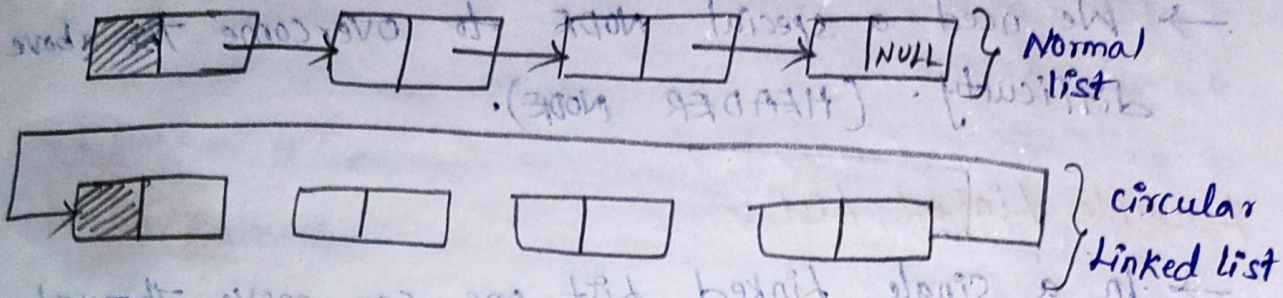
Circular Linked list :-



* In a single linked list the link field of the last NODE is NULL.
* If we utilize the NULL link field to store the pointer to the header node, we can gain an advantage.

Def :- A linked list where the last NODE is pointed to the header node is called as circular linked list.

* The main advantage of circular linked list are
  - → every node is accessible from a given NODE.
  - → While deleting a element in single linked list we have to capture the address of the previous NODE, And a search is made to delete the element. But in circular linked list it is not necessary, to search for previous NODE while deletion.
  - → Accessibility in circular linked list is more faster than single linked list.

→ Certain operations on circular linked linked list is comparitively more effective than single list.

Draw Back :-

The circular Linked list requires some extra care to detect end of the list when only one element is available.

→ We need a special NODE to overcome the above difficulty. (HEADER NODE).

Various operations that can be performed on circular linked list are

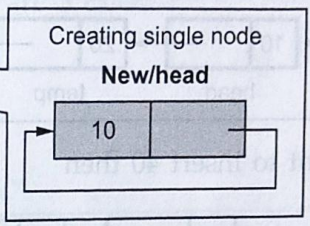
1. Creation of circular linked list.
2. Insertion of a node in circular linked list.
3. Deletion of any node from linked list.
4. Display of circular linked list.

We will see each operation along with some example.

Creation of circular linked list

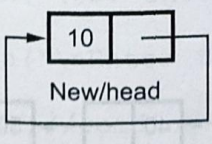
```
ublic node  create(int val,node New)

 node temp;
 temp=head;
 if ( New == null )
     System.out.println("\nMemory is not allocated");
    New.data   = val ;
 if (head==null)  /* Executed only for the first
                              time */
 {
     head = New;
     New.next=head;
 }
 else
  {
      while(temp.next!=head)
         temp=temp.next;
        /* temp keeps track of the most recently
         created node */
        temp.next = New;
        New.next=head;
  }
  return head;
```



Initially we will allocate memory for **New** node using a new operator.

Initially



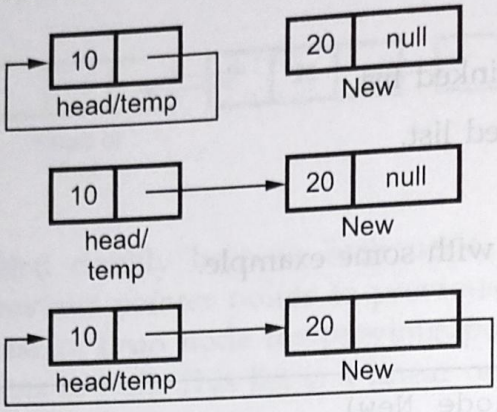
Suppose we have taken element '10' then

```
head = New;
New.next = head;
```

