UNIT-III
Linked List

We have some drawbacks of using sequential storage to represent stacks and queues.

1. Fixed amount of storage.
2. In one array represent at most two stacks and only one queue.
3. In sequential representation, the items of a stack or queue are implicitly ordered by the sequential order of storage.

Singly Linked List

Array representation of list:

A list is simply a collection of nodes.

```
#define max 20
struct node
{
        int info, next;
};
struct node a[max];
```

index		info	next
	0	26	12
	1	8	3
	2	4	18
	3	76	14
List-2	4	12	6
	5	46	0
	6	7	17
	7	22	13
List-1	8	11	2
	9	6	-1
	10		
	11		
	12	28	-1
	13	98	19
	14	42	9
	15		
	16		
	17	32	1

18	65	7
19	55	5

List-1: 11 → 4 → 65 →22 →98 →55 →46 →26 →28

List-2: 12 →7 →32 →8 →76 →42 →6

Dis-advantages:

1. The number of nodes that are needed often cannot be predicated when a program is written.

Linked list representation using dynamic memory allocation:

List is a collection of nodes. The structure of node is

```
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr p;
```

getnode():

getnode is a function allocate memory for node dynamically and initialize the fields of node and return address of node.

```
nodeptr getnode()
{
        nodeptr p;
        p = (nodeptr) malloc(sizeof(struct node));
        p→info = 0;
        p→next = NULL;
        return p;
}
```

Operations on singly linked list:

1. **Create a singly linked list**
2. **insertion**
3. **deletion**
4. **searching**
5. **traversal**
6. **reversing singly linked list**

create a singly linked list:

```
nodeptr create()
{

      nodeptr p,p1,p2;
      p1=getnode();
      p=p1;
      p2=getnode();
      printf("\nEnter the at end -999\n");
      printf("\nEnter the number:");
      scanf("%d",&p2->info);
      while(p2->info!=-999)
      {
                  p1->next=p2;
                  p1=p2;
                  p2=getnode();
                  printf("\nEnter the number:");
                  scanf("%d",&p2->info);
      }
          return p->next;
}
```

Traversal:

Traversing in the list and print the data in the list.

```
void traversal(nodeptr p)
{
          nodeptr p1;
          p1=p;
          printf("\nelements are:\n");
          while(p1!=NULL)
          {
                  printf("%d-->",p1->info);
                  p1=p1->next;
          }
}
```

/* Create a singly linked list Program */

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

```
typedef struct node *nodeptr;
nodeptr getnode();
void traversal(nodeptr);
nodeptr create();
main()
{
        nodeptr first;
        clrscr();
        first=create();
        traversal(first);
        getch();
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=0;
        return p;
}
void traversal(nodeptr p)
{
        nodeptr p1;
        p1=p;
        printf("\nelements are:\n");
        while(p1!=NULL)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
}
nodeptr create()
{

    nodeptr p,p1,p2;
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
```

```
                p1->next=p2;
                p1=p2;
                p2=getnode();
                printf("\nEnter the number:");
                scanf("%d",&p2->info);
        }
            return p->next;
}
```

Insertion:

insert at 1. Beginning 2. End 3. Given position

```
nodeptr insert(nodeptr p)
{
        nodeptr temp,temp1;
        int ch,pos,i;
        temp=getnode();
        temp1=p;
        printf("\nenter the insert number:");
        scanf("%d",&temp->info);
        printf("\n1.at begining\n2. at given position\n3. at end");
        printf("\nenter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: temp->next=temp1;
                        return temp;
                case 2: printf("\nEnter the position to insert:");
                        scanf("%d",&pos);
                        if(pos==1)
                        {
                                temp->next=temp1;
                                p=temp;
                        }
                        else
                        {
                                for(i=1;i<pos-1;i++)
                                        temp1=temp1->next;
                                temp->next=temp1->next;
                                temp1->next=temp;
                        }
                        break;
                case 3: while(temp1->next!=NULL)
                                temp1=temp1->next;
```

```
                        temp1->next=temp;
                        break;

        }
        return p;
}
```

/* INSET OPERATIONS in Singly Linked List Program */

```
/* Insert operations on Singly LINKED LIST */
#include<stdio.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode();
nodeptr create();
nodeptr insert(nodeptr);
void display(nodeptr);
main()
{
        int ch;
        nodeptr start;
        clrscr();
        start=create();
        display(start);
        while(1)
        {
                printf("\n**************\nMENU\n***********\n");
                printf("\n1.insert\n2.display\n3.exit\n");
                printf("\nenter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: start=insert(start);
                                display(start);
                                break;
                        case 2: display(start);
                                break;
                        case 3: exit(0);
                }
        }
```

```
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=NULL;
        return p;
}
nodeptr create()
{

    nodeptr temp1,temp2,temp;
    temp1=getnode();
    temp2=getnode();
    temp=temp1;
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&temp2->info);
    while(temp2->info!=-999)
    {
            temp1->next=temp2;
            temp1=temp2;
            temp2=getnode();
            printf("\nEnter the number:");
            scanf("%d",&temp2->info);
    }
    temp2=temp;
    temp=temp->next;
    temp2->next=NULL;
    return temp;
}
void display(nodeptr p)
{
        nodeptr temp;
        temp=p;
        printf("\nelements are:");
        while(temp!=NULL)
        {
            printf("%d-->",temp->info);
            temp=temp->next;
        }
}
```

```
nodeptr insert(nodeptr p)
{
        nodeptr temp,temp1;
        int ch,pos,i;
        temp=getnode();
        temp1=p;
        printf("\nenter the insert number:");
        scanf("%d",&temp->info);
        printf("\n1.at begining\n2. at given position\n3. at end");
        printf("\nenter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: temp->next=temp1;
                        return temp;
                case 2: printf("\nEnter the position to insert:");
                        scanf("%d",&pos);
                        if(pos==1)
                        {
                                temp->next=temp1;
                                p=temp;
                        }
                        else
                        {
                                for(i=1;i<pos-1;i++)
                                        temp1=temp1->next;
                                temp->next=temp1->next;
                                temp1->next=temp;
                        }
                        break;
                case 3: while(temp1->next!=NULL)
                                temp1=temp1->next;
                        temp1->next=temp;
                        break;

        }
        return p;
}
```

Deletion:

Delete at 1. Beginning 2. End 3. Delete at given position.

```
nodeptr del( nodeptr p)
{

        int c,i,pos;
        nodeptr temp;
        temp=p;
        if(p==NULL)
        {
                printf("\nlist is empty");
                return p;
        }
        else
        {
                printf("\nDelete\n1.at begining\n2.at end\n3.given position");
                printf("\nEnter your choice:");
                scanf("%d",&c);
                switch(c)
                {
                        case 1: p=p->next;
                                temp->next=NULL;
                                break;
                        case 2: while(temp->next->next!=NULL)
                                        temp=temp->next;
                                temp->next=NULL;
                                break;
                        case 3: printf("\nenter the position to be deleted:");
                                scanf("%d",&pos);
                                if(pos==1)
                                {
                                        p=temp->next;
                                        temp->next=NULL;
                                }
                                else
                                {
                                        for(i=1;i<pos-1;i++)
                                                temp=temp->next;
                                        temp->next=temp->next->next;
                                }
                                break;
```

```
                }
                return p;
        }
}
```

/* Deletion operations in Singly Linked List Program*/

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode();
nodeptr del(nodeptr);
void traversal(nodeptr);
nodeptr create();
main()
{
        nodeptr first;
        int choice;
        clrscr();
        first=create();
        traversal(first);
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("\n1.delete a node\n2.traversal\n3.exit\n*******************\nEnter
your choice:");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: first=del(first);
                                traversal(first);
                                break;
                        case 2: traversal(first);
                                break;
                        case 3: exit(0);
                }
        }
}
```

```
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=0;
        return p;
}
nodeptr del( nodeptr p)
{

        int c,i,pos;
        nodeptr temp;
        temp=p;
        if(p==NULL)
        {
                printf("\nlist is empty");
                return p;
        }
        else
        {
                printf("\nDelete\n1.at begining\n2.at end\n3.given position");
                printf("\nEnter your choice:");
                scanf("%d",&c);
                switch(c)
                {
                        case 1: p=p->next;
                                temp->next=NULL;
                                break;
                        case 2: while(temp->next->next!=NULL)
                                        temp=temp->next;
                                temp->next=NULL;
                                break;
                        case 3: printf("\nenter the position to be deleted:");
                                scanf("%d",&pos);
                                if(pos==1)
                                {
                                        p=temp->next;
                                        temp->next=NULL;
                                }
                                else
                                {
                                        for(i=1;i<pos-1;i++)
```

```
                                        temp=temp->next;
                                    temp->next=temp->next->next;
                                }
                                break;


                        }
                        return p;
                }
}
void traversal(nodeptr p)
{
        nodeptr p1;
        p1=p;
        printf("\nelements are:\n");
        while(p1!=NULL)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
}
nodeptr create()
{

    nodeptr p,p1,p2;
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
                p1->next=p2;
                p1=p2;
                p2=getnode();
                printf("\nEnter the number:");
                scanf("%d",&p2->info);
    }
        return p->next;
}
```

Searching:

Search an element in the list.

