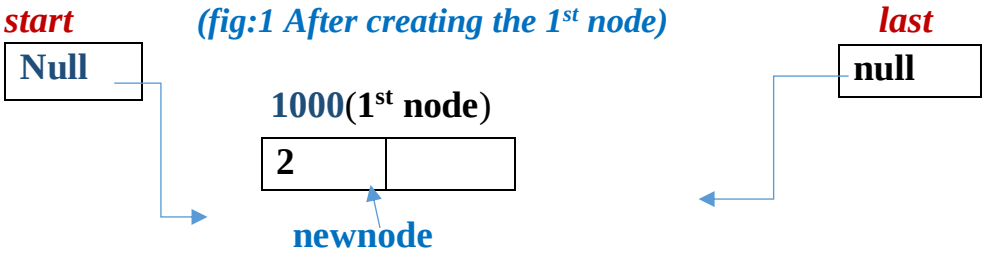

```
/*Structure of a node in Circular single linked list*/
class NODE
{
        int info;
        NODE link;
}
/* The above  class 'NODE' will be used to create nodes of the circular single linked
list which is same as nodes in single linked list: nodes are nothing but class type
objects that you were creating in ICP by using new operator and constructor call */

/*Each node of the above class type will hold an integer value in the circular list*

public class CIRCULAR_SINGLE_LL_DEMO
{
      static NODE start=null;

      public static void main(String[] args)
      {
            Scanner sc=new Scanner(System.in);
            char ch;
            int opt;
            int data;
       /*The following do…while loop implements the concept of menu driven program*/
            do
            {   /* Display the menu consisting of different operations on single linked list*/

                  System.out.println("1. Create list 2. Display list 9. Count nodes");
                   System.out.println("3.Insert at beginning 4. Insert at end");
                  System.out.println("5. Insert at any position 6. Delete at beginning");
                   System.out.println(" 7. Delete at end 8. Delete at any position ");

                    System.out.println("Enter your option");
                    opt=sc.nextInt();

                    switch(opt)
                   {
                        case 1: create_single_LL();
                                  break;
                        case 2: display_list();
                                 break;
                        case 3: System.out.println("Enter the info of new node:");
                                  data=sc.nextInt();
```

```
                    insert_at_beg(data);
                    break;
                case 4: System.out.println("Enter info of the new node:");
                    data=sc.nextInt();
                    insert_at_end(data);
                    break;
                case 5: System.out.println("Enter the info of new node:");
                    data=sc.nextInt();
                    System.out.print("Enter the info key node "
                      + "\n after which you want to insert the new node:");

                    int node_info=sc.nextInt();
                    insert_at_any_pos(data , node_info);
                    break;
                case 6: delete_at_beg();
                    break;
                case 7: delete_at_back_end();
                    break;
                case 8: System.out.println("Enter info of the node to delete”);
                    node_info=sc.nextInt();
                    delete_at_any_pos(node_info);
                    break;
                case 9:
                    int c=count_node();
                    System.out.println("No. of nodes in the list: ”+ c);
                    break;

                  default:
                    System.out.println("Invalid option” );
            }  /* End of switch */

          System.out.println("\nDo you want to perform another operation(y/n)");
          ch=sc.next().charAt(0);
        }while(ch=='y'|| ch== 'Y');  /* End of do---while loop */

    }  /* End of MAIN method */

/* CREATE SINGLE LINKED LIST METHOD*/

/* The following figure shows the status of a circular single list when it is empty initially,
before execution of create list method */
```

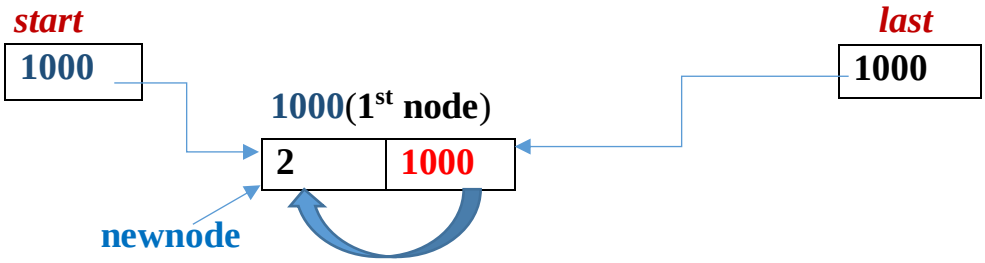
```
public static void create_single_LL()
{
        Scanner sc=new Scanner(System.in);
        char ch;
        NODE newnode=new NODE(); /*←Creates the first node as shown in fig: 1*/
        System.out.println("Enter the info of first node");
        newnode.info=sc.nextInt();
```



/*After execution of above statements the new node is created and the entered value from keyboard gets stored in info part of the new node, as shown in the figure: 1*/

```
        start=last=newnode; /*←Stores new nodes address in start and last pointer, figure:1*/
        last.link=start; /* ←Stores 1st nodes address in the link part of current last node*/
```

(Fig: 2 After attaching the 1st node into the list : the initial list containing one node only)

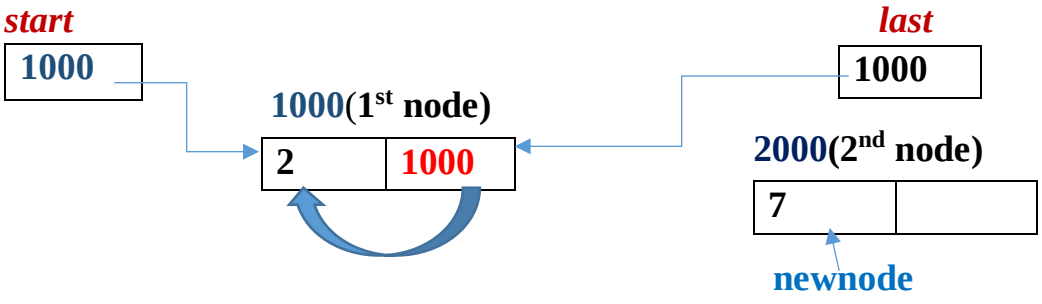



```
        System.out.println("Do u want to create another node(y/n)");
        ch=sc.next().charAt(0);
```

*/*If you want to create more nodes then enter 'y' to execute the while…loop repeatedly*/*

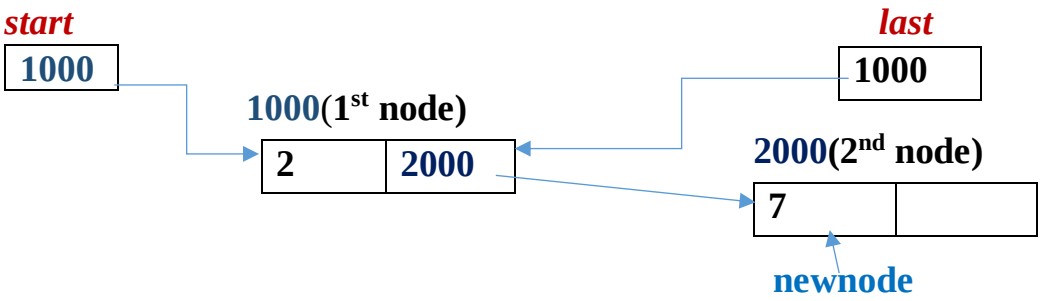
```
        while(ch=='y' || ch=='Y')
        {
            newnode=new NODE(); /* ← Creates another new node */
            System.out.println("Enter the info of next new node:");
            newnode.info=sc.nextInt(); /*←Stores entered value in info part of the new node */
```

