Empty\n");
                                  else
                                      display();
                                  break;
```

```
                        case 6      :   printf("Enter Item=");
                                        scanf("%d",&ele);
                                        insertOrdered(ele);  break;
                        case 7      :   printf("Enter Item to delete=");
                                        scanf("%d",&ele);
                                        deleteOrdered(ele); break;
                        case 8      :   return 0;
                        default     :   printf("Invalid option\n");
                }
        }
        return 0;
}
```



## 18 Write a program to insert, delete from both ends and insert into ordered DLL and delete from Ordered DLL with header node, and last pointer.

```
/* Doubly Linked List (DLL) with first and last pointer  */

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

struct node
{       int  item;
        struct node *next;
        struct node *prev;
};
typedef struct node NODE;
NODE *head=NULL, *last=NULL;

void insertrear(int ele)
{       NODE *temp,*cur, *prev;
        temp= (NODE *)malloc(sizeof(NODE));
        temp->item=ele; temp->next=NULL;  temp->prev=NULL;
        if(head->next==NULL)
        {   head->next=temp; last=temp;return;
        }
        last->next =temp;  temp->prev=last;
        last=temp;
        return ;
}

void insertfront(int ele)
{       NODE *temp;
        temp= (NODE *)malloc(sizeof(NODE));
        temp->item=ele;  temp->next=NULL; temp->prev=NULL;
```

```
        if (head->next== NULL)
        {
                head->next=temp; last=temp;
        }
        else
        {       temp->next=head->next;
                head->next->prev=temp;
                head->next=temp;
        }
        return ;
}


void deletefront()
{
         NODE *temp; int num;
         temp=head->next;
         if(head->next==NULL)
         {
              printf(" List is Empty \n");return ;
         }
         if(head->next==last)
         {
              head->next=last=NULL;
         }
        else
        {     head->next=temp->next;
              temp->prev=NULL;
        }
        printf(" Element=%d\n", temp->item);
        free(temp);
        return ;
}

void deleterear()
{
        NODE *cur, *pre;
        cur=last;
        if(head->next==NULL)
        {
                printf(" List is Empty \n");return ;
        }
        printf(" Element=%d\n", last->item);
        if(head->next==last)
        {
              head->next=last=NULL;
        }
        else
        {
              pre=last->prev;
```

```
                pre->next=NULL; last=pre;
            }
            free(cur);
            return;
}

void  display()
{       NODE *r;  int count=0;
        r=head->next;
        if(r==NULL)
        {
                return;
        }
        printf("Doubly Linked List\n");
        while(r!=NULL)
        {       printf("%d-->",r->item);
                r=r->next;  count++;
        }
        printf("\n   No. Of Nodes=%d\n",count);
}

void insertOrdered(int ele)
{
        NODE *nn, *pre, *cur;
        nn=(NODE*)malloc(sizeof(NODE));
        nn->item=ele; nn->next=NULL;  nn->prev=NULL;
        if(head->next==NULL)
        {
                head->next=last=nn;   return;
        }
        if(head->next->item >ele)
        {
                nn->next=head->next;  head->next->prev=nn;
                head->next=nn; return;
        }
        pre=NULL; cur=head->next;
        while(cur->item <=ele && cur->next !=NULL)
        {   pre=cur;
                cur=cur->next;
        }
        if(cur->item >ele)
        {       pre->next=nn; nn->prev=pre;
                nn->next=cur;  cur->prev=nn;
        }
        else
        {
                cur->next=nn; nn->prev=cur; last=nn;
        }
}
```

