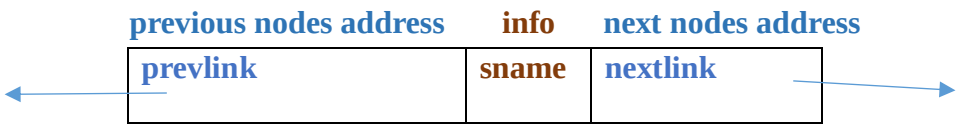
```
import java.util.Scanner;

class DLLNODE
{
        String sname;

        DLLNODE nextlink;
        DLLNODE prevlink;
}
/*The above class declaration will be used to create doubly linked list nodes where
each node will hold the name of a person in the information part*/

public class DOUBLY_LL_DEMO
{
        static DLLNODE start=null; /*Creates an empty doubly linked list*/
        static DLLNODE last=null;

        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                char ch;

                int opt;
                String data;
                do
                {
                        System.out.println("1. CREATE LIST 2.DISPLAY LIST ");
                        System.out.println("3.INSERT_AT_BEGNNING" );
                        System.out.println("3.INSERT_AT_BEGNNING ");
                        System.out.println ("4.INSERT_AT_BACK_END");
                        System.out.println("5.INSERT_AT_ANY_POSITION");
                        System.out.println("DELETE_AT_BEGNNING");
                        System.out.println("7.DELETE_AT_BACK_END");
                        System.out.println("8.DELETE_AT_ANY_POSITION");
                        System.out.println("9. COUNT TOTAL NO. OF NODES");
                        System.out.println("10.SEARCH A KEY 11.UPDATE A NODE");
                        System.out.println("12.SORT THE LIST");

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

                        switch(opt)
                        {
                           case 1: create_doubly_LL();
                                   break;
```



/*Structure of Node in Doubly linked list*/

```
case 2: display_list();
        break;
case 3:
        System.out.println("Enter the info of new node:");
        data=sc.next();
        insert_at_beg(data);
        break;
case 4:
        System.out.println("Enter info of the new node:");
        data=sc.next();
        insert_at_end(data);
        break;
case 5:
        System.out.println("Enter the info of new node:");
        data=sc.next();

        System.out.print("Enter the info of the specific node "
            + "\nafter which you want to insert the new node:");

        String node_info=sc.next();
        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 the node whcih "
           + "you want to delete");
         node_info=sc.next();
         delete_at_any_pos(node_info);
         break;
  case 9:
         int c=count_nodes();
         System.out.println("No.of nodes in the list="+c);
         break;
  case 10: System.out.println("Enter the key for search");
          String k=sc.next();
          linear_search(k);
          break;
  case 11:System.out.println("Enter info of the node to update");
          String key=sc.next();
          System.out.println("Enter new value of the node");
```

```
                    String new_val=sc.next();
                    update_node(key , new_val);
                    System.out.println("after updation...");
                    display_list();
                    break;
                case 12: System.out.println("Before sorting..");
                    display_list();
                    sort_list();
                    System.out.println("After sorting..");
                    display_list();
                    break;
                default:
                    System.out.println("invalid option");
                }/*End of switch*/

                System.out.println("\nDo you want another operation(y/n)");
                ch=sc.next().charAt(0);

        }while(ch=='y'|| ch== 'Y'); /* End of do…while loop*/

        sc.close();
    }/*End of main method*/

    public static void create_doubly_LL() /*create list method*/
    {
        Scanner sc=new Scanner(System.in);
        char ch;
```

Fig: 1

```
        DLLNODE newnode=new DLLNODE();
        System.out.println("Enter the info of first node");
        newnode.sname=sc.next();
```