```
void search(nodeptr p)
{
        int a,i;
        nodeptr p1;
        p1=p;
        printf("\nenter the searching element:");
        scanf("%d",&a);
        i=1;
        while(p1!=NULL)
        {
                if(p1->info==a)
                {
                        printf("\nsearching elment is found at location:%d",i);
                        return;
                }
                p1=p1->next;
                i++;
        }
        printf("\nsearching element is not found");
}
```

/* Searching an element in singly Linked List Program*/

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode();
void search(nodeptr);
void traversal(nodeptr);
nodeptr create();
main()
{
        nodeptr first,last;
        int choice;
        clrscr();
        first=create();
        traversal(first);
```

```
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("\n1.search an element");
                printf("\n2.traversal\n3.exit\n*******************\nEnter your choice:");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: search(first);
                                break;
                        case 2: traversal(first);
                                break;
                        case 3: exit(0);
                }
        }
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=0;
        return p;
}
void search(nodeptr p)
{
        int a,i;
        nodeptr p1;
        p1=p;
        printf("\nenter the searching element:");
        scanf("%d",&a);
        i=1;
        while(p1!=NULL)
        {
                if(p1->info==a)
                {
                        printf("\nsearching elment is found at location:%d",i);
                        return;
                }
                p1=p1->next;
                i++;
        }
```

```
        printf("\nsearching element is not found");
}
void traversal(nodeptr p)
{
        nodeptr p1;
        p1=p;
        printf("\nelements are:\n");
        while(p1!=NULL)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
}

nodeptr create()
{

    nodeptr p,p1,p2;
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
                p1->next=p2;
                p1=p2;
                p2=getnode();
                printf("\nEnter the number:");
                scanf("%d",&p2->info);
    }
        return p->next;
}
```

Reversing singly linked list:

```
nodeptr reverse(nodeptr p)
{
        nodeptr p1,p2,p3;
        p1=p;
        p2=getnode();
        p2->info=p1->info;
        while(p1->next!=NULL)
        {
                p1=p1->next;
                p3=getnode();
                p3->info=p1->info;
                p3->next=p2;
                p2=p3;
        }
        return p2;
}
```

/* Reverse Linked List */

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode();
nodeptr reverse(nodeptr);
void traversal(nodeptr);
nodeptr create();
main()
{
        nodeptr first,last;
        clrscr();
        first=create();
        traversal(first);
        printf("\nReversing list is\n");
        last=reverse(first);
        traversal(last);
        getch();
}
nodeptr getnode()
{
```

```
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=0;
        return p;
}
nodeptr reverse(nodeptr p)
{
        nodeptr p1,p2,p3;
        p1=p;
        p2=getnode();
        p2->info=p1->info;
        while(p1->next!=NULL)
        {
                p1=p1->next;
                p3=getnode();
                p3->info=p1->info;
                p3->next=p2;
                p2=p3;
        }
        return p2;
}
void traversal(nodeptr p)
{
        nodeptr p1;
        p1=p;
        printf("\nelements are:\n");
        while(p1!=NULL)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
}
nodeptr create()
{

    nodeptr p,p1,p2;
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
```

```
    while(p2->info!=-999)
    {
            p1->next=p2;
            p1=p2;
            p2=getnode();
            printf("\nEnter the number:");
            scanf("%d",&p2->info);
    }
      return p->next;
}
```

Singly linked list program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode();
nodeptr insert(nodeptr );
nodeptr del(nodeptr);
void search(nodeptr);
nodeptr reverse(nodeptr);
void traversal(nodeptr);
nodeptr create(nodeptr);
main()
{
        nodeptr first,last;
        int choice;
        clrscr();
        first=create(first);
        traversal(first);
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("\n1.adding a node\n2.delete a node\n3.search an element");
                printf("\n4reverse linked list\n5.traversal\n");
```

```
                printf("6.exit\n*******************\nEnter your choice:");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1: first=insert(first);
                                traversal(first);
                                break;
                        case 2: first=del(first);
                                traversal(first);
                                break;
                        case 3: search(first);
                                break;
                        case 4: last=reverse(first);
                                traversal(last);
                                break;
                        case 5: traversal(first);
                                break;
                        case 6: exit(0);
                }
        }
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=0;
        return p;
}
nodeptr insert(nodeptr p)
{
        nodeptr temp,temp1;
        int ch,pos,i;
        temp=getnode();
        temp1=p;
        printf("\nenter the insert number:");
        scanf("%d",&temp->info);
        printf("\n1.at begining\n2. at given position\n3. at end");
        printf("\nenter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
```

```
                case 1: temp->next=temp1;
                        return temp;
                case 2: printf("\nEnter the position to insert:");
                        scanf("%d",&pos);
                        if(pos==1)
                        {
                                temp->next=temp1;
                                p=temp;
                        }
                        else
                        {
                                for(i=1;i<pos-1;i++)
                                        temp1=temp1->next;
                                temp->next=temp1->next;
                                temp1->next=temp;
                        }
                        break;
                case 3: while(temp1->next!=NULL)
                                temp1=temp1->next;
                        temp1->next=temp;
                        break;

        }
        return p;
}

nodeptr del( nodeptr p)
{

        int c,i,pos;
        nodeptr temp;
        temp=p;
        if(p==NULL)
        {
                printf("\nlist is empty");
                return p;
        }
        else
        {
                printf("\nDelete\n1.at begining\n2.at end\n3.given position");
                printf("\nEnter your choice:");
                scanf("%d",&c);
                switch(c)
```

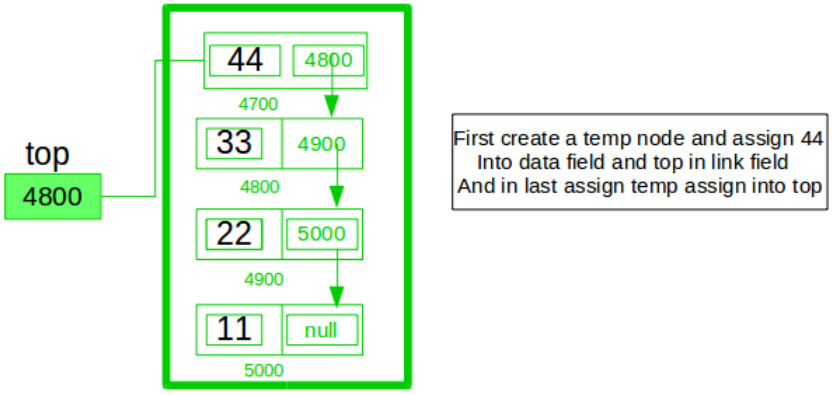
```
                {
                        case 1: p=p->next;
                                temp->next=NULL;
                                break;
                        case 2: while(temp->next->next!=NULL)
                                        temp=temp->next;
                                temp->next=NULL;
                                break;
                        case 3: printf("\nenter the position to be deleted:");
                                scanf("%d",&pos);
                                if(pos==1)
                                {
                                        p=temp->next;
                                        temp->next=NULL;
                                }
                                else
                                {
                                        for(i=1;i<pos-1;i++)
                                                temp=temp->next;
                                        temp->next=temp->next->next;
                                }
                                break;


                 }
                 return p;
        }
}
void search(nodeptr p)
{
        int a,i;
        nodeptr p1;
        p1=p;
        printf("\nenter the searching element:");
        scanf("%d",&a);
        i=1;
        while(p1!=NULL)
        {
                if(p1->info==a)
                {
                        printf("\nsearching elment is found at location:%d",i);
                        return;
                }
```

```
                p1=p1->next;
                i++;
        }
        printf("\nsearching element is not found");
}
nodeptr reverse(nodeptr p)
{
        nodeptr p1,p2,p3;
        p1=p;
        p2=getnode();
        p2->info=p1->info;
        while(p1->next!=NULL)
        {
                p1=p1->next;
                p3=getnode();
                p3->info=p1->info;
                p3->next=p2;
                p2=p3;
        }
        return p2;
}
void traversal(nodeptr p)
{
        nodeptr p1;
        p1=p;
        printf("\nelements are:\n");
        while(p1!=NULL)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
}

nodeptr create()
{

    nodeptr p,p1,p2;
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
```

```
    while(p2->info!=-999)
    {
            p1->next=p2;
            p1=p2;
            p2=getnode();
            printf("\nEnter the number:");
            scanf("%d",&p2->info);
    }
      return p->next;
}
```

Applications on singly linked list

implementation of stack:



Representation of stack:

The declaration of the stack is

```
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
struct stack
{
        nodeptr top;
};
```

Initially take **s.top = NULL.**

Stack operations:

Empty Stack:

The empty stack contains no elements. Therefore top=NULL.