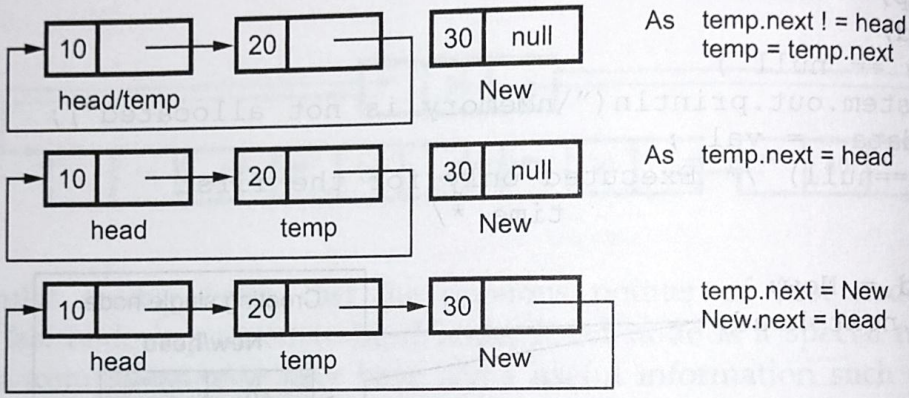
Here variable **head** indicates starting node.

Now as head!=null, we can further create the nodes and attach them as follows.

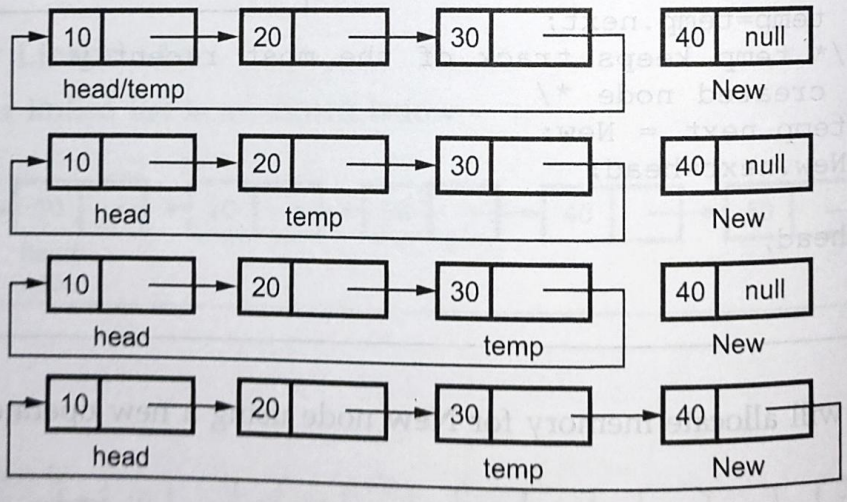
When we have taken element '20'



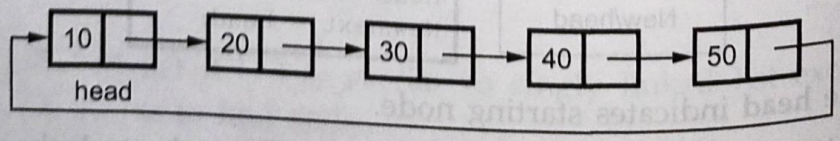
If we want to insert 30 then



If we want to insert 40 then



Thus we can create a circular linked list by inserting one more element 50. It is as shown below -



2. Display of Circular linked list

```
public void  display(node head)
{
  node temp ;
  temp = head;
  if ( temp == null )
  {
     System.out.println("\nThe list is empty\n");
  }
  else
  {
  do
  {
     System.out.print("      "+temp.data );
      temp = temp.next;
  }while ( temp != head);
 }
}
```

next pointer of last node is **head** node

3. Insertion of circular linked list

While inserting a node in the linked list, there are 3 cases –

a) inserting a node as a head node

b) inserting a node as a last node

c) inserting a node at intermediate position

The functions for these cases is as given below –

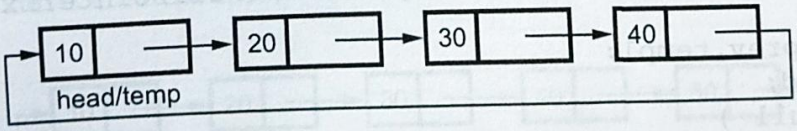
```
node insert(node head)throws IOException
{
 String str;
 int val;
 InputStreamReader input = new InputStreamReader(System.in);
 BufferedReader b = new BufferedReader(input);
 System.out.println("Enter The element which you want to insert");

 str = b.readLine();//reading the string from console
 val=Integer.parseInt(str);
 System.out.println("\nEnter the your choice for insertion of
node");
 System.out.println("\t1. Insert a node as a head node");
 System.out.println("\t2. Insert a node as a last node");
 System.out.println("\t3. Insert a node at intermediate position in
the linked list");
 System.out.println("\nEnter the your choice for insertion of
node");
 str = b.readLine();//reading the string from console
 char choice=str.charAt(0);//reading first char of console string
```