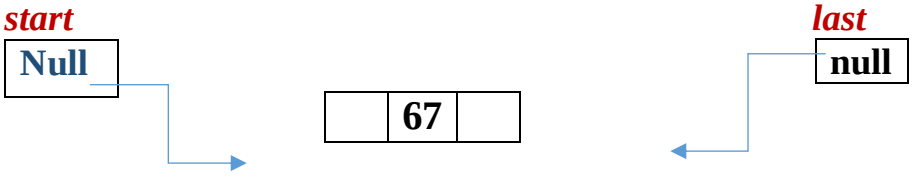
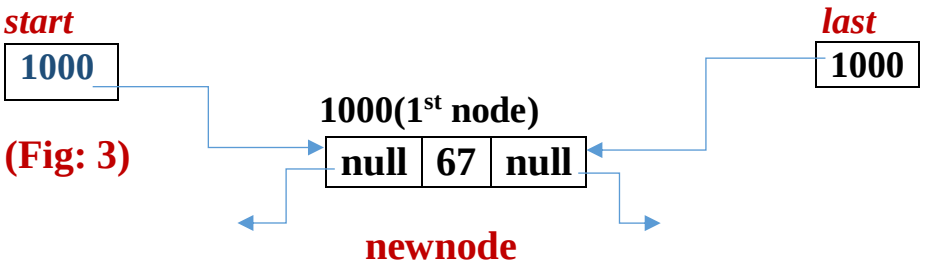


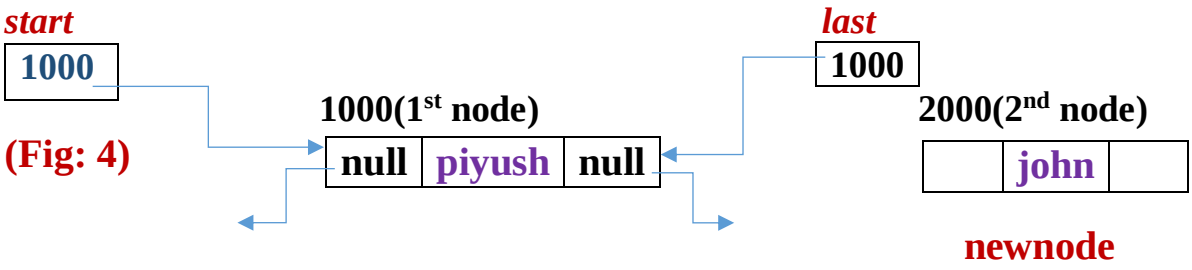
Fig: 2 newnode

```
        start=last=newnode; /*Stores new nodes address in 'start' and 'last'*/
        newnode.nextlink=newnode.prevlink=null;
```



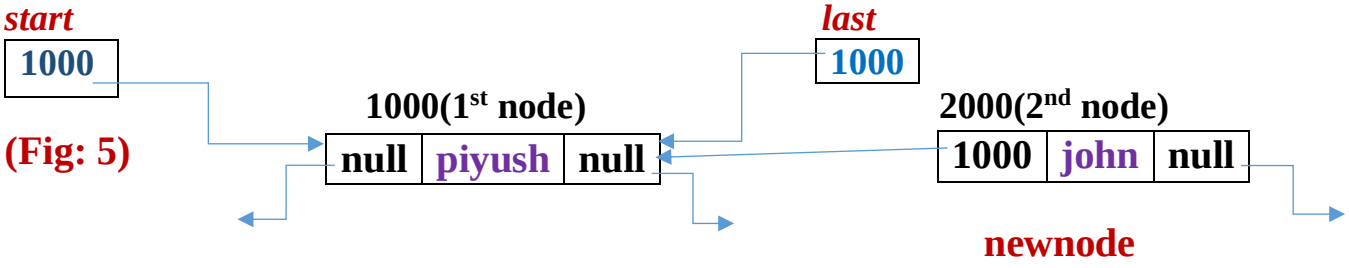
(Fig: 3)

