1. Queue

1. What is Queue and define Operation on Queue.
2. Define Types of queue.
3. Operation on Queue.
4. Write algorithm of Queue operation

   INSERT,DELETE,DISPLAY
5. WAP of Queue operation.
6. Limitation Of simple queue.
7. What is circular queue.
8. Write algorithm of circular Queue operation

   INSERT,DELETE,DISPLAY
9. WAP of circular Queue operation.
10. Difference between simple queue and circular queue.
11. Explain DEQUEUE(Double ended Queue)
12. Explain Priority Queue.
13. Write Application of queue.
14. Difference between stack and queue.

2. Linked List

1. Why we used linked list?
2. What is linked list.
3. Explain types of linked list.
4. Explain operation on link list.
5. Discuss advantages and disadvantages of linked list over array.

6. Explain operation on singly linked list.

1. Create singly linked list.
2. Traverse singly linked list.
3. Insertion into singly linked list.
   - At the beginning
   - At the end
   - After particular node.
   - In sorted.
4. Deletion from singly linked list.
   - At the beginning
   - At the end
   - Specific node.
   - Specific location.
5. Searching element into singly linked list
6. Count nodes in singly linked list.

7. Explain operation on Circular linked list.

1. Create circular linked list.
2. Traverse circular linked list.
3. Insertion into circular linked list.
   - At the beginning
   - At the end
   - After particular node
4. Deletion from circular linked list.
   - .At the beginning
   - At the end
   - Specific node

8. Explain operation on Doubly linked list.

1. Create doubly linked list.
2. Traverse doubly linked list.
   I. Forward direction
   II. Backward direction
3. Insertion into doubly linked list.
   I.At the beginning
   II.At the end
   III. After specific node

IV.In sorted list

4. Deletion from doubly linked list.
   I.From the beginning
   II.From the end
   III.From specific node

9. Application of linked list.

10. Implement stack using linked list.

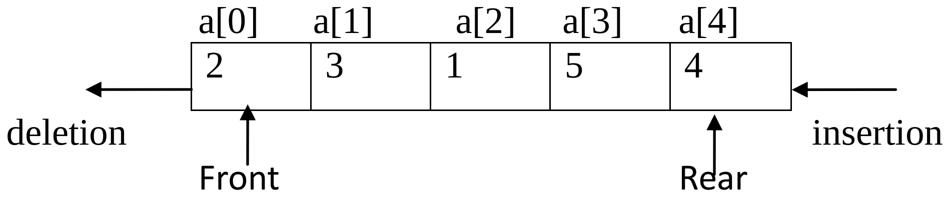
11. Implement Queue using linked list

1. Queue

1.What is Queue

Ans:

- A queue is non primitive linear data structure in which new element is added (insertion) at one end called rear end and existing element are deleted from other end called front end.
- In queue first element will be first deleted.
- So it is called FIFO(First In First Out).
- Ex:A queue at the movies,first person in queue get their tickets and left out,and new person are added at end of queue.



2. Types of Queue.

Ans:

1. Simple Queue
2. Circular Queue
3. Double ended Queue
4. Priority Queue

3.Operation on Queue.

Ans:

- Enqueue and Dequeue operation
- Insertion in queue is known as Enqueue operation
- Deletion in queue is known as Dequeue operation.

- 1.Insert

- Insert means to add element into queue.
- The insert operation insert new data at rear end.
- It increments rear pointer by one.

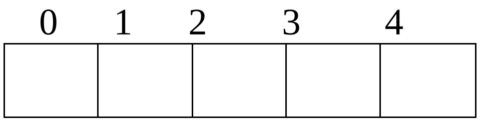
- 2.Delete

- Delete means to remove element from queue.

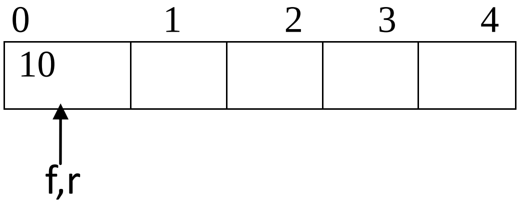
- The delete operation deletet existing data at front end .
- It increments front pointer by one.

- Process to inser or delete data in queue (f means Front ,r means Rear)

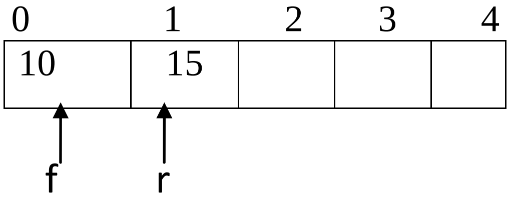
1.empty queue   f=-1 ,r=-1



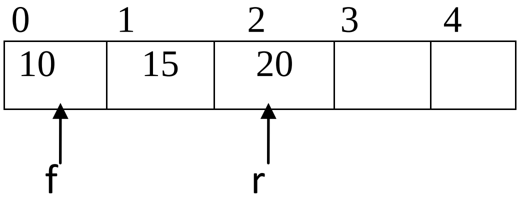
2.insert 10 in queue so   f=0,r=0



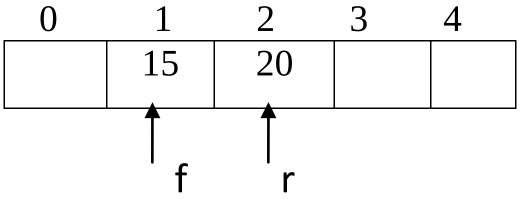
3.insert 15 in queue so   f=0,r=1(increment in rear)



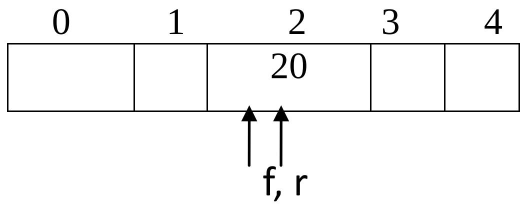
4. insert 20 in queue so   f=0,r=2(increment in rear)



5.Delete one element in queue so   f=1,r=2(increment front pointer)



6. Delete one element in queue so f=2,r=2(increment front pointer)



4.Write algorithm of Queue operation.

Ans:

<u>Insert:</u>

```
        Initially front=-1,rear= -1
  Insert(Q,front,rear,X)
  Step 1:     [Check for overflow]
              If rear>=N-1 then
              Write "queue overflow"
              Exit
  Step 2:     [increment rear pointer ]
              rear←rear+1
  Step 3:     [perform insertion]
              Q[r]=X
  Step 4:     [Is frot pointer properly set?]
              If  (front=-1) then
              front=0
              exit
  Step 5:     [finished]
              Exit
```

<u>Delete:</u>

- Initially front=-1,rear= -1

```
  Delete (Q,front,rear,X)
  Step 1:     [Check for underflow]
              If (front=-1)  then
              Write "queue underflow"
              Exit
  Step 2:     [delete element]
              Y=q[front]
  Step 3:     [increment front]
```

```
            If (front==rear) then
            front←rear←-1
            Else
            front←front+1
Step 4:     [finished]
            Exit
```

5.Implementation Queue

Ans:

```
void main()
{
     int q[10],no,n,ch,f=-1,r=-1,i,x,y;
    clrscr();
     printf("\n\nEnter the size of QUEUE:-> ");
    scanf("%d",&n);
    printf("1:INSERT \n");
    printf("2:DELETE \n");
    printf("3:DISPLAY \n");
    printf("4:EXIT \n");
     do
    {
            printf("\n\nEnter the choice:-> ");
            scanf("%d",&ch);
            switch(ch)
            {
                    case 1 :
                            if(r>=n-1)
                            {
                                     printf("\n\n----------------QUEUE OVERFLOW---------------");
                                    break;
                            }
                             printf("\n\nEnter the element:-> ");
                            scanf("%d",&no);
                            if(f==-1)
                            f=r=0;
                            else
                             r=r+1;
                            q[r]=no;
                    break;
```

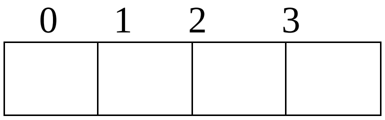
```
                case 2:
                    if(f==-1)
                    {
                        printf("\n\n------------QUEUE UNDERFLOW-------------");
                        break;
                    }
                    y=q[f];
                    printf("\n\nDeleted Element is :-> %d",y);
                    if(f==r)
                    {
                        f=r=-1;
                    }
                     else
                    {
                        f=f+1;
                    }
                 break;

                case 3:
                    printf("\n\nElements of QUEUE are :->\n");
                    printf("\n\n--------------------------\n");
                    if(f==-1 && r==-1)
                    {
                        printf("QUEUE IS UNDERFLOW\n");
                    }
                     else
                    {
                        for(i=f;i<=r;i++)
                         {
                            printf("%10d \n",q[i]);
                        }
                    }
                    printf("--------------------------");
                  break;

                 case 4:
                    exit(0);
            }
        }while(ch>=1 && ch<=4);
        getch();
  }
```

6. Limitation or disadvantages Of simple queue.

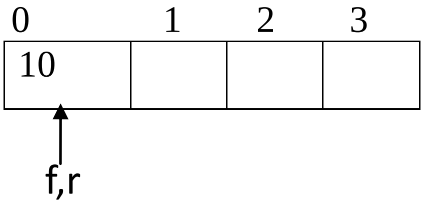
Ans:

- Trace of Simple Queue:

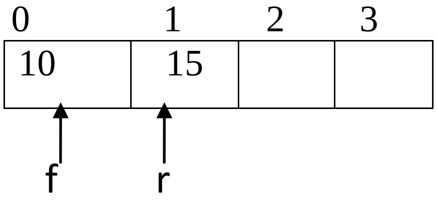
1.empty queue f=-1 ,r=-1



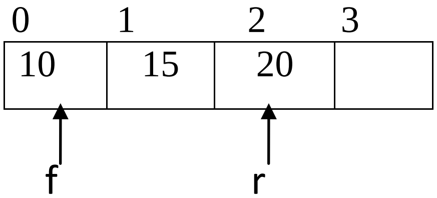
2.insert 10 in queue so f=0,r=0



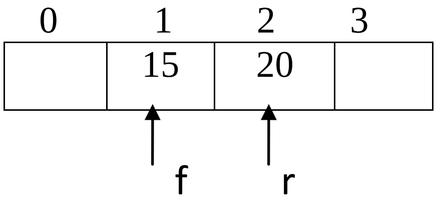
3.insert 15 in queue so f=0,r=1(increment in rear)



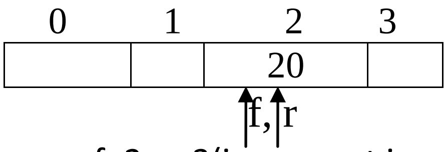
4. insert 20 in queue so f=0,r=2(increment in rear)



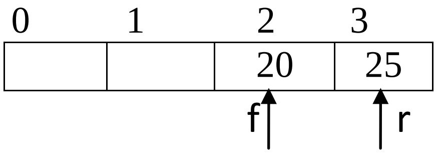
5.Delete one element in queue so f=1,r=2(increment front pointer)



6. Delete one element in queue so   f=2,r=2(increment front pointer)



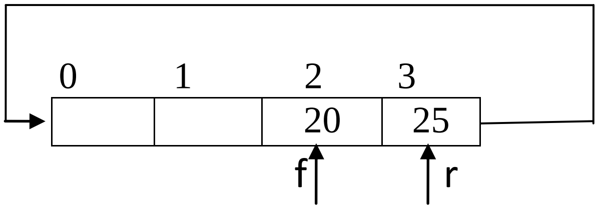
7.insert 25 in queue so   f=2 ,r=3(increment in rear)

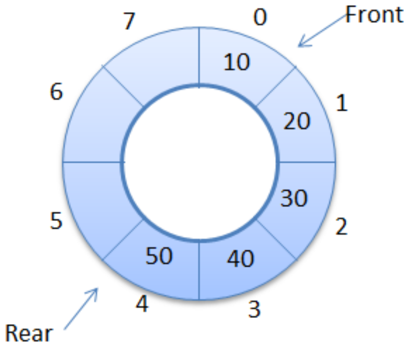


- Disadvantages is that in last operation rear pointer reaches to end of queue and queue overflow condition(rear>=n-1) is true.
- So we can not insert new element.even though there is space in queue.
- One solution is to move entire queue to the beginning of array,changing front and rear pointer accordingly and then inserting element into queue.
- But for array such operation is time consuming and expensive.
- So this type of problems can be sloved by using new concept,which is called circular queue.

7.What is Circular Queue.

Ans:

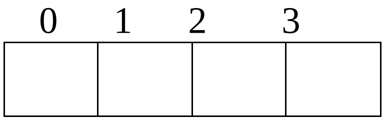




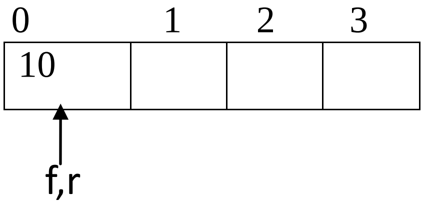
- Circular queue is one in which the insertion of new element is done at very first location of queue if last location of queue is full.
- So we can say that queue is called circular queue when first element comes just after last element.
- In circular queue both ends of queue are joined together so that once pointer either front or rear reaches to the end of queue they can move to the start of queue.
- This way we can user same location again and again.

