**Unit-4 Linked List.**

* **Linked List : -**
* Linked list is a linear data structure that contains sequence of elements such that each element links to its next element in the sequence. Each element in a linked list is called as **"Node"**.



* Thus, in a Linked list, data (actual content) and link (to point to the next data) both are required to be maintained.
* A node in linked list is usually a structure in C and it is declared as : -

struct Node

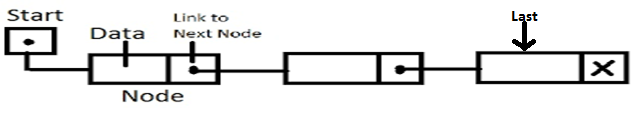
{

int data;

struct Node \*next;

}\*head = NULL;

* Each node has two parts DATA and LINK.
* The LINK part of the last node is of the iist contains a NULL value indicating the end of the list. The beginning of the list is indicated by pointer FIRST. Similarly, the end of the list can be indicated by a pointer called LAST.



**Advantages of Linked List**:

1. Linked Lists are Dynamic Data Structures: Unlike array, linked list can grow or shrink during the execution of a program.
2. Efficient Memory Utilization: Here memory is not defined (pre-allocated). Memory is allocated whenever it is required. And it is de-allocated (removed) when it is no longer needed.
3. Insertion and Deletions are Easier **and Efficient:** Linked lists provide flexibility in inserting a data item at a specified position and deletion of a data item from the given position.
4. Easy for Complex Application: Many complex applications can be easily carried out with linked list.

**Disadvantages of Linked List:**

1. More Memory Required: A linked list element requires more memory space because it also has to store address of next element in the list.
2. Time Consuming: Access to an arbitrary data item is little bit cumbersome and also time consuming.
3. Direct Access is NOT possible. Thus to access a particular element it is mandatory to traverse all elements before it.

**Basic terminologies of linked list..**

* Node: each item in Linked list is called as nodes that contains 2 fields, data and address/ next pointer/ link.
* Link/Address Field:- Address field in node is used to keep address of next node.
* Data/ information :- Data field in node is used to hold data inside linked list.
* Pointer :- it is the variable that stores the address of other variable.
* Null pointer: - the link field of the last node points to NULL rather than a valid node, this indicates the end of the linked list.
* External pointer :- it is the pointer to the very first node of the link list that enables us to access the entire list.
* Empty list :- if the nodes are NOT present in the linked list then it is called the empty list.

**Types of linked list…..**

****

**Single Linked List**

1. In this type of Linked List two **successive nodes** are linked together in **linear fashion** .
2. Each Node contain **address of the next node** to be followed.
3. In Singly Linked List only Linear or Forward **Sequential  movement is possible**.
4. Elements are accessed sequentially , **no direct access** is allowed.
5. Single linked list is a sequence of elements in which every element has link to its next element in the sequence.
6. In any single linked list, the individual element is called as **"Node"**. Every **"Node"** contains two fields, **data** and **next**. The **data** field is used to store actual value of that node and next field is used to store the address of the next node in the sequence.



**Advantages** of Singly Linked List:

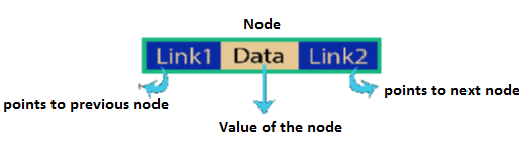
1. Accessibility of a node in the forward direction is easier.
2. Insertion and deletion of nodes are easier.

**Disadvantages of singly linked list:**

1. Accessing the preceding node of a current node is not possible as there is no backward traversal.
2. Accessing a node is time consuming.

**Doubly Linked List : Traverse Bi-directional**

* In Doubly Linked List, **each node contain two address fields** .
* One address field for storing address of **next node** to be followed and second address field contain address of **previous node** linked to it.
* So **Two way access is possible** i.e. We can start accessing nodes from start as well as from last .
* Like Singly Linked List also only Linear but Bidirectional  **Sequential  movement is possible**.
* Elements are accessed sequentially, **no direct access** is allowed.
* Double linked list is a sequence of elements in which every element has links to its previous element and next element in the sequence.

Advantages of Doubly Linked List: **BEE**

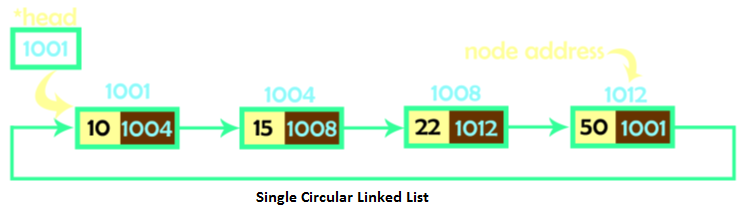
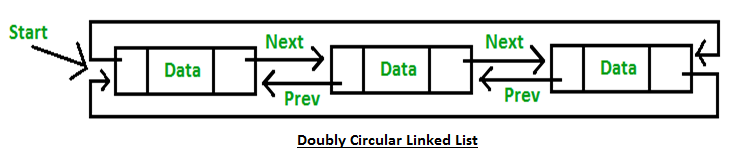
1. Doubly linked list with header node, most of the problems can be solved very easily and effectively.
2. Insertion and deletion are simple as compared to other lists.
3. Efficient utilization of memory, no wastage of memory as in Sequential representation.
4. Bidirectional (both forward and backward) traversal helps in efficient and very easily modes can be accessible.
5. Multiple stacks can be represented efficiently i.e., PUSH and POP operations are easier and efficient.
6. Doubly linked list are extensively used in trees because hierarchical structure of the tree can be easily represented.

**Disadvantages of Doubly Linked List:**

1. A nodes in a linked requires more memory than a corresponding element in an array representation.
2. Each node requires two pointers (links) one is forward link and the other backward link requires additional storage for each field.

**Circular Linked List**

* Circular linked list is a sequence of elements in which every element has link to its next element in the sequence and the last element has a link to the first element in the sequence.
* Circular Linked List is Divided into **2 Categories** .
  + Singly Circular Linked List
  + Doubly Circular Linked List
* In Circular Linked List Address field of **Last node** contain address of “**First Node**“.
* In short **First Node and Last Nodes** are adjacent.
* Linked List is made circular by linking first and last node, so it looks like **circular chain** [ shown in Following diagram ].
* **Two way access is possible** only if we are using “**Doubly Circular Linked List**”
* **Sequential  movement is possible**
* **No direct access** is allowed.

**Advantages of Circular Linked List:**

1. If we are at a node, then we can go to any node. But in linear linked list it is not possible to go to previous node.
2. It saves time when we have to go to the first node from the last node. It can be done in single step because there is no need to traverse the in between nodes. But in doubly linked list, we will have to go through in between nodes.

