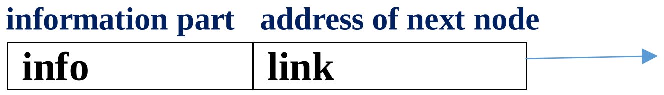
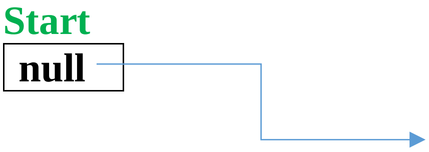


```
class NODE
{
        int info;
        NODE link;                        /*Structure of a node in single linked list*/
}

/* The above  class ‘NODE’ will be used to create nodes , nodes are nothing but
class type objects that you were creating in ICP by using new operator and
constructor call */

public class 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 ");
                  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("9. Count total number of nodes in the list");
                  System.out.println(“10. Reverse the list 11. Sorting the list”);
                  System.out.println("12. Search a node 13. Update a node");

                   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_nodes();
           System.out.println("number of nodes in the list= "+ c);
           break;
 case 10: System.out.print("\n Before reversing…  ");
            display_list();
            reverse_list();
            System.out.print("\nAfter reversing…");
            display_list();
             break;
  case 11: System.out.print("\n Before sorting…. ");
             display_list();
              sort_list();
             System.out.print("\n After sorting…");
             display_list();
             break;
   case 12: System.out.print("\nEnter the key element to”
                                          +” search");
              int k=sc.nextInt()
              linear_search(k);
              break;
    case 13: System.out.print("\nEnter the info of the node to “
                                     + ”be updated");
              k=sc.nextInt();
```

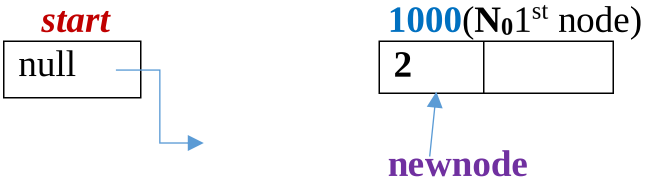
```
                              System.out.print("\nEnter new value of the node");
                               int newval=sc.nextInt();
                               update_node(k , newval);
                               break;
                        default:
                             System.out.println("Invalid 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 single list when it is empty initially, before execution of create list method */



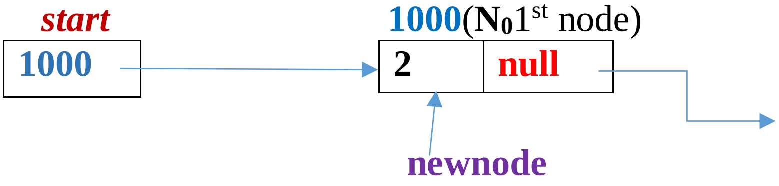
```
public static void create_single_LL()
{
    Scanner sc=new Scanner(System.in);
    char ch;

    NODE newnode=new NODE(); /*←Creates the first node as shown in the following
                                                    figure*/
    System.out.println("Enter the info of first node");
    newnode.info=sc.nextInt();
```

Fig: A

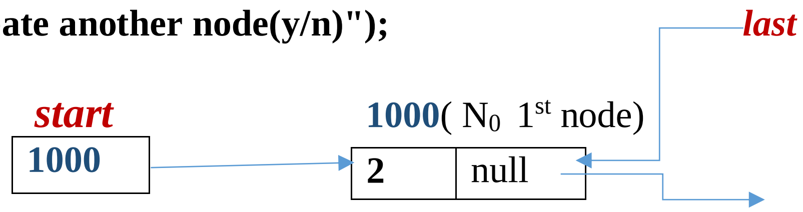


/*After execution of above statements the new node is created and the entered value from keyboard gets stored in info part of the new node, as shown in the figure: A*/

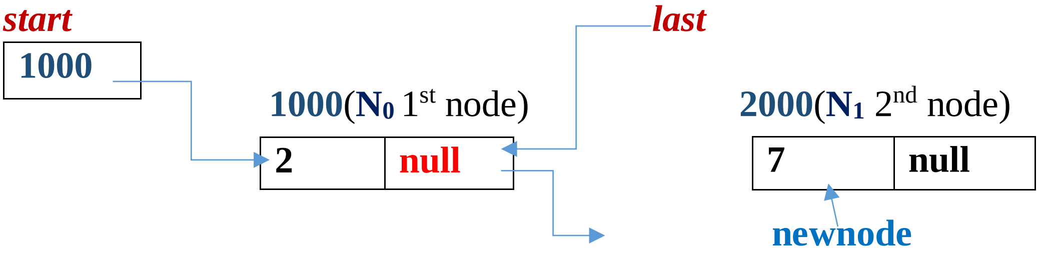
```
   start=newnode; /*←Stores new nodes address in start pointer as show in the
                                         figure:B*/
   newnode.link=null; /* ←Stores null value in the link part of newnode*/
```



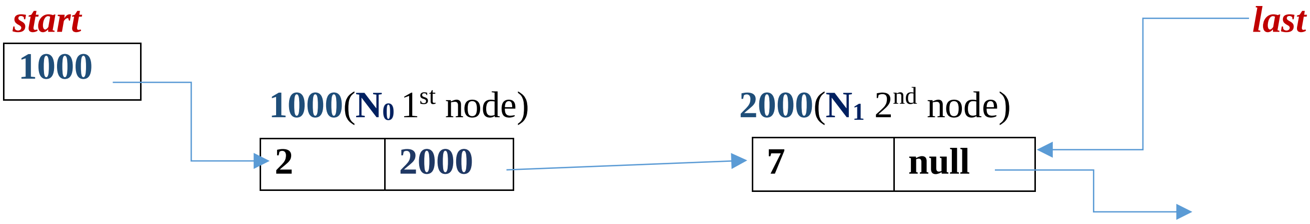
```
    System.out.println("Do u want to create another node(y/n)");
    ch=sc.next().charAt(0);
```



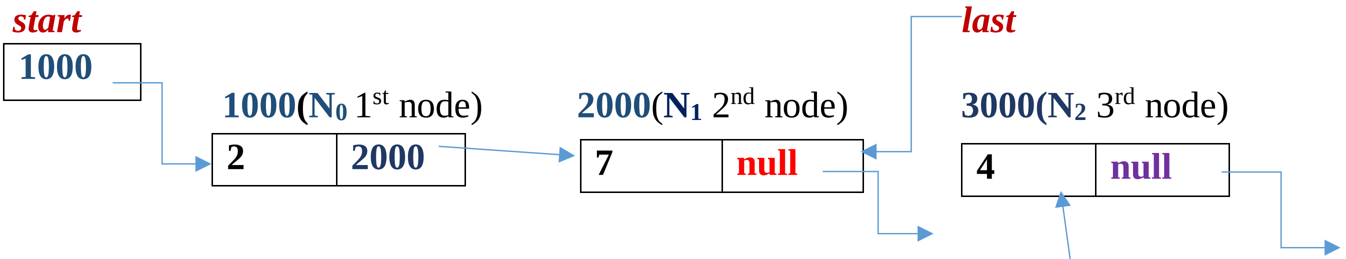
```
    NODE last=start;
/* 'last' is a NODE type reference variable like 'start', that always refers to current last
node while adding new nodes at back end of the list during create operation*/
    while(ch=='y' || ch=='Y')
    {
        newnode=new NODE(); /* ← Creates a new node */
        System.out.println("Enter the info of next new node:");

