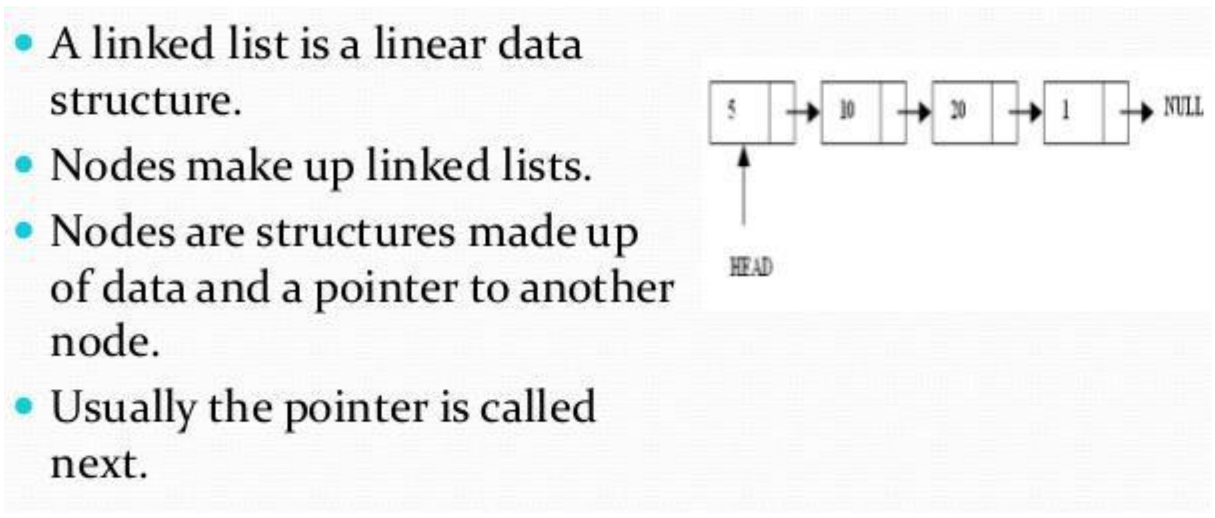
UNIT -4 - LINK LIST

• Applications of the linked lists
• Types of Linked Lists
  o Singly Linked List
  o Doubly linked list
  o Header Linked List
  o Circular Linked List
• Implementation using Singly Linked List, Doubly Linked List and Circular Singly Linked List
  o Insertion of a node at the beginning
  o Insertion of a node at the end
  o Insertion of a node after a specified node
  o Traversing the entire linked list
  o Deletion of a node from linked list
  o Updating of a specific node
• Implementation of merging of two Singly Linked List
  • Implementation of reversing of Singly Linked List