**Disadvantages of Circular Linked List:**

1. It is not easy to reverse the linked list.
2. If proper care is not taken, then the problem of infinite loop can occur.
3. If we at a node and go back to the previous node, then we can not do it in single step. Instead we have to complete the entire circle by going through the in between nodes and then we will reach the required node.

**Operations on Single Linked List : -**

1. Creating a Linked List.
2. Displaying the contents of Linked List.
3. Inserting an element into Linked List.
4. Deleting an element from Linked List.
5. Searching a node in the linked list.
6. Traverse.

**Creating linked list.**

**Step 1 :Define Node Structure**

struct node

{

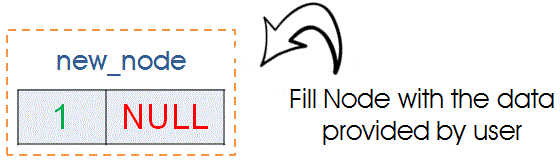
int data;

struct node \*next;

}\*start=NULL;

**Step 2 : Create Node using Dynamic Memory Allocation**

new\_node=(**struct** node \*)malloc(**sizeof**(**struct** node));



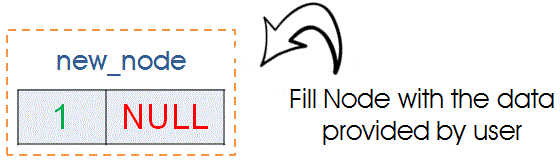
**Step 3 : Fill Information in newly Created Node.** Make its Next Field as NULL.

printf(“\n Enter the data : ");

scanf("%d",&val);

new\_node->data=val;

new\_node->next=NULL;



**Step 4 : Creating Very First Node**

If node created in the above step is very first node then we need to assign it as Starting node.

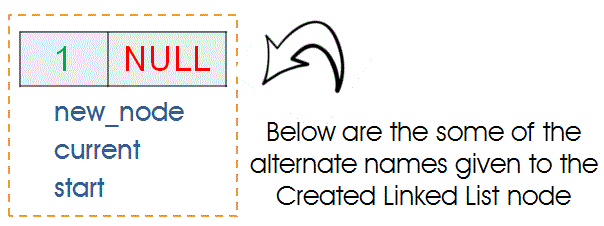
if(start == NULL)

{

start = new\_node;

curr = new\_node;

}



**Step 5 : Creating Second or nth node**.

else

{

current->next = new\_node;

current = new\_node;

}



**Traversing & display single LL : -**

Traversing linked list means visiting each and every node of the Singly linked list.

struct node \*temp;

temp = start; //Move to First Node

while(temp!=NULL)

{

printf(“%d”, temp->data);

temp = temp->next; //Move Pointer to Next Node

}

Program for creation and display of dynamic linked list.

#include <stdio.h>

#include <stdlib.h>

void display();

struct node

{

int info;

struct node \*link;

}\*start=NULL;

int main(void)

{

setbuf(stdout, NULL);

int data;

char ch;

struct node \*q, \*tmp;

do

{

printf("Enter element :\n");

scanf("%d", &data);

tmp=malloc(sizeof(struct node));

tmp->info=data;

tmp->link=NULL;

if(start==NULL) //insertion of first node

start=tmp;  
else //insertion of subsequent nodes

{

q=start; while(q->link!=NULL)

q=q->link; q->link=tmp;

}

printf("Do you want to insert more elements?");

scanf(" %c",&ch); //use a space before %c to clear from stdin }while(ch==,y'||ch==,Y");

display();

return 8;

}

void display()

{

struct node \*q; if(start==NULL)

printf("List is empty!!\n"); else

printf("\*\*\*\* Elements in Linked List \*\*\*\*\n");

q=start;

while(q!=NULL)

{

printf("%d\t",q->info);

q=q->link;

}

}

}

**Search Particular Element : Singly Linked List : -**

* Searching is a process to search the first occurrence of the given item in the list and to return the address of the node. The search operation helps to find an element in the linked list.
* Steps are :

1. Traverse the list sequentially starting from the first node.
2. Return the location of the search node as soon as a match is found.
3. Return a search failure notification if the entire list is traversed without any match.

int search(int num)

{

int flag = 0;

struct node \*temp;

temp = start;

while(temp!=NULL)

{

if(temp->data == num)

return(1); //Found

temp = temp->next;

}

if(flag == 0)

return(0); // Not found

}

**Inserting an element in single LL : -**

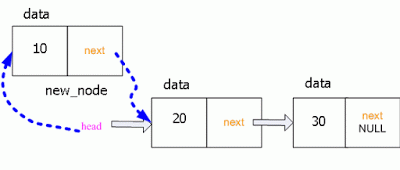
* In a single linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list.

**Inserting At Beginning of the list L –**

**Steps are : -**

1. Create New Node
2. Fill Data into “Data Field“
3. Make it’s “Pointer” or “Next Field” as NULL
4. Attach This newly Created node to Start
5. Make newnode as Starting node



void insert\_at\_beg()

{

struct node \*new\_node,\*current;

new\_node=(struct node \*)malloc(sizeof(struct node));

if(new\_node == NULL)

printf("nFailed to Allocate Memory");

printf("nEnter the data : ");

scanf("%d",&new\_node->data);

new\_node->next=NULL;

if(start==NULL)

{

start=new\_node;

current=new\_node;

}

else

{

new\_node->next=start;

start=new\_node;

}

}

**Insert node at Last Position : Singly Linked List : -**

**Steps are : -**

1. Create New Node
2. Fill Data into “Data Field“
3. Make it’s “Pointer” or “Next Field” as NULL
4. Node is to be inserted at Last Position so we need to traverse SLL upto Last Node.
5. Make link between last node and newnode.

****

void insert\_at\_end()

{

struct node \*new\_node,\*current;

new\_node=(struct node \*)malloc(sizeof(struct node));

if(new\_node == NULL)

printf("nFailed to Allocate Memory");

printf("nEnter the data : ");

scanf("%d",&new\_node->data);

new\_node->next=NULL;

if(start==NULL)

{

start=new\_node;

current=new\_node;

}

else

{

temp = start;

while(temp->next!=NULL)

{

temp = temp->next;

}

temp->next = new\_node;

}

}

**Insert node at given position : Singly Linked List : -**

** **

void insert\_mid()

{

int pos,i;

struct node \*new\_node,\*current,\*temp,\*temp1;

new\_node=(struct node \*)malloc(sizeof(struct node));

printf("nEnter the data : ");

scanf("%d",&new\_node->data);

new\_node->next=NULL;

if(start==NULL)

{

start=new\_node;

current=new\_node;

}

else

{

printf(“enter index after which element to be inserted:”)

scanf(“%d”, &pos);

temp = start;

for(i=1;i< pos-1;i++)

{

temp = temp->next;

}

temp1=temp->next;