        newnode.info=sc.nextInt(); /*←Stores entered value in info part of the new node*/
        newnode.link=null; /*←Stores null in link part of the new node */

        /*After execution of the above statements in 1st iteration of the while loop 2nd node is
created, value of info part is accepted from the keyboard and null value is stored in link
part of 2nd node, as shown in following figure*/
```



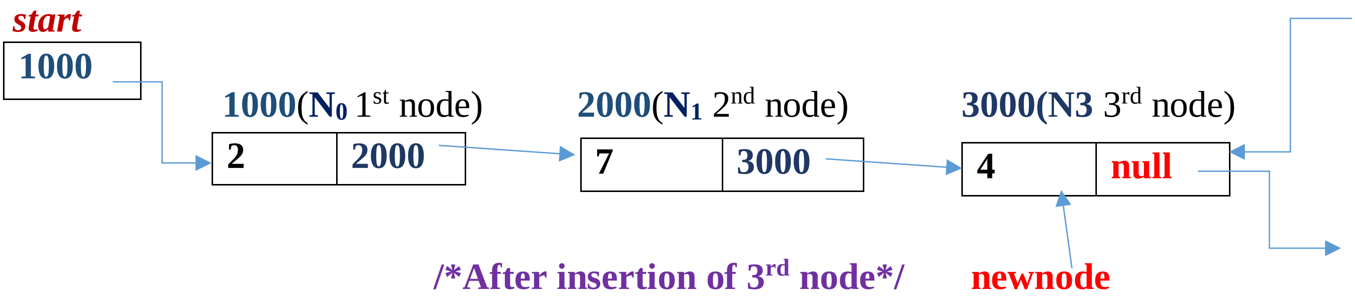
```
         last.link=newnode; /*←Stores 2nd nodes address in link part of 1st node*/
         last= newnode; /*← Updates last pointer, so that 'last' will refer to 2nd node*/

/*After execution of above two statements in 1st iteration of the while loop the new node is
attached at back end of the existing linked list, then last pointer refers to 2nd node, which
will be the new last node of the list*/
```



```
          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 */
```

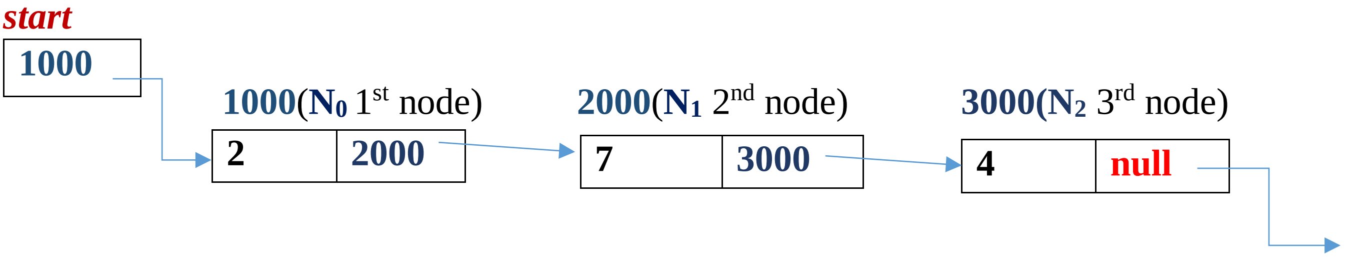


/*Before insertion of 3rd node*/

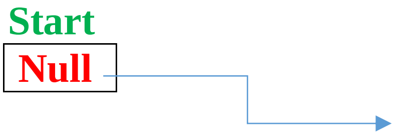


/*After insertion of 3rd node*/

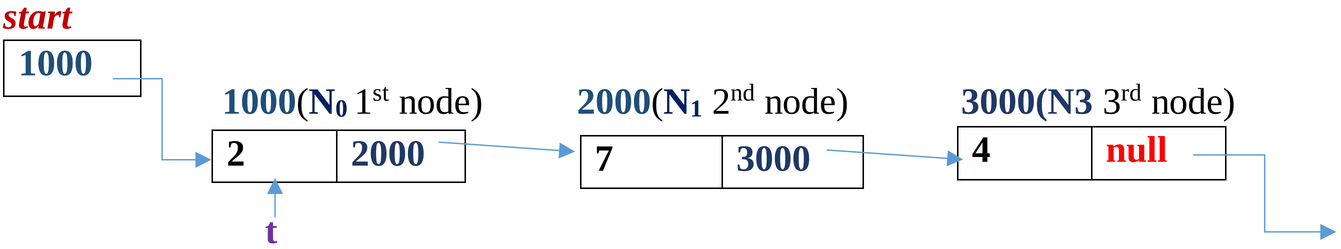
/*the following figure shows the final linked list after termination of create method where the while…loop executed twice that created 2nd and 3rd node after creation of the 1st node of the list before while…loop execution. The local temporary variables ‘last’ and ‘newnode’ get deleted after termination of the create list method*/



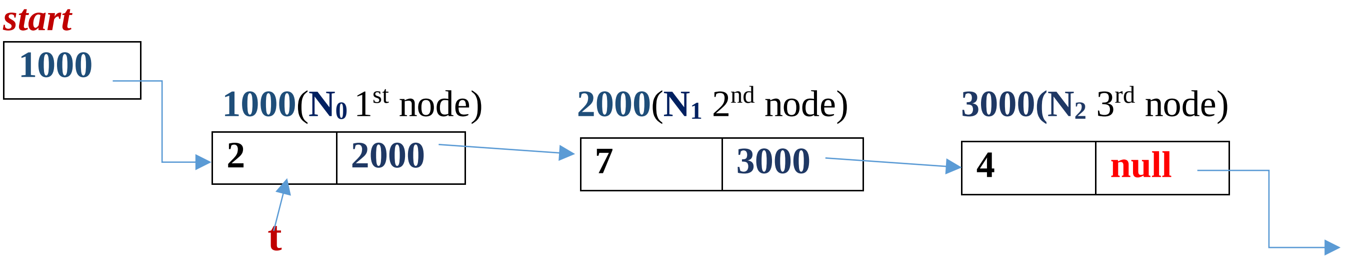
```
/* DISPLAY SINGLE LINKED LIST METHOD*/
public static void display_list()
{
```



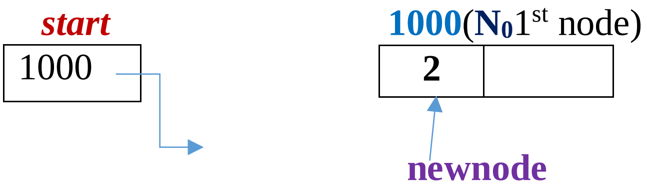
```
        if(start==null)  /*← Checks whether the list is empty or not */
        {
            System.out.println("list is empty");
            return;
         }
        else
        {
             System.out.println("\n The Single Linked List you have created is…..\n");
```



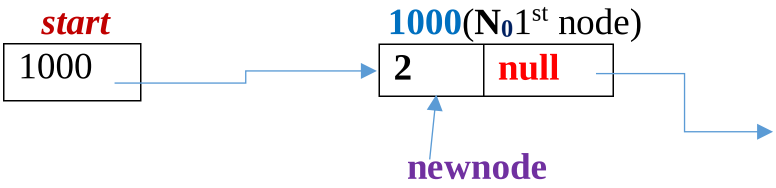
```
             NODE t=start; /*← t refers to 1st node before the while loop starts execution*/
             while(t != null) /*← checks whether we have reached end of the list?*/
             {
                System.out.print(t.info + " → "); /*←prints info part of current node*/
                t = t.link; /*←moves the reference variable t to the next node */
            }/*End of while loop*/