- Trace of Circular Queue:

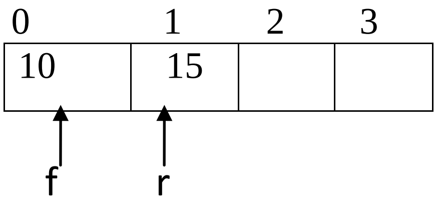
1.empty queue f=-1 ,r=-1



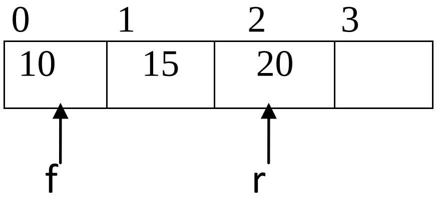
2.insert 10 in queue so f=0,r=0



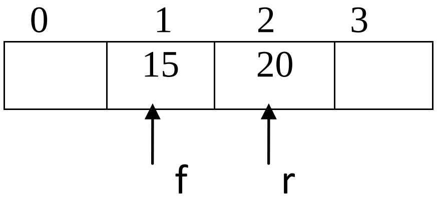
3.insert 15 in queue so f=0,r=1(increment in rear)



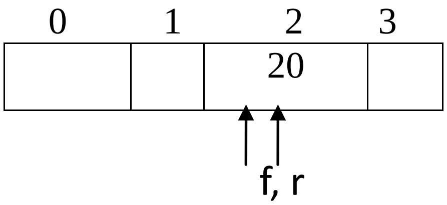
4. insert 20 in queue so f=0,r=2(increment in rear)



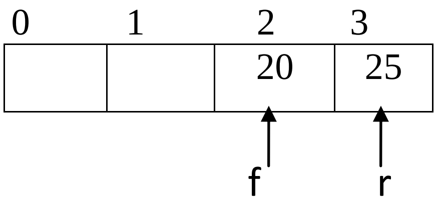
5.Delete one element in queue so f=1,r=2(increment front pointer)



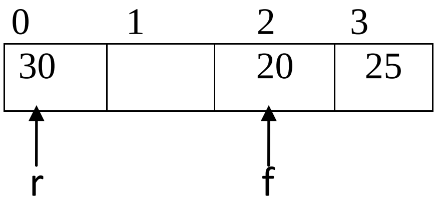
6. Delete one element in queue so f=2,r=2(increment front pointer)



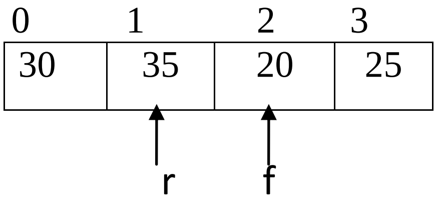
7.insert 25 in queue so f=2,r=3(increment in rear)



8. insert 30 in queue so f=2,**r=0**(increment in rear)



insert 35 in queue so f=2, r=1(increment in rear)



8.Write algorithm of circular Queue

Ans: **<u>Insert:</u>**

- Initially front=-1,rear= -1 ,n→no of elements

```
Insert(Q,front,rear,X)
Step 1:     [Check for overflow]
            If front=0 and rear=n-1 or front =rear+1 then
            Write “queue overflow”
            Exit
Step 2:     [Is rear  pointer properly set?]
            If  (front=-1) then
            front= rear=0
           else
            rear←(rear+1)%n

Step 3:     Q[rear]=no

Step 4:    [finished]
            Exit
```

<u>Delete:</u>

- Initially front=-1,rear= -1

```
Delete (Q,front,rear,X)
Step 1:     [Check for underflow]
            If (front=-1)  then
            Write “queue underflow”
            Exit

Step 2:     [delete element]
            Y=q[front]
```

```
Step 3:     [increment front pointer]
            If (front==rear) then
            front←rear←-1
            Else
            front←(front+1)%n

Step 4:     [finished]
            Exit
```

9.Implementation of circular Queue

Ans:

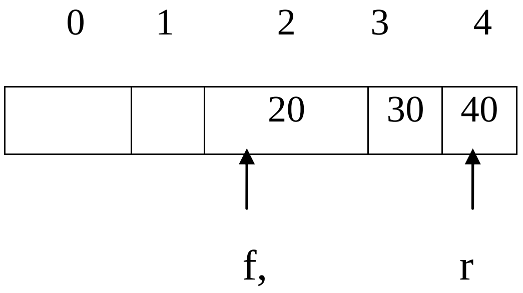
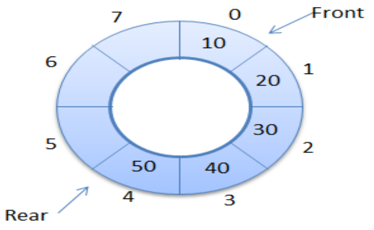
```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 5
int Q[SIZE];

int front = -1, rear =-1;


void CQ_insert()
{
   int no;
   if( (front == rear + 1) || (front == 0 && rear == SIZE-1))
   {
   printf("\n Queue is full!! \n");
   return;
   }
   else
   {
      printf("Enter value: ");
      scanf("%d",&no);
      if(front == -1)
      {
              front = 0;
              rear=0;

      }
      else
```

```
        {
        rear = (rear + 1) % SIZE;
        }
        Q[rear] = no;
        printf("\n Inserted -> %d", no);
    }
}


int CQ_delete()
{
    int no;
    if(front == -1)
    {
        printf("\n Queue is empty !! \n");
        return;
    }
    else
    {
        no = Q[front];
        if (front == rear)
        {
            front = -1;
            rear = -1;
        }
        else
        {
            front = (front + 1) % SIZE;

        }
        printf("\n Deleted element -> %d \n", no);
     }
}




void CQ_display()
{
    int i;
    if(front == -1)
    {
       printf("\n Queue is empty !! \n");
       return;
    }

    else
    {
       for( i = front; i!=rear; i=(i+1)%SIZE)
       {
          printf("%d\t",Q[i]);
       }
       printf("%d ",Q[i]);

    }
}

int main()
{
    int ch;
    while(1)
    {
       printf("\n1. insert");
       printf("\n2. delete");
       printf("\n3. display");
       printf("\n4. Exit");
       printf("Enter your choice: ");
       scanf("%d",&ch);
       switch(ch)
       {
               case 1: CQ_insert();
                       break;
               case 2: CQ_delete();
                       break;
               case 3: CQ_display();
                       break;
               case 4: exit(0);

               default: printf("Enter choice between 1 to 4");
```

```
        }
    }



    return 0;
}
```

10.Difference between simple queue and circular queue.

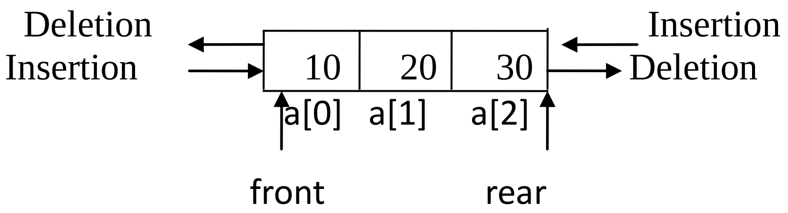
Ans

Simple Queue	Circular Queue
Insertion is performed in linear manner	Insertion is performed in circular manner
While inserting element it consider only rear pointer	while inserting element it consider both rear and front pointer.
It does not make efficient use of memory	It make efficient use of memory
It shows "queue overflow" message if rear pointer point to the last element. Queue is overflow when (r= =n-1) 	It does not show "queue overflow" message if rear pointer point to the last element circular queue overflow when (f==0 && r=n-1) || (r=f+1)  7 0 Front 10 6 20 1 30 5 2 50 40 Rear 4 3

11.Explain DEQUEUE(Double ended Queue)

Ans:

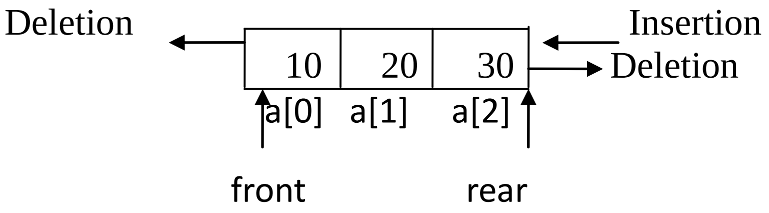
- It is lnear DS in which insertion and deletion operations are performed from both ends.
- That is we can insert elements from rear end or from front ends.
- Hence it is called double ended queue or Dequeue.



- There are two types of dequeue:

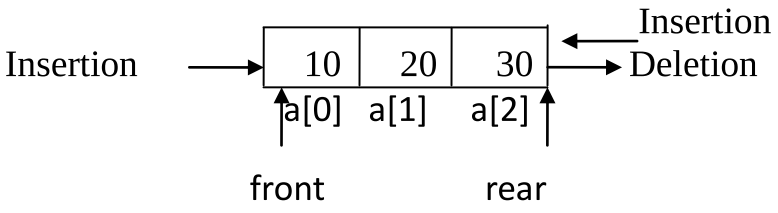
1.input restricted dequeue
2.output restricted dequeue

1. Input restricted dequeue:
Insertion at one end but deletion are possible on both end.



2.Output restricted dequeue:

Deletion at one end but insertion are possible on both end.



12.Explain Priority QUEUE

Ans:

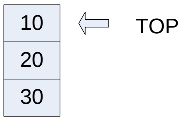
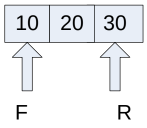
- A queue in which it is possible to insert element or remove element at any position depending on some priority is called priority queue.
- Every item has a priority associated with it.

- An element with high priority is dequeued before an element with low priority.
- If two elements have the same priority, they are served according to their order in the queue.
  - **ApplicationsofPriorityQueue:**
    1)CPUScheduling
    2) Graph algorithms like Dijkstra's shortest path algorithm, Prim's Minimum Spanning Tree,etc
    3) All queue applications where priority is involved.

.

13.Difference between Stack and Queue

Ans:

Stack	Queue
Stack is linear DS in which we inserted and deleted element from one end that is called TOP	Queue is linear DS in which we inserted element from one end(REAR) and deleted element from another end(front) .
It works in LIFO(Last In First Out) manner	It works in FIFO(First In First Out) manner
It has only one control variable i.e. TOP	It has two control variable i.e. front and rear
Stack does not have subtypes	Queue has subtypes : Simple queue circular queue double ended queue priority queue
Application: evaluation of postfix notation string reverse In recursion	Application: process scheduling Disk scheduling Incoming resource request for diff processes
	

4. Linked List

1. Why we used linked list?

Ans:

- There are many application in which linear aloocation of data is unacceptable due to following reason.

Unpredictable storage requirement:

- The exact amount of data storage required depends on the actual data being processed.
- This information may not be available at the time of writing program.

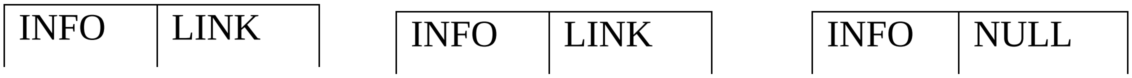
Extensive manipulation of stored data .

- If insertion and deletion frequently occurs in our program then array can not be used efficiently.
- So if we use linked list then we will get best result in terms of storage and time.

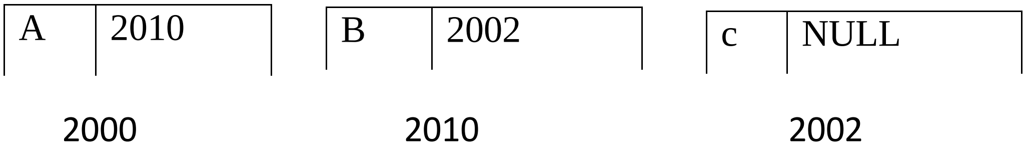
2. What is linked list.

Ans:

- In linked list we use pointer or links to refer the address of next element it means in linked list elements are logically adjacent need not to be physically adjacent.
- Linked list are special list of some data elements linked to one another.
- The logical ordering is represented by having each element pointing to next element.
- Each element is called node which has two parts.1.INFO 2. LINK



- 1.INFO: this part contains actual element of the list
- 2.LINK: this part contain address of next node in the list
- Link of last node contain special value known as “NULL” which indicate end of list
- List with no node or list at all is called empty list.
- Example:



3. Explain types of linked list.

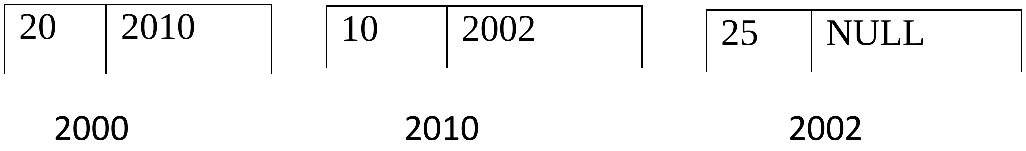
Ans:

1. Singly or linear linked list.
2. Doubly linked list.

3. Circular linked list.
4. Circular doubly linked list.

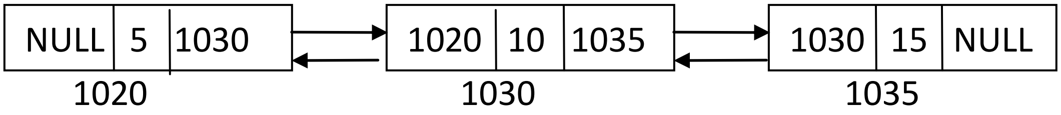
1.Singly linked list

- Singly linked list is one in which all nodes are linked together in some sequential manner
- Hence,it is also called linear linked list.
- Each node has single pointer to the next node.
- We can traverse only in one direction,forward direction.
- So we can not access previous node from current node.
- Example:



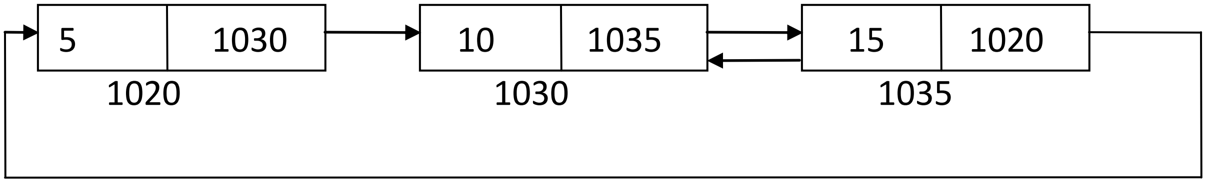
2.Doubly linked list

- It is also called two way linked list.
- Doubly linked list is one in which nodes are linked together by multiple links .
- So we can access successor(next) node and predecessor(previous)node.
- Therefore each node in a list points both node .
- This help us to traverse in both direction.
- Example:



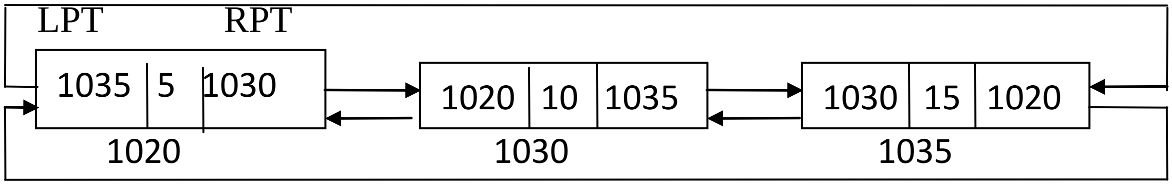
3. Circular linked list

- It likes a linear linked list in which link part of the last node contains address of first node in the list.
- i.e. second part of last node does not point to null pointer rather it points back to beginning of linked list.
- Time saving,when we want to move from last node to first node.
- Example:



4.Circular Doubly linked list

- A circular doubly linked list has both successor and predecessor pointer in circular manner.
- In this LPT(leftpart) of first node contains address of last node and RPT(right part) of last node contains address of first node.
- Example:



4. Explain operation on link list.

Ans:

.Creation.