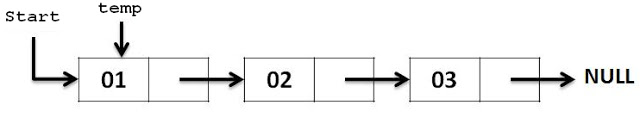
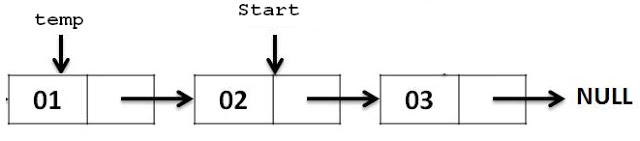
temp->next = new\_node;

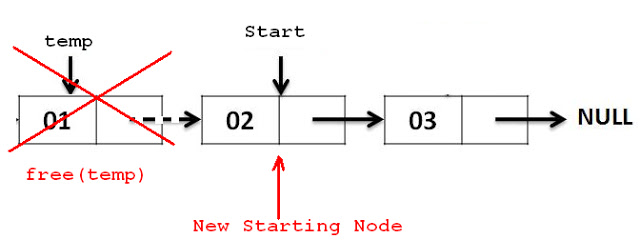
new\_node->next=temp1;

}

}

**Delete Node from First Position: Singly Linked List : -**

** **

****

void del\_beg()

{

struct node \*temp;

temp = start;

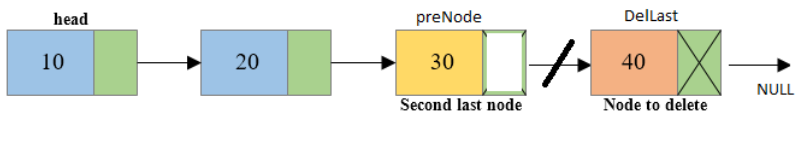
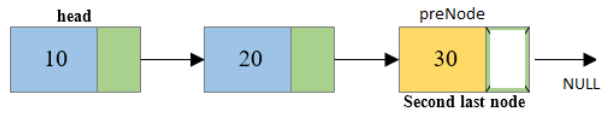
start = start->next;

free(temp);

printf("nThe Element deleted Successfully ");

}

**Delete Node from last Position : Singly Linked List : -**

** **

void LastNodeDeletion()

{

struct node \*Last, \*preNode;

if(stnode == NULL)

{

printf(" There is no element in the list.");

}

else

{

Last = stnode;

preNode = stnode;

/\* Traverse to the last node of the list\*/

while(Last->nextptr != NULL)

{

preNode = Last;

Last = Last->nextptr;

}

if(Last == stnode)

{

stnode = NULL;

}

else

{

/\* Disconnects the link of second last node with last node \*/

preNode->nextptr = NULL;

}

/\* Delete the last node \*/

free(Last);

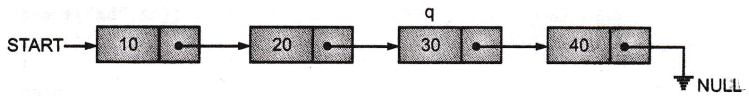
}

}

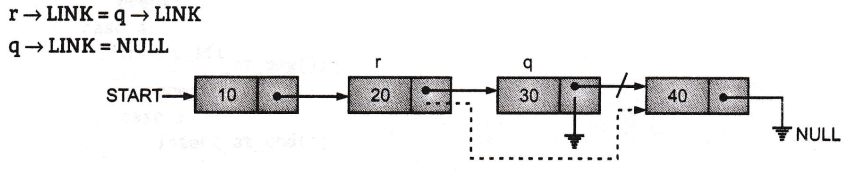
**Deleting the in between node (i.e. specified node):**

Steps are : -

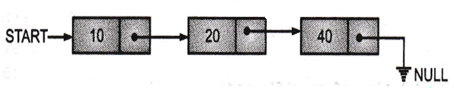
1. Traverse the list till the node to be deleted is reached and name it as q

****

1. Now change the link field of the node one before q to point to the node where q's link field is pointing. After that make q's link field to point to null with the following statements.



1. Now free the node q free (q). After which the list will look like…

****

#include <stdio.h>  
#include <malloc.h>  
void display();  
struct node  
{

int info;

struct node \*link;

};

struct node n4={40,NULL};

struct node n3={30,&n4};

struct node n2={20,&n3};

struct node nl={10,&n2};

struct node\* start=&nl;

int main()

{

int data;

struct node \*r,\*q;

setbuf(stdout, NULL);

printf("Linked List before deletion of specific node\n");

display();

printf("\nEnter the element to be deleted :\n");

scanf("%d",&data);

r=start;

q=start;

while(q!=NULL)

{

if(q->info==data)

{

r->link=q->link;

free(q);

break ;

}

r=q;

q=q->link;

}

printf("Linked List after deletion of specific node\n");

display();

}

void display()

{

struct node \*q=start;

while(q!=UNULL)

{

printf(,,%d\t", q->info);

q=q->link;

}

}

Output:

Linked List before deletion of specific node : -10 20 30 40

Enter the element to be deleted :30

Linked List after deletion of specific node: 10 20 40

**Operations on Doubly linked list : -**

1. **Create**
2. **Insertion**
3. **Deletion**

**Creating doubly LL : -**

**Step 1 :Define Node Structure**

struct node

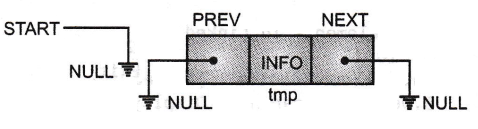
{

int info;

struct node\* next; struct node\* prev;

}\*start=NULL;

**Step 2 :Since, initially no nodes are available in the list. So start will point to NULL. Now create a new node using following function.**

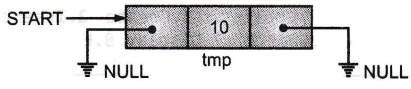
struct node \*tmp

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

tmp -> next = NULL

tmp -> prev = NULL

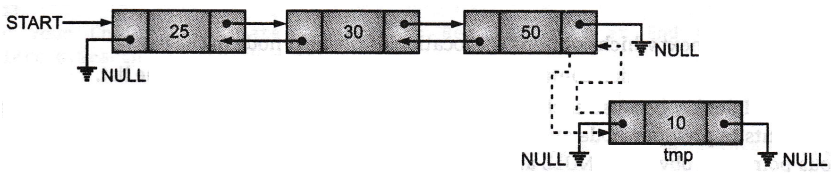
**Step 3: Since, here start is pointing to NULL as there are no elements. So tmp is the first node in list make start to point to tmp.**

If(start==NULL)

Start=tmp;

**Step 4: If some elements are already present in the list then we have to traverse till the last node and add the newly created node at the end with following statement:**

q = start

while(q->next!=NULL) 

q=q->next;

q->next=tmp

tmp->prev=q

**Program to creating a Doubly Linked List (DLL).**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int info;

struct node\* next;

struct node\* prev;