            System.out.println(); /*prints a new line*/
        }/*End of else clause*/
}/* End of Display list method */

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



```
            int c=0; /*stores the number of nodes counted */
            NODE t=start; /*←t refers to first node before while loop execution*/
           while(t != null) /*←Checks whether, we have reached at end of the list or not*/
           {
               c++; /* ←Increments c by 1 if while loop condition is true*/
               t=t.link; /*← Moves the reference variable t to next 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)
{
        NODE newnode=new NODE();
        newnode.info=data;
        if ( newnode == null )
        {    System.out.println("Memory full, u can't create new nodes");
              Return ;
         }
          else if ( start == null)
          {
```



```
start=newnode;
 newnode.link=null;
```



```
}
else
{
```

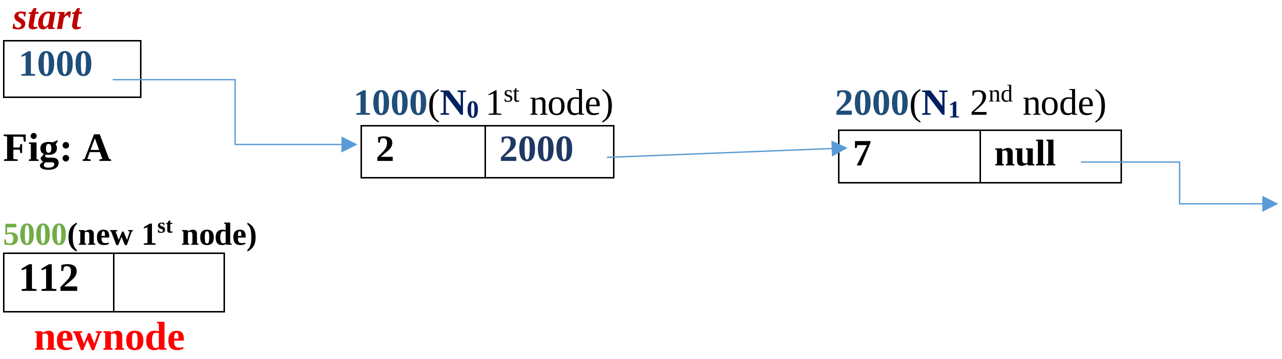


Fig: A

```
newnode.link=start;
```

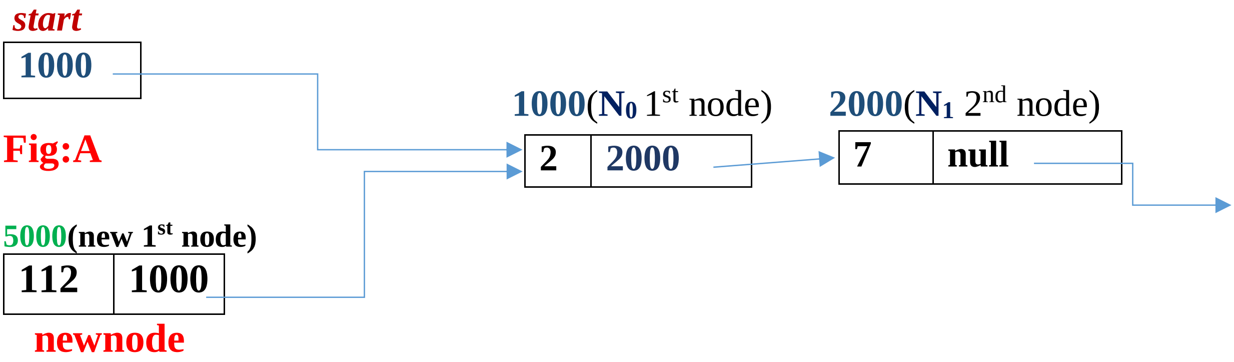


Fig:A

```
start=newnode;
```

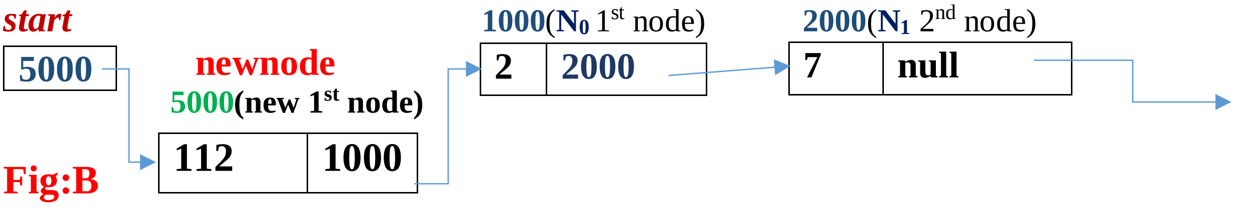