What is linked list

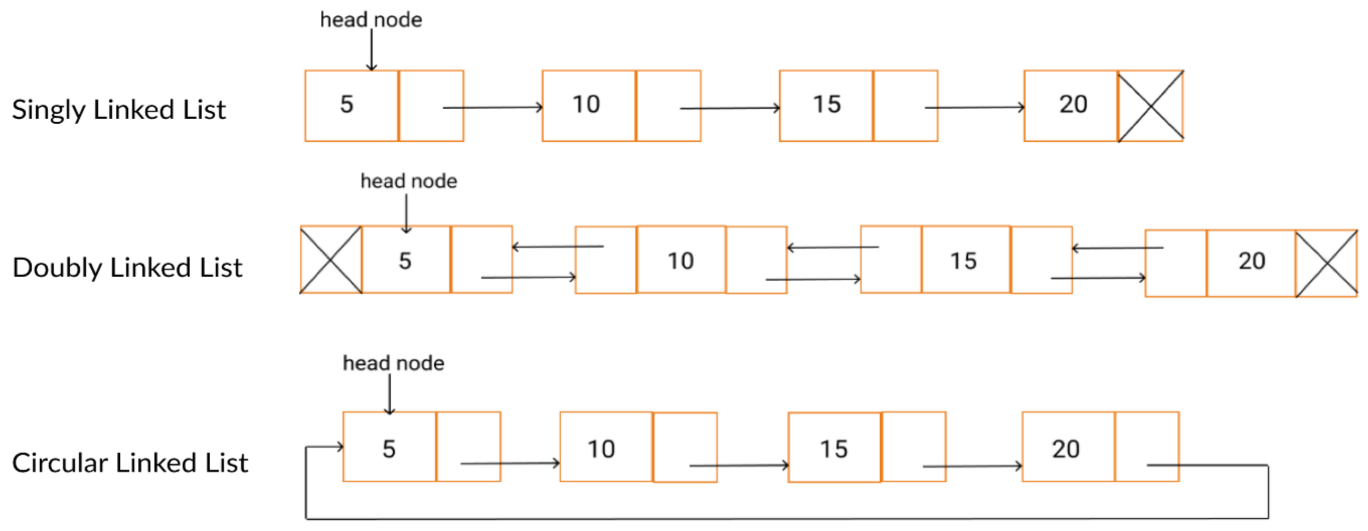


- A linked list is a linear data structure.
- Nodes make up linked lists.
- Nodes are structures made up of data and a pointer to another node.
- Usually the pointer is called next.

5
10
20
1
NULL
HEAD

- A linked list is defined as collection (sequence) of nodes. Each node has two parts
  - Information
  - Pointer to next node
- Information
  - Information part may consist of one or more than one fields.
- Pointer to next node
  - Pointer to next node contains the address of location where next information is stored
  - The last node of the list contains NULL in the pointer field

Types of Linked Lists



Applications of linked list are:

1. In line editor:

- One interesting use of linked list is line editor. We can keep a link list of lines nodes. Each containing line number, a line of text and pointer to next line information node.

2. In string manipulation:

- Variable string length can be represented as linked list. A string may be declared as a record that contains a string count & pointer to linked list of characters.

3.In implementation of space matrix:

- A space matrix is a table which relatively with few non-zero elements.

4. In operating system:

- The allocation of memory space may be managed by doubly linked list of various size block of memory. In multi-user system the operating system may keep track of user jobs waiting to execute through linked list queue.

5. Implementing stack and queue:

- It is easy to implement stack & queue operation using linked list rather than array implementations of stack & queue.

6. Polynomial manipulation:

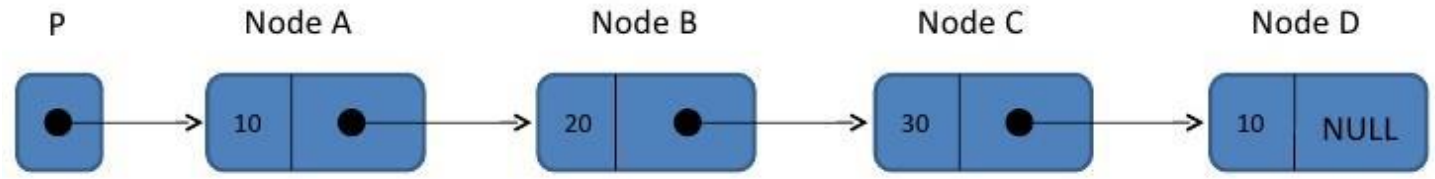
- A linked list uses as a typical term of polynomial. The common operation performs on polynomial are addition, subtraction, multiplication, division, integration and differentiation.

7. Linked dictionary:

- An important part of any compiler is the construction and maintenance of a dictionary containing name and their associated values, such dictionary is also called a symbol table

Singly Linked List

- A list implemented by each item having a link to the next item.
- Head points to the first node.
- Last node points to NULL.



Singly linked list is the most basic of all the linked data structure

A singly linked list is a collection of nodes and each node contains the pointer to next element

In singly link list we can move from left to right that is only in one dimension but we cannot return back.

Each node represents structure, containing variables for information and a structure pointer to itself.

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

Create operation

Algorithm

Step-1 Check whether any node exists in the list or not

Step-2: If list is Empty then create node

```
                if(q==NULL)
        {
                q=(node *)malloc(sizeof(node));
                q->data=info;
                q->next=NULL;
        }
```

Step-3: If the list is not Empty then create the node after the last node

```
                while(q->next!=NULL)
        {
                q=q->next;
            }
                q->next=(node *)malloc(sizeof(node));
                q->next->data=info;
                q->next->next=NULL;
    Step-4: End
```

Display operation

Algorithm

```
    Step-1: while(q!=NULL)
            {
                printf("\nElement is %d",q->info);
                q=q->next;
            }
    Step-2: End
```

1) Add node at beginning operation (Insertion of a node at the beginning)

Algorithm

```
    Step-1: Allocate the memory to the node P that is to be inserted in the beginning
            p=(node *)malloc(sizeof(node))
    Step-2:Allocate the data and pointer to next node part
            p->data=info
            p->next=q;
    Step-3:End
```