(Fig: 3 After creation of the 2nd node in the while loop)



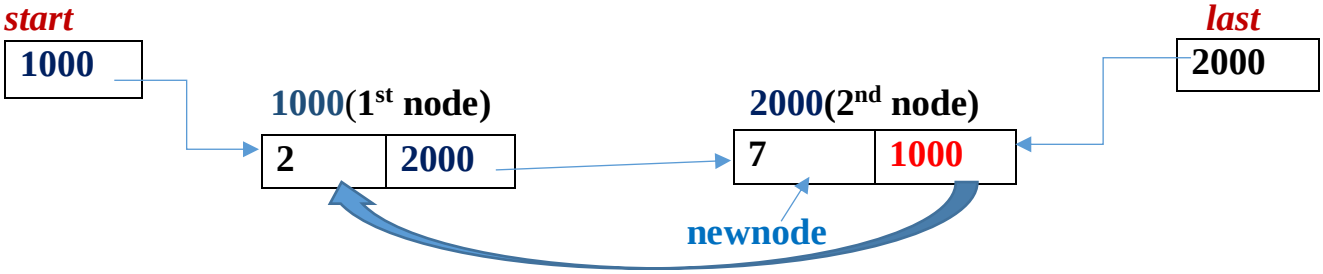

```
            last.link=newnode; /*← attach the new node end back end i.e. store new nodes address
in link part of current last node which is referred by 'last' pointer as shown in fig: 3*/
```

(Figure: 4 after storing new nodes address in link part of current last node)




```
last=newnode; /* stores new nodes address in 'last' pointer*/
last.link=start; /* stores first nodes address in new last nodes link part*/
```

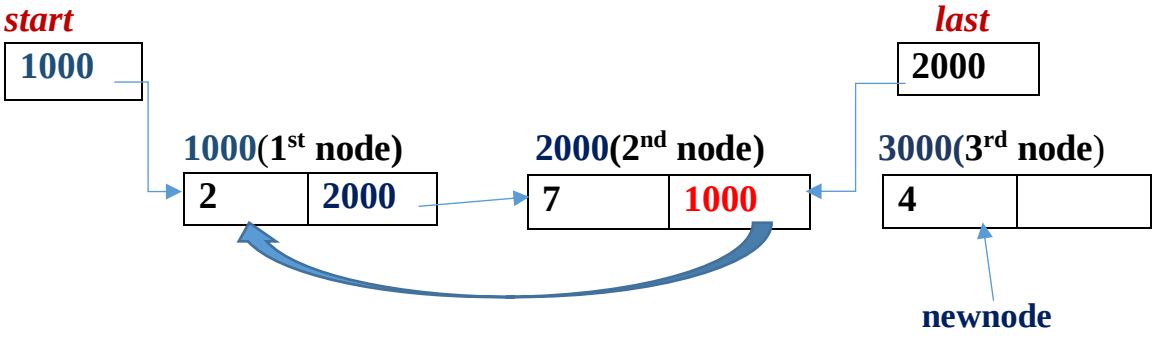
(Figure: 5 after updating 'last' pointer to new nodes address and link part of new last node)



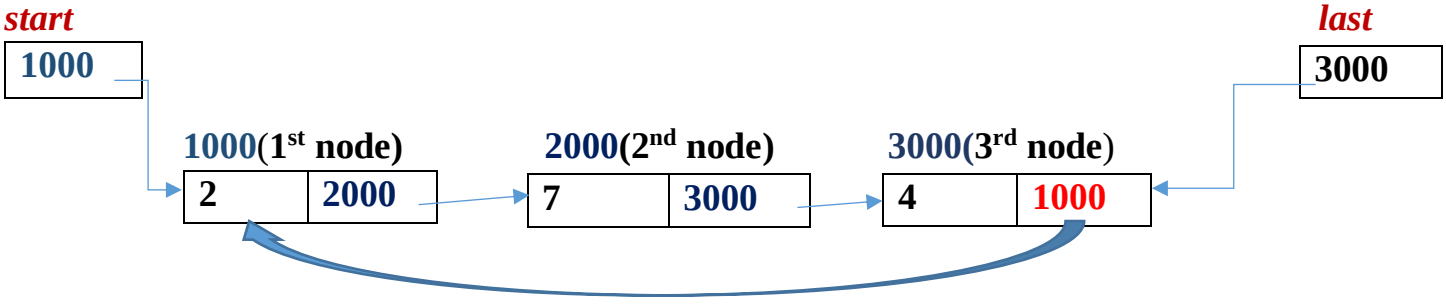
*/*Fig: 3 the final circular linked list after creating 2nd node*/*

```
System.out.println("Do u want to create another node??(y/n)");
ch=sc.next().charAt(0);

} /* End of while loop */

}   /* End of create list method */
```

*/*if you will enter 'y' again then the while…loop will execute for the 2nd time to create 3rd node of the list as shown in following figure: 6 & 7*/*

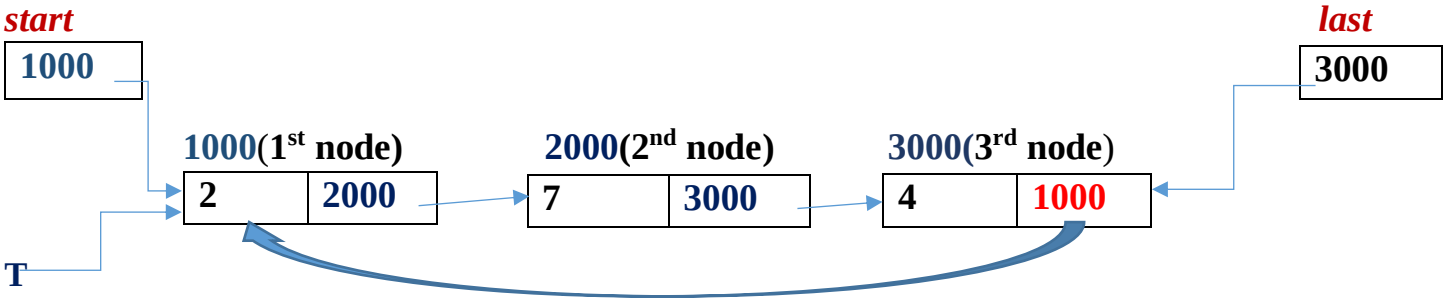


*/*Fig: 6 Before insertion of 3rd node*/*



*/*Fig: 7 after insertion of 3rd node: THE FINAL LIST */*

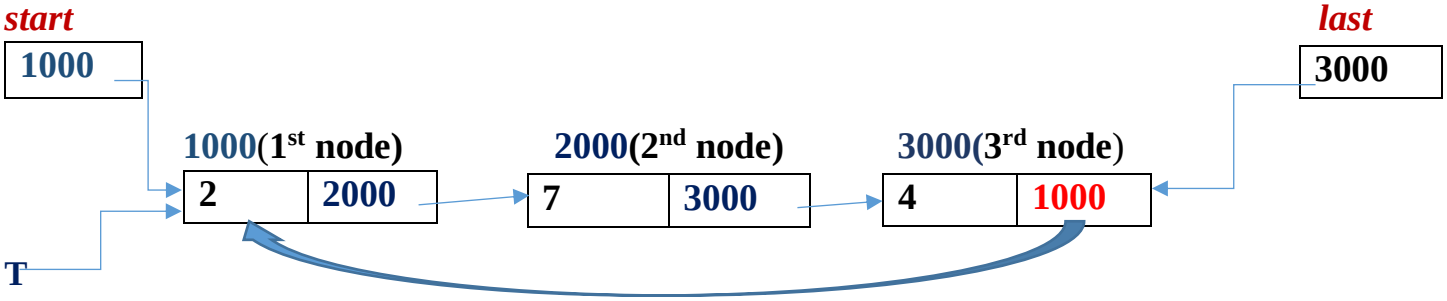
```
/* DISPLAY CIRCULAR SINGLE LINKED LIST METHOD*/
public static void display_list()
{
        if(start==null)  /*← Checks whether the list is empty or not */
        {
             System.out.println("list is empty");
             return;
         }
        else
        {
              System.out.println("\n The Circular Single Linked List is…..\n");
```