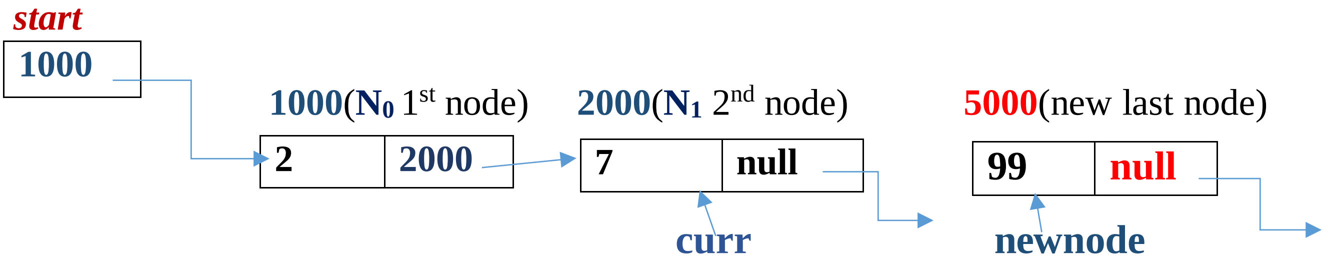


Fig:B

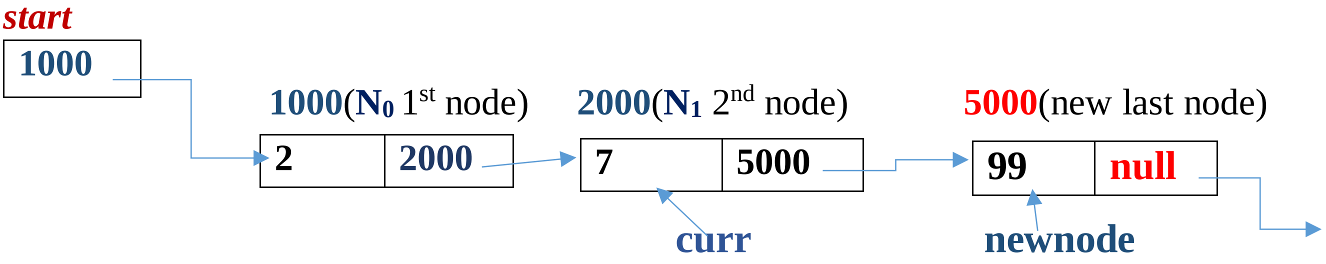
```
        } /*End of else clause*/
} /* End of insert at beginning of the list method */
```

/* INSERT AT END OF THE LIST METHOD*/

```
public static void insert_at_end(int data)
{
        NODE newnode=new NODE();
        newnode.info=data;
        newnode.link=null;

        if(start == null)
        {     start=newnode;
        }
        else
        {
```



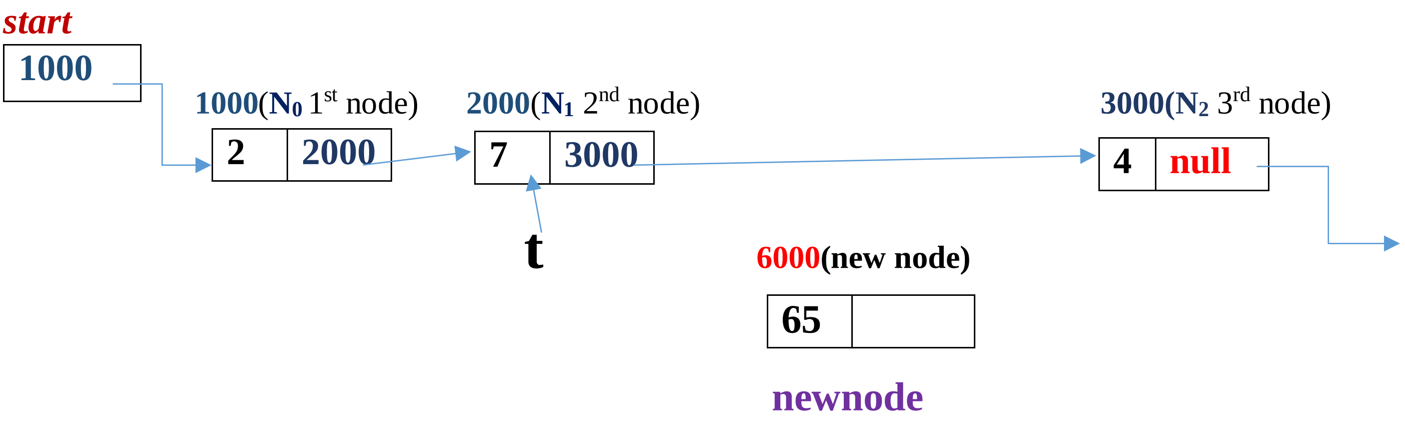
```
              NODE curr=start; /*←curr refers to first node before while…loop*/
              while(curr.link != null) /*←While..loop terminates when curr refer to last
              {                                          node*/
                    curr = curr.link;   /*moves the curr ref. variable to next node*/
              }
              curr.link=newnode; /*←In the last nodes link part stores the address of
                                                     new node*/
```



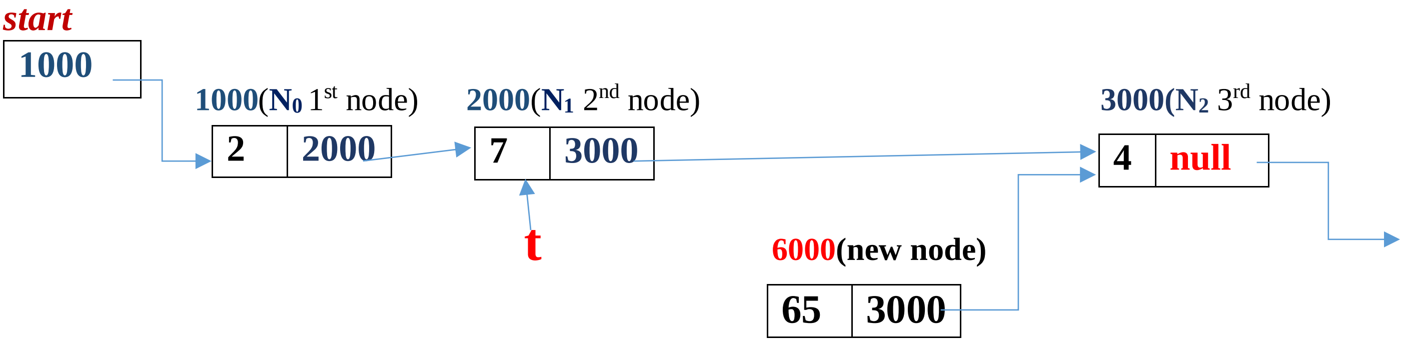
```
        }/*End of else clause*/
}/*End of insert at back end of the list*/
```