2) Add node at after operation(Insertion of a node at the end**)**

Algorithm

```
    Step-1: Take a temporary node which is to be inserted (node *temp)
    Step-2:for(i=1;i<pos;i++)
            {
                q=q->next;
                if(q==NULL)
                printf("\nout of range");
            }
    Step-3:Allocate memory to temporary node and data and next part
            temp=(node *)malloc(sizeof(node))
            temp->info=ele
            temp->next=q->next;
            q->next=temp;
    Step-4:End
```

3)]**Deletion of a node from linked list(**Deletion of a node from linked list)

```
Step:- 1    selected the node
Step:- 2    node has a element
            if(q->info==ele)
step :- 3   pointer switch next node
step:- 4    free q
step :- 5   end
```

4) Insertion of a node at the end

```
Step:- 1    if ptr = null write write overflow go to step 1
            [end of if]
step 2:      set new_node = ptr
step 3:      set ptr = ptr - > next
step 4:     set new_node - > data = val
step 5:      set new_node - > next = null
step 6:     set ptr = head
step 7:     repeat step 8 while ptr - > next != null
step 8:     set ptr = ptr - > next
            [end of loop]
step 9:      set ptr - > next = new_node
step 10:    exit
```

5) Updating of a specific node

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

// Creating node with data and a pointer
struct node {
int data;
struct node *next;
}*head;
void createList(int n);
void update_element(int data);
void displayList();

int main()
{
int n, data, element;
printf("\nEnter the total number of nodes: ");
scanf("%d", &n);
createList(n);
printf("\nThe List is \n");
displayList();
printf("\nEnter the element to be updated in the list : "); // value of the element to be
searched in the list
scanf("%d",&element);
update_element(element);
displayList();
return 0;
}

void createList(int n)
{
struct node *newNode, *temp;
int data, i;
head = (struct node *)malloc(sizeof(struct node));
// When the list is empty
if(head == NULL)
{
printf("Unable to allocate memory.");
}
else
{
printf("\nEnter the data of node 1: ");
scanf("%d", &data);
head->data = data;
head->next = NULL;
temp = head;
for(i=2; i<=n; i++)
{
newNode = (struct node *)malloc(sizeof(struct node));
if(newNode == NULL)
{
printf("Unable to allocate memory.");
break;
}
else
{
printf("\nEnter the data of node %d: ", i);
scanf("%d", &data);
newNode->data = data;
newNode->next = NULL;
temp->next = newNode;
temp = temp->next;
}}}}

void update_element(int data)
```

```
{
int count = 0;
int update_ele;
struct node* temp;
temp = head;
while(temp != NULL) // Start traversing from head node
{
if(temp -> data == data)
{
printf("\nEnter the new data to update the old data : ");
scanf("%d",&update_ele);
temp -> data = update_ele; // change the element in the list
}
else
{
count = count + 1;
temp = temp -> next;
}}}

void displayList()
{
struct node *temp;
if(head == NULL)
{
printf("List is empty.");
}
else
{
temp = head;
while(temp != NULL)
{
printf("%d\t", temp->data);
temp = temp->next;
}
printf("\n");
}}
```

6) Insertion of a node after a specified node

Algorithm

```
step 1:     if ptr = null.
step 2:     set new_node = ptr.
step 3:     new_node ? data = val.
step 4:     set temp = head.
step 5:     set i = 0.
step 6:     repeat step 5 and 6 until i.
```

```
        step 7:         temp = temp ? next.
        step 8:          if temp = null.
```

Program:

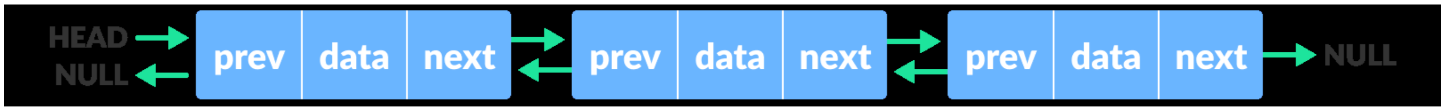
```
#include<stdio.h>
#include<conio.h>
struct list
{
        int info;
        struct list *next;
};
        typedef struct list node;
node *p;
void create(int,node *);IOOP
void display(node *);
void addbeg(int,node *);
void addafter(int,int,node *);
void deleted(int,node *);
int count(node *);
void sort(node *);
void search(int,node *);
void main()
{
        p=NULL;
        clrscr();
        create(10,p);
        create(20,p);
        create(5,p);
        display(p);
        addbeg(0,p);
        printf("\n\n");
        addafter(2,500,p);
        deleted(20,p);
        display(p);
        sort(p);
        printf("\n\n");
        display(p);
        search(15,p);
        getch();
}
void create(int ele,node *q)
{
        if(q==NULL)
        {
        p=(node *)malloc(sizeof(node));
```