}\*start=NULL;

int main()

int data;

setbuf(stdout, NULL);

struct node \*q.,\*tmp;

printf("[\nEnter](file:///nEnter) the element to be inserted :\n");

scanf("%d",&data);

tmp=malloc(sizeof(struct node));

tmp->info=data;

tmp->prev=NULL;

tmp->next=NULL;

if(start == NULL)

start = tmp;

printf("\*\*\*\* Elements in Doubly Linked List \*\*\*\*\n");

q=start;

while(q!=NULL)

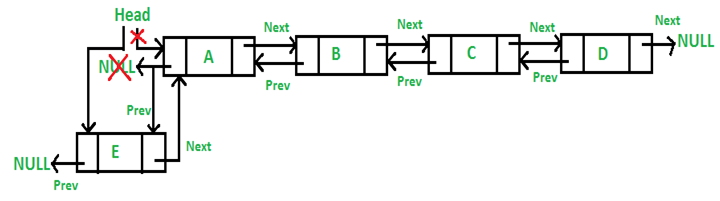
{

printf("%d\t", q->info);

q=q->next;

}

}

**Insert node at start in Doubly Linked List:**

If (start==NULL)

start=temp

else

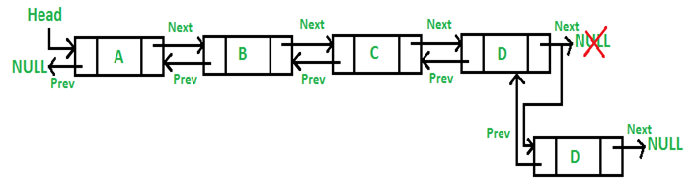
{

start->prev=temp;

temp->next=start;

start=temp;

}



**Insert node at end in Doubly Linked List : -**

If (start==NULL)

start=temp

else

{

q=start

While(q->next!=NULL)

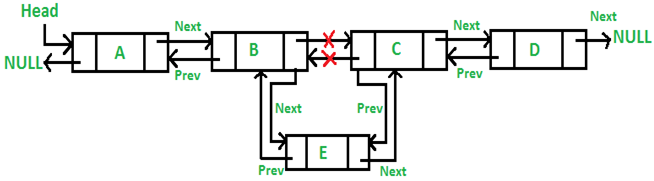
q=q->next;

q->next=temp;

temp->prev=q;

}

**Insert node in between after given location in Doubly Linked List :.**



If (start==NULL)

start=temp

else

{

printf(“enter index after which the element to be inserted:”);

scanf(“%d”,index);

q=start

for(i=0;i<index;i++)

{

q=q->next;

if(q==NULL)

{

printf(“there are less elements”)

return;

}

}

temp->next=q->next;

If(q->next!=NULL)

{

q->next->prev=temp;

}

q->next=temp;

temp->prev=q;

}

**Doubly Linked List : Delete first node : -**

* step 1: Check whether list is Empty (head == NULL)
* Step 2: If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.
* Step 3: If it is Not Empty then, define a Node pointer 'temp' and initialize with head.
* Step 4: Check whether list is having only one node (temp → next == NULL)
* Step 5: If it is TRUE then set head = NULL and delete temp (Setting Empty list conditions)
* Step 6: If it is FALSE then set head = temp → next, and delete temp.

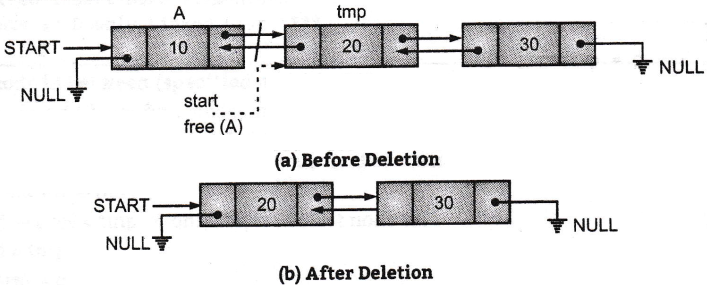
A=start;

If(A->next !=NULL)

Start->next->prev=NULL;

Start=start->next;

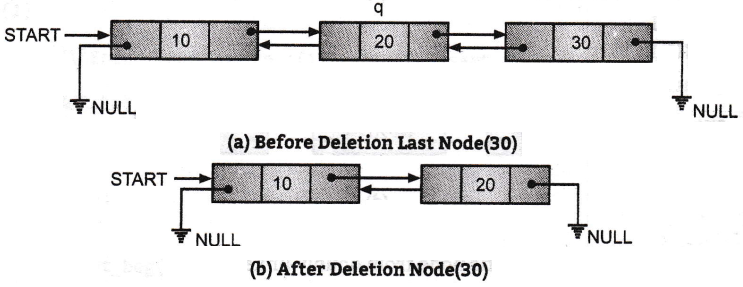
Free(A)



**Doubly Linked List : Delete last node : -**

Struct node \*temp,\*q;

q=start;

while(q->next->next !=NULL)

{

q=q->next;

}

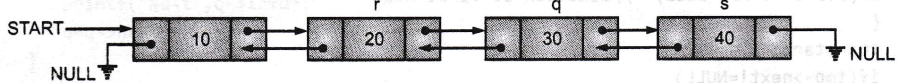
temp=q->next;

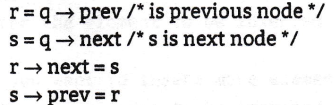
q->next=NULL;

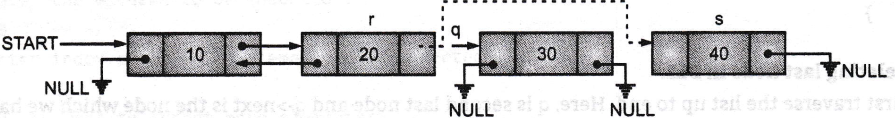
Free(temp)

**Deleting intermediate node in DLL : -**

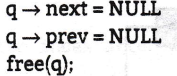
**Step 1:** Let 'q' be node to be deleted. We can point to q's previous, as well as next node, after the deletion process q's previous node must be connected to its next node.