/* 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.*/

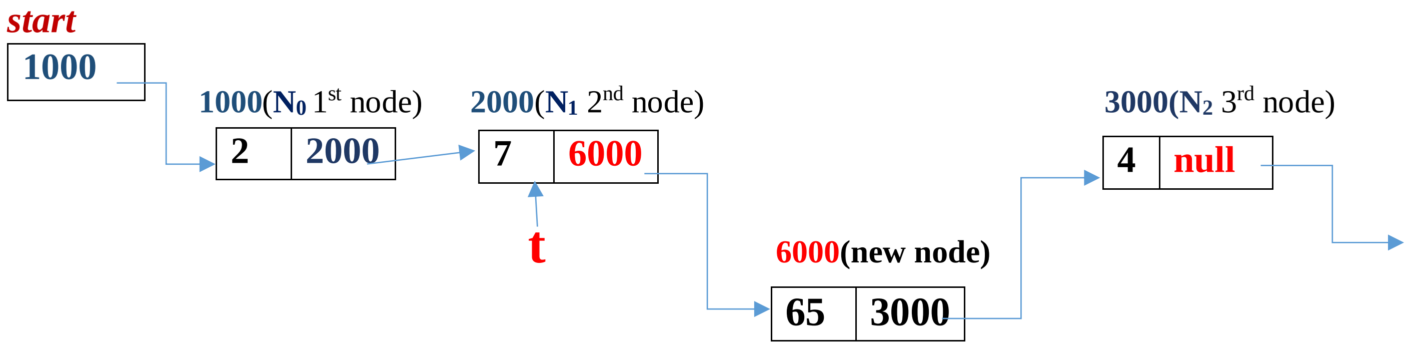
```
public static void insert_at_any_pos(int newnode_data  ,  int node_info)
{
         NODE newnode=new NODE();
         newnode.info=newnode_data;

         if ( start==null)
         {
             System.out.println("the list is empty, so the specific node is not present");
             return;
         }
         else
         {
          /* move to the specific node containing node_info after which new node will
            be inserted */
```



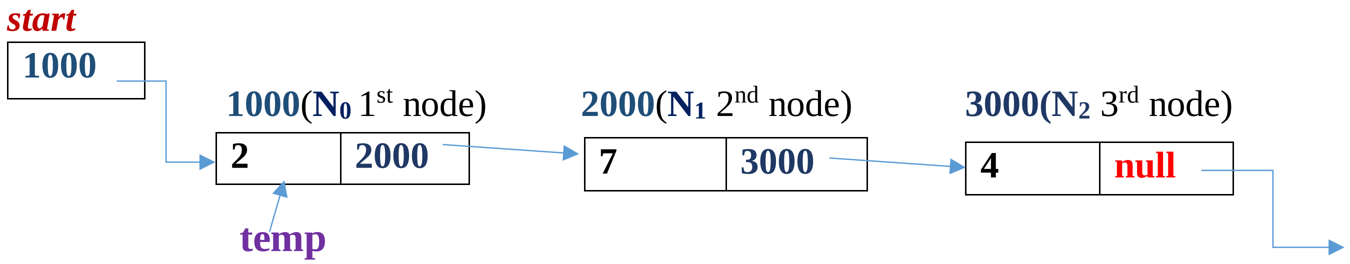
```
              NODE t=start;
               while(t != null && t.info!= node_info)
               {
                    t=t.link;
                }

               if ( t==null ) /*← checks whether we reached end of the list after the
               {                    while…loop?*/

                    System.out.print("\nthe node “ + node_info + " is not present\n");
                    return;
                }
               else
               {  /*This else clause will execute if the node_info is present in the list*/
                      newnode.link=t.link;
```



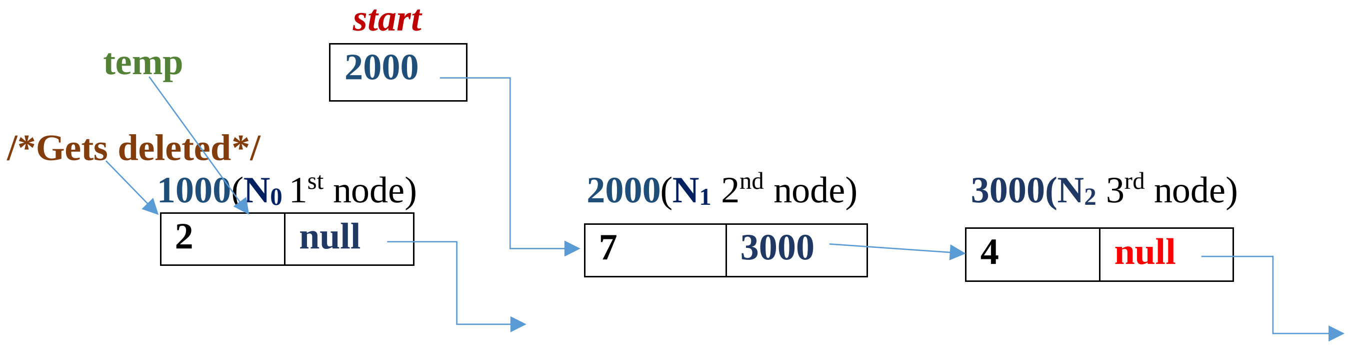
```
                       t.link=newnode;
```