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

while(ch=='y' || ch=='Y')
{

        newnode=new DLLNODE();
        System.out.println("Enter the info of next new node:");
        newnode.sname=sc.next();
```



(Fig: 4)

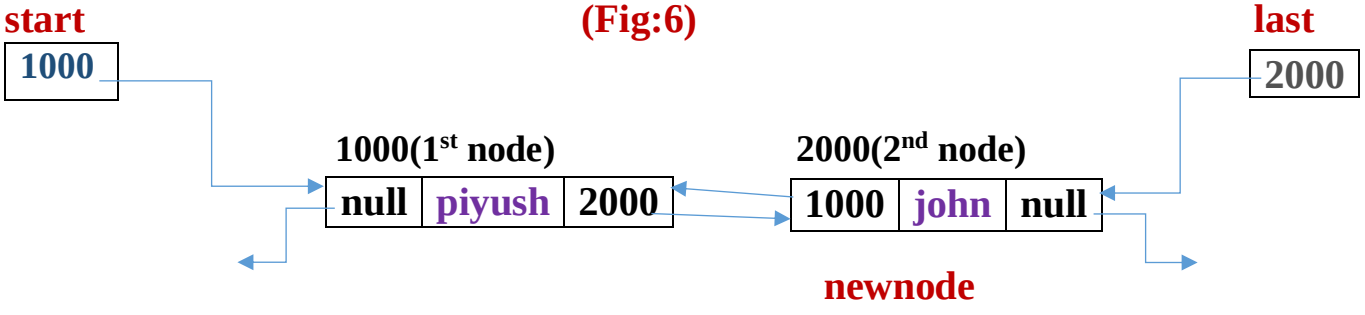
```
        newnode.nextlink=null;
        newnode.prevlink=last;
```



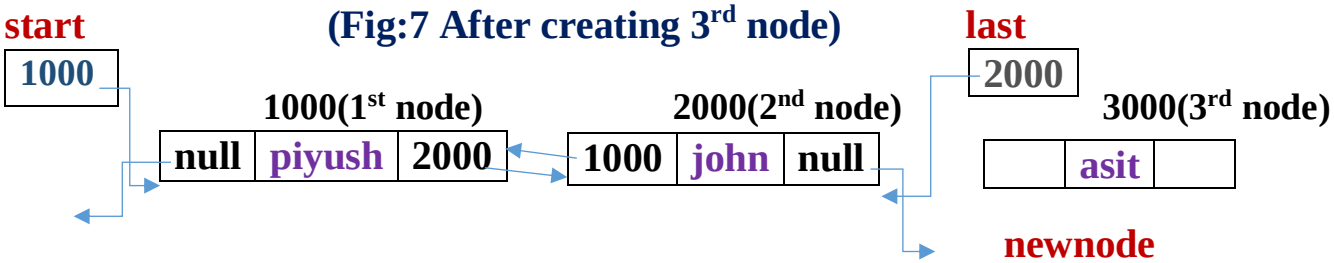
(Fig: 5)

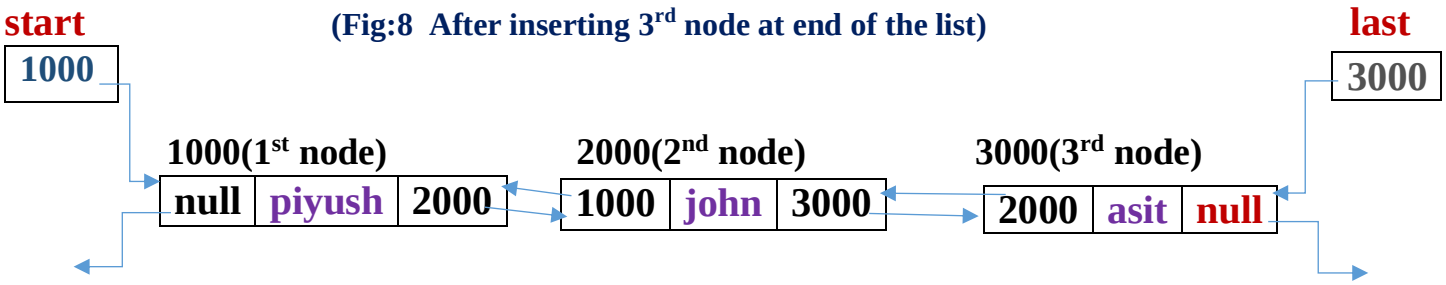
```
        last.nextlink=newnode;
        last=newnode;
```

(Fig:6)

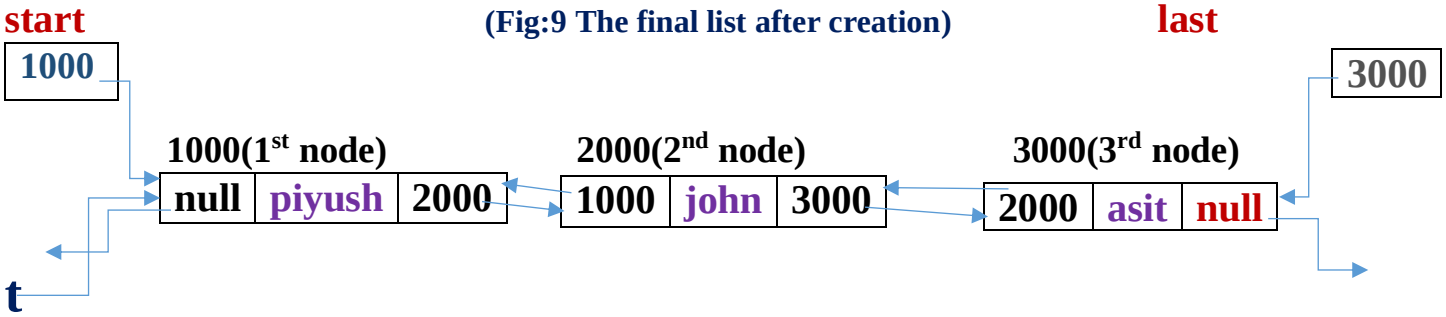