*/*Fig: 8 THE FINAL CIRCULAR SINGLE LINKED LIST */*

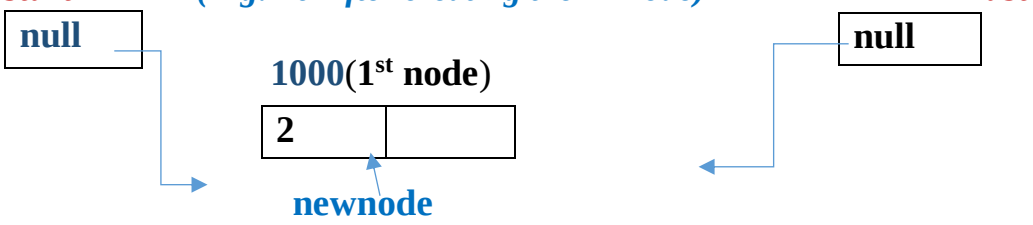
```
              NODE T=start; /*←T refers to 1st node before the while loop starts execution*/
              while(T.link != start) /*← checks whether we have reached at last node of the list?*/
              {
                    System.out.print(T.info + " → "); /*←prints info part of current node*/
                    T = T.link;           /*←moves the reference variable T to the next node */
               }/*while…loop terminates when T refers to last node, so last node is skipped in the loop*/

               System.out.print(T.info + " → "); /*←prints info part of last node after the loop*/

         }/*End of else clause*/
}/* End of Display list method */

/* COUNTING THE NUMBER OF NODES IN A CIRCULAR SINGLE LINKED LIST*/
public static int count_nodes()
{
       if(start==null)
        {
             return 0;
        }
        else
        {
```

*/*Fig: 9 THE FINAL CIRCULAR SINGLE LINKED LIST */*



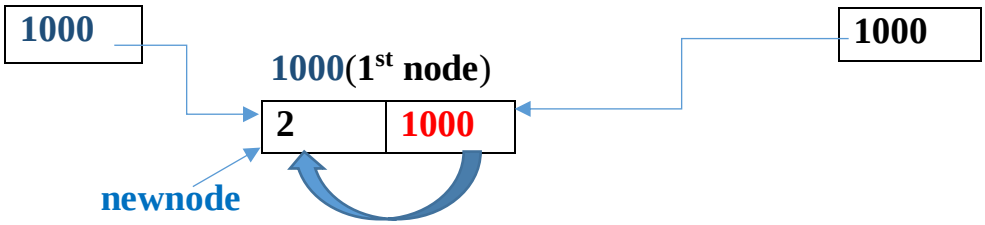
```
            int c=0; /*stores the number of nodes counted */
            NODE T=start; /*←T refers to first node before while loop execution*/
           while(T.link != start) /*←Checks whether we reached  at last nodes address*/
            {
                c++;            /* ←Increments c by 1 if while loop condition is true*/
                T=T.link;       /*← Moves the reference variable t to next node*/
            } /*the above while…loop terminates when T refers to last node*/
            c= c+1; /*finally increment c by 1 for the last node*/
            return c;
          }
    } /* End of counting method */

// INSERT A NEW NODE AT FRONT END OF THE LIST
public static void insert_at_beg(int data) /* data is value  to be stored in new node*/
{
        NODE newnode=new NODE();
        newnode.info=data;
```

start *(Fig: 10 After creating the 1st node)* *last*




```
        if ( newnode == null )  /* if newnodes value is 'null' then memory is full*/
        {    System.out.println("Memory full, u can't create new nodes");
             return ;
         }
          else if ( start == null)
          {
```

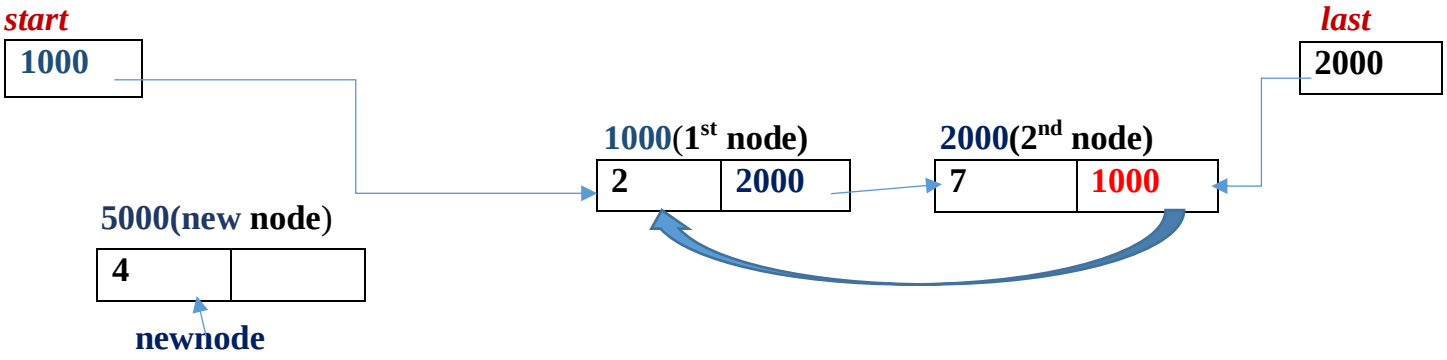
start *last*



(fig:11 After attaching the new node when the list is initially empty)

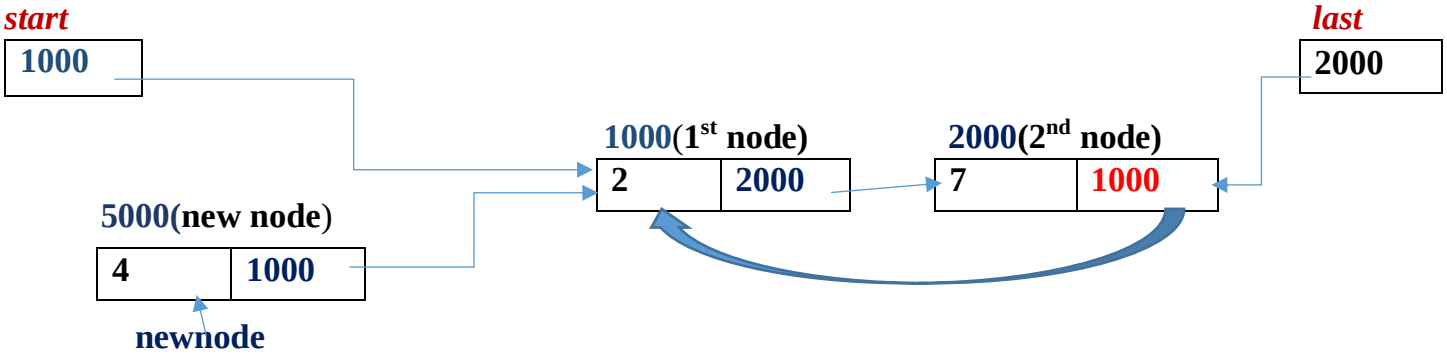
```
             start=last=newnode;
             last.link = start;
        } /*End of else if clause*/
        else
        {
             /*this else clause is executed if the list contains some nodes*/
```