The procedure is

```
int empty(struct stack *st)
{
        if(st->top==NULL)
                return 1;
        else
                return 0;
}
```

Pop operation:

The pop operation performs the following three actions.

1. If the stack is empty, print a warning message and halt the execution.
2. Remove the top element from the stack.
3. Return this element to the calling function.

The procedure is

```
int pop(struct stack *st)
{
     int t;
     if(empty(&st))
                return -1;
        else
        {
            t=st->top->info;
            st->top=st->top->next;
            return t;
        }
}
```

Push operation:

The push operation is used to insert an element into the stack. The new element is added at the topmost position of the stack.

The procedure is

```
void push(struct stack *st,int a)
{
        nodeptr p;
        p=getnode();
        p->info=a;
        if(st->top==NULL)
        {
          st->top=p;
```

```
        }
        else
        {
                p->next=st->top;
                st->top=p;
        }
 }
```

Peek operation:

Peek is an operation that return the value of the topmost element of the stack without deleting it from the stack.

The procedure is

```
int peek(struct stack *s)
{
        if (s→top = = NULL)
                return -1;
        else
                return s→top→info;
}
```

/* STACK OPERATIONS USING LINKED LISTS PROGRAM*/

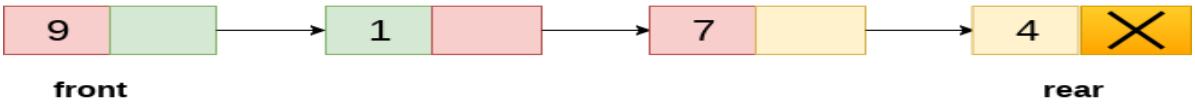
```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
struct stack
{
        nodeptr top;
};
nodeptr getnode();
void push(struct stack *,int);
int pop(struct stack *);
void display(struct stack *);
int peek(struct stack *);
main()
{
        int choice,x;
        struct stack s;
        clrscr();
```

```
        s.top=NULL;
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("1.push\n2.pop\n3.empty\n4.peek\n5.display\n");
                printf("6.exit\n*******************\nEnter your choice:");
                scanf("%d",&choice);
                switch(choice)
                {
                case 1: printf("\nenter the element:");
                        scanf("%d",&x);
                        push(&s,x);
                        display(&s);
                        break;
                case 2: x=pop(&s);
                        if(x==-1)
                                printf("\nstack is empty");
                        else
                                printf("deleted item is:%d",x);
                        display(&s);
                        break;
                case 3: x=empty(&s);
                        if(x==1)
                                printf("\nstack is empty");
                        else
                                printf("\nstack is nonempty");
                        break;
                case 4: x=peek(&s);
                        if(x==-1)
                                printf("\nStack is empty");
                        else
                                printf("\ntop element of the stack:%d",x);
                        break;
                case 5: display(&s);
                        break;
                case 6: exit(0);
                }
        }
}
nodeptr getnode()
{
        nodeptr p;
```

```
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=NULL;
        return p;
}
void push(struct stack *st,int a)
{
        nodeptr p;
        p=getnode();
        p->info=a;
        if(st->top==NULL)
        {
          st->top=p;
        }
        else
        {
                p->next=st->top;
                st->top=p;
        }
 }
int pop(struct stack *st)
{
    int t;
    if(empty(st))
                return -1;
        else
        {
            t=st->top->info;
            st->top=st->top->next;
            return t;
        }
}
void display(struct stack *st)
{
        nodeptr p1;
        p1=st->top;
        printf("\nelements are:\n");
        printf("(top)");
        while(p1!=NULL)
        {
                printf("%d<--",p1->info);
                p1=p1->next;
        }
```

```
}
int empty(struct stack *st)
{
        if(st->top==NULL)
                return 1;
        else
                return 0;
}
int peek(struct stack *st)
{
        if(st->top==NULL)
                return -1;
        else
                return st->top->info;
}
```

implementation of queues:



Linked Queue

Representation of Queue:

```
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
struct queue
{
        nodeptr front,rear;
};
```

Initially **front = NULL and rear = NULL**

Operations on Queue:

Empty operation.

```
int empty(struct queue *qu)
{
        if(qu->front==NULL)
                return 1;
```

```
        else
                return 0;
}
```

Insert operation:

```
void insert(struct queue *qu,int a)
{
        nodeptr p;
        p=getnode();
        p->info=a;
        if(qu->front==NULL)
        {
          qu->rear=p;
          qu->front=qu->rear;
        }
        else
        {
                qu->rear->next=p;
                qu->rear=p;
        }
 }
```

Remove operation:

Check empty condition.

```
int del(struct queue *qu)
{
      int t;
      if(empty(qu))
          return -1;
     else
     {
         t=qu->front->info;
         qu->front=qu->front->next;
         return t;
     }
}
```

PEEK Operation

```
int peek(struct queue *qu)
{
        if(qu->front==NULL)
                return -1;
        else
```

```
                return qu->front->info;
}
```

/* QUEUE OPERATIONS USING LINKED LISTS */

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
struct queue
{
        nodeptr front,rear;
};
nodeptr getnode();
void insert(struct queue *, int a);
int del(struct queue *);
void display(struct queue *);
int empty(struct queue *);
int peek(struct queue *);
main()
{
        int choice,x;
        struct queue q;
        clrscr();
        q.rear=NULL;
        q.front=NULL;
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("1.insert\n2.delete\n3.empty\n4.peek\n5.display\n");
                printf("6.exit\n*******************\nEnter your choice:");
                scanf("%d",&choice);
                switch(choice)
                {
                case 1: printf("\nenter the element:");
                        scanf("%d",&x);
                        insert(&q,x);
                        display(&q);
                        break;
```

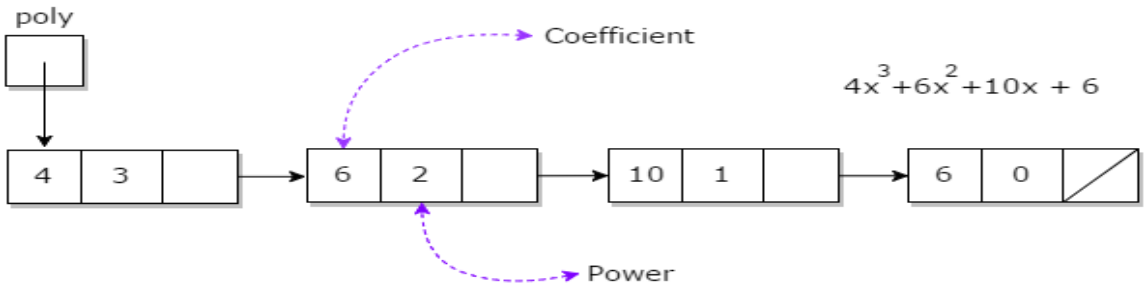
```
                case 2:x=del(&q);
                        if(x==-1)
                                printf("\nQueue is empty");
                        else
                                printf("deleted item is:%d",x);
                        display(&q);
                        break;
                case 3:if(empty(&q))
                                printf("\nQueue is empty");
                        else
                                printf("\nQueue is nonempty");
                        break;
                case 4: x=peek(&q);
                        if(x==-1)
                                printf("\nQueue is empty");
                        else
                                printf("\nfront element of Queue is:%d",x);
                        break;
                case 5:display(&q);
                        break;
                case 6:exit(0);
                }
        }

}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=NULL;
        return p;
}
void insert(struct queue *qu,int a)
{
        nodeptr p;
        p=getnode();
        p->info=a;
        if(qu->front==NULL)
        {
          qu->rear=p;
          qu->front=qu->rear;
        }
```

```
        else
        {
                qu->rear->next=p;
                qu->rear=p;
        }
 }
int del(struct queue *qu)
{
    int t;
    if(empty(qu))
        return -1;
        else
        {
            t=qu->front->info;
            qu->front=qu->front->next;
            return t;
        }
}
void display(struct queue *qu)
{
        nodeptr p1;
        p1=qu->front;
        printf("\nelements are:\n");
        while(p1!=NULL)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
}
int empty(struct queue *qu)
{
        if(qu->front==NULL)
                return 1;
        else
                return 0;
}
int peek(struct queue *qu)
{
        if(qu->front==NULL)
                return -1;
        else
                return qu->front->info;
}
```

polynomial expression representation using singly linked list

A polynomial can be thought of as an ordered list of non zero terms. Each non zero term is a two-tuple which holds two pieces of information:

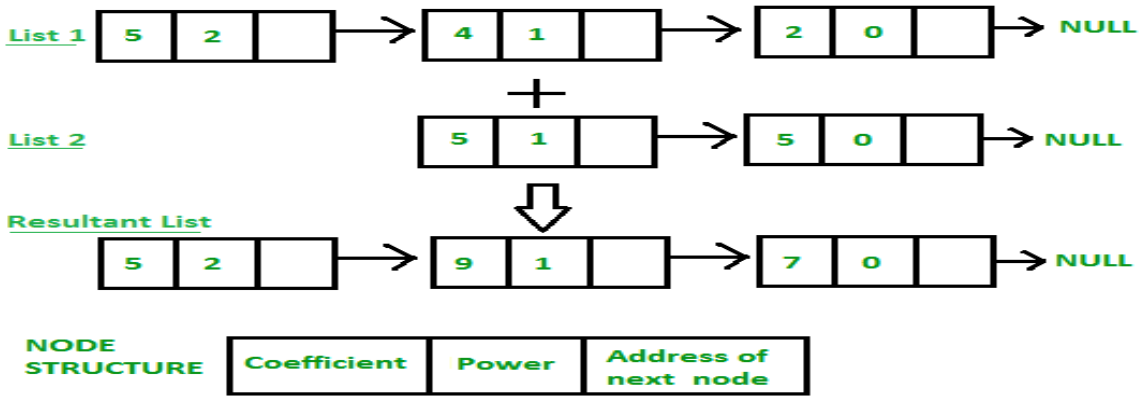
- The exponent part
- The coefficient part




