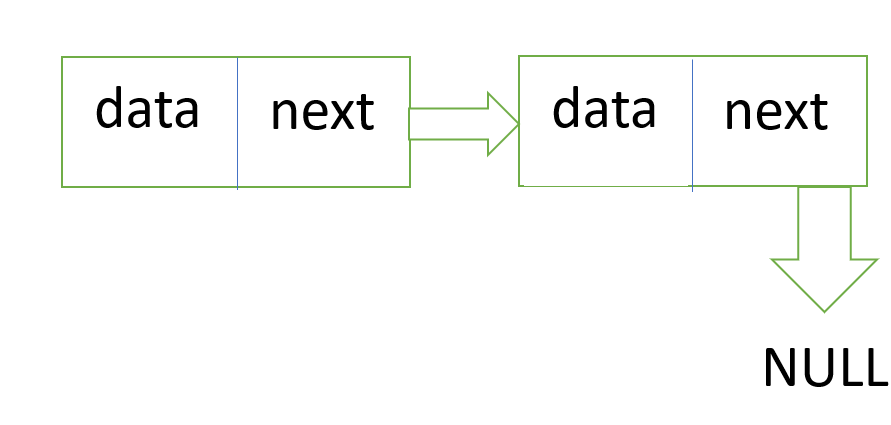
Definition:

Linked list is a linear data structure, in which elements are not stored in a contiguous location, rather they are linked using pointers. Linked list forms a series of connected nodes, where each node stores the data and the address of the next node.

VISUALIZATION:

ALGORITHMS

NODE CREATION:

Step:-1. Define a structure for the linked list node with two fields:

struct Node

{

int data;

struct Node\* next;

};

1.1 - An integer value field to store the data of the node.

1.2 - A pointer field to store the address of the next node in the linked list.

Step 2. Create a function named create Node that takes an integer value as a parameter and returns a pointer to the created node.

Step 3. Inside the create Node function:

3.1 - Allocate memory for a new node using the malloc function.

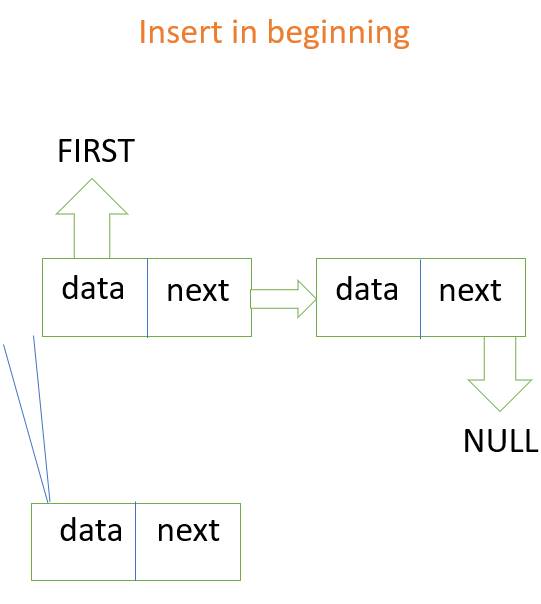
3.2- Assign the integer value to the data field of the new node.

3.3- Set the next node pointer to NULL since it is the last node in the list.

3.4 - Return the pointer to the newly created node.

Step 4. Use the create Node function to create new nodes as needed in the linked list.

VISUALIZATION



* Step 1**: memory allocation for new node**

**nn=(struct node \*)malloc(sizeof (nn))**

* Step 2: **scan the value**

**Input the value**

* Step 3: **assign the value and set link part**

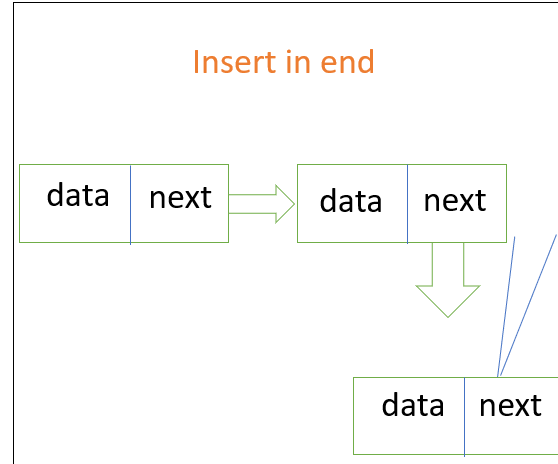
**nn->data=value**

**nn->next=NULL**

* Step 4:**connect the nodes and reset**

**nn->next=first;**

**first=nn;**



* Step1**: memory allocation for new node**

**nn=(struct node \*)malloc(sizeof (nn))**

* Step 2: **scan the value**

**Input the value**

* Step 3: **assign the value and set link part**

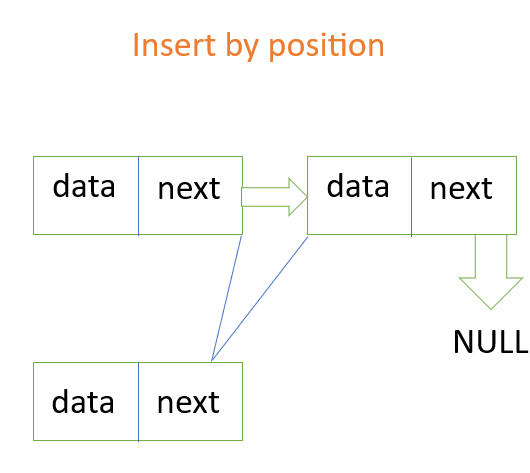
**nn->data=value**

**nn->next=NULL**

* Step 4:**connect the nodes and reset**

**last->next=first;**

**last=nn;**



* STEP 1: **struct node \*cur**,**\*prev**
* STEP 2: **initialize** 
  1. **Prev=NULL**
  2. **curr=first**
  3. **Count=1;**
* STEP 3: **input position**
* STEP 4: **create new node & assign the value and link**