- To enter value into node of linked list.

2.Traversing

- To display value of every node of linked list.

3.Searching.

- To search particular node from linked list.

4.Count.

- To count number of nodes in linked list.

5.Insertion.

- Insert node at beginning of linked list
- Insert node at end of linked list
- Insert node after/before specific node in linked list.
- Insert node in sorted linked list

6.Deletion.

- Delete node from beginning of linked list.
- Delete node from end of linked list.
- Delete node from specific node in linked list.

5. **Discuss advantages and disadvantages of linked list over array.**

Ans:

- **Advantages of array.**
- We can access any element of array directly means random access is easy.
- It can be used to create other useful data structure like stack and queue.
- It occupy less memory compared to other structures.

- **Disadvantages of array.**
- Its size is fix.
- It can not be dynamically resized.
- It is difficult to insert and delete element.
- Size of all elements must be same.

- **Advantages of Linked List.**
- Linked list are dynamic data structure it means it can grow or shrink during execution of a program.
- Efficient memory utilization;
- Memory is not pre allocated. Memory is allocated whenever it is required and it is deallocated when it is no longer needed.
- Indertion and deletion are easier and efficient.

- **Disadvantages of Linked List.**
- We can not access element randomly .We have to access elements sequentially starting from the first node.
- It can not be easily sorted.
- More complex to create than array.
- Extra memory space for pointer is required with each element of the list.

6. **Explain operation on singly linked list.**

Ans:

1. Create singly linked list.
2. Traverse singly linked list.
3. Insertion into singly linked list.

    I. At the beginning
    II. At the end
    III. After particular node.
    IV. In sorted.

4. Deletion from singly linked list.
    - I. At the beginning
    - II. At the end
    - III. Specific node.
    - IV. Specific location
5. Searching element into singly linked list
6. Count nodes in singly linked list.

- **<u>1.Algorithm : Create singly linked list</u>.**

```
Step 1: [check for avalability]
        ptr=(structnode*) malloc (sizeof(structnode))
        If ptr==NULL then
        Write “overflow”
        Exit
        End if

Step 2:  ptr →info=value
         First=ptr

Step 3: ch==’y’

Step 4: repeat step 5 to 6 while ch==’y’

Step 5: temp=(struct node*) malloc (sizeof(structnode))
        temp→info = value
        Ptr→ link = temp
        Ptr = temp

Stpe 6: repeat choice (y/n)

Stpe 7: ptr→link =NULL

Step 8 :stop
```

- **2.Algorithm : Traverse singly linked list.**

```
Step 1: [check for underflow]
            If first==NULL then
            Write “list is empty”
            Exit
            End if

Step 2:  ptr=first

Step 3: Repeat step 3 to 4 while (ptr !=NULL)

Step 4: print ptr→ info

Step 5: ptr = ptr →link

Step 6 :stop
```

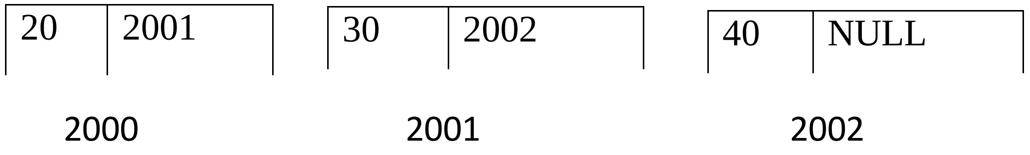
❖ **.Programme : Create and Traverse singly linked list.**

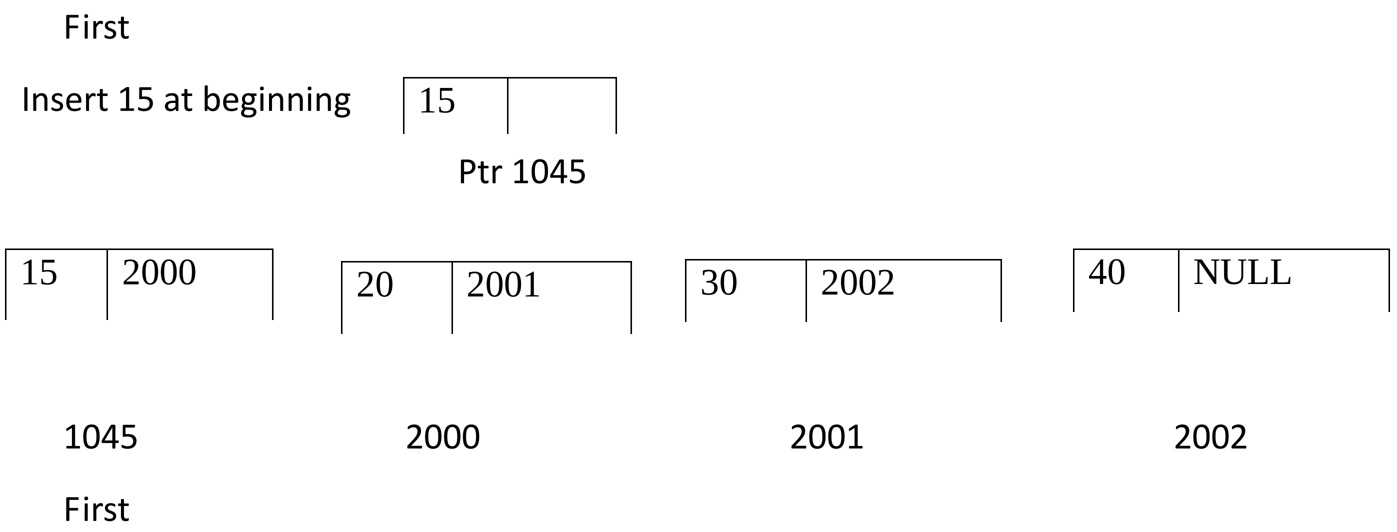
```
struct node
{
         int info;
         struct node * link;
}*temp,*ptr,*first;
void main()
{
         int ch,m;
         char ch1;
         clrscr();
        printf("\n1 create");
        printf("\n2 display");
        printf("\n3 exit");
       do
       {
                  printf("\nenter the choice");
                  scanf("%d",&ch);
                  switch(ch)
                  {
                           case 1:
                                    ptr=(struct node *)malloc(sizeof(struct node *));
```

```
                printf("\n enter the first node");
                scanf("%d",&ptr->info);
                first=ptr;
                printf("\ndo you want to continue");
                ch1=getch();
                while(ch1=='y')
                {
                    printf("\nenter node");
                    temp=(struct node *)malloc(sizeof(struct node *));
                    scanf("%d",&temp->info);
                    ptr->link=temp;
                    ptr=temp;
                    printf("\ndo you want to continue");
                    ch1=getch();
                }
                ptr->link=NULL;
            break;
            case 2:
                ptr=first;
                while(ptr!=NULL)
                {
                    printf("\n%d",ptr->info);
                    ptr=ptr->link;
                }
            break;
            case 3:
                exit(0);
            break;
        }
 }   while(ch>=1 && ch<=3);
     getch();
}
```

- **<u>3.Insertion into singly linked list.</u>**

  - **<u>At the beginning:-</u>**

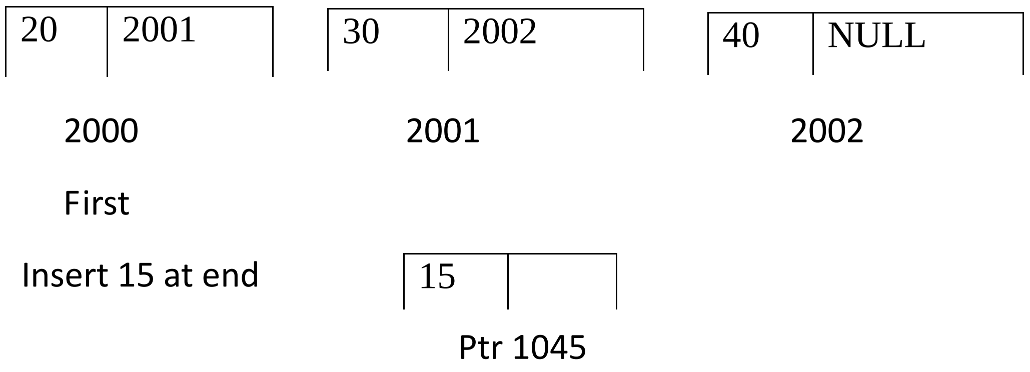


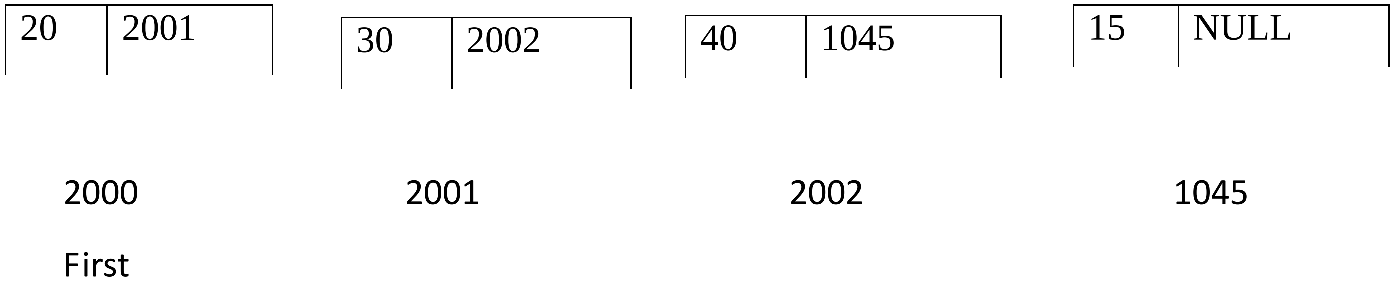


- **Algorithm : Insertion at beginning into singly linked list.**

```
Step 1: [check for availability]
          ptr=(structnode*) malloc (sizeof(structnode))
            If ptr==NULL then
            Write “overflow”
            End if
            Exit

Step 2:  ptr→info = value

Step 3: ptr→link  =first
            First=ptr

Step 4 :stop
```

- **At the end:-**

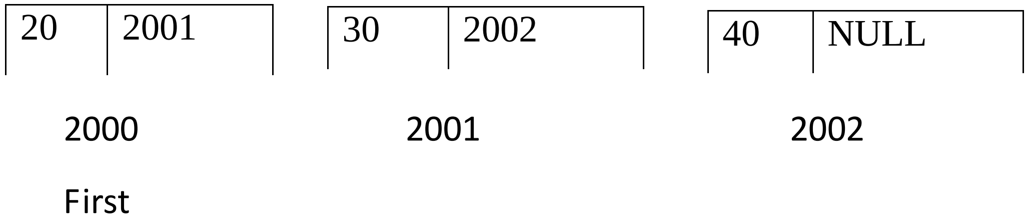


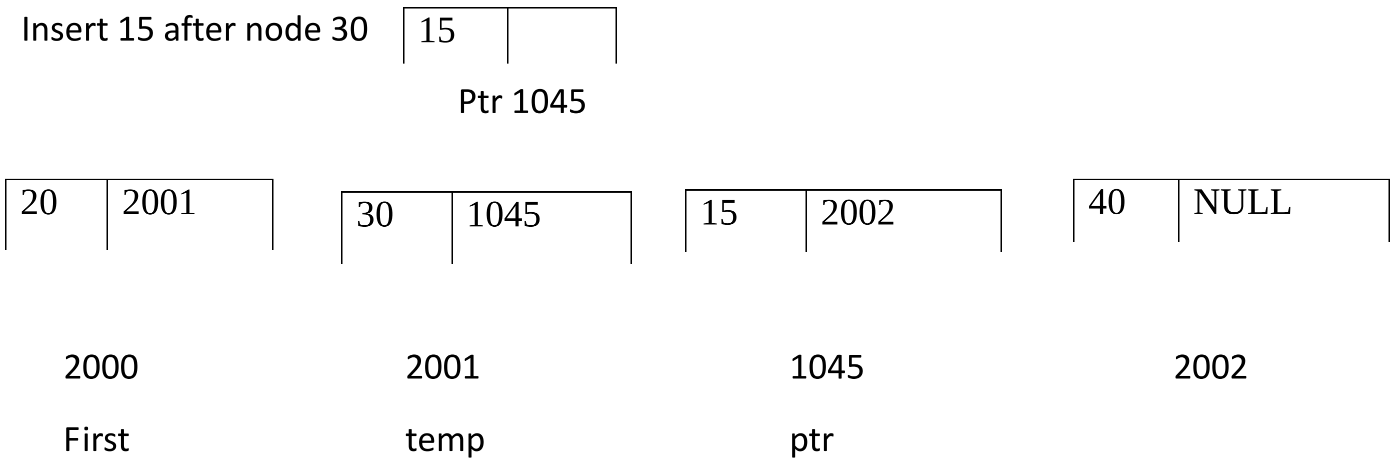


- **Algorithm : Insertion at end into singly linked list.**

```
Step 1: [check for availability]
          ptr=(structnode*) malloc (sizeof(structnode))
          If ptr==NULL then
          Write “overrflow”
           End if
           Exit

Step 2:  ptr→info = value

Step 3: temp = first

Step 4:repeat step 5 while (temp→link ! =NULL)

Step 5: temp= temp→link

Step 6: temp→link = ptr
           Ptr→link = NULL

Step 7 :stop
```