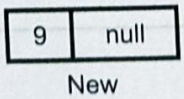
```
switch(choice)
 {
    case '1':head=insert_head(head,val);
         break;
    case '2':insert_last(head,val);
         break;
    case '3':insert_after(head,val);
         break;
}
return head;
}
//Insertion of node at first position
node insert_head(node head,int val)throws IOException
{
 node temp;
 node New=new node();
  New.data=val;
 if(head==null)
      head=New;
 else
 {
  temp=head;
  while(temp.next!=head)
      temp=temp.next;
  temp.next=New;
  New.next=head;
  head=New;
 System.out.println("\t The node is inserted!!!");
  }
 return head;
}
//Insertion of node at last position
public void insert_last(node head,int val)throws IOException
{
 node temp;
 node New=new node();
  New.data=val;
 if(head==null)
     head=New;
  else
  {
       temp=head;
       while(temp.next!=head)
                 temp=temp.next;
        temp.next=New;//attaching node at the last position
        New.next=head;
        System.out.println("\t The element inserted!!!");
   }
}
```

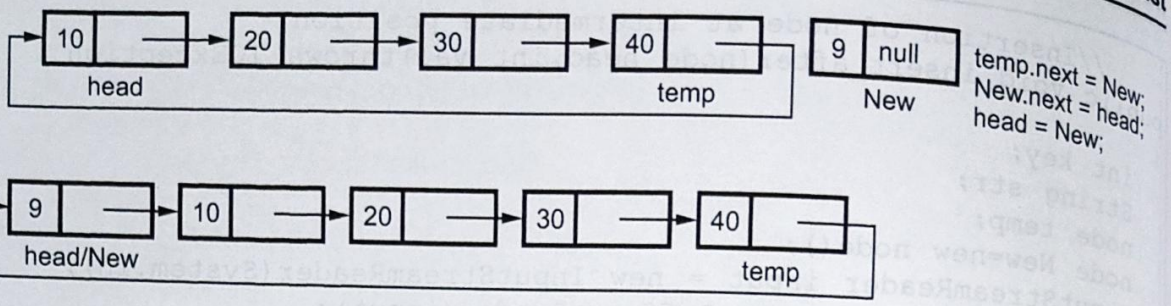
```
  //Insertion of node at intermediate position
public void insert_after(node head,int val)throws IOException
{
 int key;
 String str;
 node temp;
 node New=new node();
 InputStreamReader input = new InputStreamReader(System.in);
 BufferedReader b = new BufferedReader(input);
  New.data=val;
 if(head==null)
 {
  head=New;
 }
 else
 {
   System.out.println("\n Enter The element after which you want
to insert the node");
    str = b.readLine();
    key=Integer.parseInt(str);
    temp=head;
    do
    {
        if(temp.data==key)
         {
           New.next=temp.next;
           temp.next=New;
           System.out.println("\t The element inserted!!!");
           return;
         }
       else
             temp=temp.next;
   }while(temp!=head);
 }
}
```

Suppose linked list is already created as -

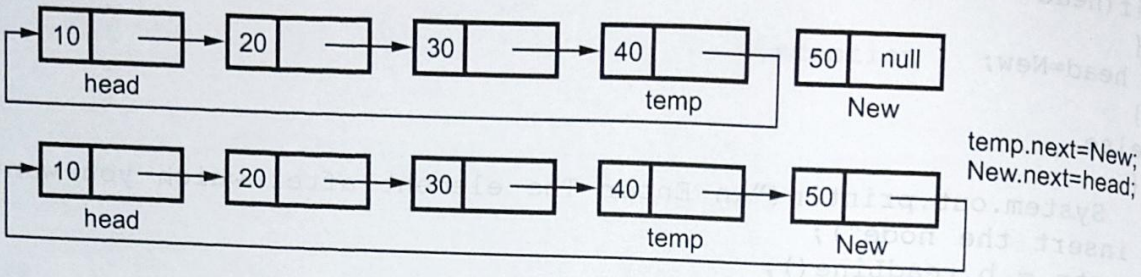


If we want to insert a **New** node as a head node then, first set **temp** to last node and then