```
	p->info=ele;
	p->next=NULL;
	}
	else
	{
	while(q->next!=NULL)
	{
	q=q->next;
}
	q->next=(node *)malloc(sizeof(node));
	q->next->info=ele;
	q->next->next=NULL;
}
}
void display(node *q)
{
		while(q!=NULL)
	{
		printf("\nElement is %d",q->info);
		q=q->next;
	}
}
int count(node *q)
{
		int c=0;
		while(q!=NULL)
	{
		q=q->next;
		c++;
	}
	return (c);
}
void addbeg(int ele,node *q)
{
	p=(node *)malloc(sizeof(node));
	p->info=ele;
	p->next=q;
}
void addafter(int c,int ele,node *q)
{
	node *temp;
	int i;
	for(i=1;i<c;i++)
	{
		q=q->next;
```

```
                if(q==NULL)
                        {
                                printf("\nposition is out of range");
                                return;
                        }
        }
                temp=(node *)malloc(sizeof(node));
                temp->info=ele;
                temp->next=q->next;
                q->next=temp;
}
void deleted(int ele,node *q)
{
        node *temp;
        if(q->info==ele)
        {
                p=q->next;
                free(q);
                return;
        }
while(q->next->next!=NULL)
{
        if(q->next->info==ele)
        {
                temp=q->next;
                q->next=q->next->next;
                free(temp);
                return;
        }
        q=q->next;
        }
}
void sort(node *q)
        {
                int t;
                node *temp;
                while(q!=NULL)
                {
                        temp=q->next;
                        while(temp!=NULL)
                        {
                                if(q->info > temp->info)
                                {
                                        t=q->info;
                                        q->info=temp->info;
                                        temp->info=t;
                                }
                                temp=temp->next;
                        }
                q=q->next;
                        }
                }
        void search(int num,node *q)
        {
                while(q!=NULL)
                {
                        if(q->info==num)
                        {
                                printf("\nSearch success");
                                printf("\nSearch element %d ",num);
                                return;
                        }
                        q=q->next;
                }
                printf("\nSearch unsuccessful");
                getch();
}
```

Doubly linked list

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer). A sample node in a doubly linked list is shown in the figure.



In C, structure of a node in doubly linked list can be given as :

```
struct node
{
   struct node *prev;
   int data;
   struct node *next;
}
```

Program