- **After the specific node:-**

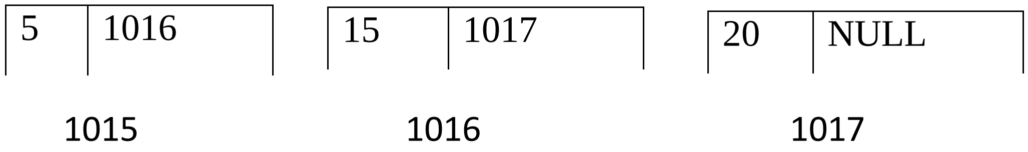


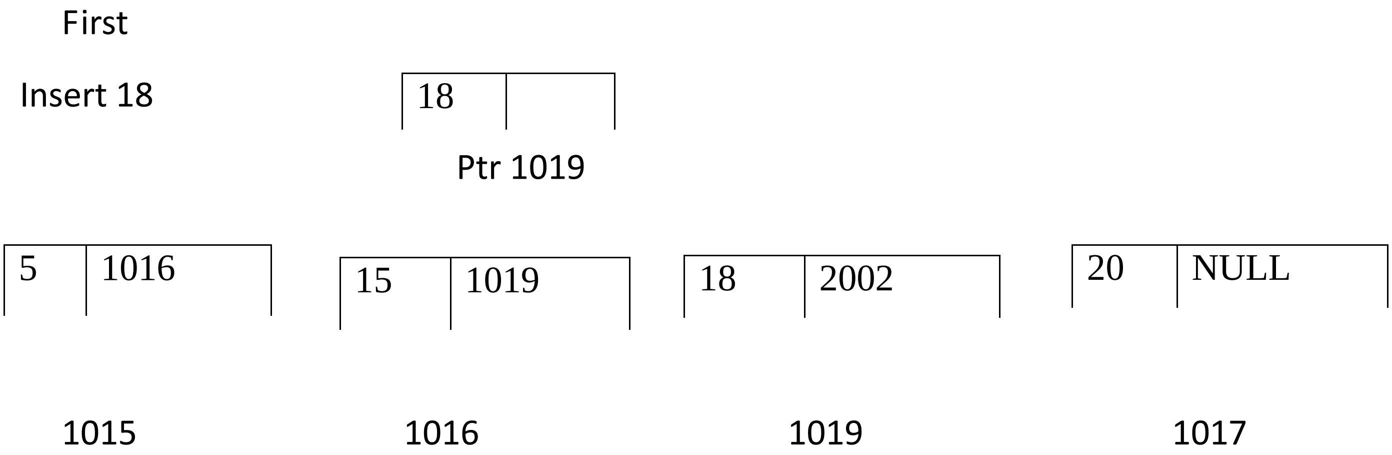


- **<u>Algorithm : Insertion After particular node into singly linked list</u>.**

```
Step 1: [check for availability]
          ptr=(structnode*) malloc (sizeof(structnode))
          If ptr==NULL then
          Write "overflow"
          End if
          Exit

Step 2:  ptr→info = value

Step 3:read data

Step 4:  temp = first

Step 5: repeat step 6 while (temp→info ! =data)

Step 6: temp= temp→link

Step 7: ptr→link = temp→link
          temp→link=ptr

Step 8 :stop
```

- **<u>In sorted list:-</u>**





- **<u>Algorithm : Insertion in sorted list into singly linked list</u>.**

```
Step 1: [check for availability]
          ptr=(structnode*) malloc (sizeof(structnode))
          If ptr==NULL then
          Write "overflow"
          Exit
          End if

Step 2:  ptr→info = value

Step 3: temp= first

Step 4:  if (ptr->info<temp->info) then
              Ptr->link=temp
              First=ptr
              Break;

 Step 5:else  repeat step 6 while (ptr→info >temp->info)

Step 6: prev=temp
          temp= temp→link
          if(temp==NULL) then break;

Step 7: prev→link = ptr
          ptr→link=temp

Step 8 :stop
```

❖ **Programme : Insertion (beginning,end,after node,insorted) into singly linked list.**

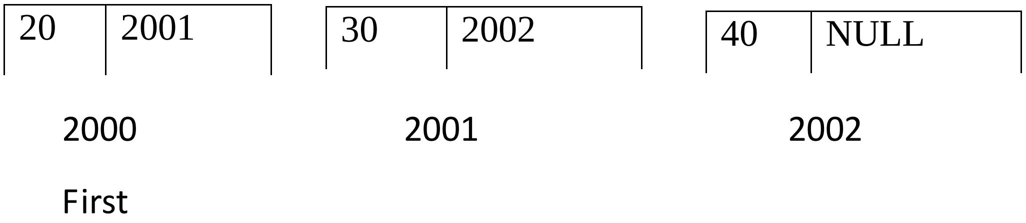
```
struct node
{
        int info;
        struct node * link;
}*temp,*ptr,*first,*prev;
 void main()
 {
        int ch,m;
        char ch1;
        clrscr();
        printf("\n1 insert at begining");
          printf("\n2 insert at end");
          printf("\n3 insert after  specific node");
         printf("\n4 insert in sorted list");
          do
           {
                 printf("\nenter the choice");
                 scanf("%d",&ch);
                 switch(ch)
                 {
                          case 1:
                                   ptr=(struct node *)malloc(sizeof(struct node *));
                                  printf("\nEnter new node");
                                  scanf("%d",&ptr->info);
                                   if(ptr==NULL)
                                  {
                                          printf("\n overflow");
                                          break;
                                  }
                                  ptr->link=first;
                                  first=ptr;
                         break;

                          case 2:
                                   ptr=(struct node *)malloc(sizeof(struct node *));
                                   printf("\nEnter new node");
                                  scanf("%d",&ptr->info);
                                   temp=first;
                                  while(temp->link!=NULL)
```

```
		{
			temp=temp->link;
		}
		temp->link=ptr;
		ptr->link=NULL;
	break;

	case 3:
		ptr=(struct node *)malloc(sizeof(struct node *));
		printf("\nEnter new node");
		scanf("%d",&ptr->info);
		printf("\n enter the data");
		scanf("%d",&m);
		temp=first;
		while(temp->info!=m)
		{
			temp=temp->link;
		}
		ptr->link=temp->link;
		temp->link=ptr;
	break;

	case 4:
		ptr=(struct node *)malloc(sizeof(struct node *));
		if(ptr==NULL)
		{
			Printf("empty");
			Break;
		}
		printf("\nEnter new node");
		scanf("%d",&ptr->info);
		temp=first;
		if(ptr->info<temp->info)
		{
			Ptr->link=temp;
			First=ptr;
			Break;
		}
		Else
		{
```

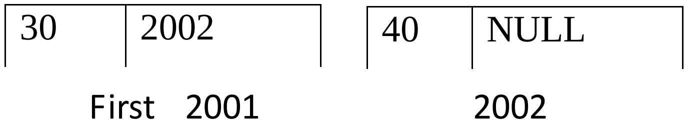
```
                              while(ptr→info>temp->info)
                              {
                                    prev=temp
                                    temp=temp→link;
                                    if(temp==NULL)
                                    break;
                              }
                              prev→link =ptr;
                              Ptr→link=temp;
                              }
                              Break;
              }while(ch>=1 && ch<=4);
               getch();
         }
```

<u>4.Deletion from singly linked list.</u>

- <u>At the beginning:-</u>



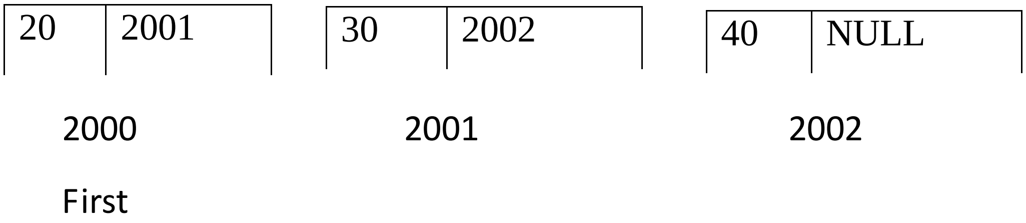
Delete node 20



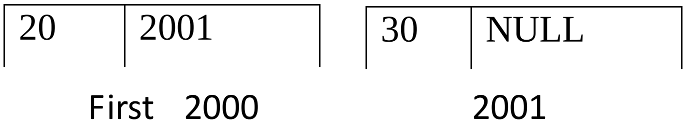
- **<u>Algorithm: Deletion at beginning from singly linked list</u>.**

```
Step 1: [check for underflow]
        If first==NULL then
        Write "list is empty"
        Exit
        End if

Step 2:  :[print deleted element]
               first→info

Step 3: first = first→link
Step 5: [free memory size]
        free(first)

stop 6: stop
```

- **<u>At the end:-</u>**



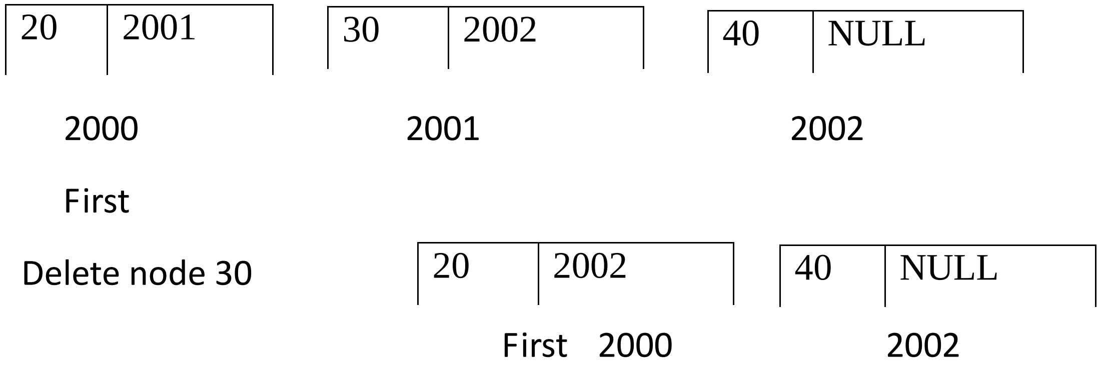
Delete node 40



- **<u>Algorithm: Deletion at end from singly linked list</u>.**

```
Step 1: [check for underflow]
        If first==NULL then
        Write "list is empty"
        Exit
        End if

Step 2: if first→link == NULL then
        Print first→info

        First=NULL
        Free(first)

Step 3:else
        temp=first
        Repeat step 4 while temp →link != NULL

Step 4 : prev = temp
         temp = temp→link

Step 5: temp →link = NULL
Step 6: free(temp)

stop 7: stop
```

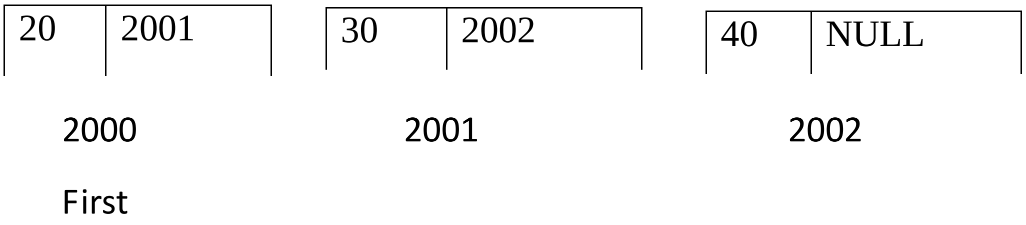
- **<u>Delete particular node:-</u>**



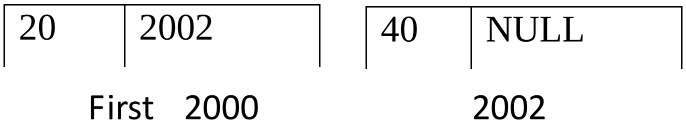
- **<u>Algorithm: Deletion specific node from singly linked list</u>.**

```
Step 1: [check for underflow]
          If first==NULL then
          Write "list is empty"
          Exit
          End if

Step 2: read data

Step 3: temp = first

Step 4:if first →info == data then
                   Print first→info

           First = first→link

Step 5 : else repeat step 6 to 8

Step 6: repeat step 7 while (temp→info !=data)

Step 7: prev = temp
          temp = temp→link

Step 8: print temp→info
Step 9: prev→link = temp→link

Step 10: free(temp)

stop 11: stop
```