```
void deleteOrdered(int ele)
{
        NODE *pre, *cur;
        if(head->next==NULL)
        {
                printf("Empty list\n"); return;
        }
        cur=head->next;
        if(cur->item ==ele)
        {
                head->next=cur->next;
                if(head->next!=NULL)
                        head->next->prev=NULL;
                printf("Deleted Rear element=%d\n",cur->item);
                free(cur);
                return ;
        }
        pre=NULL;
        cur=head->next;
        while(cur->item !=ele && cur->next !=NULL)
        {       pre=cur;
                cur=cur->next;
        }
        if(cur->item !=ele && cur->next ==NULL)
        {
                printf("Element not found\n"); return ;
        }
        pre->next=cur->next;
        return ;
}

int  main()
{  int ele;   int i,  ch;
   head= (NODE *)malloc(sizeof(NODE));
   head->item=0; head->next=NULL;  head->prev=NULL;
   while(1)
   {
       printf("\nList Operations\n");
       printf("===============\n");
       printf("1.Insert at Front end\n");
       printf("2.Deletion from the Front end\n");
       printf("3.Insertion at the Rear end of DLL\n");
       printf("4.Deletion from the rear end End of DLL\n");
       printf("5.Display DLL\n");
       printf("6.Perform Insertion into Ordered DLL\n");
       printf("7.Deletion from the Ordered DLL\n");
       printf("8.Exit\n");
       printf("Enter your choice : ");
       scanf("%d",&i);
```

```
        switch(i)
        {
                case 1      :   printf("Enter Item="); scanf("%d",&ele);
                                insertfront(ele);  break;
                case 2      :   deletefront(); break;
                case 3      :   printf("Enter Item="); scanf("%d",&ele);
                                insertrear(ele);  break;
                case 4      :   deleterear(); break;
                case 5      :   if(head->next==NULL)
                                printf("List is Empty\n");
                                else
                                display();
                                break;

                case 6      :   printf("Enter Item="); scanf("%d",&ele);
                                insertOrdered(ele);  break;
                case 7      :   printf("Enter Item to delete=");
                                scanf("%d",&ele);
                                deleteOrdered(ele); break;
                case 8      :   return 0;
                default     :   printf("Invalid option\n");
                }
        }
    return 0;
    }
```



## 19 Write a program to create linked to represent polynomial equation and evaluate the equation with one variable($x7 + 8x - 43$).

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

struct node
{
    int coef, pwr;
    struct node *next;
};

typedef struct node NODE;

NODE* insertrear(NODE *first, int c, int p)
{
    NODE *temp, *cur;
    temp=(NODE*)malloc(sizeof(NODE));
    if(temp==NULL)
```

```
        {
                printf("insufficient memory\n");  return  NULL;
        }
        temp->coef=c; temp->pwr=p;
        temp->next=NULL;
        if(first==NULL)
                first=temp;
        else
        {
                cur=first;
                while(cur->next!=NULL)
                {
                        cur=cur->next;
                }
                cur->next=temp;
        }
        return first;
}

void display(NODE *first)
{   NODE *cur;
    if(first==NULL)
    {
            printf(" Empty List\n"); return ;
    }
    cur=first;
    while(cur!=NULL)
    {
            printf("%dx^%d + ", cur->coef, cur->pwr);
            cur=cur->next;
    }
    printf("NULL\n\n");
}

void evaluate(NODE *third, int x)
{
    NODE *cur;
    int s=0;
    cur=third;
    if(third==NULL)
    {
            printf(" empty list\n");
            return;
    }
    while(cur!=NULL)
    {
```

```
            s=s+cur->coef*pow(x, cur->pwr);
            cur=cur->next;
        }
        printf("Value=%d\n",s);
        return;
    }

    int main()
    {   int c, p, x,ln1,ln2,i;
        NODE *first=NULL;
        NODE *second=NULL;
        NODE *third=NULL;
        printf(" Enter no of nodes  for first List=");
        scanf("%d", &ln1);
        for(i=0;i<ln1;i++)
        {
            printf("\n Enter Coef & power=");
            scanf("%d%d", &c,&p);
            first=insertrear(first,c,p);
            //display(first);

        }
        display(first);
        printf(" Enter no of nodes  for second List=");
        scanf("%d", &ln2);
        for(i=0;i<ln2;i++)
        {
            printf("\n Enter Coef & power=");
            scanf("%d%d", &c,&p);
            second=insertrear(second,c,p);
            //display(second);
        }
        printf("\n Polynomial addition using Linked List is \n");
        third=polyadd(first, second,third);
        display(third);
        printf("Enter the value of x=");
        scanf("%d",&x);
        evaluate(third,x);

        return 0;
    }
```