(Figure:12 Before attaching the new node into the list when the list is not empty)



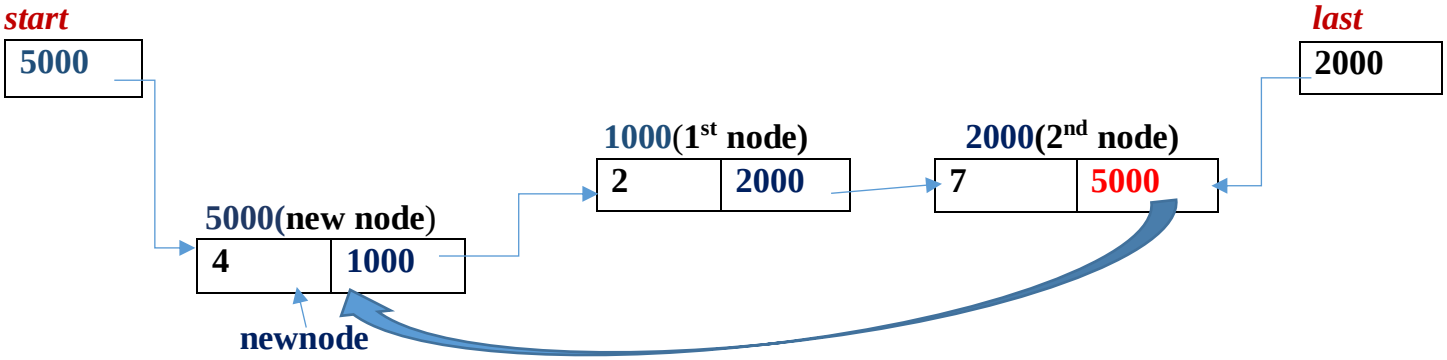
newnode.link=start;

(Figure:12 after storing the address of current 1st node in the link part of new node)



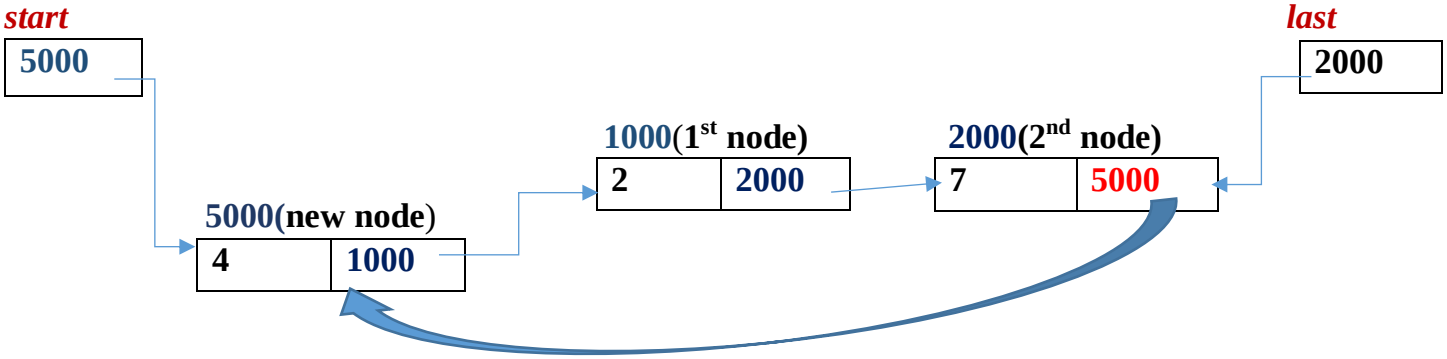
start=newnode;
last.link=start;

(Figure: 13 after updating 'start' pointer and link part of current last node)



} */*End of else clause*/*

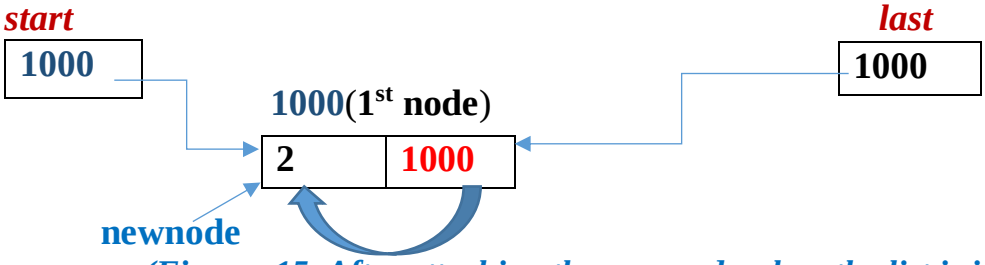
(Figure: 14 after successful insertion of new node at front end of the list: the final list)



} /* End of insert at beginning of the list method */

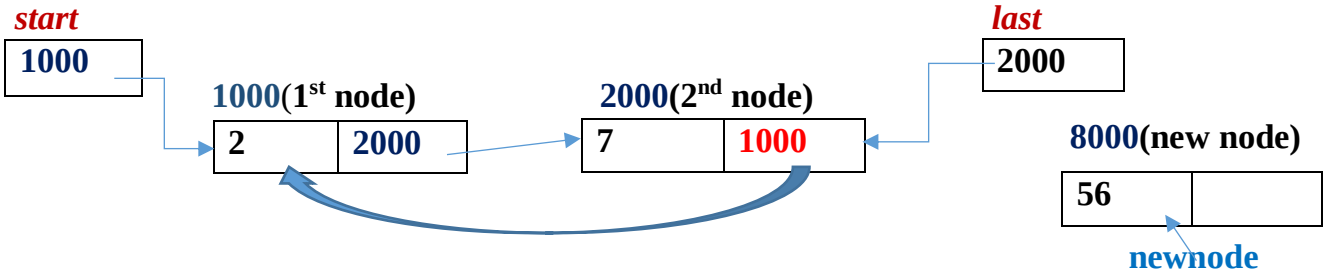
```
/* INSERT AT END OF THE LIST METHOD*/
public static void insert_at_end(int data)
{
        NODE newnode=new NODE();
```

```
newnode.info=data;
if(start == null)  /*if the list empty initially : same case as insert at beginning */
{     start=last=newnode;
      last.link=start;
```