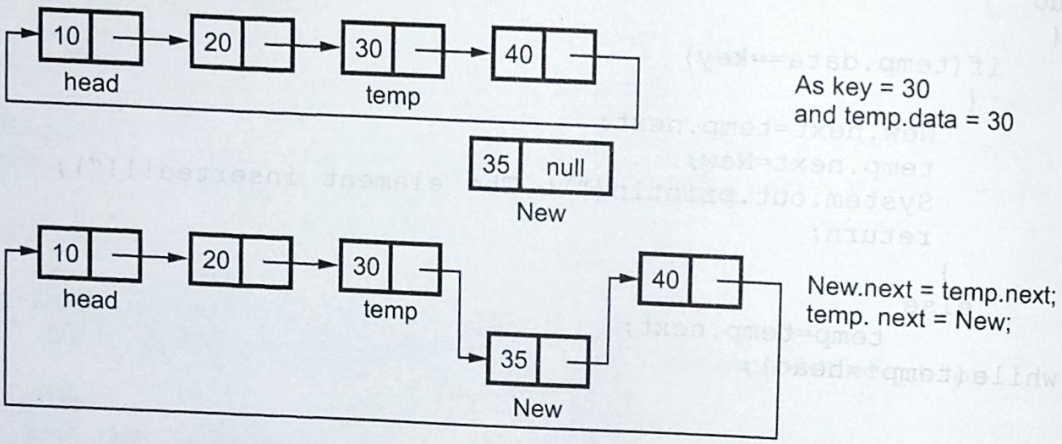




If we want to insert a New node as a last node consider a linked list



If we want to insert an element 35 after node 30 then

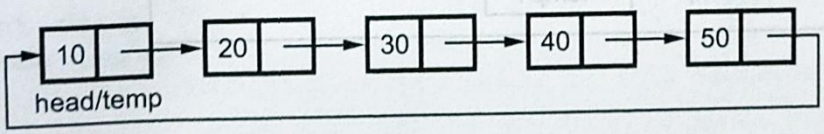


4. Deletion of any node

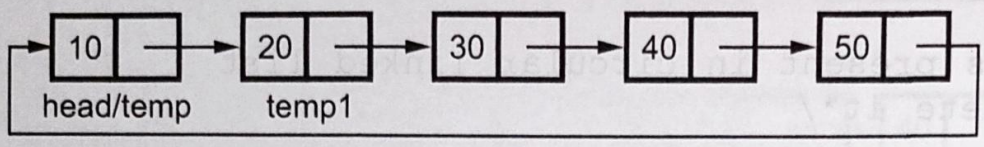
```
public node dele(node head,int key)throws NullPointerException
{
   node temp,prev,temp1;
   temp = head;
   if(temp==null )
   {
      System.out.println("\nThe list is empty.Can not delete");
      return null;
   }
```

```
if(temp.data==key)
{
temp1=temp.next;
if(temp1==temp)
/*if single node is present in circular linked list
and we want to delete it*/
 {
  temp=null;
  head=temp;
  System.out.println("\n The node is deleted");
  }
  else   /*otherwise*/
  {
       while(temp.next!=head)
            temp=temp.next;/*searching for the last node*/
       temp.next=temp1;
       head=temp1;/*new head*/
       System.out.println("\n The node is deleted");
    }
 }
 else
 {
       while(temp.next!=head) /* if intermediate node is to
                                 be deleted*/
 {
  if((temp.next).data==key)
  {
      temp1=temp.next;
      temp.next=temp1.next;
      temp1.next=null;
      temp1=null;
      System.out.println("\n The node is deleted");
  }
  else
      temp=temp.next;
 }  }
```

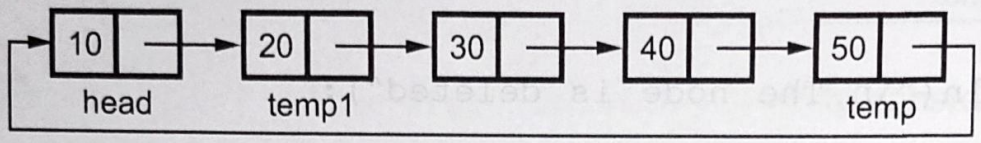
Suppose we have created a linked list as