**nn->data=value**

* STEP 5: **traversal**

5.**1 Prev=curr**

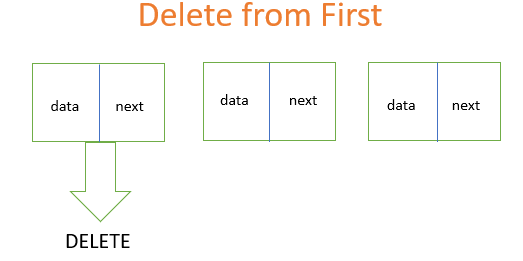
**5.2 curr=curr->next**

**5.3 count=count+1**

* STEP 6: Repeat step 5 till count<pos
* STEP 7: **connect nodes**

**prev->next=nn**

**nn->next=curr**



Step 1: **struct node \*temp**

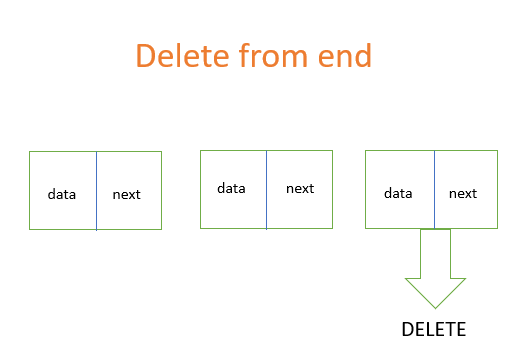
Step 2: **store first in temp variable**

**temp=first**

Step 3: **delete node and reset first node**

**first=first->next**

**free(temp)**



Step 1: **struct node \*temp**

Step 2: **store first in temp variable**

**temp=first**

Step 3: **while(temp->next!=last)**

**{**

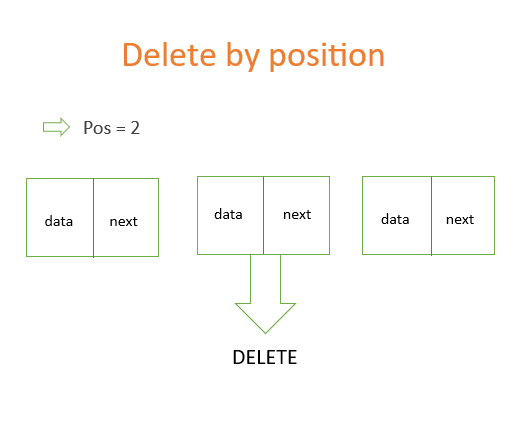
**temp=temp->next**

**}**

Step 4: **free(last)**

**last->temp**

**last->next=NULL**



* STEP 1: **struct node \*cur**,**\*prev**
* STEP 2: **initialize**

**Prev=NULL**

**curr=first**

**Count=1;**

* STEP 3: **input position**
* STEP 4: **traversal**

4.**1 Prev=curr**

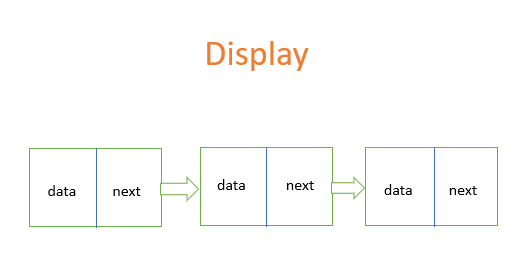
**4.2 curr=curr->next**

**4.3 count=count+1**

* STEP 5: Repeat step 5 till count<pos
* STEP 6: **display** **curr->data**
* STEP 6: **set link and deallocate memory**

**prev->next=curr->next**

**free(cur)**



Step 1**: take variable temp**

**Temp=first**

Step 2: **traverse**

**Display temp->data**

**temp=temp->next**

Step 3: **Repeat step 2 till temp->next!=NULL**

Step 4: **display last->data**

Code For SLL:

#include<stdio.h>

#include<stdlib.h>

**struct** node

{

**int** data;

**struct** node \*next;

}\*first=NULL,\*last=NULL,\*nn=NULL,\*temp,\*cur,\*pre;

**int** ch,x;

**void** **main**()

{

create();

**do**

{

printf("**\n**1.insert at first");

printf("**\n**2.insert at middle");

printf("**\n**3.insert at last");

printf("**\n**4.delete from first");

printf("**\n**5.delete from middle");

printf("**\n**6.delete from last");

printf("**\n**7.display");

printf("**\n**enter choice");

scanf("%d",&ch);

**switch**(ch)

{

**case** **1**:

insertf();

**break**;

**case** **2**:

insertm();

**break**;

**case** **3**:

insertl();

**break**;

**case** **4**:

deletef();