```
            System.out.println("Do you want to create another node(y/n)");
            ch=sc.next().charAt(0);
        }/*End of while…loopt*/
```





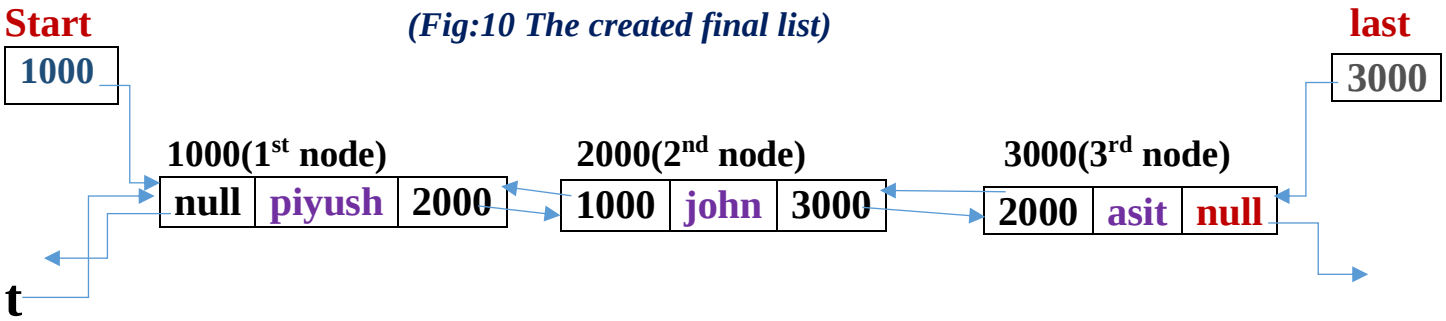

```
}/*End of create double linked list method*/

    public static void display_list() /*Traverse or display the nodes of  the list*/
    {
        if(start==null)
        {
            System.out.println("list is empty");
            return;
        }
        else
        {    System.out.println("\nthe DLL  in forward direction is….\n");
```



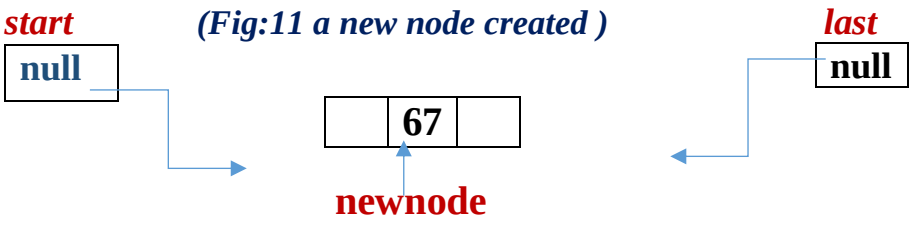