```
#include<stdio.h>
#include<conio.h>
struct list
{
        int info;
        struct list *next,*prev;
};
        typedef struct list node;
        node *p;
        void create(int,node *);
        void display(node *);
        void addbeg(int,node *);
        void addafter(int,int,node *);
        void deleted(int,node *);
        int count(node *);
        void sort(node *);
        void search(int,node *);
        void main()
        {
                p=NULL;
                clrscr();
                create(10,p);
                create(20,p);
                create(30,p);
                display(p);
                addbeg(0,p);
                printf("\n\n");
                addafter(2,500,p);
                deleted(20,p);
                display(p);
                sort(p);
                printf("\n\n");
                display(p);
                search(15,p);
                getch();
}
void create(int ele,node *q)
{
        node *temp;
        if(q==NULL)
        {
                p=(node *)malloc(sizeof(node));
                p->prev=NULL;
                p->info=ele;
                p->next=NULL;
        }
        else
        {
        while(q->next!=NULL)
        {
                q=q->next;
        }
                temp=(node *)malloc(sizeof(node));
                temp->next=NULL;
                temp->info=ele;
                temp->prev=q;
                q->next=temp;
        }
        }
void display(node *q)
        {
                while(q!=NULL)
                {
                        printf("\nElement is %d",q->info);
                        q=q->next;
                }
        }
        int count(node *q)
                {
                        int c=0;
                        while(q!=NULL)
                {
                        q=q->next;
                        c++;
        }
        return (c);
}
void addbeg(int ele,node *q)
{
        p=(node *)malloc(sizeof(node));
        p->prev=NULL;
        p->info=ele;
        p->next=q;
        q->prev=p;
}
void addafter(int c,int ele,node *q)
{
        node *temp;
        int i;
        for(i=1;i<c;i++)
        {
                q=q->next;
                if(q==NULL)
                {
                        printf("\nposition is out of range");
                        return;
                }
        }
        temp=(node *)malloc(sizeof(node));
        temp->prev=q;
        temp->next=q->next;
        temp->info=ele;
        temp->next->prev=temp;
        q->next=temp;
        return;
        }
void deleted(int ele,node *q)
{
        node *temp;
        if(q->info==ele)
        {
                p=q->next;
                q->next->prev=NULL;
                free(q);
                return;
        }
        while(q->next->next!=NULL)
{
        if(q->next->info==ele)
{
        temp=q->next;
        q->next=q->next->next;
        q->next->prev=temp->prev;
        free(temp);
        return;
}
        q=q->next;
}
}
        void sort(node *q)
{
        int t;
        node *temp;
        while(q!=NULL)
        {
                temp=q->next;
                while(temp!=NULL)
        {
                if(q->info>temp->info)
                {
                        t=q->info;
                        q->info=temp->info;
                        temp->info=t;
                }
                temp=temp->next;
        }
                q=q->next;
}
}
void search(int num,node *q)
{
        while(q!=NULL)
        {
                if(q->info==num)
                        {
                                printf("\nSearch success");
                                printf("\nSearch element %d ",num);
                                return;
                        }
                q=q->next;
        }
        printf("\nSearch unsuccessful");
        getch();
}
```

Header Linked List

Header Linked List is a modified version of Singly Linked List.In the Header linked list, we have a special node, the Header Node present at the beginning of the linked list. The Header Node is an extra node at the front of the list storing meaningful information about the list. Such a node is not similar in structure to the other nodes in the list. It does not represent any items of the list like other nodes, rather the information present in the Header node is global for all nodes such as Count of Nodes in a List, Maximumamong all Items, Minimum value among all Items etc.

Types of Header Linked List :

1. Grounded Header Linked List

In this type of Header Linked List, the last node of the list points to NULL or holds the reference to NULL Pointer.

The head pointer points to the Header node of the list. If there is no node to the next of head pointer or head.next equals NULL then we know that the Linked List is empty.

The operations performed on the Header Linked List are the same as Singly Linked List such as Insertion, Deletion, and Traversal of nodes.

Let us understand this with an example:

2. Circular Header Linked List :

A Linked List whose last node points back to the First node or the Head Node of the list is called a Circular Linked List. Similarly, if the last node of the Header Linked List points back to Header Node then it is a Circular Header Linked List.

The last node of the list does not hold NULL reference. In this case, We have to use external pointers to maintain the last node.The end of the list is not known while traversing.

It can perform the same operations as its counterpart such as Insert, Delete and Search. Let us see an example.

Applications of linked list are:

- Implementation of stacks and queues
- Implementation of graphs: Adjacency list representation of graphs is the most popular which uses a linked list to store adjacent vertices.
- Dynamic memory allocation: We use a linked list of free blocks.
- Maintaining a directory of names
- Performing arithmetic operations on long integers
- Manipulation of polynomials by storing constants in the node of the linked list
- representing sparse matrices In Doubly linked list :
- Redo and undo functionality.
- Use of the Back and forward button in a browser.
- The most recently used section is represented by the Doubly Linked list.
- Other Data structures like Stack, HashTable, and BinaryTree can also be applied

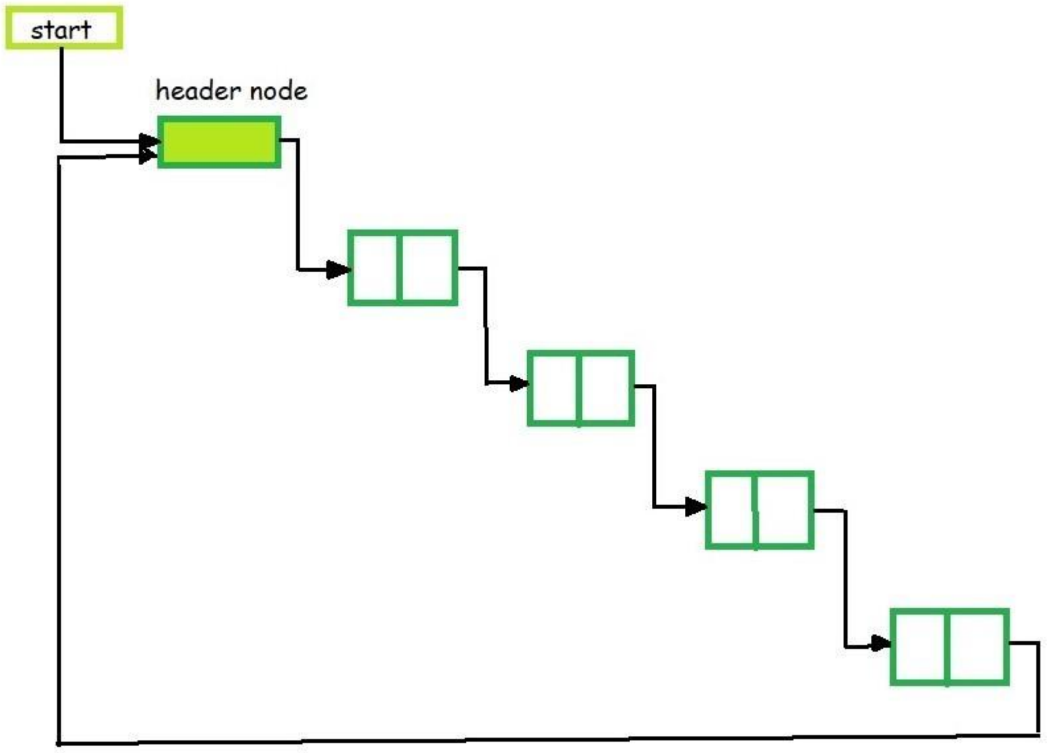
by Doubly Linked List.

Applications of linked list in the real world:

1. Image viewer – Previous and next images are linked and can be accessed by the next and previous buttons.

2. Previous and next page in a web browser – We can access the previous and next URL searched in a web browser by pressing the back and next buttons since they are linked as a linked list.

3. Music Player – Songs in the music player are linked to the previous and next songs. So you can play songs either from the start or ending of the list. operations on this type of linked list are Insertion, Deletion and Traversing.



Program of header link list

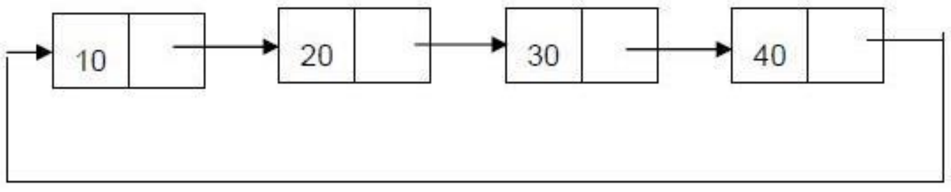
```
// C program for a Header Linked List
#include <malloc.h>
#include <stdio.h>