**break**;

**case** **5**:

deletem();

**break**;

**case** **6**:

deletelast();

**break**;

**case** **7**:

display();

}

}**while**(ch!=**8**);

}

**void** **create**()

{

**int** x=**0**;

printf("**\n**enter value (-1 for end) :");

scanf("%d",&x);

**while**(x!=-**1**)

{

nn=(**struct** node \*)malloc(**sizeof**(**struct** node));

nn->data = x;

nn ->next=NULL;

**if**(first == NULL)

{

first=nn;

last=nn;

}

**else**

{

last->next=nn;

last=nn;

}

printf("**\n**enter value (-1 for end ):");

scanf("%d",&x);

}

}

**void** **display**()

{

temp=first;

**while**(temp!=last)

{

printf("linked list elements are %d**\n**",temp->data);

temp=temp->next;

}

printf("linked list elements are %d**\n**",last->data);

}

**void** **insertf**()

{

printf("**\n**enter value :");

scanf("%d",&x);

nn=(**struct** node \*)malloc(**sizeof**(nn));

nn->data=x;

nn->next=first;

first=nn;

}

**void** **insertm**()

{

**int** pos,cnt;

printf("**\n**enter position");

scanf("%d",&pos);

cnt=**1**;

printf("enter value :");

scanf("%d",&x);

nn=(**struct** node \*)malloc(**sizeof**(nn));

nn->data=x;

pre=NULL;

cur=first;

**while**(cnt<pos)

{

pre=cur;

cur=cur->next;

cnt++;

}

pre->next=nn;

nn->next=cur;

}

**void** **insertl**()

{

printf("**\n**enter the value");

scanf("%d",&x);

nn=(**struct** node \*)malloc(**sizeof**(nn));

nn->data=x;

last->next=nn;

last=nn;

}

**void** **deletef**()

{

**if**(first==NULL)

{

printf("**\n**List is Empty");

}

**else**

{

printf("**\n**deleted element %d",first->data);

temp=first;

first=first->next;

free(temp);

}

}

**void** **deletem**()

{

**int** cnt,pos;

printf("**\n**enter pos");

scanf("%d",&pos);

cnt=**1**;

pre=NULL;

cur=first;

**while**(cnt<pos)

{

pre=cur;

cur=cur->next;

cnt++;

}

printf("deleted element %d",cur->data);

**if**(cur==last)

{

free(last);

last=pre;

last->next= NULL;

}

**else**

{

pre->next = cur->next;

free(cur);

}

}

**void** **deletelast**()

{

**if**(first==NULL)

{

printf("**\n**List is Empty!!!!");

}

**else**

{

printf("**\n**deleted element %d",last->data);

temp=first;

**while**(temp->next !=last)

{

temp=temp->next;

}

free(last);

last=temp;

last->next=NULL;

}

}

**CIRCULAR SINGLY LINKED LIST**

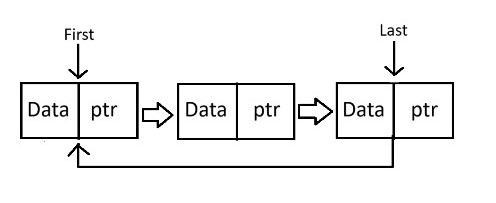
A circular linked list is a type of linked list in which the first and the last nodes are also connected to each other to form a circle.

There are basically two types of circular linked list:

1. Circular Singly Linked List

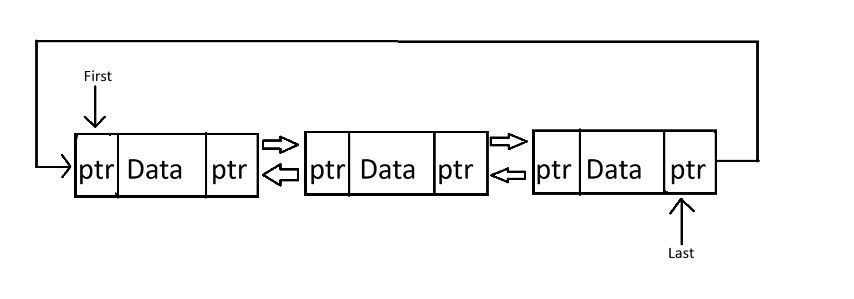
Here, the address of the last node consists of the address of the first node.

**Circular Singly Linked List Representation :**



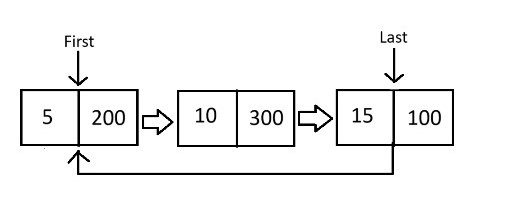
2. Circular Doubly Linked List

Here, in addition to the last node storing the address of the first node, the first node will also store the address of the last node.