- **Delete from specific position:-**



Delete position 2



- **Algorithm: Deletion from specific position from singly linked list.**

```
Step 1: [check for underflow]
        If first==NULL then
        Write "list is empty"
        Exit
        End if

Step 2: read location

Step 3: if loation = =1 then perform step 4

Step 4: print first->info
        if(first->link == NULL)
        {
            first=NULL;
            free(first);
        }
        else
        {
                ptr = first;
                first=first->next;
                free(ptr);
        }

Step 5: temp = first;

Step 6: for(i=1;i<loc;i++) perform step 7

Step 7: prev = temp;
        if(temp->link!=NULL)
        {
                temp = temp->link;
        }
        else
        {
                printf("\n \tNo node at specified location \n");
        }

Step 8: printf temp->info;
         prev->link = temp->link;
         free(temp);

stop 9: stop
```

- **<u>Programme : Deletion (beginning,end,particular position) from singly linked list</u>.**

```
struct node
{
     int info;
     struct node * link;
     }*cpt,*ptr,*first;

     void main()
     {
         int ch,m,data;
         char ch1;
         clrscr();
           printf("\n1 delete at begining");
           printf("\n2 delete at end");
         printf("\n3 delete specific node");
         printf("\n4 delete specific position");
```

```
do
{
            printf("\nenter the choice");
           scanf("%d",&ch);
           switch(ch)
           {
                      case 1:
                                 if(first==NULL)
                                 {
                                            printf("\n underflow");
                                            break;
                                 }
                                 printf("deleted no.is %d",first->info);
                                 temp=first;
                                  first=first->link;
                                 free(temp);
                      break;

                      case 2:
                                  if(first==NULL)
                                 {
                                            printf("\n underflow");
                                            break;
                                 }
                                 if(first->link==NULL)
                                 {
                                            printf("deleted no.is %d",first->info);
                                            first=NULL;
                                            free(first);
                                 }
                                  else
                                 {
                                            temp=first;
                                            while(temp->link!=NULL)
                                            {
                                                       prev=temp;
                                                       temp=temp->link;
                                            }
                                            prev->link=NULL;
                                 printf("deleted no.is %d",temp->info);
```

```
         free(temp);
        }
break;

case 3:
         if(first==NULL)
        {
                printf("\n underflow");
                break;
         }
        temp=first
        Printf(“enter deleted node”)
        Scanf(“%d”,&data);
         if(first->info==data)
        {
                 first=first->link
                 printf("deleted no.is %d",temp->info);
        }
        else
        {
                while(temp->info!=data)
                {
                        prev=temp;
                        temp=temp->link;
                }
                  prev->link=temp->link;
         printf("deleted no.is %d",temp->info);
        }
        free(temp);

break;
case 4:
        if(first==NULL)
        {
                printf("\n underflow");
                break;
         }
        printf("\n \tEnter a Location : ");
        scanf("%d",&loc);
        if(loc == 1)
```

```
                {
                    printf("\n \tDeleted node is  : %d \n",first->info);
                    if(first->link == NULL)
                    {
                         first=NULL;
                        free(first);
                    }
                    else
                    {
                        ptr = first;
                        first=first->link;
                        free(ptr);
                    }
                }
                else
                {
                    temp = first;
                    for(i=1;i<loc;i++)
                    {
                        prev = temp;
                        if(temp->link!=NULL)
                        {
                            temp = temp->link;
                        }
                        else
                        {
                            printf("\n \tNo node at specified location \n");
                        }
                    }
                printf("\n \tDeleted node is  : %d \n",temp->info);
                prev->link = temp->link;
                free(temp);
                }
        }
  }while(ch>=1 && ch<=4);
  getch();
}
```

5. Searching node into singly linked list.

- Algorithm: searching node from singly linked list.

```
Step 1: [check for empty list]
        If first==NULL then
        Write “list is empty”
        Exit
        End if

Step 2: read data
          read X

Ste p3: [initialize]
          Flag ← 0

Step 4: [search entire list]
           temp=first
          Repeat step 5 while temp!= NULL

Step 5:if temp→info==X then
        Flag←1
        printf("node found”);
        Break;
        Else
        temp = temp←link

step 6: If(flag ==0)
        printf(“node not found”);

stop 7: stop
```

6. Count node into singly linked list.

- Algorithm: count node from singly linked list.

```
Step 1: [check for empty list]
        If first==NULL then
        Write “list is empty”
        Exit
```

```
          End if

Step 2: count ←0
            temp=first
            Repeat step 3 while  temp!= NULL

Step 3: count ← count +1
           temp = temp←link

step 4: print count

Step 5:stop
```

- **Programme : Searching ,counting into singly linked list.**

```
struct node
{
      int info;
      struct node * link;
      }*temp,*ptr,*first;

      void main()
      {
           int ch,m,data,flag=0,count;
           char ch1;
           clrscr();
            printf("\n1 searching");
           printf("\n2 count");
           do
           {
                    printf("\nenter the choice");
                     scanf("%d",&ch);
                    switch(ch)
                    {
                             case 1:
                                      if(first==NULL)
                                      {
                                               printf("\n list is empty");
                                               break;
                                      }
```

```
                Printf(“enter search data”);
                Scanf(“%d”,&X);
                temp=first;
                while(temp != NULL)
                {
                    If (temp→info = = X)
                    {
                        printf("node found”);
                        Flag=1;
                        Break;
                    }
                 temp=temp->link;
                 }
                 If(flag ==0
                    printf(“node not found”);
            break;

            case 2:
                 if(first==NULL)
                {
                    printf("\n underflow");
                    break;
                }
                Count=0;
                temp=first;
                while(temp !=NULL)
                {
                    Count=count+1;
                    temp=temp→link;
                }
                    printf("total number of nodes is is %d",count);
            break;
            case 3:
                 exit(0)
            break;

  }while(ch>=1 && ch<=3);
  getch();
}
```

7. **Explain operation on Circular linked list.**

Ans :

1. Create circular linked list.
2. Traverse circular linked list.
3. Insertion into circular linked list.
   i. At the beginning
   ii. At the end
   iii.After particular node.
4. Deletion from circular linked list.
   i.At the beginning.
   ii.At the end.
   iiiAt Specific Node

1.Algorithm : Create circular linked list.

```
Step 1: [check for overflow]
          ptr=(structnode*) malloc (sizeof(structnode))
          If ptr==NULL then
          Write "overflow"
          Exit
          End if

Step 2:   ptr →info=value
            First=ptr

Step 3: ch=='y'

Step 4: repeat step 5 to 6 while ch=='y'

Step 5: temp=(struct node*) malloc (sizeof(structnode))
          temp →info = value
          Ptr→ link = temp
          Ptr = temp

Stpe 6: repeat choice (y/n)

Stpe 7: ptr→link =first

Step 8 :stop
```

2.Algorithm : Traverse circular linked list.

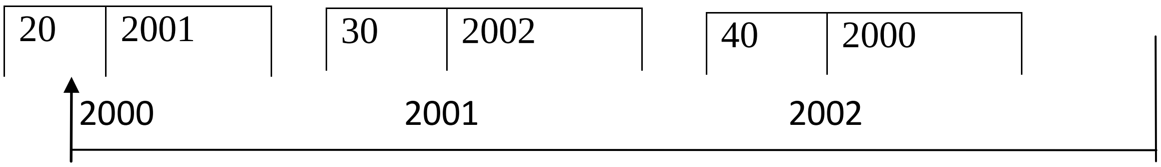
```
Step 1: [check for underflow]
          If first==NULL then
          Write "list is empty"
          Exit
          End if

Step 2:    ptr=first
             print ptr -> info
            ptr = ptr →link

Step 3: Repeat step 2  while (ptr !=first)

Step 4 :stop
```

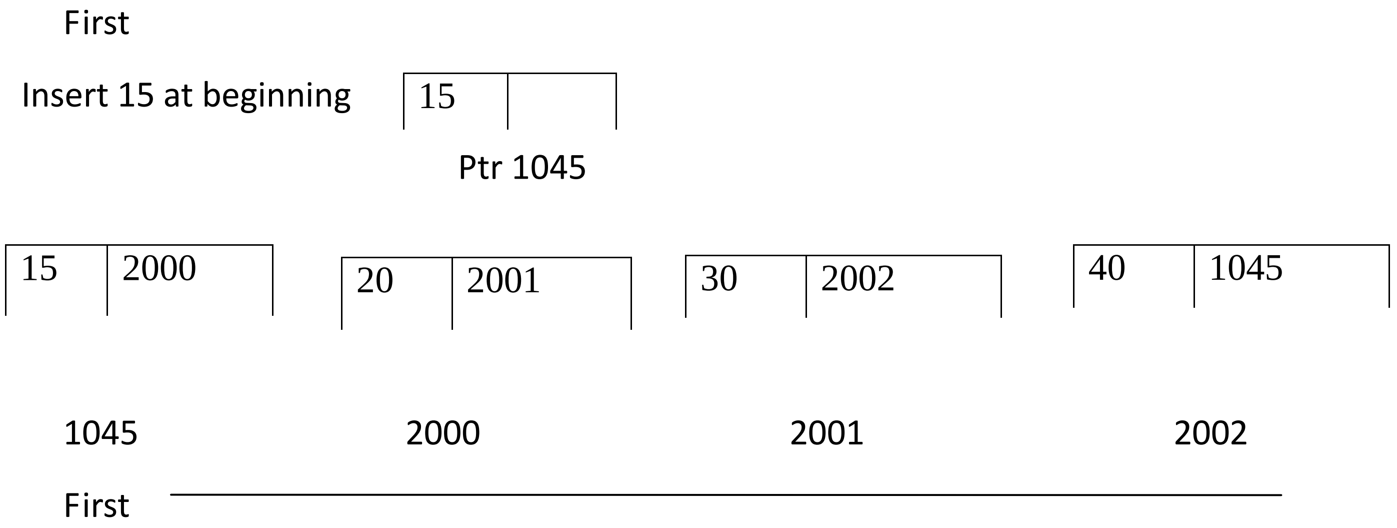
❖ **.Programme : Create and Traverse circular linked list.**

```
struct node
{
        int info;
        struct node * link;
}*temp,*ptr,*first;
void main()
{
        int ch,m;
        char ch1;
        clrscr();
      printf("\n1 create");
      printf("\n2 display");
      printf("\n3 exit");
      do
      {
                printf("\nenter the choice");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                  ptr=(struct node *)malloc(sizeof(struct node *));
                                 printf("\n enter the first node");
                scanf("%d",&ptr->info);
                first=ptr;
                printf("\ndo you want to continue");
                ch1=getch();
                while(ch1=='y')
                {
                    printf("\nenter node");
                    temp=(struct node *)malloc(sizeof(struct node *));
                    scanf("%d",&temp->info);
                    ptr->link=temp;
                    ptr=temp;
                     printf("\ndo you want to continue");
                    ch1=getch();
                }
                 ptr->link=first;
            break;
            case 2:
                ptr=first;
                do
                {
                    printf("\n%d",ptr->info);
                    ptr=ptr->link;
                } while(ptr!=first);

            break;
            case 3:
                exit(0);
            break;
        }
    }while(ch>=1 && ch<=3);
    getch();
}
```

3.Insertion into circular linked list.

- **At the beginning:-**

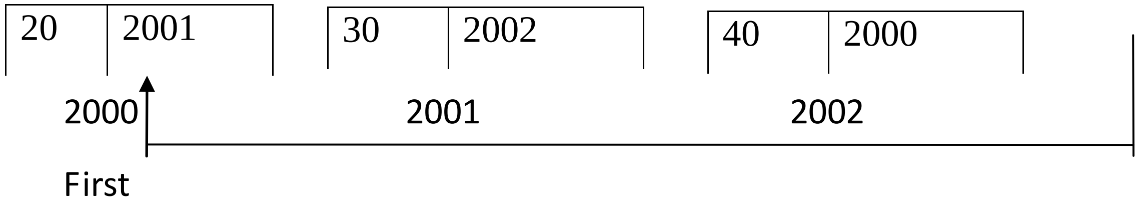




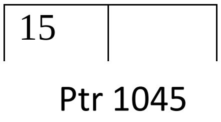
- **<u>Algorithm : Insertion at beginning into circular linked list.</u>**

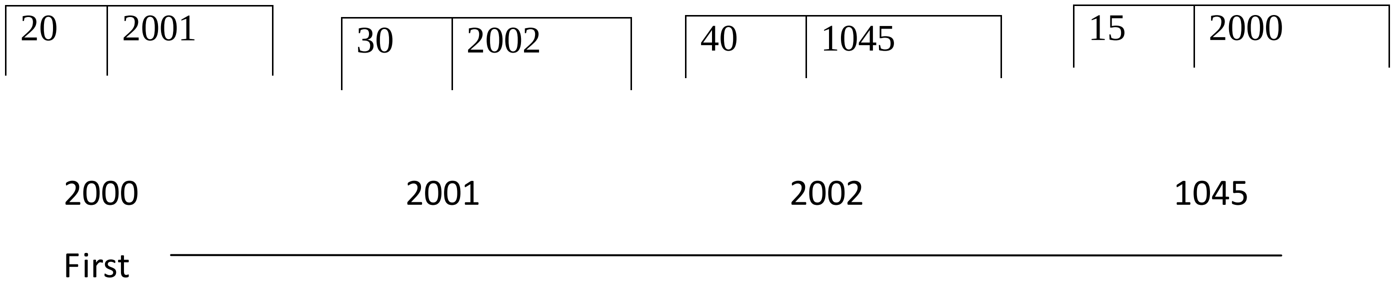
```
Step 1: [check for availability]
        If ptr==NULL then
        Write "overrflow"
        Exit
        End if

Step 2:  ptr→info = value

Step 3:  temp = first

Step 4: repeat step 5 while temp →link != first

Step5: temp= temp→link

Step 6: ptr→link = first

Step 7: first = ptr

Step 8: temp→link  = first

Step 9 :stop
```

At the end :-



Insert 15 at beginning

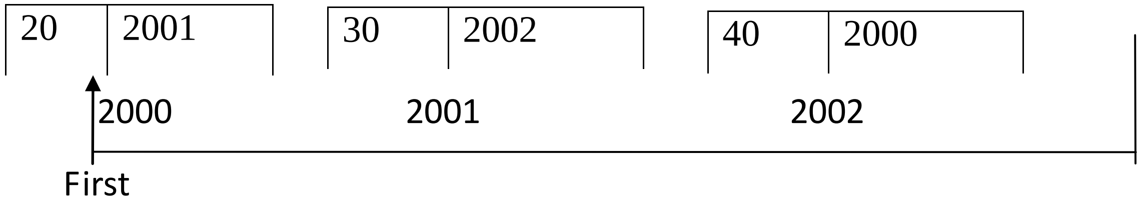




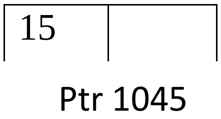
Algorithm : Insertion at end into circular linked list.