```
struct node
{
        int exp,coef;
        struct node *next;
};

nodeptr create()
{
        nodeptr p,p1,p2;
        p1=getnode();
        p2=getnode();
        p=p1;
        printf("\nenter the expantions in order\n");
        printf("\nEnter the exp -1 at END\n");
        printf("enter the coef &  exp :");
        scanf("%d%d",&p2->coef,&p2->exp);
        while(p2->exp!=-1)
        {
                p1->next=p2;
```

```
            p1=p2;
            p2=getnode();
            printf("\nenter the coef & exp :");
            scanf("%d%d",&p2->coef,&p2->exp);
        }
        p=p->next;
        return p;
}
```

Addition




```
nodeptr addpoly(nodeptr p,nodeptr q)
{
        nodeptr p1,p2,p3,p4,fst;
        p1=p;
        p2=q;
        p3=getnode();
        fst=p3;
        while((p1!=NULL)&&(p2!=NULL))
        {
            p4=getnode();
            if(p1->exp==p2->exp)
            {
                p4->exp=p1->exp;
                p4->coef=p1->coef+p2->coef;
                p1=p1->next;
                p2=p2->next;
            }
```

```
                else if(p1->exp>p2->exp)
                {
                        p4->exp=p1->exp;
                        p4->coef=p1->coef;
                        p1=p1->next;
                }
                else
                {
                        p4->exp=p2->exp;
                        p4->coef=p2->coef;
                        p2=p2->next;
                }
                p3->next=p4;
                p3=p4;
        }
        if(p1==NULL)
        {
                while(p2!=NULL)
                {
                        p4=getnode();
                        p4->exp=p2->exp;
                        p4->coef=p2->coef;
                        p2=p2->next;
                        p3->next=p4;
                        p3=p4;
                }
        }
        else
        {
                while(p1!=NULL)
                {
                        p4=getnode();
                        p4->exp=p1->exp;
                        p4->coef=p1->coef;
                        p1=p1->next;
                        p3->next=p4;
                        p3=p4;
                }
        }
        fst=fst->next;
        return fst;
}
```

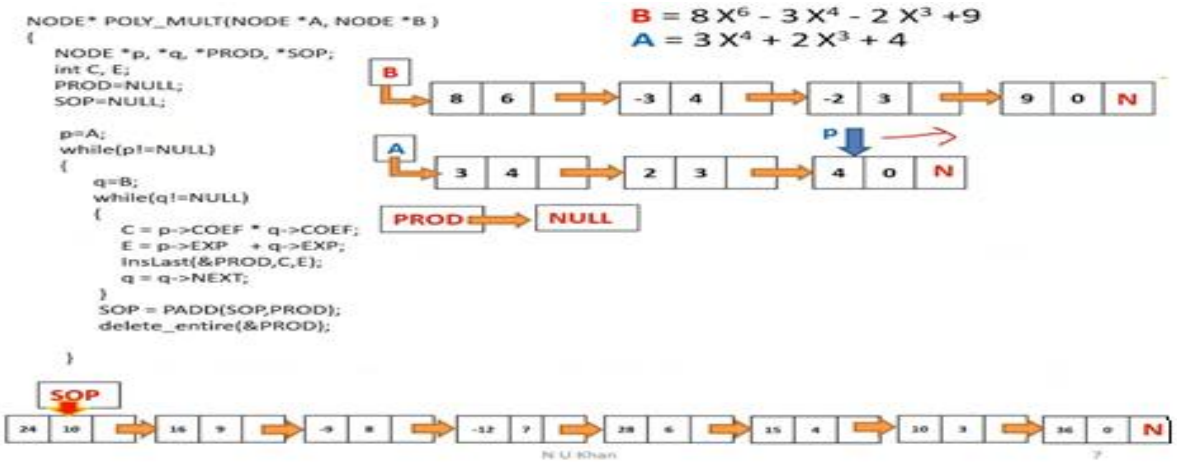
/* POLYNOMIAL ADDITION PROGRAM */

```
#include<stdio.h>
struct node
{
        int exp,coef;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr create();
nodeptr addpoly();
void display();
nodeptr getnode();
main()
{
        nodeptr fst,sec,first;
        clrscr();
        fst=create();
        printf("\nFirst polynomial is:");
        display(fst);
        sec=create();
        printf("\nSecond polynomial is:");
        display(sec);
        first=addpoly(fst,sec);
        printf("\nAddition of given two polynomials is:\n");
        display(first);
        getch();
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->exp=p->coef=0;
        p->next=NULL;
        return p;
}
nodeptr create()
{
        nodeptr p,p1,p2;
        p1=getnode();
        p2=getnode();
        p=p1;
        printf("\nenter the expantions in order\n");
        printf("\nEnter the exp -1 at END\n");
```

```
        printf("enter the coef &  exp :");
        scanf("%d%d",&p2->coef,&p2->exp);
        while(p2->exp!=-1)
        {
                p1->next=p2;
                p1=p2;
                p2=getnode();
                printf("\nenter the coef & exp :");
                scanf("%d%d",&p2->coef,&p2->exp);
        }
        p=p->next;
        return p;
}
void display(nodeptr p)
{
        nodeptr p1;
        p1=p;
        while(p1!=NULL)
        {
                printf("%d*X^%d+",p1->coef,p1->exp);
                p1=p1->next;

        }
}
nodeptr addpoly(nodeptr p,nodeptr q)
{
        nodeptr p1,p2,p3,p4,fst;
        p1=p;
        p2=q;
        p3=getnode();
        fst=p3;
        while((p1!=NULL)&&(p2!=NULL))
        {
                p4=getnode();
                if(p1->exp==p2->exp)
                {
                        p4->exp=p1->exp;
                        p4->coef=p1->coef+p2->coef;
                        p1=p1->next;
                        p2=p2->next;
                }
                else if(p1->exp>p2->exp)
                {
```

```
                p4->exp=p1->exp;
                p4->coef=p1->coef;
                p1=p1->next;
        }
        else
        {
                p4->exp=p2->exp;
                p4->coef=p2->coef;
                p2=p2->next;
        }
        p3->next=p4;
        p3=p4;
    }
    if(p1==NULL)
    {
        while(p2!=NULL)
        {
                p4=getnode();
                p4->exp=p2->exp;
                p4->coef=p2->coef;
                p2=p2->next;
                p3->next=p4;
                p3=p4;
        }
    }
    else
    {
        while(p1!=NULL)
        {
                p4=getnode();
                p4->exp=p1->exp;
                p4->coef=p1->coef;
                p1=p1->next;
                p3->next=p4;
                p3=p4;
        }
    }
    fst=fst->next;
    return fst;
}
```

Polynomial Multiplication




```
nodeptr polymul(nodeptr p,nodeptr q)
{
        nodeptr p1,p2,p3,p4,fst,fst1;
        p1=p;
        fst1=getnode();
        p3=getnode();
        fst=p3;
        while(p1!=NULL)
        {
                p2=q;
                p3=fst;
                while(p2!=NULL)
                {
                        p4=getnode();
                        p4->coef=p1->coef*p2->coef;
                        p4->exp=p1->exp+p2->exp;
                        p2=p2->next;
                        p3->next=p4;
                        p3=p4;
                }
                p4=fst->next;
                fst1=polyadd(p4,fst1);
                p1=p1->next;
        }
        return fst1;
}
```

/* POLYNOMIAL MULTIPLICATIONS */

```
#include<stdio.h>
struct node
{
        int coef,exp;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode();
nodeptr create();
void display();
nodeptr polymul();
nodeptr polyadd();
main()
{
        nodeptr fst,sec,first;
        clrscr();
        fst=create();
        printf("\n1st polynomial is:");
        display(fst);
        sec=create();
        printf("\n2nd polynomial is:");
        display(sec);
        first=polymul(fst,sec);
        printf("\nmultiplication of given two polynomials is:\n");
        display(first);
        getch();
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->coef=p->exp=0;
        p->next=NULL;
        return p;
}
nodeptr create()
{
        nodeptr p,p1,p2;
        p1=getnode();
        p2=getnode();
        p=p1;
        printf("\nenter the expantions inorder\n");
```