**Circular Doubly Linked List Representation:**

**Algorithms of Circular Singly Linked List**

**Creation Algorithm :-**

****

**Step 1 :- Struct node{**

**int data;**

**struct node \*next;**

**};**

**Step 2 :-struct node first,\*last,\*nn.**

**Step 3 :- Initialize first=last=nn=Null.**

**Step 4 :- Input value .**



**Step 5 :- Allocate memory nn=(struct node\*)malloc(sizeof(struct node)**

**Step 6 :- Assign value and set link nn data = value.**

**Step 7 :- if(first== NULL)**

**first = nn;**

**last = nn;**

**Step 8 :- else connect nodes**

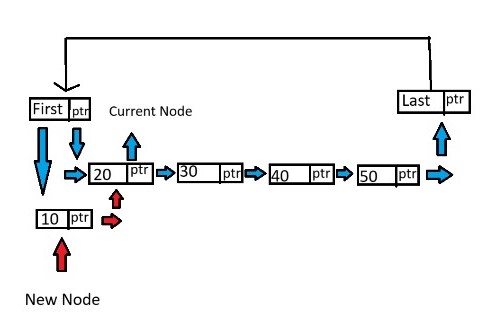
**last next = nn;**

**last = nn;**

**last next = first;**

**Step 9 :- Continue step 1 to 8 to create other nodes.**

**Insert Before First Algorithm :-**

****

**Step 1 :- Input value**



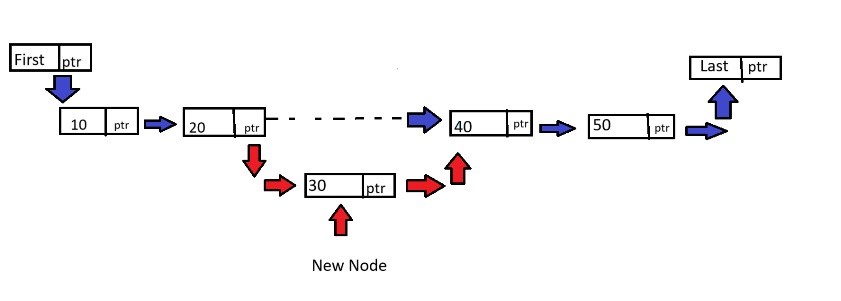
**Step 2 :- Allocate memory nn=(struct node\*)malloc(sizeof(struct node)**

**Step 3 :- Assign value and set link nn data = value.**

**Step 4 :- Now link the node at the begining.**

**nn next = first; first = nn;**

**Insert in Middle Algorithm :-**

****

**Step 1 :- create count and position variables of integer type**

**Step 2 :- Input position and set count to 1.**

**Step 3 :- Input value.**



**Step 4 :- Allocate memory nn=(struct node\*)malloc(sizeof(struct node)**

**Step 5 :- Assign value and set link nn data = value.**

**Step 6 :- Set cur pointer to first and prev pointer to null.**

**Step 7 :- start a while loop.**

**while(count != position)**

**{**

**prev = cur;**

**cur = cur next;**

**count ++;**

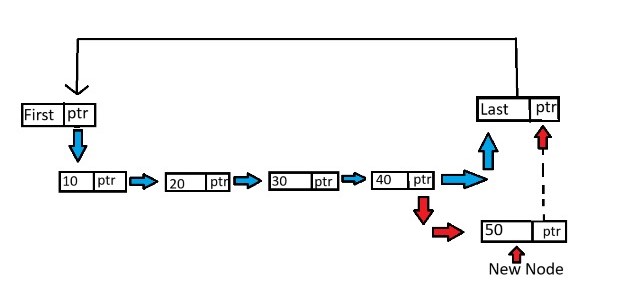
**}**

**Step 8 :- Now set the link between the new node and old nodes.**

**prev next = nn;**

**nn next = cur;**

**Insert at Last Algorithm :-**

****

**Step 1 :- Input value**

**Step 2 :- Allocate memory nn=(struct node\*)malloc(sizeof(struct node)**

**Step 3 :- Assign value and set link nn data = value.**

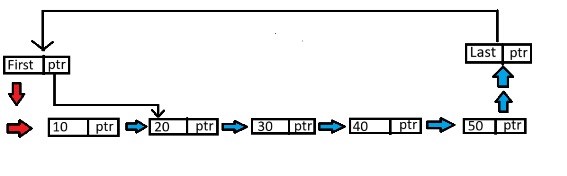
**Step 4 :- Now link the node at the last.**

**last next = nn;**

**last = nn;**

**nn next = first;**

**Delete First Algorithm :-**

****

**Step 1 :- Check if the link list is empty or not.**

**if(first == NULL)**

**display “list is empty”.**

**Step 2 :- else**

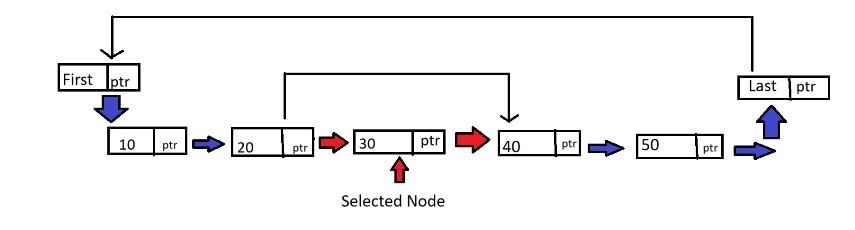
**print the value of the first link list.**