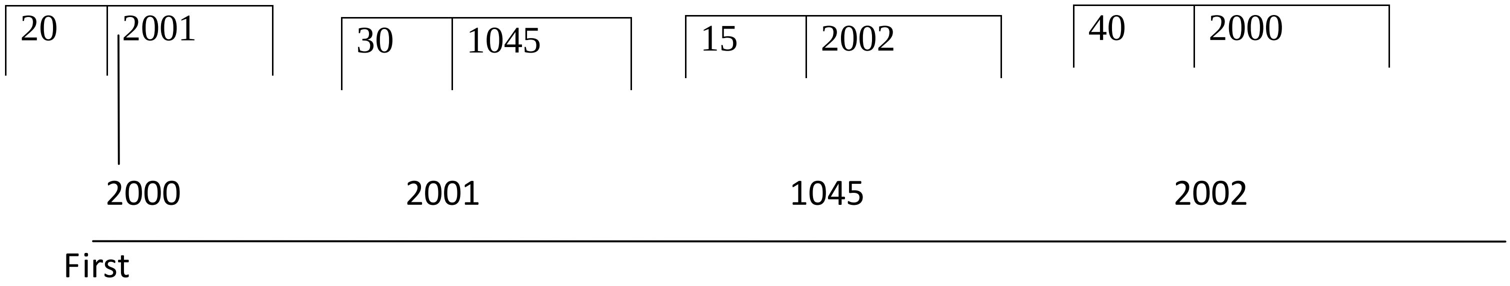
```
Step 1: [check for availability]
        If ptr==NULL then
        Write “overrflow”
        Exit
        End if

Step 2:  ptr→info = value

Step 3: temp = first

Step 4:repeat step 5 while (temp→link ! =first)

Step 5: temp= temp→link

Step 6: temp→link = ptr

Step 7 :ptr→link = first

Stop 8:stop
```

➢ **After particular data:-**



Insert 15 after node 30





➢ **Algorithm : Insertion at particular position into circular linked list.**

```
Step 1: [check for availability]
        If ptr==NULL then
        Write “overrflow”
        Exit
        End if

Step 2:  ptr→info = value

Step 3:read data

Step 4:  temp = first

Step 5: repeat step 6 while (temp→info ! =data)

Step 6: temp= temp→link

Step 7: ptr→link = temp→link
        temp→link=ptr

Step 8 :stop
```

➢ **Programme : Insertion (beginning,end,after particular node) into circular linked list.**

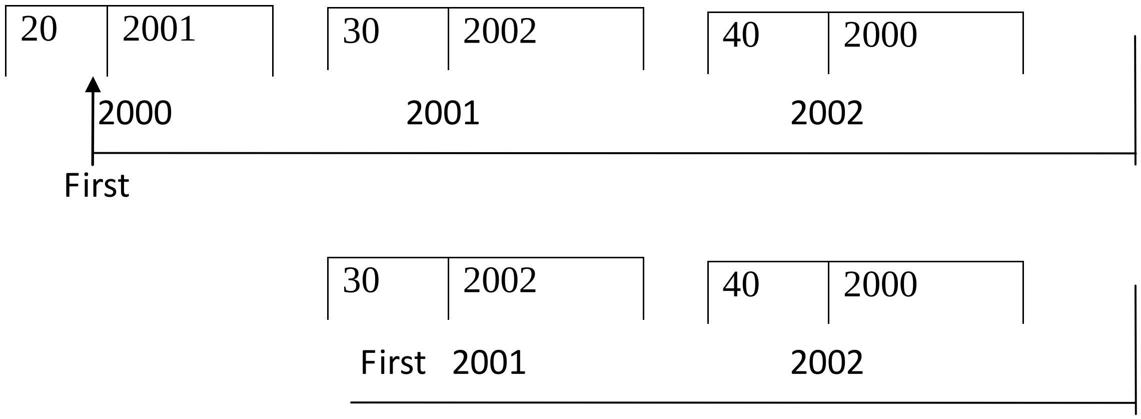
```
struct node
{
    int info;
    struct node * link;
}*temp,*ptr,*first;
void main()
{
    int ch,m;
    char ch1;
    clrscr();
   printf("\n1 insert at begining");
   printf("\n2 insert at end");
   printf("\n3 insert after specified position");
   printf("\n4 exit");
   do
   {
        printf("\nenter the choice");
        scanf("%d",&ch);
       switch(ch)
        {
             case 1:
                  ptr=(struct node *)malloc(sizeof(struct node *));
                  printf("\nEnter new node");
                  scanf("%d",&ptr->info);
                  if(ptr==NULL)
                  {
                       printf("\noverflow");
                       break;
                  }
                  temp=first;
                   while(temp->link!=first)
                  {
                       temp=temp->link;
                  }
                  ptr->link=first
                   first=ptr;
                   temp->link=first;
             break;
```

```
 case 2:
          ptr=(struct node *)malloc(sizeof(struct node *));
           printf("\nEnter new node");
          if(ptr==NULL)
           {
                    printf("\noverflow");
                    break;
           }
          scanf("%d",&ptr->info);
           temp=first;
          while(temp->link!=first)
          {
                    temp=temp->link;
          }
           temp->link=ptr;
          ptr->link=first;
 break;

case 3:
          ptr=(struct node *)malloc(sizeof(struct node *));
           if(ptr==NULL)
          {
                    printf("\noverflow");
                    break;
          }
          Printf("enter new node information");
          scanf("%d",&ptr->info);
          printf("enter data");
          scanf("%d",&data);
          temp = first
          while(temp→info!=data)
          temp = temp→link
          ptr →link  =temp → link
          temp→link=ptr
break;

case 4:
          exit(0);
break;
```

```
                }
        }while(ch>=1 && ch<=4);
          getch();
  }
```

4.Deletion from circular linked list.

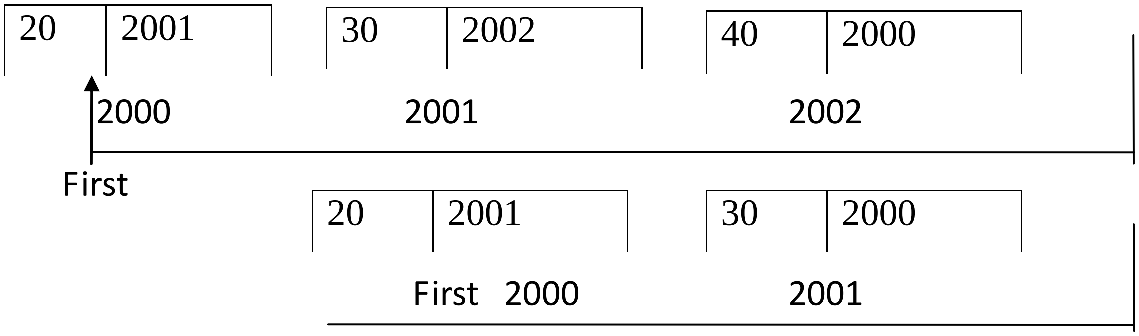
1. At the beginning:-



➢ Algorithm: Deletion at beginning from circular linked list.

```
Step 1: [check for underflow]
            If first==NULL then
            Write "list is empty"
            Exit
            End if

Step 2:  temp= first

Step 3: if( first →link = =first) then
            Print first→info
            First=NULL
            free(first);

Step 4 :else perform step 5 -7

Step 5: repeat step  6 while(temp→link != first)

Step 6: temp= temp→link
Step 7:ptr = first
            print first->info
            First= first→link
            temp →link = first
            Free(ptr)

stop 8: stop
```

2. **<u>At the end:-:-</u>**

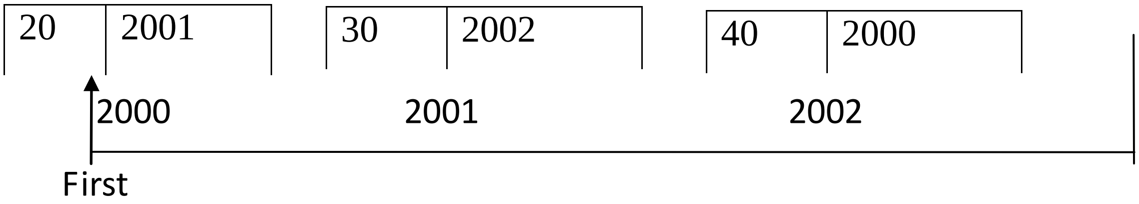


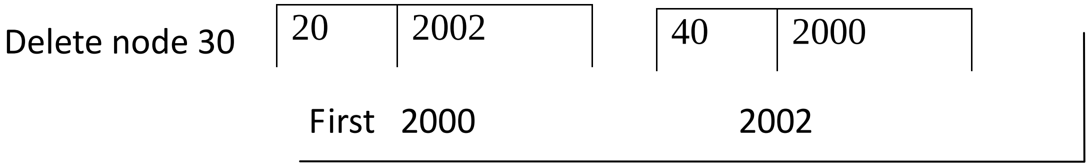
➢ **<u>Algorithm: Deletion at end from circular linked list.</u>**

```
Step 1: [check for underflow]
          If first==NULL then
          Write "list is empty"
          Exit
          End if

Step 2: if first→link = = first then
          Print first→info
          First=NULL
          free(first);

Step 3:else perform step 4 –7

Step 4: temp= first

Step 5: repeat step 6 while(temp→link !=first)

Step 6: prev = temp
           temp= temp→link

Step 7: prev→link=first
```

```
        Print temp→info
        Free(temp)

stop 8: stop
```

- **After given data:-**





➢ **Algorithm: Deletion at particular position from circular linked list.**

```
Step 1: [check for underflow]
        If first==NULL then
        Write "list is empty"
        Exit
        End if

Step 2: read data

Step 3: temp = first

Step 4: if first →link == first then
         Print   first→info
         First=NULL
         free(first);

Step 5 : else if  first->info==data then do step  6

Step 6: While (temp->link !=first) do temp=temp->link
        print  temp→info"
         first=first->link
        temp->link=first

Step 7: else do step 8 to 11
Step 8:  while (temp→info !=data) do step 9

Step 9: prev = temp
        temp = temp→link

Step 10: print "deleted element is temp→info"

Step 11: prev→link = temp→link

Step 12: free(temp)

stop 13: stop
```

- **Programme : Deletion (beginning,end,particular data) from circular linked list.**

```
struct node
{
    int info;
    struct node * link;
    }*temp,*ptr,*first;
    void main()
    {
        int ch,m;
        char ch1;
        clrscr();
          printf("\n1 delete at begining");
        printf("\n2 delete at end");
        printf("\n3 delete particular data”);
        printf("\n4 exit");
        do
        {
                printf("\nenter the choice");
                scanf("%d",&ch);
                switch(ch)
               {
                        case 1:
                                if(first==NULL)
                                {
                                        printf("underflow");
                                        break;
                }
                 if(first->link==first)
                 {
                          printf("\ndeleted no is%d",first->info);
                         first=NULL;
                         free(first);
                 }
                 else
                {
                          temp=first;
                         while(temp->link!=first)
                         temp=temp->link;
                         printf("\ndeleted no is%d",first->info);
                         ptr=first;
                         first=first->link;
                         temp->link=first;
                         free(ptr);
        break;

        case 2:
                if(first==NULL)
                 {
                          printf("underflow");
                         break;
                 }
                temp=first;
                if(first->link==first)
                {
                         printf("\ndeleted no is%d",ptr->info);
                         first=NULL;
                         free(first);
                }
                 else
                {
                         while(temp->link!=first)
                         {
                                  prev=temp;
                                 temp=temp->link;
                          }
                prev->link=temp;
        printf("\ndeleted no is%d",temp->info);
        }
         free(temp);
break;

 case 3:
         if(first==NULL)
        {
                 printf("underflow");
                break;
         }
                Printf(“enter data”);
                Scanf(“%d”,&data);
                if(first->link==first)
                {
                        printf("\ndeleted no is%d",first->info);
                first=NULL
                        free(first);

                }
                else if(first->info==data)
                {
                        ptr=firsr;
                        temp=first;
                        While(temp->link!=first)
                        temp=temp->link;
                        printf("\ndeleted no is%d",first->info);
                        first=first->link;
                        temp->link=first;
                        free(ptr);
                }
                else
                {
                        temp=first;
                        While(temp →info ! =data)
                      {
                        prev=temp
                        temp=temp→link;
                     }
                       Printf(“deleted node is %d”,temp→info);
```

```
                                                prev→link=temp → link;
                                            Free(temp);
                                            }
                          break;

                          case 4:
                                 exit(0);
                          break;
                   }
         }while(ch>=1 && ch<=4);
         getch();
    }
```

8. Explain operation on Doubly linked list.

Ans :