```
            DLLNODE t=start; /*t starts from 1st nodes*/
            while(t != null) /*← This loop visits each node in forward direction*/
            {
                System.out.print(t.sname + " → ");
                t=t.nextlink; /*← t moves to successor node*/
            }

            System.out.println("\nthe DLL in backward direction..\n");
```

```
                t=last; /*t starts from last node of the list*/
                while(t != null) /*←this loop visits each node in backward direction*/
                  {
                        System.out.print(t.sname + " → ");
                        t=t.prevlink; /*← t moves to predecessor node*/
                  }
                  System.out.println();
          }
    }/*END OF DISPLAY METHOD*/

    public static int count_nodes() /*← Counts the number of nodes in the list*/
    {     if(start==null)
          {
                return 0;
          }
          else
          {
```

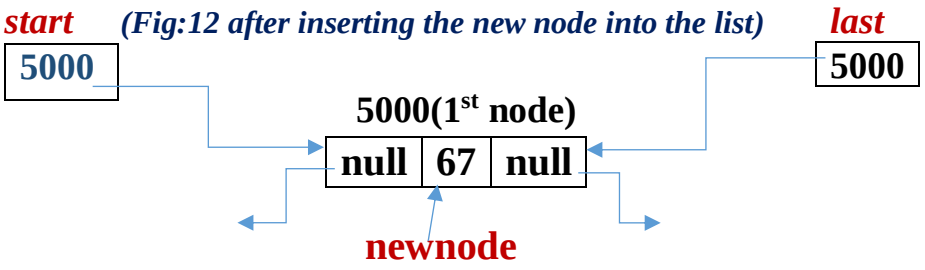
(Fig:10 The created final list)



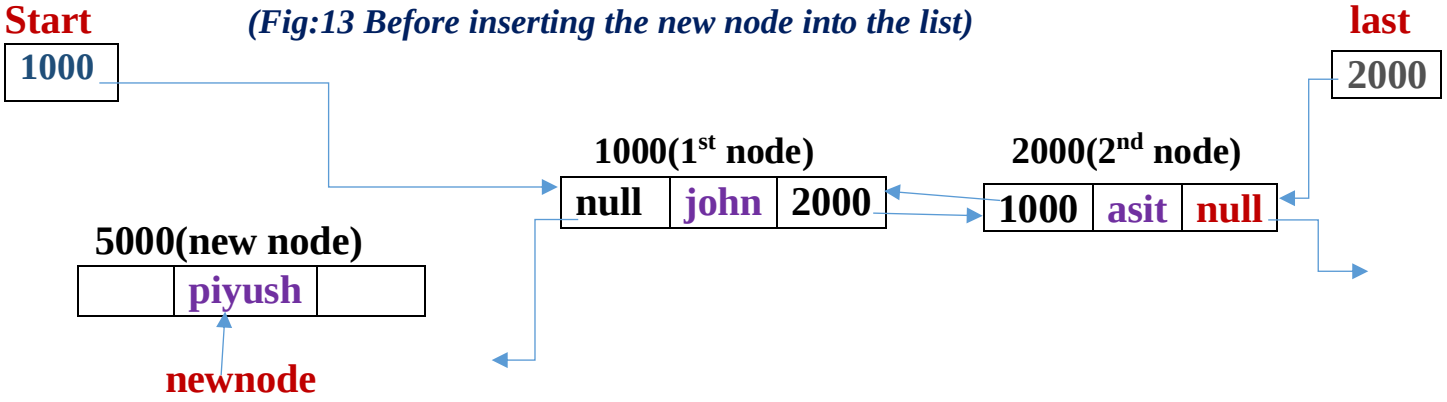

```
                int c=0; /*← initializes counter to 0, to count nodes*/
                DLLNODE t=start; /*← t starts from first node to count nodes*/
                while(t != null) /*← this loop counts the number of nodes*/
                {
                        c++; /*← increments c by 1 in each iteration of the loop*/
                        t=t.nextlink;
                }
            return c; /*← returns the final value of c i.e. the total no. of nodes*/
          }
    }/*END OF COUNT MEHTOD*/