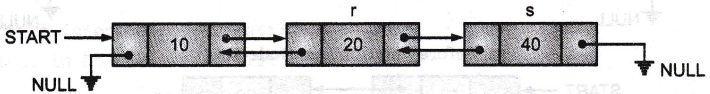


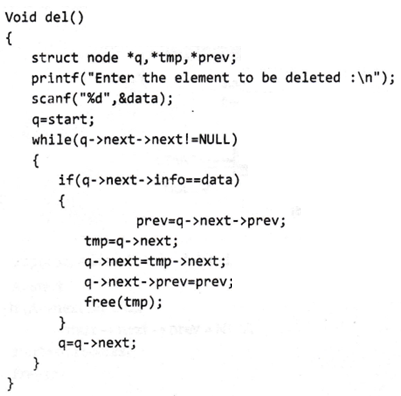




**Step 2:**





****

**Menu driven program for doubly linked list (create, insert, delete, display) : -**

**#include <stdio.h>**

**#include <malloc.h>**

**#include <process.h>**

**void create();**

**void insert\_at\_beg();**

**void insert\_at\_end();**

**void insert\_after\_pos();**

**void del();**

**void search();**

**void display();**

**struct node**

**{**

**int info;**

**struct node\* next;**

**struct node\* prev;**

**}\*start=NULL;**

**int data, item, n1, pos, i, m;**

**int main()**

**{**

**int n;**

**setbuf( stdout, NULL);**

**printf("\n\*\*\*\*Doubly Linked List\*\*\*\*\*\n");**

**printf("\n1.Create\n2.Insert at Beginning\n3.Insert at End\n4.Insert After Position\n5.Delete\n 6.Display\n 7.Exit\n");**

**while(1)**

**{**

**printf("\n Enter Your Choice :(1.Create 2.Insert at Beg. 3.Insert at End 4.Insert after Pos. 5.Delete 6.Display 7.Exit)\n");**

**scanf("%d",&n);**

**switch(n)**

**{**

**case 1:**

**create();break;**

**case 2:**

**insert\_at\_beg();break;**

**case 3:**

**insert\_at\_end(); break;**

**case 4:**

**insert\_after\_pos(); break;**

**case 5:**

**del(); break;**

**case 6:**

**display(); break;**

**case 7:**

**exit(e);**

**default:**

**printf("\nWrong Choice !!\n");**

**}**

**}**

**return 0;**

**}**

**void create()**

**{**

**int data;**

**struct node \*tmp;**

**printf("\n Enter the first element to be inserted :\n");**

**scanf("%d",&data);**

**tmp=malloc(sizeof(struct node));**

**tmp->info=data;**

**tmp->prev=NULL;**

**tmp->next=NULL;**

**if(start == NULL)**

**start-=tmp;**

**display();**

**}**

**void insert\_at\_beg()**

**{**

**int data;**

**struct node \*tmp;**

**printf("\n Enter the element to be inserted :\n");**

**scanf("%d",&data);**

**tmp=malloc(sizeof(struct node));**

**tmp->info=data;**

**tmp->prev=NULL;**

**tmp->next=NULL;**

**if(start == NULL)**

**start = tmp;**

**else**

**{**

**start->prev = tmp;**

**tmp->next = start;**

**start = tmp;**

**}**

**display();**

**}**

**void insert\_at\_end()**

**{**

**int data;**

**struct node \*q,\*tmp;**

**printf("\n Enter the element to be inserted :\n");**

**scanf("%d%&data);**

**tmp=malloc(sizeof(struct node));**

**tmp->info=data;**

**tmp->prev=NULL;**

**tmp->next=NULL;**

**if(start == NULL)**

**start = tmp;**

**else**

**{**

**q=start;**

**while(q->next != NULL)**

**q = q->next; // Go To last Node**

**q->next = tmp;**

**tmp->prev = q;**

**}**

**display();**

**}**

**void insert\_after\_pos()**

**{**

**int data;**

**struct node \*q,\*tmp;**

**tmp=malloc(sizeof(struct node));**

**printf("\nEnter the element to be inserted :\n");**

**scanf("%d",&data);**

**tmp->info=data;**

**tmp->prev=NULL;**

**tmp->next=NULL;**

**if(start==NULL)**

**{**

**start=tmp;**

**}**

**else**

**printf("Enter index after which element to be inserted :\n");**

**scanf("%d",&pos);**

**q=start;**

**for(i=0;i<pos;i++)**

**{**

**q = q->next;**

**}**

**tmp->next = q->next;**

**if(q->next!=NULL)**

**{**

**q->next->prev=tmp;**

**}**

**q->next = tmp;**

**tmp->prev=q;**

**display();**

**}**

**}**

**void del()**

**{**

**struct node \*tmp,\*q,\*prev;**

**printf("Enter the element to be deleted :\n");**

**scanf("%d", &data);**

**if(start->info==data) //deletion of first node**

**{**

**tmp=start;**

**if(tmp->next!=NULL)**

**{**

**start->next->prev=NULL;**

**}**

**start=start->next;**

**free(tmp);**

**display();**

**return;**

**}**

**q=start;**

**while(q->next->next!=NULL) //deletion of middle node**

**{**

**if(q->next->info==data)**

**{**

**prev=q->next->prev;**

**tmp=q->next;**

**q->next=tmp->next;**

**q->next->prev=prev;**

**free(tmp);**

**displayQ;**

**return;**

**}**

**q=q->nextj**

**}**

**if(q->next->info==data) //deletion at end**

**{**

**tmp=q->next;**

**q->next=NULL;**

**free(tmp);**

**display();**

**return;**

**}**

**printf("\n Element not found \n");}**

**void display()**

**{**

**struct node \*q;**

**if(start==NULL)**

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

**else**

**{**

**printf("\*\*\*\* Elements in Doubly Linked List \*\*\*\*\n");**

**q=start;**

**while(q!=NULL)**

**{**

**printf("%d\t", q->info);**

**q=q->next;**

**}**

**}**

**}**

**Menu driven program for Singly linked list (create, insert, delete, display) : -**

#include <stdio.h>

#include <malloc.h>

#include <process.h>

void create();

void insert\_at\_beg();

void insert\_at\_end();

void insert\_after\_pos();

void del();

void search();

void display();