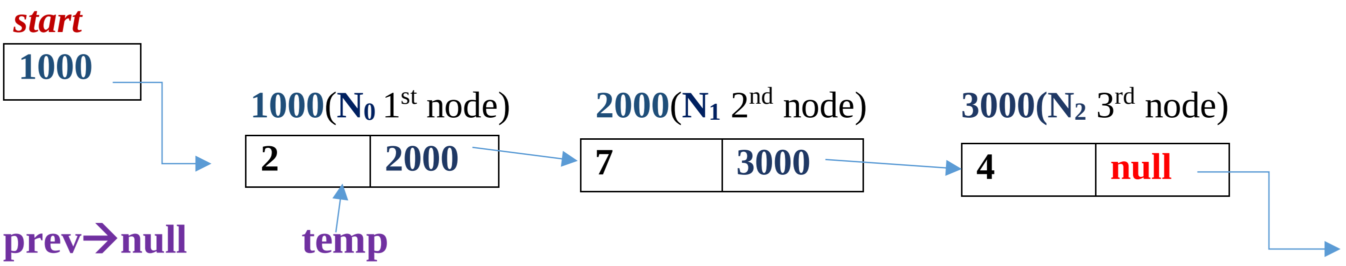
```
        }/* End of inner else clause*/
    }/*End of outer else clause*/
} /*End of Insertion at any position method */

/* 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== null) /*← checks whether the list contains only one node?*/
     {
         start=null; /*← set start to null to make the list empty */
     }
     else
     {    /* this else clause is executed if the list contains more than one node*/
```



```
            start=start.link;
            temp.link=null;
        /*After the execution of above statements the list will be as follows*/
```



```
        } /* End of else clause*/
         System.out.println("The deleted node is:" + temp.info);
}/* 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 = start;
           if( start.link == null ) /* if the list contains only one node*/
           {     start=null;
           }
           else
            {   /* if the list contains more than one node the move to last node and
                   And previous node of the last node*/
```



```
                 NODE prev=null;
                 while(temp.link != null) /*←Loop will stop when temp will refer to last
                 {                                       node */
                        prev=temp; /*←stores the current value of temp*/
                        temp=temp.link; /*←moves temp to the next node*/
                 }
              /*after termination of while…loop prev will refer to 2nd node and temp will refer

                to 3rd node which is the last node in list that is to be deleted*/
```

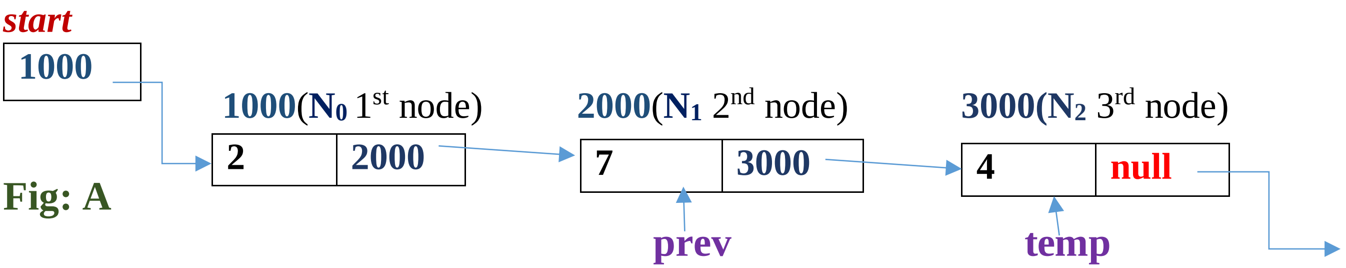


Fig: A

```
                   prev.link=null; /*stores null value in link part of 2nd node*/
```

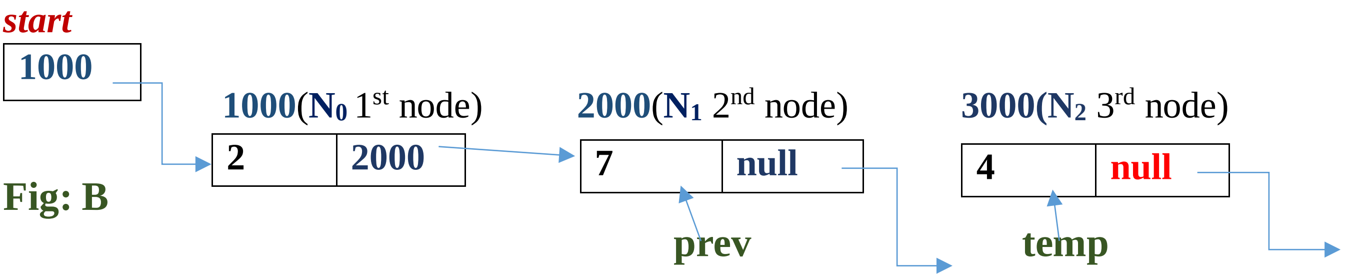


Fig: B

```
         }
        System.out.println("The deleted node is:" + temp.info);
/*After deletion of the 3rd node which was the last node the final linked list is as
shown in the following figure: C*/
```

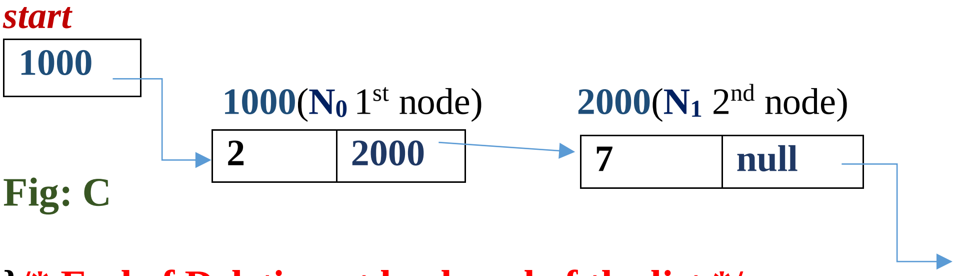


Fig: C

```
}/* 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
        {      /*this else clause is executed, if the key element is not present at first
                  node of the list*/
```

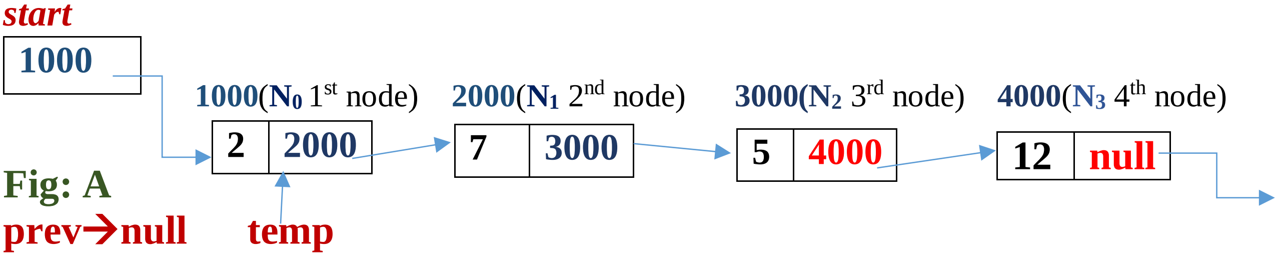


Fig: A

```
/*move the temp to the that node which contains the key and prev to previous node
of the key node which we want to delete*/

                  NODE prev=null;
                  NODE temp=start;
```

```
                while(temp != null && temp.info != key)
                {
                        prev=temp;
                         temp=temp.link;
                 }
                 if ( temp == null)/*after loop if we have reached at end of the list*/
                 {
                        System.out.println("the node is not present in the list");
                        return;
                 }
                 else
                 {   /* if the key=5, then the loop will terminate when temp refers to
            3rd node and prev refers to 2nd node, then this else clause will execute*/
```