**first = first next;**

**temp = first;**

**free(temp);**

**Delete Middle Algorithm :-**

****

**Step 1 :- Check if the link list is empty or not.**

**if(first == NULL)**

**display “list is empty”.**

**Step 2 :- create count and position variables of integer type.**

**Step 3 :- Input position and set count to 1.**

**Step 4 :- Input value.**



**Step 5 :- Allocate memory nn=(struct node\*)malloc(sizeof(struct node).**

**Step 6 :- Set cur pointer to first and prev pointer to null.**

**Step 7 :- start a while loop.**

**while(count != position)**

**{**

**prev = cur;**

**cur = cur next;**

**count ++;**

**}**

**Step 8 :- check if current is at first or last**

**if(cur==first)**

**{**

**free(first);**

**first = cur next;**

**last next = first;**

**}**

**else if(cur == last)**

**{**

**free(last);**

**last=prev;**

**last next=first;**

**}**

**else**

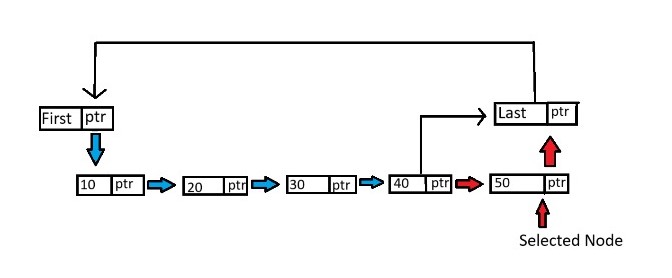
**{**

**prev next=cur next;**

**free(cur);**

**}**

**Delete Last Algorithm :-**

****

**Step 1 :- Check if the link list is empty or not.**

**if(first == NULL)**

**display “list is empty”.**

**Step 2 :- else**

**print the value of the last link list.**

**Step 5 :- use temp pointer**

**temp = first;**

**Step 6 :- start a while loop**

**while(temp != last)**

**{**

**temp = temp next;**

**}**

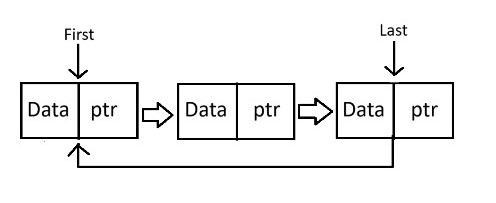
**Step 7 :- clear the memory of the deleted data and link the last the first.**

**free(last);**

**last = temp;**

**last next = first;**

**Display**

****

**Display From First Algorithm :-**

**Step 1 :- Check if the link list is empty or not.**

**if(first == NULL)**

**display “list is empty”.**

**Step 2 :- else use temp variable start it from the last.**

**temp = first.**

**Step 3 :- start a while loop and print the linked list inside the loop.**

**while(temp next != first)**

**{**

**display the values : temp data**

**temp = temp next;**

**}**

**Step 4 :- print the data in the temp one more time but outside the loop.**

**Display temp data.**

**Display From Last Algorithm :-**

**Step 1 :- Check if the link list is empty or not.**

**if(first == NULL)**

**display “list is empty”.**

**Step 2 :- else use temp variable start it from the last.**

**temp = last.**

**Step 3 :- start a while loop and print the linked list inside the loop.**

**while(temp next != last)**

**{**

**display the values : temp data**

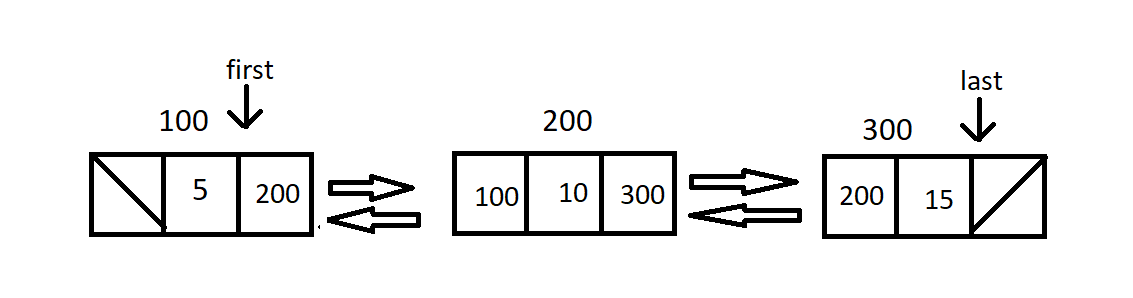
**temp = temp next;**

**}**

**Step 4 :- print the data in the temp one more time but outside the loop.**

**Display temp data.**

**Doubly linked list**

****VISUALIZATION

Here there are 2 pointers left and right pointing to left and right nodes of the respective nodes.

**PRE- REQUISITES:**

**struct node{**

**int data;**

**struct node \*left,\*right;**

**}\*first,\*last,\*nn;**

**Algorithm for insert before first in DDL:**

Step 1: Allocate memory to a new node

nn=(struct node\*)malloc(sizeof(struct node))

Step 2: Set value

nn->data=value

Step 3: connect nodes and reset first node

first->left=nn

nn->right=first

first=nn

Step 4: If you want to repeat go to step1

**Algorithm for insert after last in DDL:**

Step 1: Allocate memory to a new node

nn=(struct node\*)malloc(sizeof(struct node))