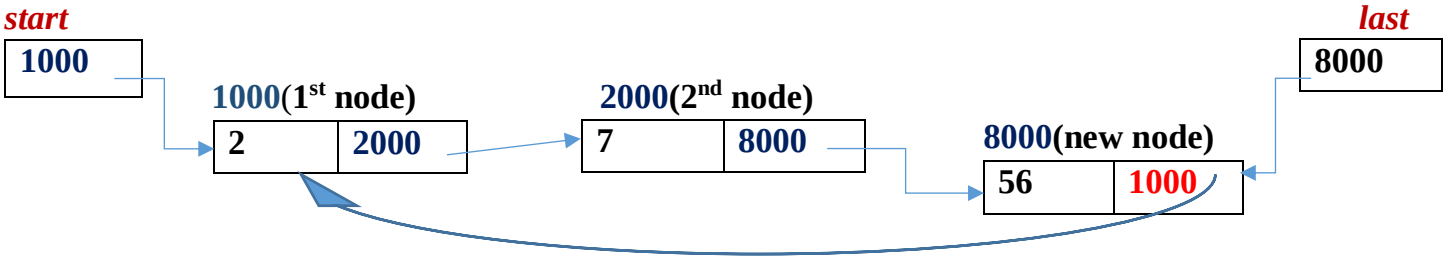


(Figure: 15 After attaching the new node when the list is initially empty)

```
}/*End of if clause*/
else
{   /*Fig: 16 before attaching the new node into the list at back end*/
```



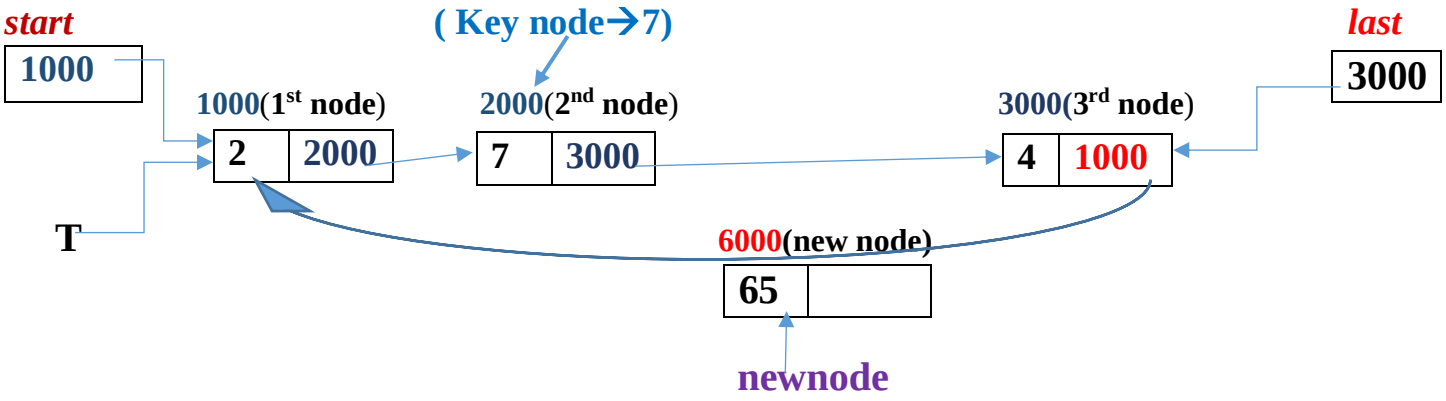

```
last.link=newnode; /*←Store new nodes address in link part of current last node*/
last=newnode;  /*←update 'last pointer to new nodes address*/
last.link=start;  /*←store 1st nodes address in link part of new last node*/
```




```
/*Fig: 17 after attaching the new node into the list at back end: the final list*/
}

}/*End of else clause*/
}/*End of insert at back end of the list*/

/* INSERTION AT ANY POSITOIN OF A LINKED LIST */
/*Here information of a specific node in the original linked list is given, we
need to insert a new node after the that specific node if present in the list.*/
public static void insert_at_any_pos(int newnode_data , int key)
{       /*if key value is present in any node than insert*/
        if ( start==null)
         {
              System.out.println("the list is empty, so the key node is not present");
```

```
            return;
        }
        else
        {          /* if the key value is 7*/
```

*/*Fig: 18 move to the specific node containing key→7 after which new node will be inserted */*




```
                NODE T=start; /* T starts from 1st node*/
```

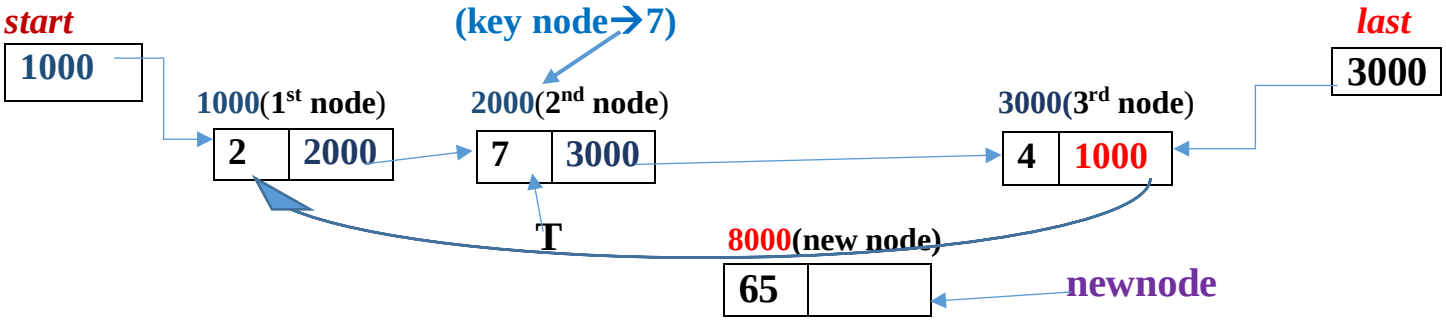
*/*The while…loop moves T from one to another node until last node is not reached or the key value is not fount*/*

```
                while(T.link != start && T.info!= key)
                {
                     T=T.link;
                 }
```

/*if the key is not present in any node (except last node) then while…loop terminates when T refers to last node*/

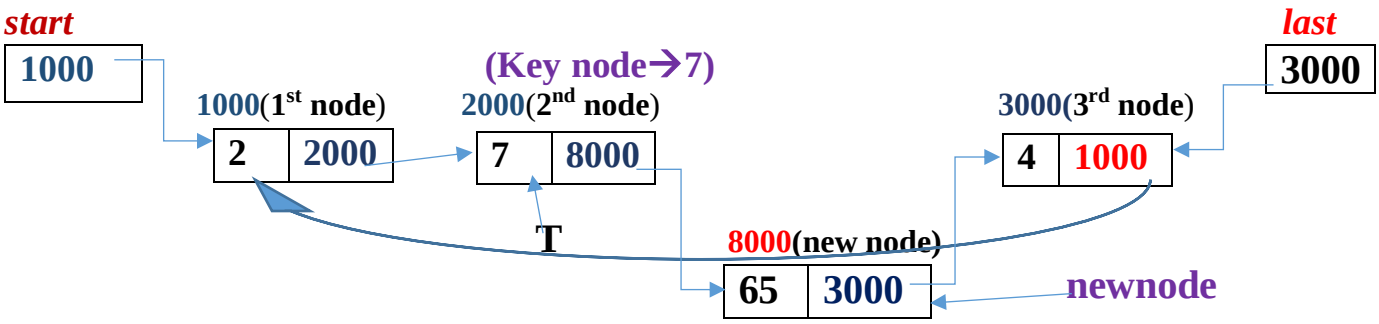
```
                if (last.info== key ) /*← checks whether key is present at last node of the list*/
                {
                   insert_at_end(newnode_data); /*same as insertion at back end of the list*/
                }
              else if ( T.link == start) /*if T reaches  at last node then key is not present in the  list*/
                {
                     System.out.print("The key is not present in any node”);
                      return;
                 }
                  else    /*Fig: 19 insert at any other position after the key→7 node*
                  {
```