1. Create doubly linked list.
2. Traverse doubly linked list.
    - I. Forward direction
    - II. Backward direction
3. Insertion into doubly linked list.
    - I. At the beginning
    - II. At the end
    - III. At particular position
    - IV. In sorted list
4. Deletion from doubly linked list.
    - I. From the beginning
    - II. From the end
    - III. From specific node

<u>1. Algorithm : Create doubly linked list.</u>

```
Step 1: [check for overflow]
        ptr=(structnode*) malloc (sizeof(structnode))
        If ptr==NULL then
        Write “overflow”
        Exit
        End if

Step 2:  ptr →info=value
         Ptr→lpt=NULL
```

```
            First =ptr

Step 3: ch==’y’

Step 4: repeat step 5 to 6 while ch==’y’

Step 5: temp=(struct node*) malloc (sizeof(structnode))
             temp→info = value
            ptr→ rpt = temp
            temp→lpt=ptr
            ptr = temp

Stpe 6: repeat choice (y/n)

Stpe 7: ptr→rpt =NULL

Step 8 :stop
```

2.**Algorithm : Traverse doubly linked list.**

- Forward direction:-

➢ Algorithm: forward direction into doubly linked list.

```
Step 1: [check for underflow]
            If first==NULL then
            Write “list is empty”
            Exit
            End if

Step 2: ptr = first

Step 3 : repeat step 4  while( ptr != NULL)

Step 4: print ptr →info
             ptr=ptr→rpt

stop 5: stop
```

- backward direction:-

- Algorithm: backward direction into doubly linked list.

```
Step 1: [check for underflow]
          If first==NULL then
          Write “list is empty”
          Exit
          End if

Step 2: ptr = first

Step 3 : repeat step 4  while( ptr→rpt != NULL)

Step 4: ptr= ptr → rpt

Step 5: repeat step 6 while(ptr!=NULL)

Step 6: print ptr →info
           Ptr=ptr→lpt

stop 7: stop
```

- **Programme : create,display doubly linked list.**

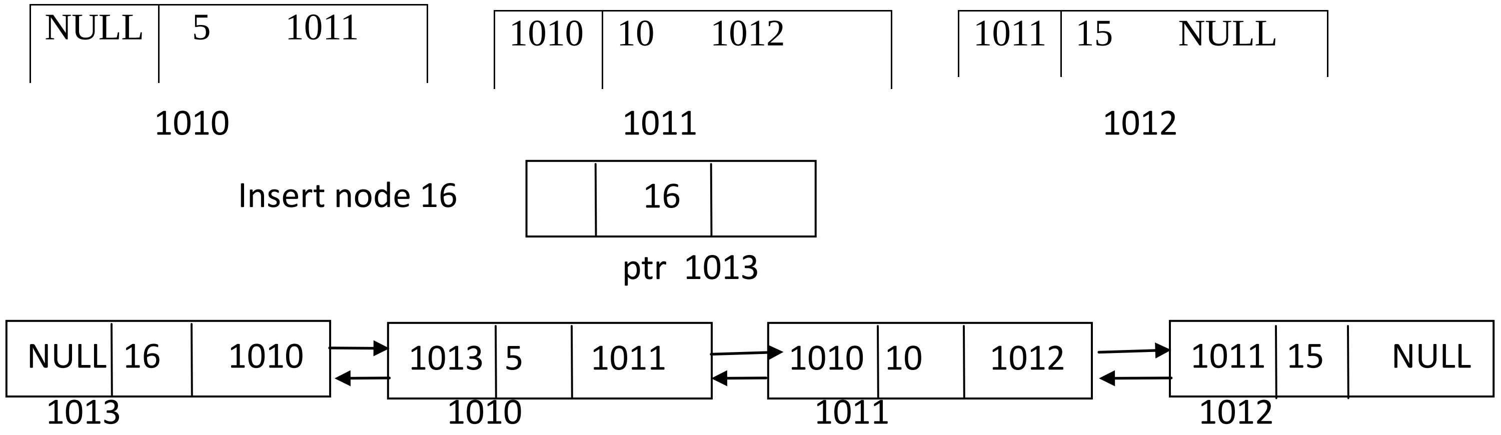
```
struct node
{
    int info;
    struct node *lpt,*rpt;
} *ptr,*first,*temp,*prev;

void main()
{
     int ch1,data;
     char ch;
     clrscr();
      printf("\n 1:create");
       printf("\n 2:display forward");
       printf("\n 3:display backward");
      printf("\n 4:exit");
      do
```

```
{
    printf("\nEnter your choice=");
    scanf("%d",&ch1);
    switch(ch1)
     {
            case 1:
             ptr=(struct node*)malloc(sizeof(struct node));
             printf("\nEnter node");
            scanf("%d",&ptr->info);
             if(ptr==NULL)
            {
                     printf("\noverflow");
                    break;
             }
            first=ptr;
            ptr->lpt=NULL;
            printf("\nDo you want to continew");
            ch=getch();
            while(ch=='y')
             {
                     temp=(struct node*)malloc(sizeof(struct node));
                    printf("\nEnter the node");
                     scanf("%d",&temp->info);
                    ptr->rpt=temp;
                     temp->lpt=ptr;
                    ptr=temp;
                    printf("\nDo you want to continew");
                    ch=getch();
            }
            ptr->rpt=NULL;
            break;

            case 2:
             ptr=first;
            while(ptr!=NULL)
             {
                      printf("\n%d",ptr->info);
                     ptr=ptr->rpt;
          }
          break;
```

```
                case 3:
                 ptr=first;
                while(ptr->rpt!=NULL)
                {
                        ptr=ptr->rpt;
                }
                while(ptr!=NULL)
                {
                         printf("\n%d",ptr->info);
                        ptr=ptr->lpt;
                }
              break;

                case 4:
                 exit(0);
          }
      }while(ch1>=1 && ch1<=4);
      getch();
}
```

3 Insertion into doubly linked list.

- **At the beginning:-**

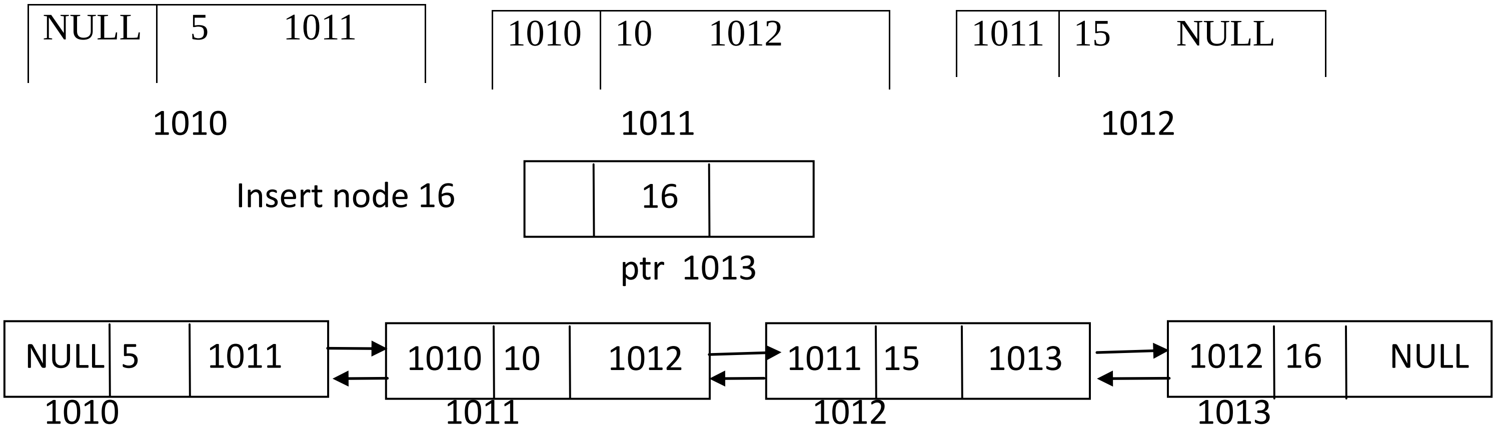


- **Algorithm : Insertion at beginning into doubly linked list.**

```
Step 1: [check for availability]
          ptr=(structnode)*malloc (sizeof(structnode))
```

```
        If ptr==NULL then
        Write “overrflow”
        Exit
        End if

Step 2:  ptr→info = value

Step 3:ptr→rpt=first

Step 4:  first→lpt = ptr
          ptr→lpt=NULL

Step 5: first = ptr

Step 6 :stop
```

➢ **At the end:-**

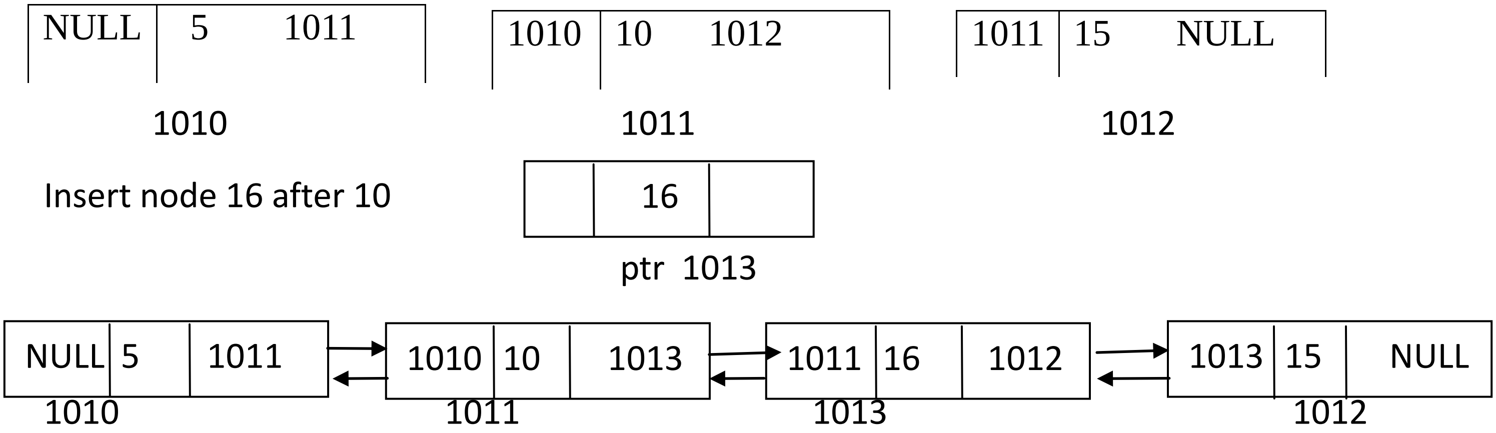


➢ **Algorithm : Insertion at end into doubly linked list.**

```
Step 1: [check for availability]
        Ptr=(structnode)*malloc (sizeof(structnode))
        If ptr==NULL then
        Write “overrflow”
        Exit
        End if
```

```
Step 2:  ptr→info = value

Step 3: temp = first

Step 4:  repeat step 5 while(temp→rpt !=NULL)

Ste p5: temp =temp→rpt

Step 6: temp→rpt = ptr

Step7: ptr→lpt=temp

Step 8:ptr→rpt=NULL

Step 9 :stop
```

- **<u>After the specific node:-</u>**



- **<u>Algorithm : Insertion after particular node into doubly linked list.</u>**

```
Step 1: [check for availability]
        Ptr=(structnode)*malloc (sizeof(structnode))
        If ptr==NULL then
        Write "overrflow"
        Exit
        End if
```

```
Step 2:  ptr→info = value

Step 3:read node info after which insertion
          Read data

Step 4:  temp =first

Step 5: repeat step 6 while(temp →info !=data)

 Step 6: temp =temp→rpt

Step 7 : if (temp->rpt = =NULL)  then perform Step 8 else perform step 9

 Step 8:       temp →rpt=ptr
                 ptr→lpt=temp
                 ptr->rpt=NULL

Step 9:  next=temp→rpt

           temp →rpt=ptr
           ptr→lpt=temp
           ptr→rpt=next
           next→lpt=ptr

Step 10: stop
```

- **Programme : Insertion (beginning,end,after particular node) into doubly linked list.**

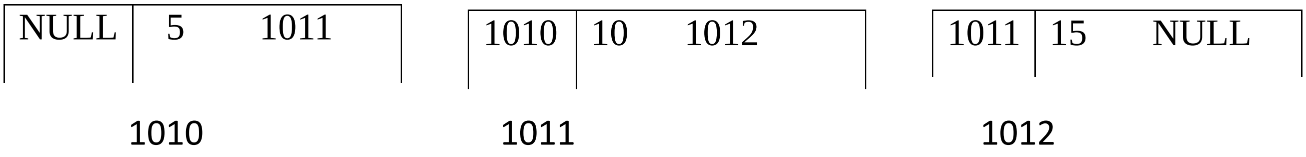
```
struct node
{
    int info;
    struct node *lpt,*rpt;
} *ptr,*first,*temp,*next;

void main()
{
   int ch1,data;
   char ch;
   clrscr();
```