**struct node**

**{**

**int info;**

**struct node \*link;**

**}\*start=NULL;**

**int data,item,nl,pos,i,m;**

**int main()**

**{**

**int n;**

**setbuf(stdout, NULL);**

**printf("\n\*\*\*\*Linked List\*\*\*\*\*\n");**

**printf("\nl. Create\n2. Insert at Beginning\n3.Insert at End\n4.Insert After**

**Position\n5.Delete\n6.Search\n7.Display\n8.Exit\n");**

**while(l)**

**{**

**printf("\n Enter Your Choice :(1.Create 2.Insert at Beg. 3.Insert at End 4.Insert**

**after Pos. 5.Delete 6.Search 7.Display 8.Exit)\n");**

**scanf("%d", &n);**

**switch(n)**

**{**

**case l:**

**create();**

**break;**

**case 2:**

**insert\_at\_beg();**

**break;**

**case 3:**

**insert\_at\_end();**

**break;**

**case 4:**

**insert\_after\_pos();**

**break;**

**case 5:**

**del();**

**break;**

**case 6:**

**search();**

**break;**

**case 7:**

**display();**

**break;**

**case 8:**

**exit(0);**

**default:**

**printf("\n Wrong Choice !!\n");**

**}**

**}**

**return 0;**

**void create()**

**{**

**struct node \*q, \*tmp;**

**printf("Enter element :\n");**

**scanf("%d",&data);**

**tmp=malloc(sizeof(struct node));**

**tmp->info=data;**

**tmp->link=NULL;**

**if(start==NULL)**

**start=tmp;**

**else**

**{**

**q=start;**

**while(q->link!=NULL)**

**q=q->link;**

**q->link=tmp;**

**}**

**}**

**void insert at beg()**

**{**

**struct node \*tmp;**

**printf("**[**\n Enter**](file:///nEnter) **the element to be inserted :\n");**

**scanf("%d", &data);**

**tmp=malloc(sizeof(struct node));**

**tmp->info=data;**

**tmp->link=start;**

**start=tmp;**

**display();**

**}**

**void insert\_at\_end()**

**{**

**struct node \*tmp,\*q;**

**printf("**[**\n Enter**](file:///nEnter) **the element to be inserted :\n");**

**scanf(M%d",&data);**

**tmp=malloc(sizeof(struct node));**

**tmp->info=data;**

**tmp->link=NULL;**

**if(start==NULL)**

**start=tmp;**

**else**

**{**

**q=start;**

**while(q->link!=NULL)**

**q=q->link;**

**q->link=tmp;**

**}**

**display();**

**}**

**void insert\_after\_pos()**

**{**

**display();**

**struct node \*q,\*tmp;**

**int index;**

**tmp=malloc(sizeof(struct node));**

**printf("**[**\n Enter**](file:///nEnter) **the element to be inserted :\n");**

**scan(“%d”, &data);**

**tmp">info=data;**

**tmp->link=NULL;**

**if(start==NULL)**

**{**

**start=tmp;**

**}**

**Else**

**{**

**printf("Enter index after which element to be inserted :\n");**

**scanf("%d", &index);**

**q=start;**

**for(i=0;i<index;i++)**

**{**

**q = q->link;**

**if(q==NULL)**

**{**

**printf("There are less elements\n");**

**return;**

**}**

**}**

**tmp->link = q->link;**

**q->link = tmp;**

**}**

**display();**

**}**

**void del()**

**{**

**struct node \*q.,\*tmp;**

**printf("Enter the element to be deleted :\n");**

**scanf(“%d”, &data);**

**if(start->info==data) //deletion of first node**

**{**

**tmp=start:**

**start=start->link;**

**free(tmp);**

**display();**

**return;**

**}**

**q=start;**

**while(q->link->link!=NULL) //deletion middle node**

**{**

**if(q->link->info==data)**

**{**

**tmp=q->link;**

**q->link=tmp->link;**

**free(tmp);**

**display();**

**return;**

**}**

**q=q->link;**

**}**

**if(q->link->info==data) //deletion of last node**

**{**

**tmp=q->link;**

**q->link=NULL;**

**free(tmp);**

**display();**

**return;**

**}**

**printf("**[**\**n Element](file:///nElement) **not found \n");**

**}**

**void search()**

**{**

**struct node \*tmp;**

**int i=0;**

**printf("**[**\n Enter**](file:///nEnter) **the element to be searched :");**

**scanf("%d",&item);**

**tmp=start;**

**while(tmp!=NULL)**

**{**

**if(tmp->info==item)**

**{**

**printf ("Element found at index: %d\n",i)j return;**

**}**

**tmp=tmp->link;**

**i++;**

**}**

**if(tmp->link==NULL)**

**printf("Element not found \n");**

**}**

**void display()**

**{**

**struct node \*q;**

**if(start==NULL)**

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

**else**

**{**

**printf("\*\*\*\* Elements in Linked List \*\*\*\*\n");**

**q=start;**

**while(q!=NULL)**

**{**

**printf("%d\t", q->info);**

**q=q->link;**

**}**

**}**

**}**

**Circular Linked List : -**

1. **Create**
2. **Insertion**
3. **Deletion**

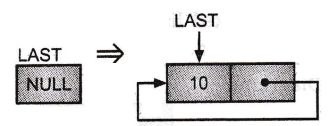
**Create**

First check the status of last pointer, if it is NULL means, we are inserting first node in the list and that new node (tmp) will be the Iast node in the list as shown in Fig

**void create()**

**{**

**struct node \*tmp;**

**printf("Enter element”); **

**scanf("%d", &data);**

**tmp=malloc(sizeof(struct node));**

**tmp->info=data;**

**if(last==NULL)**

**{**

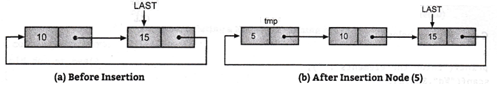
**last=tmp;**

**tmp->link=last;**

**}**

**}**

**Inserting Nodes at beginning in Circular Linked list** : -

****

* Let tmp is the node which we want to insert at beginning of iist. Store the address of (Last->link) in the

tmp->link. And update first node(last->link) as shown in Fig above.

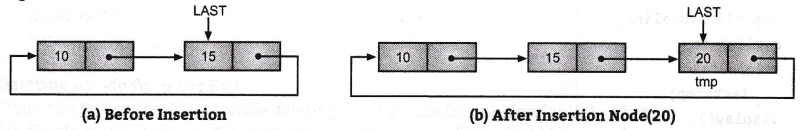
* C code is given below:

struct node \*tmp;  
printf("Enter element :\n");  
scanf("%d", &data);  
tmp=malloc(sizeof(struct node));  
tmp->info=data;  
tmp->link=last->link;

last->link=tmp;

display();

**Inserting Nodes at end in Circular Linked list : -**



* Let tmp is the node which we want to insert at the end of list. Store the address first node (Last->link) in the

tmp->link. If we are inserting node in already filled list then change the pointers as shown Fig above.

C code is given below:  
struct node \*tmp;  
printf("Enter element :\n");  
scanf("%d", &data);  
tmp = malloc(sizeof(struct node));  
tmp->info=data;  
if(last!=NULL)

{

tmp->link=last->link;

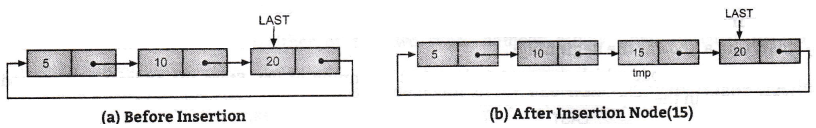
last->link=tmp;

last=tmp;

}

**Inserting Nodes after position in Circular Linked list :-**

* To insert a node after a specific position first traverse the list upto that point and let q is the node  
  after that new node (tmp) to be inserted. Change the pointers as shown in Fig.

****

C code is given below:

C code is given below:

struct node \*tmp,\*q;

printf("Enter the elements:");

scanf(“%d”, &data);

printf("Enter position after which element to be inserted : ");

scanf(“%d”,&pos);

tmp=malloc(sizeof(struct node));

tmp->info=data;

int *i;*

q =last->link;

for(i=0;i< pos-l; i++)

{

q=q->link;

if(q==last->link)

{

printf("There are less than %d elements\n",pos);

return;

}

}

tmp->link=q->link;

q->link=tmp;

if(q==last)

last=tmp;

display();

**Deleting Nodes in CLL at beginning: -**

There can be two conditions while deleting beginning node from a circular linked list.