```
        printf("\nenter exp -1 at the END\n");
        printf("\nenter the coef & exp :");
        scanf("%d%d",&p2->coef,&p2->exp);
        while(p2->exp!=-1)
        {
                p1->next=p2;
                p1=p2;
                p2=getnode();
                printf("\nenter the coef & exp :");
                scanf("%d%d",&p2->coef,&p2->exp);
        }
        p=p->next;
        return p;
}
void display(nodeptr p)
{
        nodeptr p1;
        p1=p;
        while(p1!=NULL)
        {
                printf("%d * X ^%d +",p1->coef,p1->exp);
                p1=p1->next;
        }
}
nodeptr polymul(nodeptr p,nodeptr q)
{
        nodeptr p1,p2,p3,p4,fst,fst1;
        p1=p;
        fst1=getnode();
        p3=getnode();
        fst=p3;
        while(p1!=NULL)
        {
                p2=q;
                p3=fst;
                while(p2!=NULL)
                {
                        p4=getnode();
                        p4->coef=p1->coef*p2->coef;
                        p4->exp=p1->exp+p2->exp;
                        p2=p2->next;
                        p3->next=p4;
                        p3=p4;
```

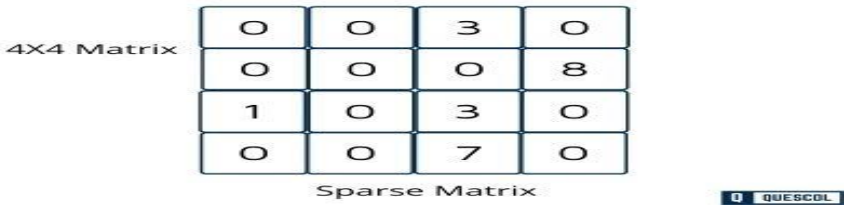
```
                }
                p4=fst->next;
                fst1=polyadd(p4,fst1);
                p1=p1->next;
        }
        return fst1;
}
nodeptr polyadd(nodeptr p,nodeptr q)
{
        nodeptr p1,p2,p3,p4,fst;
        p1=p;
        p2=q;
        p3=getnode();
        fst=p3;
        while((p1!=NULL)&&(p2!=NULL))
        {
                p4=getnode();
                if(p1->exp==p2->exp)
                {
                        p4->exp= p1->exp;
                        p4->coef=p1->coef+p2->coef;
                        p1=p1->next;
                        p2=p2->next;
                }
                else if(p1->exp<p2->exp)
                {
                        p4->exp=p2->exp;
                        p4->coef=p2->coef;
                        p2=p2->next;
                }
                else
                {
                        p4->exp=p1->exp;
                        p4->coef=p1->coef;
                        p1=p1->next;
                }
                p3->next=p4;
                p3=p4;
        }
        if(p1==NULL)
        {
                while(p2!=NULL)
                {
```

```
                p4=getnode();
                p4->exp=p2->exp;
                p4->coef=p2->coef;
                p2=p2->next;
                p3->next=p4;
                p3=p4;
        }
    }
    else
    {
        while(p1!=NULL)
        {
                p4=getnode();
                p4->exp=p1->exp;
                p4->coef=p1->coef;
                p1=p1->next;
                p3->next=p4;
                p3=p4;
        }
    }
    fst=fst->next;
    return fst;
}
```

Sparse matrix

Sparse matrix: if a matrix has most of its elements equal to zero, then the matrix is called sparse matrix.

Ex:



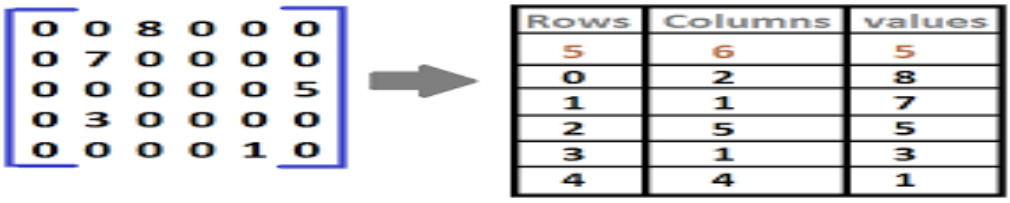
There are two types of representations

1. Array representation
2. Linked list representation

Array representation:

In this representation, we consider only non-zero values along with their row and column values. In this representation, the 0^{th} row store the total number of rows, total number of columns and total number of non-zero values in the sparse matrix.

Ex:

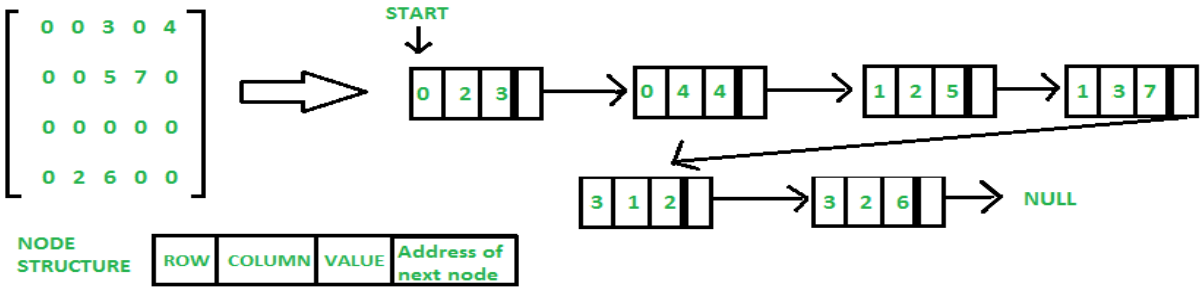


Rows	Columns	values
5	6	5
0	2	8
1	1	7
2	5	5
3	1	3
4	4	1

Linked list representation:

In linked list representation, each node consists of four fields.

1. Row: it stores the index of the row, where we have a non-zero value in the sparse matrix.
2. Column: it stores the index of column, where we have a non-zero value in the sparse matrix.
3. Value: this variable consists of the actual non-zero value being stored.
4. Next: it is a pointer to store the address of the next node.



Node Representation:

```
struct node
{
int value;
int row,column;
struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode()
{
        nodeptr p;
```

```
        p=(nodeptr)malloc(sizeof(struct node));
        p->row=0;
        p->column=0;
        p->value=0;
        p->next=NULL;
        return p;
}
```

Creation of a Linked List:

```
nodeptr create(nodeptr start, int x, int y, int z)
{
        nodeptr p,p1;
        p=start;
        p1=getnode();
        p1->row=x;
        p1->column=y;
        p1->value=z;
        if (p == NULL)
        {
                start=p1;
        }
        else
        {
                while (p->next!=NULL)
                        p=p->next;
                p->next=p1;
        }
        return start;

}
```

/* Sparse Matrix Representation using Singly Linked List */

```
/* Sparse Matrix Representation */
#include <stdio.h>
#include <stdlib.h>
struct node
{
int value;
int row,column;
struct node *next;
};
```

```
typedef struct node *nodeptr;
nodeptr create(nodeptr, int, int, int);
nodeptr getnode();
void display(nodeptr);
int main()
{
	nodeptr start;
	int i,j,a[10][10],n,m,count;
	count=0;
	start=NULL;
	printf("\nEnter the row size of array:");
	scanf("%d",&n);
	printf("\nEnter the column size of array:");
	scanf("%d",&m);
	printf("\nEnter the elements\n");
	for (i=0;i<n;i++)
		for (j=0;j<m;j++)
		{
			scanf("%d",&a[i][j]);
			if(a[i][j]!=0)
				count++;
		}
	start=create(start,n,m,count);
	for (i=0;i<n;i++)
		for (j=0;j<m;j++)
			if (a[i][j] != 0)
				start=create(start, i,j,a[i][j]);
	printf("Sparse Matrix is\n");
	for(i=0;i<n;i++)
	{
		for(j=0;j<m;j++)
			printf("%d\t",a[i][j]);
		printf("\n");
	}
	display(start);
	getch();
}
void display(nodeptr start)
{
	nodeptr temp;
	temp = start;
	printf("\nrow\tcolumn\tvalue\n");
	while(temp!=NULL)
```

```
        {
                printf("%d\t%d\t%d\n", temp->row,temp->column,temp->value);
                temp = temp->next;
        }
}
nodeptr create(nodeptr start, int x, int y, int z)
{
        nodeptr p,p1;
        p=start;
        p1=getnode();
        p1->row=x;
        p1->column=y;
        p1->value=z;
        if (p == NULL)
        {
                start=p1;
        }
        else
        {
                while (p->next!=NULL)
                        p=p->next;
                p->next=p1;
        }
        return start;