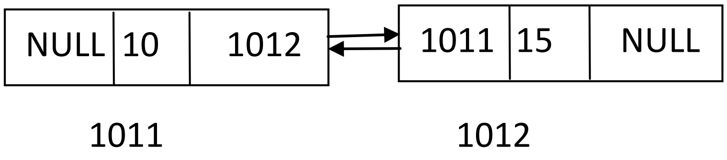
```
printf("\n 1:at begning");
printf("\n 2:at end");
printf("\n 3:after specific node");
printf("\n 5:exit");
do
{
       printf("\nEnter your choice=");
       scanf("%d",&ch1);
       switch(ch1)
      {
                case 1:
                          ptr=(struct node*)malloc(sizeof(struct node));
                           if(ptr==NULL)
                          {
                                    printf("\noverflow");
                                    break;
                          }
                          printf("\nEnter new node");
                          scanf("%d",&ptr->info);
                           ptr->lpt=NULL;
                           ptr->rpt=first;
                          first→lpt=ptr;
                            first=ptr;
             break;

           case 2:
                          ptr=(struct node*)malloc(sizeof(struct node));
                          if(ptr==NULL)
                          {
                                      printf("\noverflow");
                                     break;
                          }
                          printf("\nEnter new node");
                          scanf("%d",&ptr->info);
                          temp=first;
                          while(temp->rpt!=NULL)
                          temp=temp->rpt;
                            temp->rpt=ptr;
                          ptr→lpt=temp;
                          ptr->rpt=NULL;
```

```
            break;
        case 3:
                ptr=(struct node*)malloc(sizeof(struct node));
                 if(ptr==NULL)
                {
                        printf("\noverflow");
                        break;
                }
                printf("\nEnter new node");
               scanf("%d",&ptr->info);
               printf("\nEnter data");
               scanf("%d",&data);
               temp=first;
               while(temp->info!=data)
               {
               temp=temp->rpt;
               }
               if(temp->rpt==NULL)
               {
               temp→rpt=ptr
               ptr → lpt =temp;
               ptr->rpt=NULL
               }
               else
               {
               next=temp→rpt
               temp→rpt=ptr
               ptr → lpt =temp;
               ptr→rpt=next
               next→lpt=ptr;
               }
            break;

         case 4:
                exit(0);
        }
    }while(ch1>=1 && ch1<=4);
    getch();
}
```

4. Deletion from doubly linked list.

- At the beginning:-

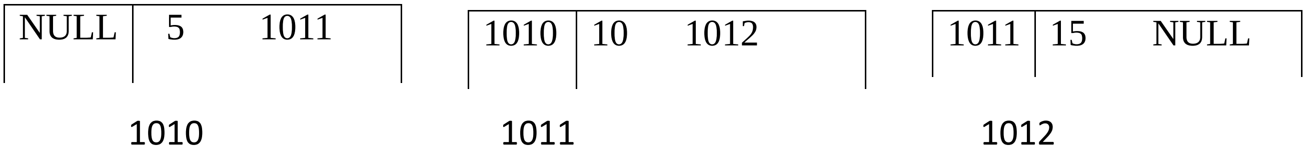


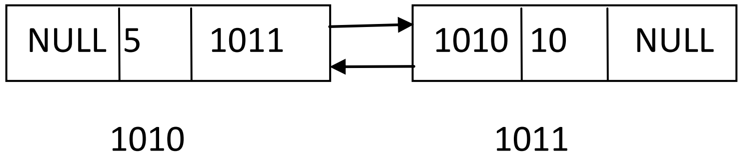


Algorithm : Deletion at beginning into doubly linked list.

```
Step 1: [check for underflow]
           If first==NULL then
           Write “list is empty”
           Exit
           End if

Step 2:  temp =first

Step 3:  print first ->info
            first = first →rpt
           if(first!=NULL)

 first→lpt=NULL

             free(temp);

Step 4 :stop
```

- At the end:-

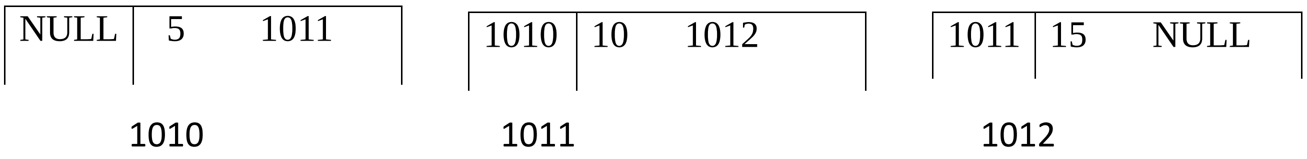




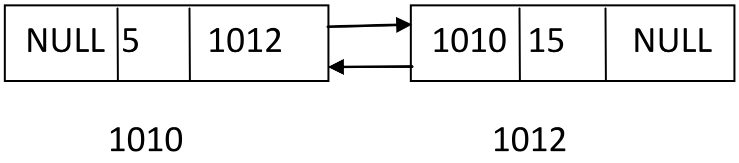
- **Algorithm : Deletion at end from doubly linked list.**

```
Step 1: [check for underflow]
        If first==NULL then
        Write “list is empty”
        Exit
        End if

Step 2:  temp =first

Step 3:  if (first→rpt == NULL) then
                print first ->info
                First =NULL
                free(first);

Step 4: else repeat  step 5 to 7

Step 5: repeat step 6  while(temp →rpt !=NULL)

Step 6:  temp =temp→rpt

Step 7:  prev =temp→lpt

Step8: prev→rpt=NULL

Step 9: free(temp)

Step 10 :stop
```

- **After the particular node:-**



Delete node 10



- **<u>Algorithm : Deletion particular node from doubly linked list.</u>**

```
Step 1: [check for underflow]
          If first==NULL then
          Write "list is empty"
          Exit
          End if

Step 2:  read data

Step 3: temp =first

Step 4:  if (first→info == data) then
                   print first ->info
                   First =first->rpt
                   if(first!=NULL)
                   {
                   first->lpt=NULL
                   }
                   free(temp);

Step 5:else repeat step  6  to 7

Step 6: repeat step 7 while( temp→info ! =data)

Step 7: temp =temp→rpt

Step 8: prev=temp →lpt
           next=temp →rpt

          prev→rpt=next
          next→lpt=prev

Step 9: free(temp)
```

Step 10 :stop

- **<u>Programme : Deletion (beginning,end,particular node) from doubly linked list.</u>**

```
struct node
{
    int info;
    struct node *lpt,*rpt;
} *ptr,*first,*temp,*next,*prev;
void main()
{
         int ch1,data;
         char ch;
         clrscr();
        printf("\n 1:delete at begning");
         printf("\n 2:delete at end");
        printf("\n 3:delete spcific node");
        printf("\n 4:exit");
         do
        {
                printf("\nEnter your choice=");
                scanf("%d",&ch1);
                switch(ch1)
                {
                        case 1:
                                if(first==NULL)
                                {
                                        printf("\n underflow");
                                        break;
                                }
                                temp=first;
                                printf("deleted no.is %d",first->info);
                                first=first->rpt;
                                if(first!=NULL)
                                {
                                first->lpt=NULL;
                                }
                                free(temp);
                        break;
```

```
case 2:
        if(first==NULL)
        {
                printf("\n underflow");
                break;
        }
        if(first->rpt==NULL)
        {
                temp=first;
                printf("deleted no.is %d",temp->info);
                first=NULL;
                free(first);
        }
        else
        {
                temp=first;
                while(temp->rpt!=NULL)
                {
                        temp=temp->rpt;
                }
        prev=temp->lpt;
        prev->rpt=NULL;
        printf("deleted no.is %d",temp->info);
        free(temp);
        }
        break;

case 3:
        if(first==NULL)
        {
                printf("\n underflow");
                break;
        }
        Printf(“enter data”);
        Scanf(“%d”,&data);
        if(first->info==data)
        {
                printf(“%d”, first ->info)
                first =first->rpt
```

```
                    if(first!=NULL)
                    {
                    first->lpt=NULL
                    }
                    free(temp);
                }
                else
                {
                    temp=first;
                    while(temp->info!=data)
                    {
                        temp =temp→rpt
                    }
                    prev=temp →lpt
                    next=temp →rpt

                    prev→rpt=next
                    next→lpt=prev
                }

                printf("deleted no.is %d",temp->info);
                free(temp);
            break;

            case 4:
            exit(0);
        }
    }while(ch1>=1 && ch1<=4);
     getch();
}
```

9. Application of linked list.

Ans :

- Polynomial representation ,automatic polynomial manipulations are performed by linked list
- Addition subtraction operations of polynomial are easily implemented using linked list
- Symbol table creation
- Multiple precision arithmetic and representation of sparse matrices

10. Implement stack using linked list.

Ans :

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
          int info;
          struct node * link;
          }*ptr,*top;
          void main()
          {
          int ch1,m,data,ch;
             printf("\n1.push");
            printf("\n2.traverse ");
            printf("\n3.pop ");
            printf("\n4. exit");
       do
       {
       printf("\nenter the choice");
       scanf("%d",&ch1);
       switch(ch1)
       {
          case 1:
           ptr=(struct node *)malloc(sizeof(struct node *));
           printf("\n enter the first node");
           scanf("%d",&ptr->info);
                    ptr->link=top;
                    top=ptr;
            break;
           case 2:
           if(top==NULL)
           {
                    printf("\n underflow");
                    break;
           }
           ptr=top;
           while(ptr!=NULL)
           {
           printf("\n\n%d",ptr->info);
```

```
                ptr=ptr->link;
                }
        break;
                case 3:
                if(top==NULL)
                {
                        printf("\n underflow");
                        break;
                }
                ptr=top;
                printf("deleted no.is %d",top->info);
                top=top->link;
                free(ptr);
                break;
               case 4:
               exit(0);
               break;
            }
        }while(ch1>=1 && ch1<=4);
    }
```

11. Implement Queue using linked list.

Ans :

```
        #include<stdio.h>
        #include<stdlib.h>
        struct node
        {
                int info;
                struct node * link;
               }*f,*ptr,*r;
                void main()
                {
                        int ch1,m,data;
                        char ch;
                printf("\n1.insert");
                printf("\n2.traverse ");
                printf("\n3.delete");
                printf("\n4. exit");
             do
             {
```

```
printf("\nenter the choice");
scanf("%d",&ch1);
 switch(ch1)
{
        case 1:
        {
                 ptr=(struct node *)malloc(sizeof(struct node *));
                 printf("\n enter the first node");
                 scanf("%d",&ptr->info);
                 if(f==NULL || r==NULL)
                 {
                 f=ptr;
                 r=ptr;
                 }
                else
                {
                r->link=ptr;
                r=ptr;
                }
              r->link=NULL;
 break;
 case 2:
 if(f==NULL)
 {
        printf("\n underflow");
        break;
 }
 ptr=f;
 while(ptr!=NULL)
 {
 printf("\n\n%d",ptr->info);
 ptr=ptr->link;
 }
 break;
 case 3:
 if(f==NULL)
 {
        printf("\n underflow");
        break;
 }
```

```
                 ptr=f;
                printf("deleted no.is %d",f->info);
                f=f->link;
                free(ptr);
                }
                break;
               case 4:
               exit(0);
               break;
          }
       }while(ch1>=1 && ch1<=4);
  }
```

Gtu Question

1. **1.** What is Stack? List out different operation of it and write algorithm for any two operation. -7
2. Write a C program to implement a stack with all necessary overflow and underflow checks using array -7
3. What is postfix notation? What are its advantages? Convert the following infix expression to postfix A$B-C*D+E$F/G -4
4. Write a 'C' program or an algorithm to convert infix expression without parenthesis to postfix expression -7
5. Convert (A + B) * C – D ^ E ^ (F * G) infix expression into prefix format showing stack status after every step in tabular form -7

6. Convert A+(B*C-(D/E^F)*G) infix expression into postfix format showing stack status after every step in tabular form -7.
7. Evaluate the following postfix expression using stack -3
   (a) 9 3 4 * 8 + 4 / - (b) 5 6 2 + * 1 2 4 / - +
8. Evaluate the following postfix expression using a stack. Show the stack contents.
   (a) AB*CD$-EF/G/+
   (b) A=5, B=2, C=3, D=2, E=8, F=2, G=2 - 3
9. Describe: (1) Recursion (2) Tower of Hanoi.
10. List the applications of Stack . -1
11. Write an algorithm to reverse string using stack -4,7
12. Write a program to implement stack using linked list -7
13. Write an algorithm to check if an expression has balanced parenthesis using stack. -4
14. Write an algorithm for simple queue with ENQUEUE opration -3.
15. Write an algorithm to implement insert and delete operations in a simple queue -7
16. Mention variations of the queue data structure -1
17. What is a Queue? Write down drawback of simple queue. Also write an algorithm for deleting an element from circular queue -7
18. Is Queue a priority queue? Justify
19. Explain the concept of circular queue. Compare circular queue with simple queue -4
20. Write a program to implement circular queue using array -7.
21. Write user defined C function for inserting an element into circular queue -7
22. Write a C program to implement a circular queue using array with all necessary overflow and underflow checks -7
23. Perform following operations in a circular queue of length 4 and give the Front, Rear and Size of the queue after each operation. -4
    1) Insert A, B
    2) Insert C
    3) Delete
    4) Insert D
    5)Insert E
    6) Insert F
    7) Delete
24. Difference between Circular queue and Simple queue.
25. Explain double ended queue -3
26. Define priority queue -1
27. Explain various applications of queue -3
28. Mention one operation for which use of doubly linked list is preferred over the singly linked list.-1
29. Write an algorithm/steps to traverse a singly linked list-1
30. What is a header node and what is its use?-1

31. Write a program to insert and delete an element after a given node in a singly linked list.
32. Create a doubly circularly linked list and write a function to traverse it
33. Write ‘C’ functions to: (1) insert a node at the end (2) delete a node from the beginning of a doubly linked list.
34. Write ‘C’ structure of Singly linked list.-1
35. Write a ‘C’ functions to: (1) insert a node at beginning in singly linked list
36. Write an algorithm for insert operation at end of Linked List.-7
37. Write down advantages of linked list over array and explain it in detail.
38. Write an algorithm to delete a node from doubly linked list.
39. Explain delete operation of doubly linked list.
40. What are the advantages of doubly linked list? Write a C function to find maximum element from doubly linked list.
41. Briefly explain advantages of doubly link list over singly link list. Write function delete (p, &x) which delete the node pointed by p in doubly link list.
42. Write a program to implement stack using linked list.