**(a) Single node in the entire list:** The condition last->link==last&8dast->info==data specifies the  
there is only one node that is first and last node. To delete this node, make last pointer to NULL.

if(last->link==last&&last->info==data) //single node deletion

{

last=NULL;

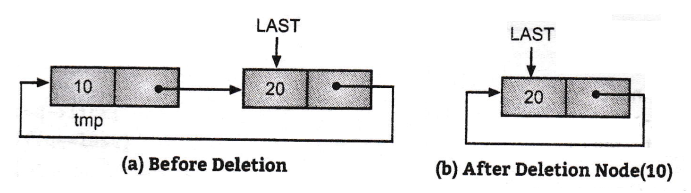
free(last);

printf("List is empty!!");

return;

}

**(b) Multiple nodes in list:** As the last->link in point to the starting node of the list. To delete this  
node, first save last->link in q variable and then update the last->links (as after removal of first  
node, second node will be the starting node after that) as shown below :

****

**C code is given below:**

**void del\_at\_beg()**

**{**

**struct node \*q;**

**if(last->link==last) //single node in the list**

**{**

**last=NULL;**

**free(last);**

**printf("List is empty!!");**

**return;**

**}**

**if(last==NULL)**

**{**

**printf("List is empty"); }**

**}**

**q=last->link;**

**last->link=q->link;**

**free(q);**

**printf("First element deleted!!\n");**

**display();**

**return;**

**}**

**Deleting a node from the End:**

There can be two conditions while deleting ending node from a circular linked list.

1. **Single node in the entire list:** The condition last->link==last&Slast->info==data specifies the there is only one node that is first and last node. To delete this node, make last pointer to NULL.

if(last->link==last&&last->info==data) //single node deletion

{

last=NULL;

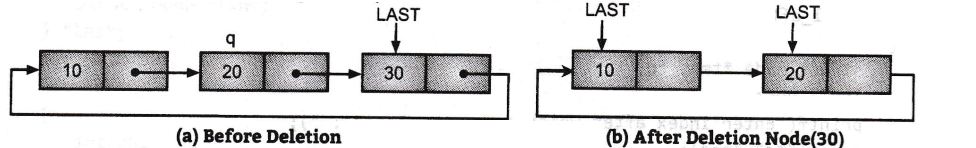
free(last);

printf("List is empty!!");

return;

}

**(b) Multiple nodes in list:** To delete last node in the list, first traverse the list up to last pointer, and  
store the address of second last node in q pointer. Then store starting node address in q and  
update the value of last pointer (i.e. set as q).

****

void del\_at\_end()

{

struct node \*q;

q=last->link;

if(last==NULL)

{

printf("List is empty");

}

if(last->link==last) //single node in the list

{

last=NULL;

free(last);

printf("List is empty!!");

return;

}

while(q->link!=last)//traverse upto Last pointer

{

q=q->link;

}

q->link=last->link;

free(last);

last=q;

printf("Last element deleted!!\n");

display();

return;

**}**

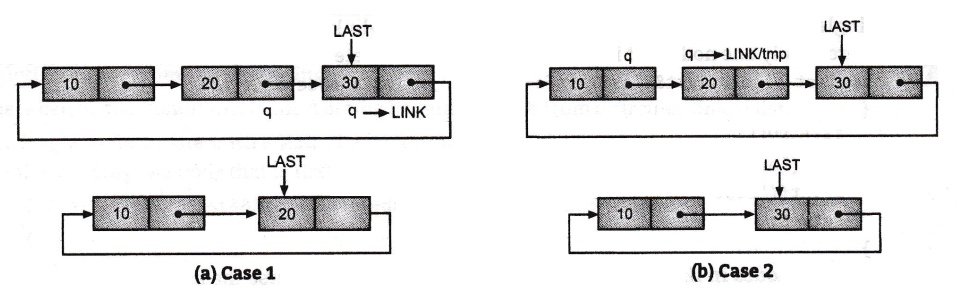
**Deleting a node after a position:**

* To delete a node after a specific position, first traverse the list up to specified position.
* After traversing, you will get the address of node(q) after which deletion operation has to perform.
* Let q is the node after that deletion has to perform so q->link is the node which we have to delete.

**Case 1:**

* First check that q is second last node or q->link is the last node. If q->link is last node then perform the steps of deletion of last node otherwise it is in between node.

**Case 2:** If it is in between node then save the q->link to tmp and update q->link.

****

**C code is given below:**

void del\_after\_pos()

{

struct node \*tmp, \*q;

int i,pos;

printf("Enter index after which element to be deleted : ");

scanf("%d",&pos);

q=last->link;

for(i=0;i<pos;i++)

{

q=q->link;

if(q==last)

{

printf("There are less elements\n");

return;

}

}

if(q->link==last) // Case 1

{

q->link=last->link;

free(last);

last=q;

displayQ;

return;

}

tmp=q->link; // Case 2

q->link=tmp->link;

free(tmp);

display();

return;

**}**

**Difference between Single Linked List and Doubly linked List:**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Singly Linked List (SLL)** | **Doubly Linked List (DLL)** |
| **1.** | Singly linked list allows us to go one way direction. | Doubly linked list has two way directions next and previous. |
| 2. | Singly linked list uses less memory per node (one pointer). | Doubly linked list uses more memory per node. |
| 3. | Complexity of Insertion and Deletion at known position is O(n). | Complexity of Insertion and Deletion at known position is 0(1). |
| **4.** | If we need to save memory in need to update node values frequently and searching is not required, we can use singly linked list. | If we need faster performance in searching and memory is not a limitation we use doubly linked list. |
| **5.** | In single list each node contains at least two  parts:  (i) INFO, and  (ii) LINK. | In doubly linked list Each node contains at  least three parts:  (i) INFO,  (ii) LINK to next node, and  (iii) LINK to previous node. |
| 6. | Singly linked list is unidirectional i.e., only one direction. | It is bidirectional. |
| **7.** | Singly linked list can mostly be used for stacks. | They can be used to implement stacks, heaps, binary trees. |
| **8.** | Diagram | Diagram |

Menu driven program for circular single Linked list **(create, insert, delete, search,** display).