// Structure of the list
struct link {
        int info;
        struct link * next;
};

// Empty List
struct link * start = NULL;

// Function to create a header linked list
```

```
struct link * create_header_list(int data)
{

        // Create a new node
        struct link  *new_node,  *node;
        new_node = (struct link*) malloc(sizeof(struct link));
        new_node->info = data;
        new_node->next = NULL;

        // If it is the first node
        if (start == NULL) {

                // Initialize the start
                start = (struct link*)
                        malloc(sizeof(struct link));
                start->next = new_node;
        }
        else {

                // Insert the node in the end
                node = start;
                while (node->next != NULL) {
                        node = node->next;
                }
                node->next = new_node;
        }
        return start;
}

// Function to display the
// header linked list
struct link* display()
{
        struct link* node;
        node = start;
        node = node->next;
        while (node != NULL) {
                printf("%d ", node->info);
                node = node->next;
        }
        printf("\n");
        return start;
}

// Driver code
```

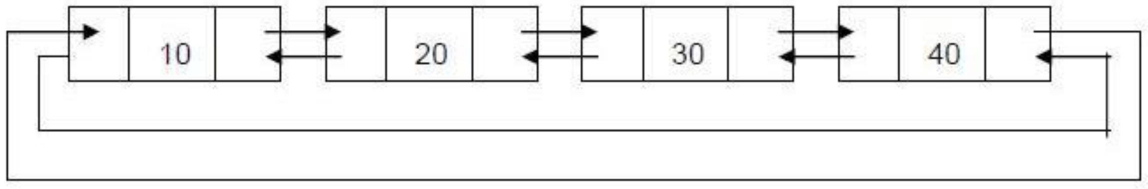
```
int main()
{

        // Create the list
        create_header_list(11);
        create_header_list(12);
        create_header_list(13);

        // Print the list
        display();
        create_header_list(14);
        create_header_list(15);

        // Print the list
        display();

        return 0;
}
```

Circular Linked List

- The linked list where the last node points the header node is called circular linked list.



Circular singly linked list



Circular doubly linked list

Circular Singly Linked List:
Program