Step 2: Set value

nn->data=value

Step 3: connect nodes and reset last node

last->right=nn

nn->left=last

nn->right=NULL

last=nn

Step 4: If you want to repeat go to step1

**Algorithm for insert at position number in DDL:**

Step 1: Input position

Step 2: Intialize variable

prev=NULL

curr=NULL

count =1

Step 3: Allocate memory to a new node

nn=(struct node\*)malloc(sizeof(struct node))

Step 4: Assign value

nn->data=value

Step 5: Traversal

while(count<pos)

{

prev=curr

curr=curr->right

count++

}

Step 6: Connect nodes

(with 2 variables)

nn->left=prev

nn->right=curr

prev->right=nn

curr->left=nn

(with one variable)

while(count<pos)

{

curr=curr->right

count++

}

Connect nodes

nn->right=curr

nn->left=curr->left

curr->left->right=nn

curr->left=nn

Step 7: If you want to repeat go to step1

**Algorithm for delete first node in DDL:**

Step 1: initialize and declare a temp variable

temp=first

display first->data

Step 2: Reset first node

first=first->right

first->left=NULL

Step 3: Delete first node

free(temp)

Step 4: If you want to repeat go to step1

**Algorithm for delete last node in DDL:**

Step 1: initialize and declare a temp variable

temp=last

display last->data

Step 2: Reset last node

last=last->left

last->right=NULL

Step 3: Delete last node

free(temp)

Step 4: If you want to repeat go to step1

**Algorithm for delete at position number in DDL:**

Step 1: Input position

Step 2: Intialize variable

prev=NULL

curr=NULL

count=1

Step 3: Traversal

while(count<pos)

{

prev=curr

curr=curr->right

count++

}

Step 4: Delete a node & connect nodes

Display curr->data

prev->right=curr->right

curr->right->left=curr->left

free(curr)

Step 5: If you want to repeat go to step1

**Algorithm for display in DDL:**

**(first to last)**

Step 1: Intialize temp

temp=first

Step 2: while(temp!=NULL)

{

display temp->data

temp=temp->right

}

**Algorithm for display in DDL:**

**(last to first)**

Step 1: Intialize temp

temp=last

Step 2: while(temp!=NULL)

{

display temp->data

temp=temp->left

}

CODE:

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node \*right,\*left;

}\*last,\*first,\*nn,\*curr,\*temp,\*temp1;

int ch,pos,count=1,value,cnt=0;

void create()

{

     nn=(struct node\*)malloc(sizeof(struct node));

     printf("\n Enter the value of new node\n ");

     scanf("%d",&value);

     nn->data=value;

    if(first==NULL)

    {

        first=nn;

        last=nn;

        cnt++;

    }

}

void insert()

{

     nn=(struct node\*)malloc(sizeof(struct node));

     printf("\n Enter the value of new node\n ");

     scanf("%d",&value);

     nn->data=value;

    printf("Enter 1 to insert before first node:\n");

    printf("Enter 2 to insert after last node:\n");

    printf("Enter 3 to insert at position number:\n");

    printf("Enter choice:\n");

    scanf("%d",&ch);

    switch(ch)

    {

        case 1:

        first->left=nn;

        nn->right=first;

        first=nn;

        first->left=NULL;

        cnt++;

        break;

        case 2:

        last->right=nn;

        nn->left=last;

        last=nn;

        last->right=NULL;

        cnt++;

        break;

        case 3:

        printf ("Enter the position where you want to enter a node:");

        scanf("%d",&pos);

        curr=first;

        if(pos==cnt+1)

        {

            last->right=nn;

            nn->left = last ;

            last = nn;

            last->right=NULL;

            cnt++;

        }

        else if(pos>0 && pos<=cnt)

        {

        while(count<pos)

        {

            curr=curr->right;

            count++;

        }

        nn->right=curr;

        nn->left=curr->left;

        curr->left->right=nn;

        curr->left=nn;

        cnt++;

        }

        else if(pos==1)

        {

            nn->right=first;

            first->left=nn;

            first=nn;

            first->left=NULL;

            cnt++;

        }

        break;

        default:

        printf("Invalid Input");

        break;

    }

}

void delete1()

{

    int value;

    printf("Enter 1 to delete before first node:\n");

    printf("Enter 2 to delete after last node:\n");

    printf("Enter 3 to delete at position number:\n");

    printf("Enter choice:\n");

    scanf("%d",&ch);

    switch(ch)

    {

        case 1:

        printf("Deleted node:%d",first->data);

        first=first->right;

        free(first->left);

        first->left=NULL;

        cnt--;

        break;

        case 2:

        printf("Deleted node:%d",last->data);

        last=last->left;

        free(last->right);

        last->right=NULL;

        cnt--;

        break;

        case 3:

        printf("Enter position to be deleted at:\n");

        scanf("%d",&pos);

        curr=first;

        if(pos==cnt+1 || pos==cnt)

        {

            printf("Deleted node:%d",last->data);

            last=last->left;

            free(last->right);

            last->right=NULL;

            cnt--;

        }

         else if(pos==1)

        {

            printf("Deleted node:%d",first->data);

            first=first->right;

            free(first->left);

            first->left=NULL;

            cnt--;

        }

        else if(pos>0 && pos<cnt)

        {

        while(count<pos)

        {

            curr=curr->right;

            count++;

        }

        printf("Deleted node:%d",curr->data);

        curr->right->left=curr->left;

        curr->left->right=curr->right;

        free(curr);

        cnt--;