/*if value of key is 7 then the while loop will terminate when T refers to 2nd node as shown figure:19*/




```
                NODE newnode=new NODE(); /* create the new node*/
                newnode.info=newnode_data; /*store new data in new node*/
                newnode.link=T.link;
               T.link = newnode;
```

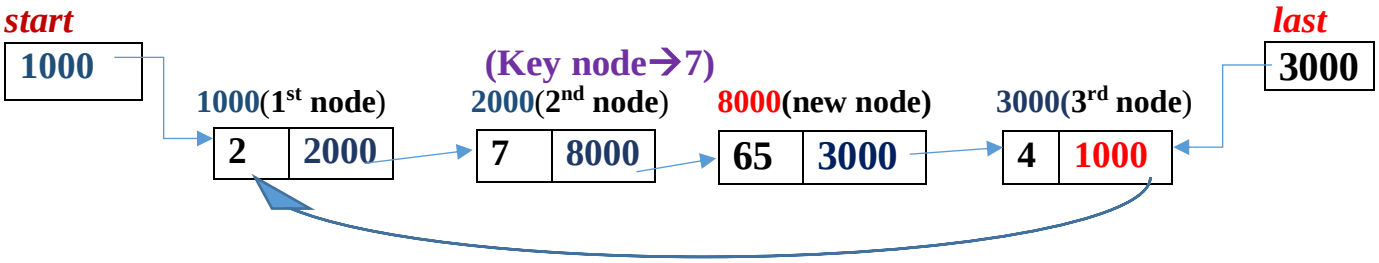
*/*Fig: 20 attaching the new node after the key node*/*



}*/* End of the inner else clause*/*

}*/*End of the outer Else clause*/*

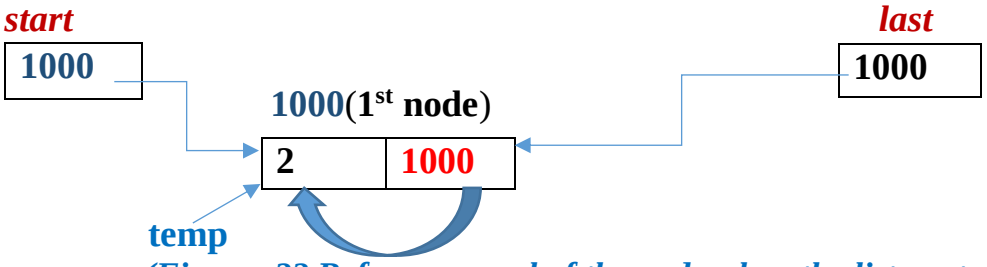
*/*Fig: 21 after successful insertion of the new node: The final list*/*



} /*End of Insertion at any position method */

/* DELETION AT FRONT END OF THE LIST: DELETES 1ST NODE*/

```
public static void delete_at_beg()
{       if(start == null)
        {
            System.out.println("list is empty ....");
            return;
         }
          NODE temp=start;  /*←Stores 1st nodes address in temp if list is not empty*/
          if ( start.link== start) /*←checks whether the list contains only one node?*/
          {
```



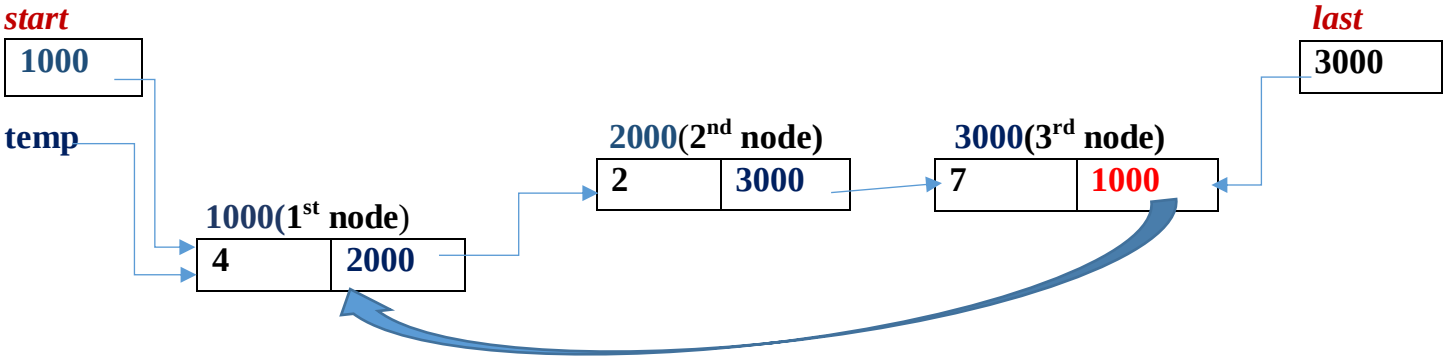
(Figure: 22 Before removal of the node when the list contains only one node)

```
          start=last=null; /*←After deletion of only node the list will be empty*/
```

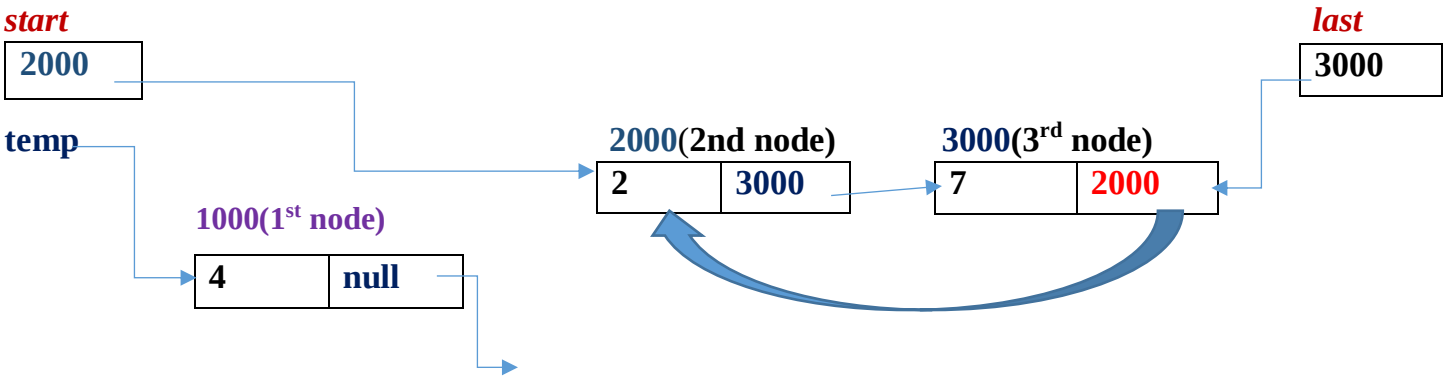

```
/*   (Figure: 23 After removal of the only node: the final list is empty */
          } /*End of if clause*/
```

```
else
{    /* this else clause is executed if the list contains more than one node*/
```