}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->row=0;
        p->column=0;
        p->value=0;
        p->next=NULL;
        return p;
}
```

Advantages and disadvantages of singly linked list

ADVANTAGE: -

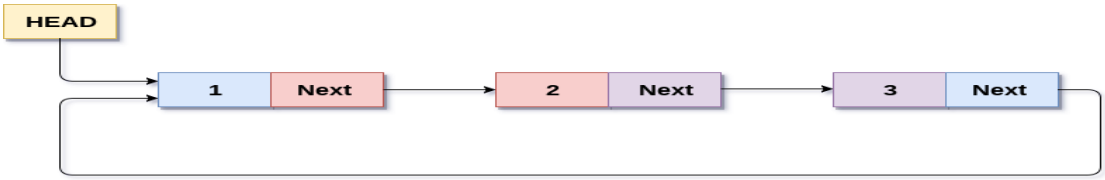
1) Insertions and Deletions can be done easily.
2) It does not need movement of elements for insertion and deletion.
3) It space is not wasted as we can get space according to our requirements.
4) Its size is not fixed.

5) It can be extended or reduced according to requirements.
6) Elements may or may not be stored in consecutive memory available, even then we can store the data in computer.
7) It is less expensive.

DISADVANTAGE: -

1) It requires more space as pointers are also stored with information.
2) Different amount of time is required to access each element.
3) If we have to go to a particular element then we have to go through all those elements that come before that element.
4) we cannot traverse it from last & only from the beginning.
5) It is not easy to sort the elements stored in the linear linked list.

circular Singly linked list



Circular Singly Linked List

```
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=p;
        return p;
}
```

Create a Circular Linked List:

```
nodeptr create()
{

    nodeptr p,p1,p2;
```

```
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
            p1->next=p2;
            p1=p2;
            p2=getnode();
            printf("\nEnter the number:");
            scanf("%d",&p2->info);
    }
      p=p->next;
      p1->next=p;
      return p;
}
```

Traverse:

```
  void traversal(nodeptr p)
  {
      nodeptr p1;
      p1=p;
      printf("\nelements are:\n");
      while(p1->next!=p)
      {
            printf("%d-->",p1->info);
            p1=p1->next;
      }
      printf("%d",p1->info);
  }
```

Insertion operation

Insert at 1. beginning 2. End 3. Given position

```
nodeptr insert(nodeptr p)
{
      nodeptr p1,p2;
      int ch,k,i;
      p1=p;
```

```
p2=getnode();
printf("\nenter inserted element:");
scanf("%d",&p2->info);
if(p==NULL)
{
        p2->next=p2;
        p=p2;
        return p;
}
else
{
        printf("\ninsert\n1.at beg\n2.at end\n3.insert at given position\nenter your
choice:");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: while(p1->next!=p)
                                p1=p1->next;
                        p2->next=p;
                        p1->next=p2;
                        return p2;
                case 2: while(p1->next!=p)
                                p1=p1->next;
                        p2->next=p;
                        p1->next=p2;
                        return p;
                case 3: printf("\nEnter the position to insert Node:");
                        scanf("%d",&k);
                        if(k==1)
                        {
                                while(p1->next!=p)
                                        p1=p1->next;
                                p2->next=p;
                                p1->next=p2;
                                p=p2;
                        }
                        else
                        {
                                for(i=1;i<k-1;i++)
                                        p1=p1->next;
                                p2->next=p1->next;
                                p1->next=p2;
                        }
```

```
                        return p;

                }
        }
}
```

/* Insertion operations in CIRCULAR Linked List */

```
/* Insertion operations in CIRCULAR Linked List  */
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode();
nodeptr create();
nodeptr insert(nodeptr);
void traversal(nodeptr);
main()
{
        nodeptr first;
        int choice;
        clrscr();
        first=create();
        traversal(first);
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("\n1.adding a node\n2.traversal\n3.exit\n*******************\nEnter
your choice:");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1: first=insert(first);
                                traversal(first);
                                break;
                        case 2: traversal(first);
                                break;
                        case 3: exit(0);
```

```
                }
        }
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=p;
        return p;
}
nodeptr insert(nodeptr p)
{
        nodeptr p1,p2;
        int ch,k,i;
        p1=p;
        p2=getnode();
        printf("\nenter inserted element:");
        scanf("%d",&p2->info);
        if(p==NULL)
        {
                p2->next=p2;
                p=p2;
                return p;
        }
        else
        {
                printf("\ninsert\n1.at beg\n2.at end\n3.insert at given position\nenter your
choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: while(p1->next!=p)
                                        p1=p1->next;
                                p2->next=p;
                                p1->next=p2;
                                return p2;
                        case 2: while(p1->next!=p)
                                        p1=p1->next;
                                p2->next=p;
                                p1->next=p2;
                                return p;
                        case 3: printf("\nEnter the position to insert Node:");
```

```
                                scanf("%d",&k);
                                if(k==1)
                                {
                                        while(p1->next!=p)
                                                p1=p1->next;
                                        p2->next=p;
                                        p1->next=p2;
                                        p=p2;
                                }
                                else
                                {
                                        for(i=1;i<k-1;i++)
                                                p1=p1->next;
                                        p2->next=p1->next;
                                        p1->next=p2;
                                }
                                return p;

                        }
                }
}

void traversal(nodeptr p)
{
        nodeptr p1;
        p1=p;
        printf("\nelements are:\n");
        while(p1->next!=p)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
        printf("%d",p1->info);
}

nodeptr create()
{

    nodeptr p,p1,p2;
    p1=getnode();
        p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
```

```
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
            p1->next=p2;
            p1=p2;
            p2=getnode();
            printf("\nEnter the number:");
            scanf("%d",&p2->info);
    }
      p=p->next;
      p1->next=p;
      return p;
}
```

Deletion operation

Deletion at 1. Beginning 2. End 3. Given Position

```
nodeptr del( nodeptr p)
{
      int c,k,i;
      nodeptr p1,p2;
      p1=p;
      if(p1==NULL)
      {
            printf("\nlist is empty");
            return p;
      }
      else
      {
            printf("\nDelete\n1.at begining\n2.at end\n3.position to delete ");
            printf("\nEnter your choice:");
            scanf("%d",&c);
            switch(c)
            {
                  case 1: if(p1->next==p1)
                        {
                              p=NULL;
                              return p;
                        }
                        else
                        {
                              while(p1->next!=p)
                                    p1=p1->next;
```

```
                        p2=p->next;
                        p1->next=p2;
                        p->next=NULL;
                        return p2;
                }
        case 2: if(p1->next==p1)
                {
                        p=NULL;
                        return p;
                }
                while(p1->next->next!=p)
                        p1=p1->next;
                p1->next=p;
                return p;
        case 3: printf("\nenter the position to be delete :");
                scanf("%d",&k);
                if(k==1)
                {
                        while(p1->next!=p)
                                p1=p1->next;
                        p1->next=p->next;
                        p=p->next;
                }
                else
                {
                        for(i=1;i<k-1;i++)
                                p1=p1->next;
                        p2=p1->next;
                        p1->next=p2->next;
                        p2->next=NULL;
                }
                return p;

        }
    }
}
```

/* Deletion operations in CIRCULAR Linked List */

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

```
        struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode();
nodeptr create();
nodeptr del(nodeptr);
void traversal(nodeptr);
main()
{
        nodeptr first;
        int choice;
        clrscr();
        first=create();
        traversal(first);
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("\n1.delete a node\n2.traversal\n");
                printf("3.exit\n*******************\nEnter your choice:");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1: first=del(first);
                                traversal(first);
                                break;
                        case 2: traversal(first);
                                break;
                        case 3: exit(0);
                }
        }
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=p;
        return p;
}
nodeptr del( nodeptr p)
{
```

```
int c,k,i;
nodeptr p1,p2;
p1=p;
if(p1==NULL)
{
        printf("\nlist is empty");
        return p;
}
else
{
        printf("\nDelete\n1.at begining\n2.at end\n3.position to delete ");
        printf("\nEnter your choice:");
        scanf("%d",&c);
        switch(c)
        {
                case 1: if(p1->next==p1)
                        {
                                p=NULL;
                                return p;
                        }
                        else
                        {
                                while(p1->next!=p)
                                        p1=p1->next;
                                p2=p->next;
                                p1->next=p2;
                                p->next=NULL;
                                return p2;
                        }
                case 2: if(p1->next==p1)
                        {
                                p=NULL;
                                return p;
                        }
                        while(p1->next->next!=p)
                                p1=p1->next;
                        p1->next=p;
                        return p;
                case 3: printf("\nenter the position to be delete :");
                        scanf("%d",&k);
                        if(k==1)
                        {
```

```
                                        while(p1->next!=p)
                                                p1=p1->next;
                                        p1->next=p->next;
                                        p=p->next;
                                }
                                else
                                {
                                        for(i=1;i<k-1;i++)
                                                p1=p1->next;
                                        p2=p1->next;
                                        p1->next=p2->next;
                                        p2->next=NULL;
                                }
                                return p;