```
#include<stdio.h>
#include<conio.h>
	struct list
	{
		int info;
		struct list *next;
	};
typedef struct list node;
node *p;
void create(int,node *);
void display(node *);
void addbeg(int,node *);
void addafter(int,int,node *);
void deleted(int,node *);
int count(node *);
void sort(node *);
void search(int,node *);
void main()
{
	p=NULL;
	clrscr();
	create(10,p);
	create(11,p);
	create(12,p);
	display(p);
	printf("\n\n");
	addbeg(0,p);
	addafter(3,12,p);
	deleted(11,p);
	sort(p);
	search(5,p);
	display(p);
	getch();
}
void create(int ele,node *q)
	{
		if(q==NULL)
		{
			p=(node *)malloc(sizeof(node));
			p->info=ele;
```

```
                        p->next=p;
        }
else
        {
                while(q->next!=p)
                {
                        q=q->next;
                        }
                        q->next=(node *)malloc(sizeof(node));
                        q->next->info=ele;
                        q->next->next=p;
                }
        }
void display(node *q)
{
        do
                {
                        printf("\nElement is %d",q->info);
                        q=q->next;
                }       while(q!=p);
}
int count(node *q)
        {
                int c=0;
                do
                        {
                                q=q->next;
                                c++;
                        }while(q!=p);
                        return (c);
        }
void addbeg(int ele,node *q)
{
        node *temp;
        temp=(node *)malloc(sizeof(node));
        temp->info=ele;
        temp->next=q;
        while(q->next!=p)
        {
                q=q->next;
        }
        q->next=temp;
        p=temp;
}
void addafter(int c,int ele,node *q)//pos is same as c var.
```

```
{
        node *temp;
        int i;
        for(i=1;i<c;i++)
        {
                q=q->next;
                if(q==p)
                {
                        printf("\nposition is out of range");
                        return;
                }
        }
        temp=(node *)malloc(sizeof(node));
        temp->info=ele;
        temp->next=q->next;
        q->next=temp;
        }
void deleted(int ele,node *q)
{
node *temp;
if(q->info==ele)
{
do
{
q=q->next;
}while(q->next!=p);
q->next=p;
p=p->next;
return;
}
while(q->next->next!=p)
{
if(q->next->info==ele)
{
temp=q->next;
q->next=temp->next;
free(temp);
return;
}
q=q->next;
}
}
void sort(node *q)
{
int t;
node *temp;
do
{
temp=q->next;
while(temp!=p)
{
if(q->info>temp->info)
{
t=q->info;
q->info=temp->info;
temp->info=t;
}
temp=temp->next;
}
q=q->next;
}while(q!=p);
}
void search(int num,node *q)
{
do
{
if(q->info==num)
{
printf("\nSearch success");
printf("\nSearch element %d",num);
return;
}
q=q->next;
}while(q!=p);
printf("\nSearch unsuccessful");
getch();
}
```

Circular Doubly Linked List:

Program

```
#include<stdio.h>
#include<conio.h>
        struct list
        {
                int info;
                struct list *next,*prev;
        };
typedef struct list node;
node *p;
void create(int,node *);
void display(node *);
void addbeg(int,node *);
void addafter(int,int,node *);
void deleted(int,node *);
int count(node *);
void sort(node *);
void search(int,node *);
void main()
{
        p=NULL;
        clrscr();
        create(10,p);
        create(5,p);
        create(15,p);
        addbeg(0,p);
        //search(5,p);
        //sort(p);
        //deleted(15,p);
        display(p);
        getch();
}
void create(int ele,node *q)
{
        node *temp;
        if(q==NULL)
        {
                p=(node *)malloc(sizeof(node));
                p->prev=p;
                p->info=ele;
                p->next=p;
        }
        else
        {
                while(q->next!=p)
        {
                q=q->next;
        }
                temp=(node *)malloc(sizeof(node));
                temp->next=p;
                temp->info=ele;
                temp->prev=q;
                q->next=temp;
        }
}
void display(node *q)
```