(Figure: 24 before removal of the 1st or front node at front end of the list: the final list)

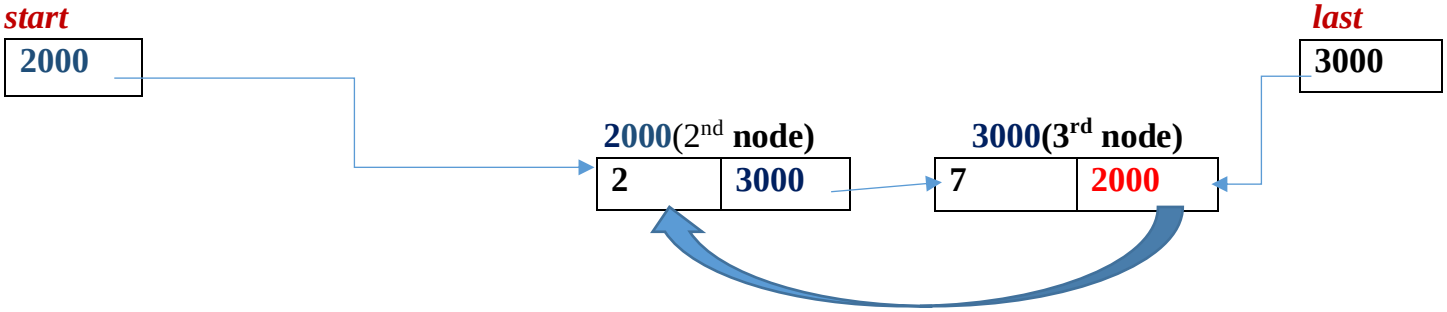



```
start=start.link;
temp.link=null;
last.link=start;
```



*/*Fig: 25 After the execution of above statements the 1st node gets removed from the list */*

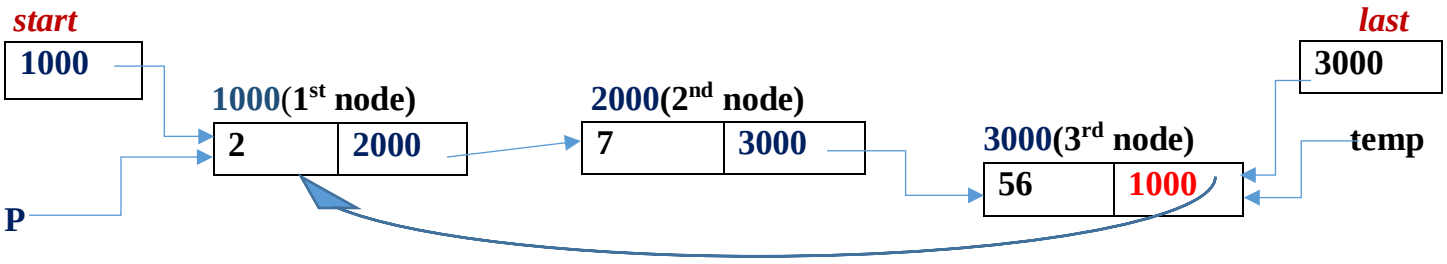
```
} /* End of else clause*/

System.out.println("The deleted node is:" + temp.info); /*←prints 4*/
```



*/*Fig: 26 The final list after deletion of 1st node */*

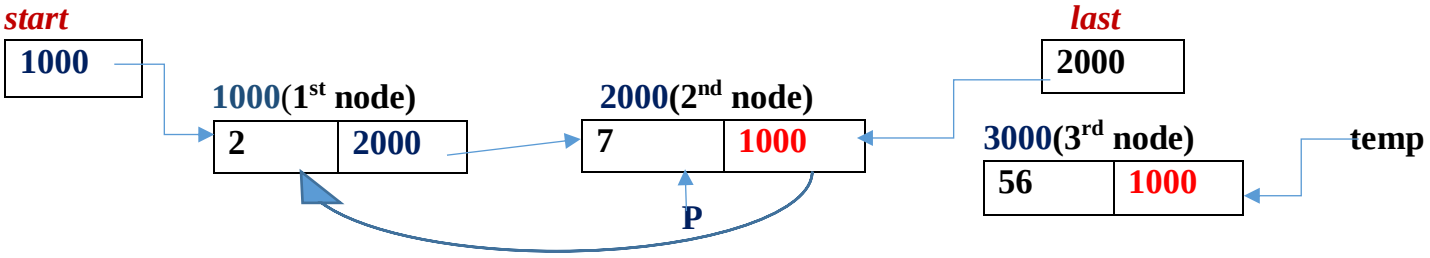
```
}/* End of Delete at front end method */

/* DELETION AT BACK END OF THE LIST: DELTES THE LAST NODE */
public static void delete_at_back_end()
{
        if ( start == null)
        {    System.out.println("the list is empty");
             return;
        }
```

```
NODE temp = last;    /* last nodes address gets stored in temp*/
if( start.link == null ) /* if the list contains only one node*/
{
   start=last=null; /*←set both 'start' & 'last' to 'null value to make the list empty*/
}
else
{  /* if the list contains more than one node, this else clause is executed*/
```



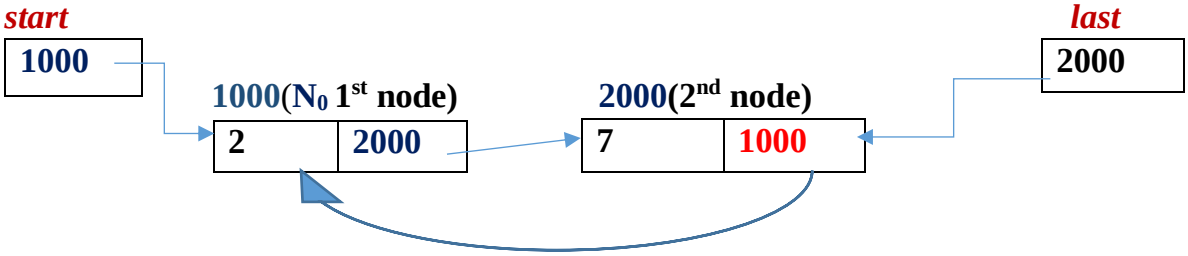
*/*Fig: 27 Before removal of last node when the list is contains more than one node*/*

```
NODE P=start;
while(P.link != last) /*←Loop will stop when P will refer to previous node
{                                   of the last node*/
      P = P.link; /*←moves P to next node*/
}    /*after termination of while…loop prev will refer to 2nd node*/
 P.link=start;
 last=P;
```