                        }
                }
}

void traversal(nodeptr p)
{
        nodeptr p1;
        p1=p;
        printf("\nelements are:\n");
        while(p1->next!=p)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
        printf("%d",p1->info);
}

nodeptr create()
{

    nodeptr p,p1,p2;
    p1=getnode();
        p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
```

```
    {
            p1->next=p2;
            p1=p2;
            p2=getnode();
            printf("\nEnter the number:");
            scanf("%d",&p2->info);
    }
      p=p->next;
      p1->next=p;
      return p;
}
```

/* CIRCULAR Singly Linked List operations */

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
      int info;
      struct node *next;
};
typedef struct node *nodeptr;
nodeptr getnode();
nodeptr insert(nodeptr);
nodeptr del(nodeptr);
void traversal(nodeptr);
nodeptr create();
main()
{
      nodeptr first;
      int choice;
      clrscr();
      first=create();
      traversal(first);
      while(1)
      {

      printf("\n*************************\n\n\tMENU\n**********************\n");
            printf("\n1.adding a node\n2.delete a node");
            printf("\n3.traversal\n");
            printf("4.exit\n*******************\nEnter your choice:");
            scanf("%d",&choice);

            switch(choice)
```

```
                {
                        case 1: first=insert(first);
                                traversal(first);
                                break;
                        case 2: first=del(first);
                                traversal(first);
                                break;
                        case 3: traversal(first);
                                break;
                        case 4: exit(0);
                }
        }
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=p;
        return p;
}
nodeptr insert(nodeptr p)
{
        nodeptr p1,p2;
        int ch,k,i;
        p1=p;
        p2=getnode();
        printf("\nenter inserted element:");
        scanf("%d",&p2->info);
        if(p==NULL)
        {
                p2->next=p2;
                p=p2;
                return p;
        }
        else
        {
                printf("\ninsert\n1.at beg\n2.at end\n3.insert at given position\nenter your
choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: while(p1->next!=p)
```

```
                        p1=p1->next;
                    p2->next=p;
                    p1->next=p2;
                    return p2;
                case 2: while(p1->next!=p)
                        p1=p1->next;
                    p2->next=p;
                    p1->next=p2;
                    return p;
                case 3: printf("\nEnter the position to insert Node:");
                    scanf("%d",&k);
                    if(k==1)
                    {
                        while(p1->next!=p)
                            p1=p1->next;
                        p2->next=p;
                        p1->next=p2;
                        p=p2;
                    }
                    else
                    {
                        for(i=1;i<k-1;i++)
                            p1=p1->next;
                        p2->next=p1->next;
                        p1->next=p2;
                    }
                    return p;

            }
        }
}
nodeptr create()
{

    nodeptr p,p1,p2;
    p1=getnode();
      p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
```

```
                p1->next=p2;
                p1=p2;
                p2=getnode();
                printf("\nEnter the number:");
                scanf("%d",&p2->info);
        }
        p=p->next;
        p1->next=p;
        return p;
}
nodeptr del( nodeptr p)
{

        int c,k,i;
        nodeptr p1,p2;
        p1=p;
        if(p1==NULL)
        {
                printf("\nlist is empty");
                return p;
        }
        else
        {
                printf("\nDelete\n1.at begining\n2.at end\n3.position to delete ");
                printf("\nEnter your choice:");
                scanf("%d",&c);
                switch(c)
                {
                        case 1: if(p1->next==p1)
                                {
                                        p=NULL;
                                        return p;
                                }
                                else
                                {
                                        while(p1->next!=p)
                                                p1=p1->next;
                                        p2=p->next;
                                        p1->next=p2;
                                        p->next=NULL;
                                        return p2;
                                }
                        case 2: if(p1->next==p1)
```

```
                        {
                                p=NULL;
                                return p;
                        }
                        while(p1->next->next!=p)
                                p1=p1->next;
                        p1->next=p;
                        return p;
                case 3: printf("\nenter the position to be delete :");
                        scanf("%d",&k);
                        if(k==1)
                        {
                                while(p1->next!=p)
                                        p1=p1->next;
                                p1->next=p->next;
                                p=p->next;
                        }
                        else
                        {
                                for(i=1;i<k-1;i++)
                                        p1=p1->next;
                                p2=p1->next;
                                p1->next=p2->next;
                                p2->next=NULL;
                        }
                        return p;

                }
        }
}

void traversal(nodeptr p)
{
        nodeptr p1;
        p1=p;
        printf("\nelements are:\n");
        while(p1->next!=p)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
        printf("%d",p1->info);
}
```

doubly linked list-insertion, deletion

Doubly Linked List

```
struct node
{
        int info;
        struct node *next, *prev;
};
typedef struct node *nodeptr;
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=NULL;
        p->prev=NULL;
        return p;
}
```

Create a Doubly Linked List:

```
void create()
{

    nodeptr p,p1,p2,p3;
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
            p1->next=p2;
            p2->prev=p1;
```

```
                p1=p2;
                p2=getnode();
                printf("\nEnter the number:");
                scanf("%d",&p2->info);
        }
        p3=p->next;
        p->next=NULL;
        p3->prev=NULL;
        first=p3;
        last=p1;
}
```

Traversal:

```
void traversal()
{
        nodeptr p1;
        p1=first;
        printf("\nelements are:\n");
        while(p1!=NULL)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
        printf("\nelements in reverse order\n");
        p1=last;
        while(p1!=NULL)
        {
            printf("<--%d",p1->info);
            p1=p1->prev;
        }

}
```

Insertion operation

Insert at 1. beginning 2. End 3. Given Position

```
void insert()
{
        nodeptr p1,p2,p3;
        int k,ch,i;
        p2=getnode();
        printf("\nenter inserted element:");
        scanf("%d",&p2->info);
        if(first==NULL)
```

```
        {
                first=p2;
                last=p2;
        }
        else
        {
                printf("\ninsert\n1.at beg\n2.at end\n3.insert at given position\nenter your
choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: p2->next=first;
                                first->prev=p2;
                                first=p2;
                                break;
                        case 2: last->next=p2;
                                p2->prev=last;
                                last=p2;
                                break;
                        case 3: printf("\nEnter the position to insert:");
                                scanf("%d",&k);
                                if(k==1)
                                {

                                        p2->next=first;
                                        first->prev=p2;
                                        first=p2;
                                }
                                else
                                {
                                        p1=first;
                                        for(i=1;i<k-1;i++)
                                                p1=p1->next;
                                        if(p1==last)
                                        {
                                                p1->next=p2;
                                                p2->prev=p1;
                                                last=p2;
                                        }
                                        else
                                        {
                                                p3=p1->next;
                                                p1->next=p2;
```

```
                                        p2->prev=p1;
                                        p2->next=p3;
                                        p3->prev=p2;
                                    }
                                }
                                break;

                }
        }
}
```

/* Insertion operations in DOUBLE Linked List */

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next, *prev;
};
typedef struct node *nodeptr;
nodeptr getnode();
void insert();
void traversal();
void create();
nodeptr first,last;
main()
{
        int choice;
        clrscr();
        create();
        traversal();
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("\n1.adding a node");
                printf("\n2.traversal\n");
                printf("3.exit\n*******************\nEnter your choice:");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1: insert();
                                traversal();
```

```
                        break;
                case 2: traversal();
                        break;
                case 3: exit(0);
            }
        }
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=NULL;
        p->prev=NULL;
        return p;
}
void create()
{

    nodeptr p,p1,p2,p3;
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
            p1->next=p2;
            p2->prev=p1;
            p1=p2;
            p2=getnode();
            printf("\nEnter the number:");
            scanf("%d",&p2->info);
    }
        p3=p->next;
        p->next=NULL;
        p3->prev=NULL;
        first=p3;
        last=p1;
}
void insert()
{
```

```
        nodeptr p1,p2,p3;
        int k,ch,i;
        p2=getnode();
        printf("\nenter inserted element:");
        scanf("%d",&p2->info);
        if(first==NULL)
        {
                first=p2;
                last=p2;
        }
        else
        {
                printf("\ninsert\n1.at beg\n2.at end\n3.insert at given position\nenter your
choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: p2->next=first;
                                first->prev=p2;
                                first=p2;
                                break;
                        case 2: last->next=p2;
                                p2->prev=last;
                                last=p2;
                                break;
                        case 3: printf("\nEnter the position to insert:");
                                scanf("%d",&k);
                                if(k==1)
                                {

                                        p2->next=first;
                                        first->prev=p2;
                                        first=p2;
                                }
                                else
                                {
                                        p1=first;
                                        for(i=1;i<k-1;i++)
                                                p1=p1->next;
                                        if(p1==last)
                                        {
                                                p1->next=p2;
                                                p2->prev=p1;
```

```
                                        last=p2;
                                }
                                else
                                {
                                        p3=p1->next;
                                        p1->next=p2;
                                        p2->prev=p1;
                                        p2->next=p3;
                                        p3->prev=p2;
                                }
                        }
                        break;