**20 Write a program to create linked to represent polynomial equation and add two polynomial the equation with one variable P1: 2x7 + 83x +4x+5 & p2: 4x7 +5x2+18x +3.**



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

struct node
{
    int coef, pwr;
    struct node *next;
};

typedef struct node NODE;

NODE* insertrear(NODE *first, int c, int p)
{
    NODE *temp, *cur;
    temp=(NODE*)malloc(sizeof(NODE));
    if(temp==NULL)
    {
            printf("insufficient memory\n");  return  NULL;
    }
    temp->coef=c; temp->pwr=p;
    temp->next=NULL;
    if(first==NULL)
            first=temp;
    else
    {
            cur=first;
            while(cur->next!=NULL)
            {
                    cur=cur->next;
            }
            cur->next=temp;
    }
    return first;
}

void display(NODE *first)
{   NODE *cur;
    if(first==NULL)
    {
            printf(" Empty List\n"); return ;
    }
    cur=first;
```

```
    while(cur!=NULL)
    {
        printf("%dx^%d + ", cur->coef, cur->pwr);
        cur=cur->next;
    }
    printf("NULL\n\n");
}

NODE* polyadd(NODE *first, NODE *second, NODE *third)
{
    NODE *nn, *cur1, *cur2;
    int sum;
    nn=(NODE*)malloc(sizeof(NODE));
    cur1=first; cur2=second;
    if(cur1==NULL && cur2==NULL)
        return NULL;
    if(cur1==NULL)  return cur2;
    if(cur2==NULL)  return cur1;
    while(cur1!=NULL && cur2!=NULL)
    {
        if(cur1->pwr >cur2->pwr)
        {   third=insertrear(third, cur1->coef, cur1->pwr);
            display(third);
            cur1=cur1->next;
        }
        else
        if(cur1->pwr==cur2->pwr)
        {   sum=cur1->coef+cur2->coef;
            third=insertrear(third, sum, cur1->pwr);
            display(third);
            cur1=cur1->next;cur2=cur2->next;
        }
        else
        {   third=insertrear(third, cur2->coef, cur2->pwr);
            display(third);
            cur2=cur2->next;
        }
    }
    if(cur1==NULL)
    {
        while(cur2!=NULL)
        {
            third=insertrear(third, cur2->coef, cur2->pwr);
            cur2=cur2->next;
        }
```

```
        }
        else
        if(cur2==NULL)
        {
                while(cur1!=NULL)
                {
                        third=insertrear(third, cur1->coef, cur1->pwr);
                        cur1=cur1->next;
                }
        }
        return third;
}

int main()
{  int c, p, x,ln1,ln2,i;
        NODE *first=NULL;
        NODE *second=NULL;
        NODE *third=NULL;
        printf(" Enter no of nodes  for first List=");
        scanf("%d", &ln1);
        for(i=0;i<ln1;i++)
        {
                printf("\n Enter Coef & power=");
                scanf("%d%d", &c,&p);
                first=insertrear(first,c,p);
                //display(first);

        }
        display(first);
        printf(" Enter no of nodes  for second List=");
        scanf("%d", &ln2);
        for(i=0;i<ln2;i++)
        {       printf("\n Enter Coef & power=");
                scanf("%d%d", &c,&p);
                second=insertrear(second,c,p);
                //display(second);
        }
        printf("\n Polynomial addition using Linked List is \n");
        third=polyadd(first, second,third);
        display(third);
        return 0;
 }
```

## Questions on Linked List

What is list? How list can be represented in memory.

What is Linked List? Explain different types of Linked Lists.

What are the difference between linked list and arrays?

Mention the advantages and disadvantages of arrays and linked list.



Explain dynamic memory allocation techniques which are used to allocate memory.

What is heap?

Why do you need double linked lists?

Why do you want circular linked list and header linked lists.