*/*Fig: 28 after removal of last node from the list*/*

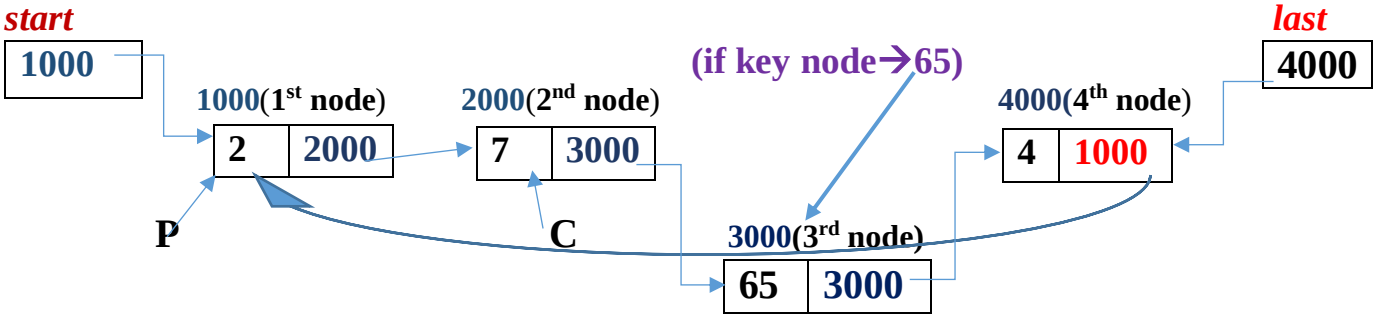
```
System.out.println("The deleted node is:" + temp.info); /*prints  56*/
```



/*Fig: 29 After successful deletion of the last node: the final list */

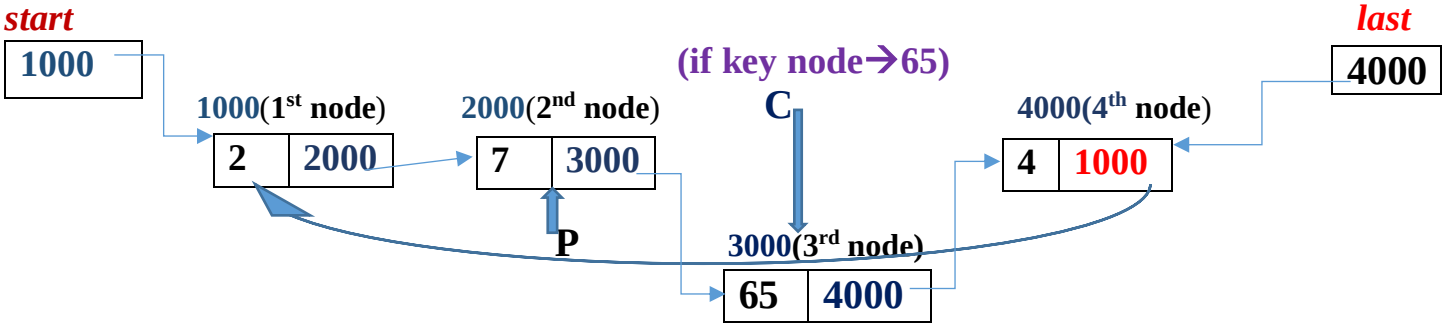
```
} /* End of Deletion at back end of the list */

/* DELETION AT ANY POSITION OF THE LIST
/* Here information or value of a specific node is given, that is to be deleted*/
public static void delete_at_any_pos(int  key)   /*key is info of the node to be deleted*/
{       if ( start==null )
        {
            System.out.println("the list is empty");
            return ;
```

```
}

if( start.info == key) /* if key is present at first node of the list*/
{
    delete_at_beg();    /*← same as deletion at beginning */
}
else if ( last.info == key) /*if the key is present at last node of the list*/
{
     delete_at_back_end();   /*←same as deletion at back end of the list*/
}
else   /*this else clause is executed if the key is present at any other node*/
{          /*Fig: 30 before removal of the key node from the list*/
```



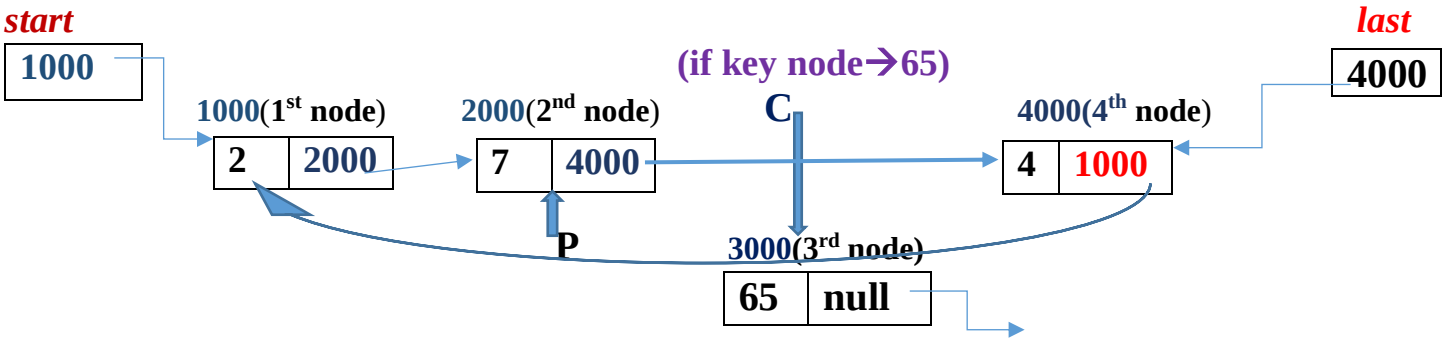

```
NODE P=null;
NODE  C=start;    /* Before loop C starts from 1st  node*/
while(C.link != start && C.info != key)
{
       P=C;
       C=C.link;
} /*this loop will terminate, if C reaches at last node or key is found*/
if ( C.link == start) /*after loop if we have reached at last node of the list*/
{
     System.out.println("the node is not present in the list");
     return;
}
else
{   /* if the key is present at any other node other than 1st and last node*/
```

*/*Fig: 31 after termination of the loop P→ 2nd node and C→3rd node : before removal of key node*/*



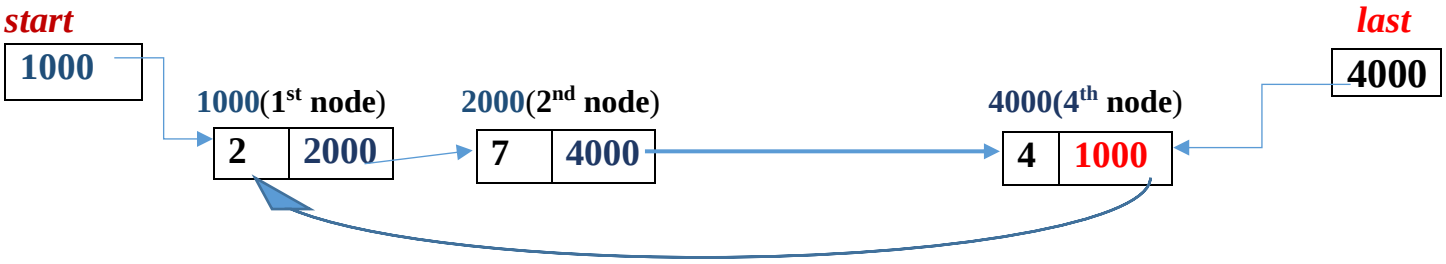

```
P.link = C.link;
C.link = null;
```

*/*Fig: 32 after the loop P→ 2nd node and C→3rd node : After removal of key node*/*




```
                System.out.println("The deleted node is:" + temp.info);
            }/*End of inner else clause*/

        }/* End of outer Else clause*/
```

*/*Fig: 34 After removal of key node i.e. the 3rd node : the final list*/*




```
    }/*End of deletion at any position method*/


} /*END OF CIRCULAR_SINGLE_LL_DEMO CLASS*/
```