        }

        break;

        default:

        printf("Invalid Input");

        break;

    }

}

void display()

{

    printf("Enter 1 to display from first to last:\n");

    printf("Enter 2 to display from last to first:\n");

    printf("Enter choice:\n");

    scanf("%d",&ch);

    switch(ch)

    {

        case 1:

        temp=first;

        while(temp!=NULL)

        {

            printf("Linked elements are:%d\n",temp->data);

            temp=temp->right;

        }

        break;

        case 2:

        temp=last;

        while(temp!=NULL)

        {

            printf("Linked elements are:%d",temp->data);

            temp=temp->left;

        }

        break;

        default:

        printf("Invalid Input\n");

        break;

    }

}

int main()

{

     create();

    do

    {

        printf("Enter 1 to insert:\n");

        printf("Enter 2 to delete:\n");

        printf("Enter 3 to display:\n");

        printf("Enter choice:\n");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1:

                 insert();

            break;

            case 2:

                delete1();

            break;

            case 3:

                 display();

            break;

            default:

                 printf("Invalid Input");

            break;

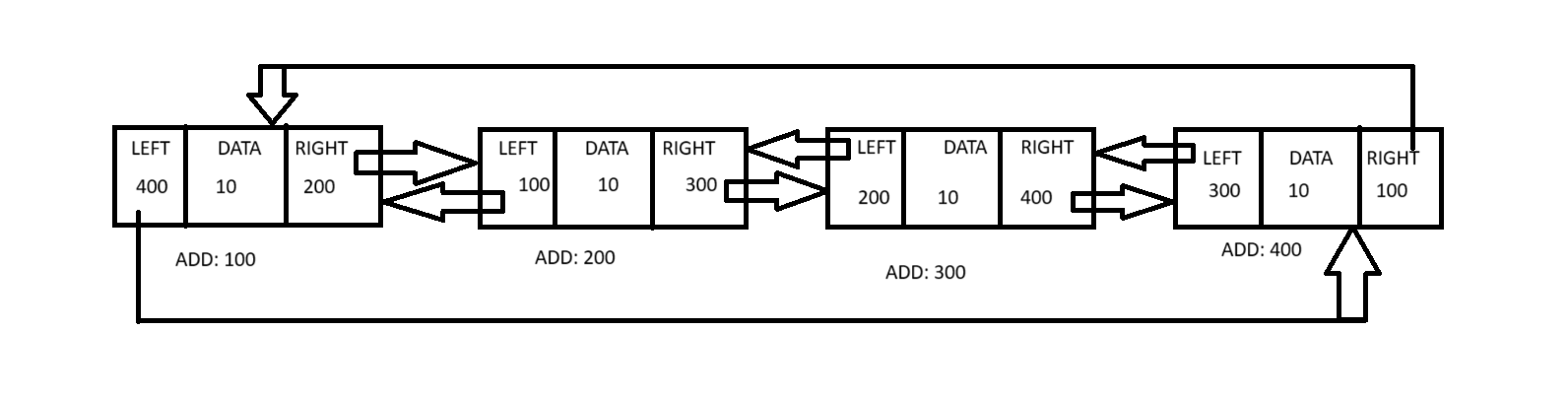
        }

    } while (ch!=4);

   return 0;

}

Circular Doubly Linked List

VISUALIZATION

This linked list has its first and last nodes linked to each other using left and right pointers

**Prerequisites**:

Struct node{

Int data;

Struct node \*left,\*right;

}\*first,\*last,\*nn;

**Algorithm 1: Insert at first node**

STEPS :

1.Allocate memory to new node:

Nn=(struct node\*) malloc(sizeof(nn));

2.Input nn-> data

3.Check if(first==NULL)

Then set first & last nodes:

First=last=nn;

Nn->right=left;

Nn->left=first

4.ELSE:

Nn->right=first;

Last->right=nn;

Nn->left=last;

First=nn;

**Algorithm 2: Insert at last node**

 STEPS :

1.Allocate memory to new node:

Nn=(struct node\*) malloc(sizeof(nn));

2.Input nn-> data

3.Check if(first==NULL)

Then set first & last nodes:

First=last=nn;

Nn->right=left;

Nn->left=first

4.ELSE:

Nn->right=first;

Last->right=nn;

Nn->left=last;

last=nn;

**Algorithm 3: Insert at position**

STEPS :

1.Allocate memory to new node:

Nn=(struct node\*) malloc(sizeof(nn));

2.Input nn-> data

3.Input pos

3.Check if(first==NULL)

Then set first & last nodes:

First=last=nn;

Nn->right=left;

Nn->left=first

5.ELSE:

Initialize

Count=1;

Temp=first;

5.1 temp=temp->right

5.2 count=count+1

Repeat steps 5.1 and 5.2 till count<pos

6.Set and link

Nn->right=temp

Nn->left=temp->left->right

Temp->left->right=nn

temp->left=nn

**Algorithm 4: Delete first node**

Steps:

1. Check if(first==NULL)

Display "EMPTY LIST"

1. Else

Set

First=first->right

Deallocate first->left

First->left=last

Last->right=first

**Algorithm 5: Delete last node**

Steps:

1. Check if(first==NULL)

Display "EMPTY LIST"

1. Else

Set

last=last->left

Deallocate last->right

last->right=first

**Algorithm 6: Delete by position**

 Steps:

1. Check if(first==NULL)

Display "EMPTY LIST"

1. Initialize count=1
2. Input pos
3. Else
4. Initialize

Temp=first

5.1 temp=temp->right

5.3 count=count+1

Repeat till count<pos

Set

Temp->right->left=temp->left

temp->left->right=temp->right

1. Deallocate temp

**Algorithm 7: DISPLAY first node to last node**

Steps:

1. Check if(first==NULL)

Display "EMPTY LIST"

1. Else
2. Nn=first;
3. Display nn->data
4. Nn=nn->right
5. Repeat steps 4 and 5 till nn!=first

**Algorithm 8: DISPLAY last node to first node**

Steps:

1. Check if(first==NULL)

Display "EMPTY LIST"

1. Else
2. Nn=last;
3. Display nn->data
4. Nn=nn->last
5. Repeat steps 4 and 5 till nn!=last

CODE:-

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int no;

    struct node \*left,\*right;

}\*l,\*f,\*nn,\*t;

int cnt=0;

void insert()

{

    int ch,pos,count;

    nn=(struct node\*) malloc(sizeof(struct node));

    printf("INSERT NO:");

    scanf("%d",&nn->no);

    printf("ENTER 1 for insert at first\n");

    printf("ENTER 2 for insert at last\n");

    printf("ENTER 3 for insert at position\n");

    scanf("%d",&ch);

    switch(ch)

    {

        case 1:

            if(f==NULL)

                f=l=nn;

            else

            {

                nn->right=f;

                f->left=nn;

                f=nn;

                f->left=l;

            }

            cnt++;

            break;

        case 2:

            if(f==NULL)

                f=l=nn;

            else{

                l->right=nn;

                nn->left=l;

                l=nn;

                l->right=f;

            }

            cnt++;

            break;

        case 3:

            printf("ENTER POSITION");

            scanf("%d",&pos);

            count=1;

            if(f==NULL)

            {

                f=l=nn;

                cnt++;

            }

            else if(pos==cnt+1)

            {

                l->right=nn;

                nn->left=l;

                l=nn;

                 l->right=f;

                cnt++;

            }

            else if(pos>0 && pos<=cnt)

            {

                t=f;

                while(pos>count)

                {

                    t=t->right;

                    count++;

                }

                nn->right=t;

                t->left->right=nn;

                nn->left=t->left;

                t->left=nn;

                cnt++;

            }

            else if(pos==1)

            {

                nn->right=f;

                f->left=nn;

                f=nn;

                f->left=l;

                cnt++;

            }

            else{

                printf("INVALID POSITION\n");

            }

            break;

        default:

                printf("INVALID INPUT");

                break;

    }

}

void delete()

{

    int ch,pos,count;

    printf("ENTER 1 for delete at first\n");

    printf("ENTER 2 for delete at last\n");

    printf("ENTER 3 for delete at position\n");

    scanf("%d",&ch);

    switch(ch)

    {

        case 1:

            if(f==NULL)

                printf("EMPTY LIST\n");

            else

            {

                f=f->right;

                free(f->left);

                f->left=l;

                l->right=f;

                cnt--;

            }

            break;

        case 2:

            if(f==NULL)

                printf("EMPTY LIST\n");

            else

            {

                l=l->left;

                free(l->right);

                l->right=f;

                f->left=l;

                cnt--;

            }

            break;

        case 3:

            printf("ENTER POSITION");

            scanf("%d",&pos);

            count=1;

            if(f==NULL)

            {

                printf("EMPTY LIST \n");

            }

            else if(pos==cnt+1)

            {

                l=l->left;

                free(l->right);

                l->right=f;

                f->left=l;

                cnt--;

            }

            else if(pos>0 && pos<=cnt)

            {

                t=f;

                while(pos>count)

                {

                    t=t->right;

                    count++;

                }

                t->right->left=t->left;

                t->left->right=t->right;

                free(t);

                cnt--;

            }

            else if(pos==1)

            {

                f=f->right;

                free(f->left);

                f->left=l;

                l->right=f;

                cnt--;

            }

            else

            {

                printf("INVALID POSITION\n");

            }

            break;

        default:

                printf("INVALID INPUT");

                break;

    }

}

void display()

{

    if(f!=NULL)

    {

        int ch;

        printf("ENTER 1 to display first to last\n");

        printf("ENTER 2 to display last to first\n");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1:

                t=f;

                do

                {

                    printf("DATA IS: %d\n",t->no);

                    t=t->right;

                }while(t!=f);

                break;

            case 2:

                t=l;

                do

                {

                    printf("DATA IS: %d\n",t->no);

                    t=t->left;

                }while(t!=l);

                break;

            default:

                printf("INVALID INPUT");

                break;

        }

    }

}

void main()

{

    int ch;

    while(1)

    {

        printf("\nENTER 1 for insert\n");

        printf("ENTER 2 for delete\n");

        printf("ENTER 3 for display\n");

        printf("ENTER 4 to exit");

        scanf("%d",&ch);

        switch(ch)

        {

            case 1:

                insert();

                break;

            case 2:

                delete();

                break;

            case 3:

                display();

                break;

            case 4:

                exit(0);

            break;

            default:

                printf("INVALID INPUT");

                break;

        }

    }

}