                }
        }
}
void traversal()
{
        nodeptr p1;
        p1=first;
        printf("\nelements are:\n");
        while(p1!=NULL)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
        printf("\nelements in reverse order\n");
        p1=last;
        while(p1!=NULL)
        {
            printf("<--%d",p1->info);
            p1=p1->prev;
        }

}
```

Deletion operation

Deletion at 1. Beginning 2. End 3. Given Position

```
void del( )
{

        int c,i,k;
        nodeptr p1,p2,p3;
        if(first==NULL)
```

```
{
        printf("\nlist is empty");

}
else if(first==last)
        first=last=NULL;
else
{
        printf("\nDelete\n1.at begining\n2.at end\n3.delete at given position");
        printf("\nEnter your choice:");
        scanf("%d",&c);
        switch(c)
        {
                case 1: p1=first;
                        first=first->next;
                        p1->next=NULL;
                        first->prev=NULL;
                        break;
                case 2: p1=last;
                        last=last->prev;
                        last->next=NULL;
                        p1->prev=NULL;
                        break;
                case 3: printf("\nenter the position to delete:");
                        scanf("%d",&k);
                        if(k==1)
                        {
                                p1=first;
                                first=first->next;
                                p1->next=NULL;
                                first->prev=NULL;
                        }
                        else
                        {
                                p1=first;
                                for(i=1;i<k;i++)
                                        p1=p1->next;
                                if(p1==last)
                                {
                                        last=last->prev;
                                        p1->prev=NULL;
                                        last->next=NULL;
                                }
```

```
                                        else
                                        {
                                                p2=p1->prev;
                                                p3=p1->next;
                                                p2->next=p3;
                                                p3->prev=p2;
                                                p1->next=NULL;
                                                p1->prev=NULL;
                                        }
                                }
                                break;


                        }
                }
}
```

/* Deletion operations in DOUBLE Linked List */

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next, *prev;
};
typedef struct node *nodeptr;
nodeptr getnode();
void del();
void traversal();
void create();
nodeptr first,last;
main()
{
        int choice;
        clrscr();
        create();
        traversal();
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("\n1.delete a node");
                printf("\n2.traversal\n");
```

```
                printf("3.exit\n*******************\nEnter your choice:");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1: del();
                                traversal();
                                break;
                        case 2: traversal();
                                break;
                        case 3: exit(0);
                }
        }
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=NULL;
        p->prev=NULL;
        return p;
}
void create()
{

    nodeptr p,p1,p2,p3;
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
                p1->next=p2;
                p2->prev=p1;
                p1=p2;
                p2=getnode();
                printf("\nEnter the number:");
                scanf("%d",&p2->info);
    }
        p3=p->next;
```

```
        p->next=NULL;
        p3->prev=NULL;
        first=p3;
        last=p1;
}
void del( )
{

        int c,i,k;
        nodeptr p1,p2,p3;
        if(first==NULL)
        {
                printf("\nlist is empty");

        }
        else if(first==last)
                first=last=NULL;
        else
        {
                printf("\nDelete\n1.at begining\n2.at end\n3.delete at given position");
                printf("\nEnter your choice:");
                scanf("%d",&c);
                switch(c)
                {
                        case 1: p1=first;
                                first=first->next;
                                p1->next=NULL;
                                first->prev=NULL;
                                break;
                        case 2: p1=last;
                                last=last->prev;
                                last->next=NULL;
                                p1->prev=NULL;
                                break;
                        case 3: printf("\nenter the position to delete:");
                                scanf("%d",&k);
                                if(k==1)
                                {
                                        p1=first;
                                        first=first->next;
                                        p1->next=NULL;
                                        first->prev=NULL;
                                }
```

```
                                else
                                {
                                        p1=first;
                                        for(i=1;i<k;i++)
                                                p1=p1->next;
                                        if(p1==last)
                                        {
                                                last=last->prev;
                                                p1->prev=NULL;
                                                last->next=NULL;
                                        }
                                        else
                                        {
                                                p2=p1->prev;
                                                p3=p1->next;
                                                p2->next=p3;
                                                p3->prev=p2;
                                                p1->next=NULL;
                                                p1->prev=NULL;
                                        }
                                }
                                break;


                        }
                }
}
void traversal()
{
        nodeptr p1;
        p1=first;
        printf("\nelements are:\n");
        while(p1!=NULL)
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
        printf("\nelements in reverse order\n");
        p1=last;
        while(p1!=NULL)
        {
            printf("<--%d",p1->info);
            p1=p1->prev;
```

```
        }

}
```

/* DOUBLY Linked List operations */

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int info;
        struct node *next, *prev;
};
typedef struct node *nodeptr;
nodeptr getnode();
void insert();
void del();
void traversal();
void create();
nodeptr first,last;
main()
{
        int choice;
        clrscr();
        create();
        traversal();
        while(1)
        {

        printf("\n*************************\n\n\tMENU\n**********************\n");
                printf("\n1.adding a node\n2.delete a node");
                printf("\n3.traversal\n");
                printf("4.exit\n*******************\nEnter your choice:");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1: insert();
                                traversal();
                                break;
                        case 2: del();
                                traversal();
                                break;
                        case 3: traversal();
```

```
                                        break;
                                case 4: exit(0);
                        }
                }
}
nodeptr getnode()
{
        nodeptr p;
        p=(nodeptr)malloc(sizeof(struct node));
        p->info=0;
        p->next=NULL;
        p->prev=NULL;
        return p;
}
void create()
{

    nodeptr p,p1,p2,p3;
    p1=getnode();
    p=p1;
    p2=getnode();
    printf("\nEnter the at end -999\n");
    printf("\nEnter the number:");
    scanf("%d",&p2->info);
    while(p2->info!=-999)
    {
                p1->next=p2;
                p2->prev=p1;
                p1=p2;
                p2=getnode();
                printf("\nEnter the number:");
                scanf("%d",&p2->info);
    }
        p3=p->next;
        p->next=NULL;
        p3->prev=NULL;
        first=p3;
        last=p1;
}
void insert()
{
        nodeptr p1,p2,p3;
        int k,ch,i;
```

```
        p2=getnode();
        printf("\nenter inserted element:");
        scanf("%d",&p2->info);
        if(first==NULL)
        {
                first=p2;
                last=p2;
        }
        else
        {
                printf("\ninsert\n1.at beg\n2.at end\n3.insert at given position\nenter your
choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: p2->next=first;
                                first->prev=p2;
                                first=p2;
                                break;
                        case 2: last->next=p2;
                                p2->prev=last;
                                last=p2;
                                break;
                        case 3: printf("\nEnter the position to insert:");
                                scanf("%d",&k);
                                if(k==1)
                                {

                                        p2->next=first;
                                        first->prev=p2;
                                        first=p2;
                                }
                                else
                                {
                                        p1=first;
                                        for(i=1;i<k-1;i++)
                                                p1=p1->next;
                                        if(p1==last)
                                        {
                                                p1->next=p2;
                                                p2->prev=p1;
                                                last=p2;
                                        }
```

```
                                        else
                                        {
                                                p3=p1->next;
                                                p1->next=p2;
                                                p2->prev=p1;
                                                p2->next=p3;
                                                p3->prev=p2;
                                        }
                                }
                                break;

                }
        }
}
void del( )
{

        int c,i,k;
        nodeptr p1,p2,p3;
        if(first==NULL)
        {
                printf("\nlist is empty");

        }
        else if(first==last)
                first=last=NULL;
        else
        {
                printf("\nDelete\n1.at begining\n2.at end\n3.delete at given position");
                printf("\nEnter your choice:");
                scanf("%d",&c);
                switch(c)
                {
                        case 1: p1=first;
                                first=first->next;
                                p1->next=NULL;
                                first->prev=NULL;
                                break;
                        case 2: p1=last;
                                last=last->prev;
                                last->next=NULL;
                                p1->prev=NULL;
                                break;
```

```
                case 3: printf("\nenter the position to delete:");
                        scanf("%d",&k);
                        if(k==1)
                        {
                                p1=first;
                                first=first->next;
                                p1->next=NULL;
                                first->prev=NULL;
                        }
                        else
                        {
                                p1=first;
                                for(i=1;i<k;i++)
                                        p1=p1->next;
                                if(p1==last)
                                {
                                        last=last->prev;
                                        p1->prev=NULL;
                                        last->next=NULL;
                                }
                                else
                                {
                                        p2=p1->prev;
                                        p3=p1->next;
                                        p2->next=p3;
                                        p3->prev=p2;
                                        p1->next=NULL;
                                        p1->prev=NULL;
                                }
                        }
                        break;


                }
        }
}

void traversal()
{
        nodeptr p1;
        p1=first;
        printf("\nelements are:\n");
        while(p1!=NULL)
```

```
        {
                printf("%d-->",p1->info);
                p1=p1->next;
        }
        printf("\nelements in reverse order\n");
        p1=last;
        while(p1!=NULL)
        {
            printf("<--%d",p1->info);
            p1=p1->prev;
        }

}
```