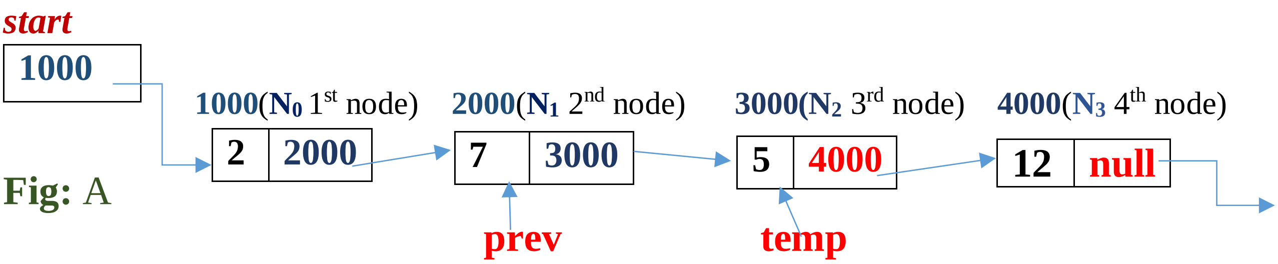


Fig: A

```
                   prev.link=temp.link;
                   temp.link=null;
/*After execution of above two statements the list is as follows: 3rd node gets
deleted*/
```

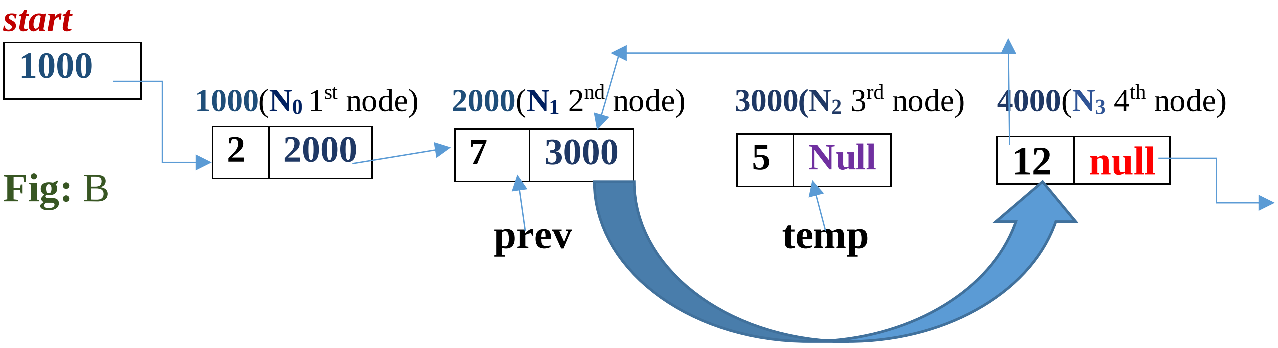


Fig: B

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

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

/* REVERSING A SINGLE LINKED LIST */
/* Input: Before reversing the list*/
```

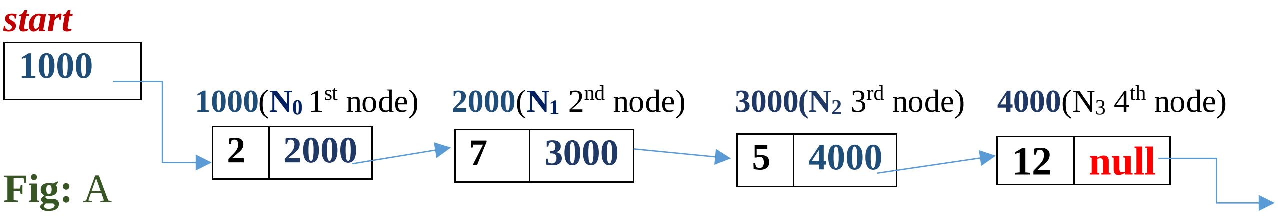


Fig: A

/* Output: After reversing the list */

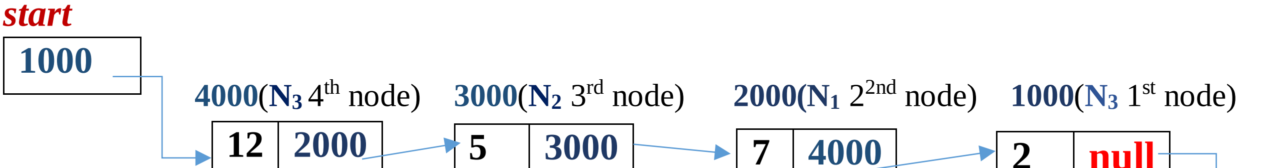


Fig: B