```
{
do
          {
          printf("\nElement is %d",q->info);
          q=q->next;
          }while(q!=p);
          }
int count(node *q)
{
          int c=0;
          do
          {
                    q=q->next;
                    c++;
          }while(q!=p);
          return (c);
}
void addbeg(int ele,node *q)
{
          node *temp;
          while(q->next!=p)
                    {
                              q=q->next;
                    }
                    temp=(node *)malloc(sizeof(node));
                    temp->prev=q;
                    temp->info=ele;
                    temp->next=p;
                    p->prev=temp;
                    q->next=temp;
                    p=temp;
          }
          void addafter(int c,int ele,node *q)
          {
                    node *temp;
                    int i;
                    for(i=1;i<c;i++)
                    {
                    q=q->next;
                    if(q==p)
                    {
                    printf("\nposition is out of range");
                    return;
}
}
```

```
        temp=(node *)malloc(sizeof(node));
        temp->prev=q;
        temp->next=q->next;
        temp->info=ele;
        temp->next->prev=temp;
        q->next=temp;
        return;
}
void deleted(int ele,node *q)
        {
                node *temp;
                if(q->info==ele)
        {
                temp=q;
                p=temp->next;
                while(q->next!=p)
        {
                q=q->next;
        }
                p->prev=q;
                return;
        }
        do
        {
                if(q->next->info==ele)
                {
                temp=q->next;
                q->next=temp->next;
                temp->next->prev=q;
                // free(temp);
                return;
                }
                q=q->next;
                }while(q!=p);
        }
        void sort(node *q)
        {
                int t;
                node *temp;
                do
        {
                temp=q->next;
                while(temp!=p)
                        {
                if(q->info>temp->info)
                {
                t=q->info;
                q->info=temp->info;
                temp->info=t;
                }
                temp=temp->next;
        }
                q=q->next;
        }       while(q!=p);
        }
        void search(int num,node *q)
        {
        do
        {
                if(q->info==num)
                {
                        printf("\nSearch success");
                        printf("\nSearch element %d",num);
                        return;
                }
                        q=q->next;
                }while(q!=p);
                printf("\nSearch unsuccessful");
                getch();
}
```

NOTE: All the algorithms will be same only conditions will be changed.

Implementation of merging of two Singly Linked List

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

struct Node
{
   int data;
   struct Node *next;
} *temp = NULL, *first = NULL, *second = NULL, *third = NULL, *last = NULL;

struct Node* Create (int A[], int n)
{
   int i;
   struct Node *t, *last;
   temp = (struct Node *) malloc(sizeof(struct Node));
   temp->data = A[0];
   temp->next = NULL;
   last = temp;
```

```
    for (i = 1; i < n; i++)
    {
        t = (struct Node *) malloc(sizeof(struct Node));
        t->data = A[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
    return temp;
}

void Display(struct Node *p)
{
    while (p != NULL)
    {
        printf ("%d ", p->data);
        p = p->next;
    }
}

void Merge(struct Node *first, struct Node *second)
{
    if (first->data < second->data)
    {
        third = last = first;
        first = first->next;
        last->next = NULL;
    }
    else
    {
        third = last = second;
        second = second->next;
        last->next = NULL;
    }

    while (first != NULL && second != NULL)
    {
        if (first->data < second->data)
        {
            last->next = first;
            last = first;
            first = first->next;
            last->next = NULL;
        }
        else
        {
            last->next = second;
            last = second;
            second = second->next;
            last->next = NULL;
        }
    }

    if (first != NULL)
        last->next = first;
    else
        last->next = second;
}

int main()
{
    int A[] = { 3, 4, 7, 9 };
    int B[] = { 2, 5, 6, 8 };
    first = Create (A, 4);
    second = Create (B, 4);

    printf ("1st Linked List: ");
    Display (first);

    printf ("\n2nd Linked List: ");
    Display (second);

    Merge (first, second);

    printf ("\n\nMerged Linked List: \n");
    Display (third);
  return 0;
}
```

Implementation of reversing of Singly Linked List

```
#include <stdio.h>

#include <stdlib.h>

/* Link list node */

struct Node {

    int data;
    struct Node* next;
};
/* Function to reverse the linked list */
static void reverse(struct Node** head_ref)
{
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next = NULL;
    while (current != NULL) {
        // Store next
        next = current->next;

        // Reverse current node's pointer
        current->next = prev;

        // Move pointers one position ahead.
        prev = current;
        current = next;
    }
    *head_ref = prev;
}

/* Function to push a node */
```

```
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node
        = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

/* Function to print linked list */
void printList(struct Node* head)
{
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

/* Driver code*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;


    push(&head, 20);
    push(&head, 4);
    push(&head, 15);
    push(&head, 85);


    printf("Given linked list\n");
    printList(head);
    reverse(&head);
    printf("\nReversed linked list \n");
    printList(head);
    return 0;
}
```