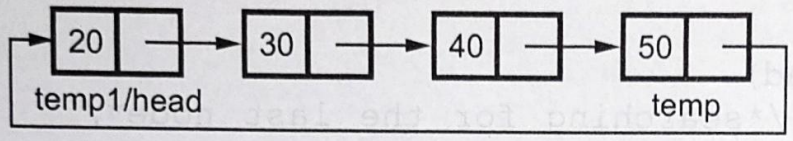
If we want to delete temp → data i.e. node 10 then,



temp1 = temp.next;

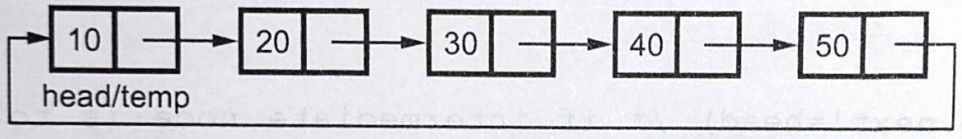


while (temp.next ! head)
temp = temp.next;

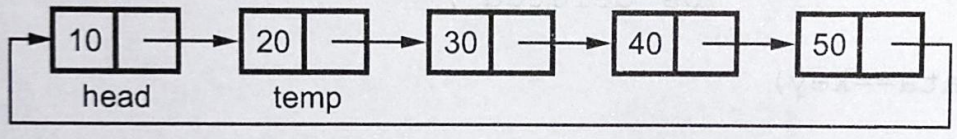


temp. next = temp1;
head = temp1;

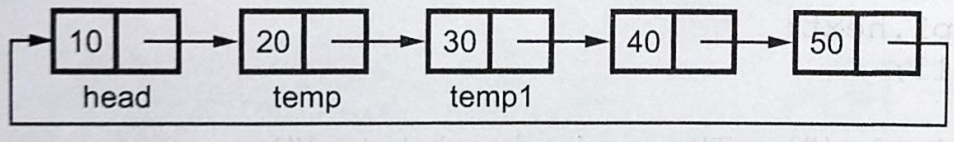
If we want to delete an intermediate node from a linked list which is given below



We want to delete node with '30' then

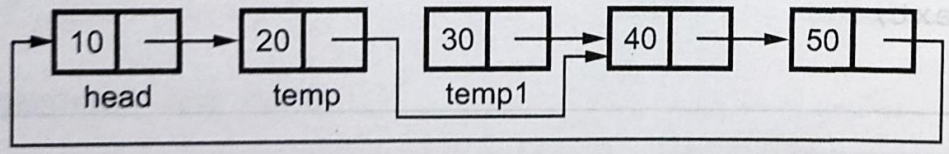


Key = 30
and
temp.next.data
= key, hence

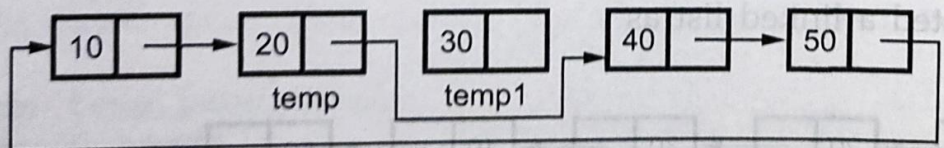


temp1 = temp.next;

Now



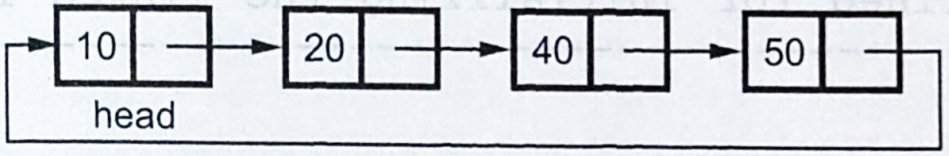
temp.next
= temp1.next;



temp1. next = null

```
temp1=null   / * to deallocate memory */
```

The linked list can be -



Thus node with value 30 is deleted from CLL.

5. Searching a node from circular linked list

```
node search(node head, int key)
{
 node temp;
 temp = head;
 while ( temp.next!=head)
 {
  if( temp.data ==key)
      return temp;
  else
     temp=temp.next;
 }
 return null;
}
```