```
/*
public static void reverse_list()
{
       if(start == null)
      {
           System.out.println("the list is empty");
           return ;
       }
        else if ( start.link == null)
        {
                System.out.println("the list contains one node");
                System.out.println("reverse is same as the original list");
                return ;
        }
        else
        {
                NODE next= null;
                NODE prev=null;
                NODE curr=start;
                while(curr != null)
                {
                        next= curr.link;
                        curr.link = prev;
                        prev=curr;
                         curr=next;
                }
                start=prev;
         }
}/*End of Reverse method */
```

/*SORTING THE SINGLE LINKED LIST*/

/*Input: Before sorting, the original linked list as shown in fig: A*/

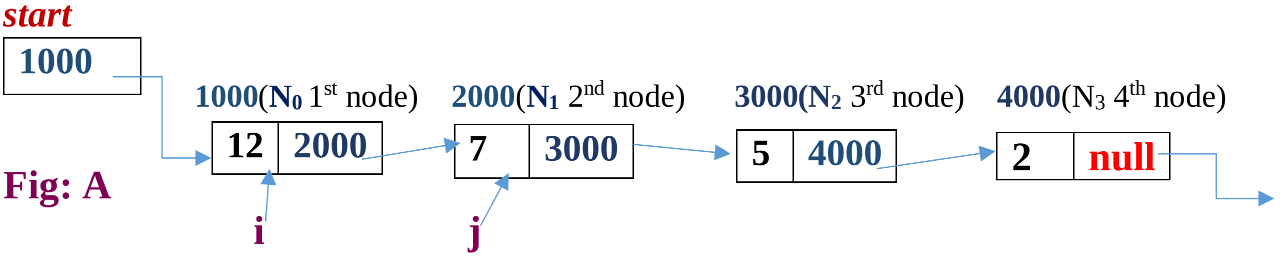


Fig: A

/*Output: After sorting, the sorted linked list as shown in fig: B*/

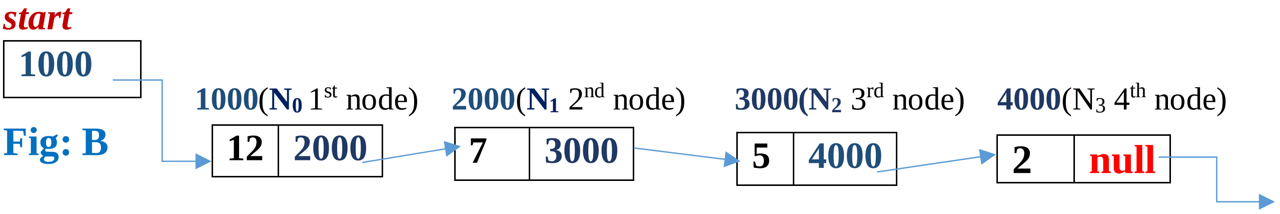
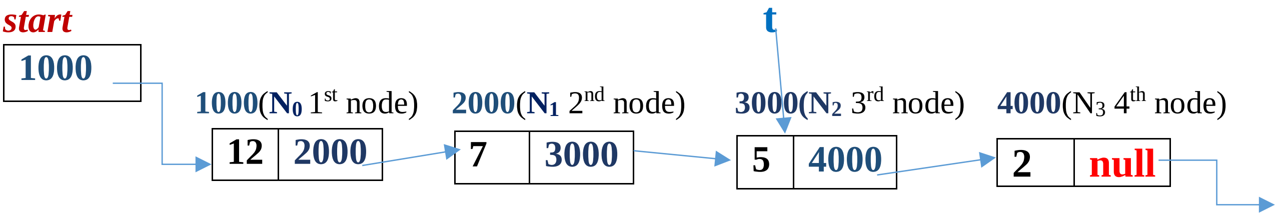


Fig: B

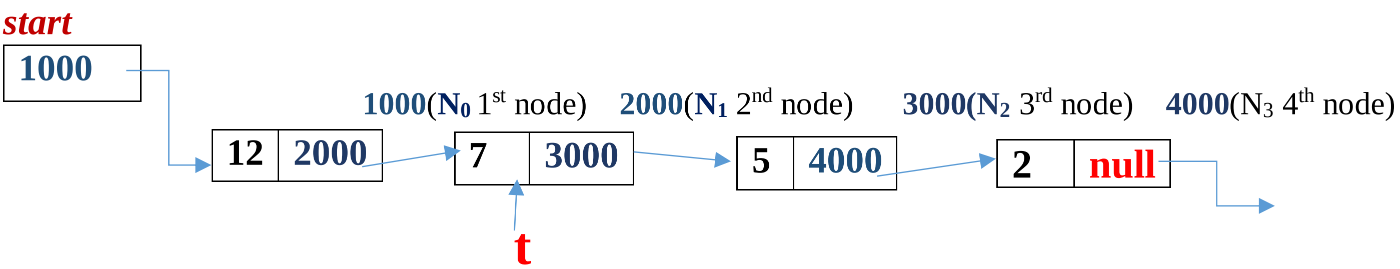
```
public static void sort_list()
        {
                for( NODE i=start ; i.link != null ; i = i.link)
                {
                        for(NODE j=i.link ; j != null ; j=j.link)
                        {
                                if( j.info > i.info )
                                {
                                        int t = i.info;
                                        i.info=j.info;
                                        j.info=t;
                                }
                        }
                }
        } /* End of sorting method */
```

```
public static void linear_search(int key)
{
         If ( start == null)
        {
             System.out.println("list empty");
             Return;
        }
        else   /*(if the key= 5: while loop will terminate when t refers to 3rd node)*/
        {
```



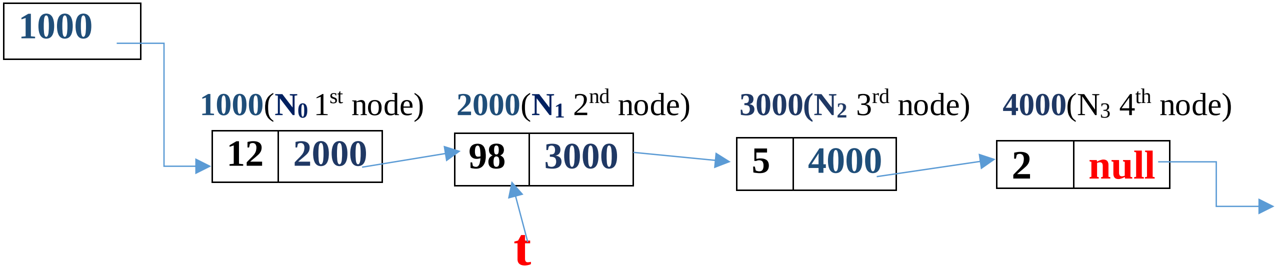
```
                boolean flag=false;
                NODE t=start;
               while( t != null)
               {
                     if( t.info == key)/*checks whether key is found at current node*/
                    {
                         flag=true;
                         break;
                     }
                     t = t.link;
```

```
                  }

                 if( flag == true)
                    System.out.println(“The key element is present in the list”);
                 else
                    System.out.println(“The key element is not present in the list”);
        }/*End of linear search method*/

public static void update_node(int key , int new_val)
{
         if ( start == null)
        {
             System.out.println(“list empty”);
             return;
        }
        else   /*(if the key= 7: while loop will terminates when t refers to 2nd  node)*/
        {      /*if the key is present in the list then this else clause is executed*/
```



```
             boolean flag=false;
             NODE t=start;
             while( t! = null)
              {
                    If( t.info == key)
                    {
                          t.info=new_val;
                          flag=true;
                          Break;
                     }
                    t= t.link;
              }
```

/*(if the key= 7: after updating 2nd node by the new_val=98*/



```
              if( flag==true)
                 System.out.println(“key node updated successfully”);
```

```
                else
                        Sstem.out.println(“the key node is not present ”);
        }/*End of update node method*/

} /*END OF SINGLE_LL_DEMO CLASS*/
```