#include<stdio.h> #include<conio.h> #include<malloc.h> #include<stdlib.h>

void create();

void insert\_at\_beg();

void insert\_at\_end();

void insert\_after\_pos();

void del\_at\_beg();

void del\_after\_pos();

void del\_at\_end();

void search();

void display();

struct node

{

int info;

sruct node \*link;

} \*last;

int data,pos;

int main()

{

int ch;

last=NULL;

setbuf(stdout, NULL);

printf("\n\*\*\*\*Circular Linked List\*\*\*\*\*\n");

printf("\nl. Create\n2.Insert at Beginning\n 3.Insert at End\n 4.Insert After

Position\n 5.Delete at Beginning\n 6.Delete at End\n 7.Delete after Position\n

8.Search\n 9.Display\nl0.Exit\n");

while(l)

{

printf("\nEnter Your Choice :(1.Create 2.Insert at Beg. 3.Insert at End 4.Insert

after Pos. 5.Delete at Beginning 6.Delete at end 7.Delete after Pos. 8.Search

9.Display 10.Exit)\n");

Scanf(“%d”,&ch);

switch(ch)

{

case 1:

create();

break;

case 2:

insert\_at\_beg();

break;

case 3:

insert\_at\_end();

break;

case 4:

insert\_after\_pos();

break;

case 5:

del\_at\_beg();

break;

case 6:

del\_at\_end();

break;

case 7:

del\_after\_pos();

break;

case 8:

search();

break;

case 9:

display();

break;

case 10:

exit(0);

default:

printf("\n Wrong Choice !!\n");

}

}

return 0;

}

void create()

{

struct node \*tmp;

printf("Enter element :\n");

scanf("%d",&data);

tmp=malloc(sizeof(struct node));

tmp->info=data;

if(last==NULL)

{

last=tmp;

tmp->link=last;

}

else

{

tmp->link=last->link;

last->link=tmp;

last=tmp;

}

}

void insert\_at\_beg()

{

struct node \*tmp;

printf("Enter the element to be inserted :\n");

scanf("%d",&data);

tmp=malloc(sizeof(struct node));

tmp->info=data;

tmp->link=last->link;

last->link=tmp;

display();

}

void insert\_at\_end()

{

struct node \*tmp;

printf("Enter element :\n");

scanf("%d",&data);

tmp=malloc(sizeof(struct node));

tmp->info=data;

tmp->link=last->link;

last->link=tmp;

last=tmp;

display();

}

void insert\_after\_pos()

{

struct node \*tmp, \*q;

printf("Enter the elements : ");

scanf("%d",&data);

printf("Enter index after which element to be inserted

scanf("%d",&pos);

int i;

q=last->link;

for(i=0;i<pos;i++)

{

q=q->link;

if(q==last->link)

{

printf("There are less elements\n");

return;

}

}

tmp=malloc(sizeof(struct node));

tmp->link=q->link;

tmp->info=data;

q->link=tmp;

if(q==last)

last=tmp;

display();

}

void del\_at\_beg()

{

struct node \*q;

if(last->link==last) //single node in the list

{

last=NULL;

free(last);

printf("List is empty!!");

return;

}

if(last==NULL)

{

printf("List is empty");

}

q=last->link;

last->link=q->link;

free(q);

printf("First element deleted!!\n");

display();

return;

}

void del\_at\_end()

{

struct node \*q;

q=last->link;

if(last==NULL)

{

printf("List is empty");

}

if(last->link==last) //single node in the list

{

last=NULL;

free(last);

printf("List is empty!!");

return;

}

while(q->link!=last)//traverse upto Last pointer

{

q=q->link;

}

q->link=last->link;

free(last);

last=q;

printf("Last element deleted!!\n);

display();

return;

}

void del\_after\_pos()

{

struct node \*tmp, \*q;

int i,pos;

printf("Enter index after which element to be deleted : ");

scanf("%d",&pos);

q=last->link;

for(i=0;i<pos;i++)

{

q=q->link;

if(q==last)

{

printf("There are less elements\n");

return;

}

}

if(q->link==last)

{

q->link=last->link;

free(last);

last=q;

display();

return;

}

tmp=q->link;

q->link=tmp->link;

free(tmp);

display();

return;

}

void display()

{

struct node \*q;

if(last==NULL)

{

printff'List is empty!!\n");

return;

}

printf("\*\*\*\* Elements in Circular Linked List \*\*\*\*\n");

q=last->link;

while(q!=last)

{

printf("%d\t",q->info);

q=q->link;

}

printf("%d\t", last->info);

}

void search()

{

struct node \*tmp;

int i=0,item;

printf("\n Enter the element to be searched :");

scanf("%d", &item);

tmp=last->link;

while(tmp!=last)

{

if(tmp->info==item)

{

printf("\n Element %d is found at index : %d \n", item, i);

return;

}

tmp=tmp->link;

++i;

}

if(last->info==item)

printf("\n Element %d is found at index : %d \n", item, i);

else

printf("\n Element not found \n");

}

**Following table differentiate between Singly, Doubly and Circularly linked lists**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sr No | **Parameter** | **Single Linked List** | **Doubly Linked Lists** | **Circular Linked List (CLL)** |
| 1 | Structure | struct Node  {  int Data;  struct Node \*Next;  }; | struct Node  {  int Data;  struct Node \*Next;  struct Node \*Previous;  }; | Depends on the type of circular  linked listening  **Single Circular**:  struct Node  {  int Data;  struct Node \*Next;  };  **Double Circular**:  struct Node  {  int Data;  struct Node \*Next;  struct Node \*Previous;  }; |
| 2 | Address | Pointers contains the address of next node in the list. | Pointers contains the address of next node as well as  previous Node in the list. | Pointer can or cannot contains  the address of previous node asit depends on type of circular linked list. |
| 3 | Connection | Last element is linked to a null object. | Last element is linked to a null object. | The last element is linked to the first element. |
| 4 | No of pointer | One pointer. | Two pointers. | Can have one or two pointers. |
| 5 | Link to other  node | Each node (item) of the list is connected to the next node. | The next node also knows about the previous node. | The last node knows about the first node and first node knows about the last node. |
| 6 | Mobility | We can not move in backward direction. | We can move backward as well as forward direction.  ■ | We can or cannot move in  backward direction as it depends on type of circular linked list. |
| 7 | Insertion in  between | Two address need to  be updated. | Four address need to be updated. | Two or four address need to be updated. |

**Applications of Linked list:**

1. **Polynomial Addition and Multiplication:** Polynomial equations can be added and multiplied easily using linked lists. Linked lists hold each coefficient and exponential part as information parts.
2. **Radix Sort:** Radix sorts, and address calculation sort use linked lists to perform sorting.
3. **To perform Arithmetic Operations on Large Numbers:** Using existing data types such as int, long, it is not possible to perform arithmetic operations large numbers. Linked lists solve this problem by splitting them into digits and holding them in nodes.
4. **Implementation of Stacks and Queues:** The data structures such as stacks and queues are implementations of linked list.
5. **Memory Management:** The unused nodes can be maintained as a free list. There will not be any memory wastage in linked list.
6. **Symbol Table Implementation:** Symbol tables are used in compiler construction and related fields. These can be used as dictionary for the values. Linked list are best suited for implementing this tables.
7. **Garbage Collection:** They are used in garbage collection and linked dictionary.