/*Insert a new node at beginning or front end of the list method*/
    public static void insert_at_beg(String data)
    {
          DLLNODE newnode=new DLLNODE();
          newnode.sname=data;
```



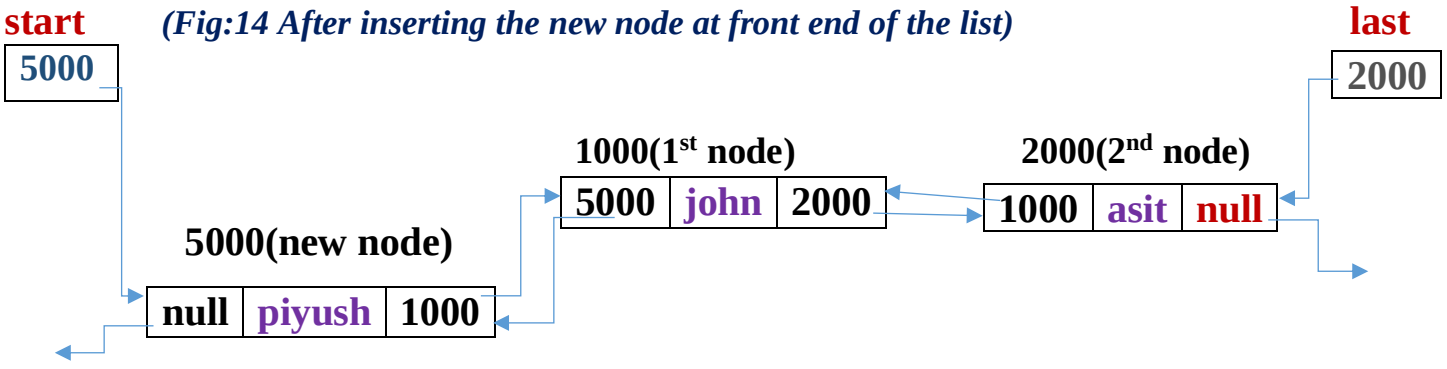

```
if ( start == null)
 {
    start=last=newnode;
    newnode.nextlink=newnode.prevlink=null;
```




```
 }
 else
 {
```

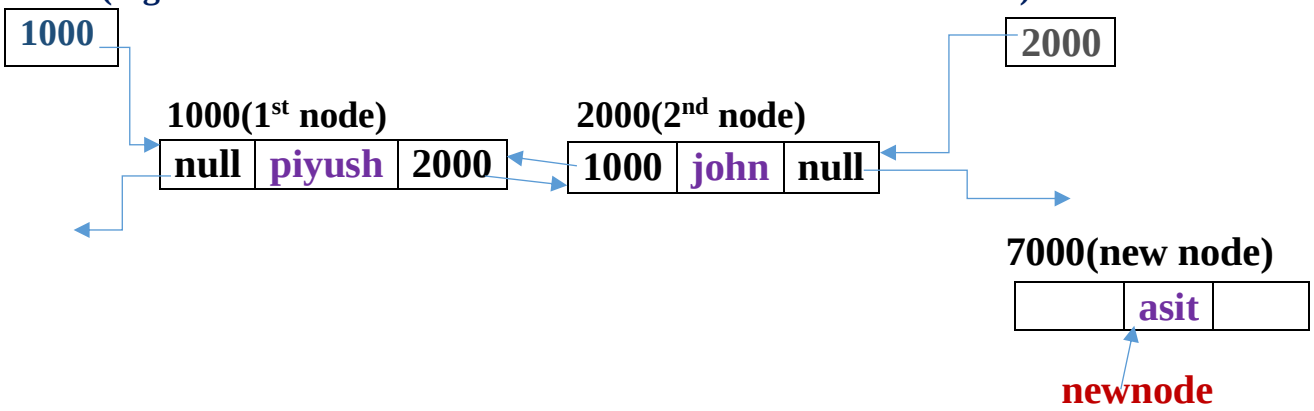



```
    newnode.nextlink=start;
    newnode.prevlink=null;
    start.prevlink=newnode;
    start=newnode;
```



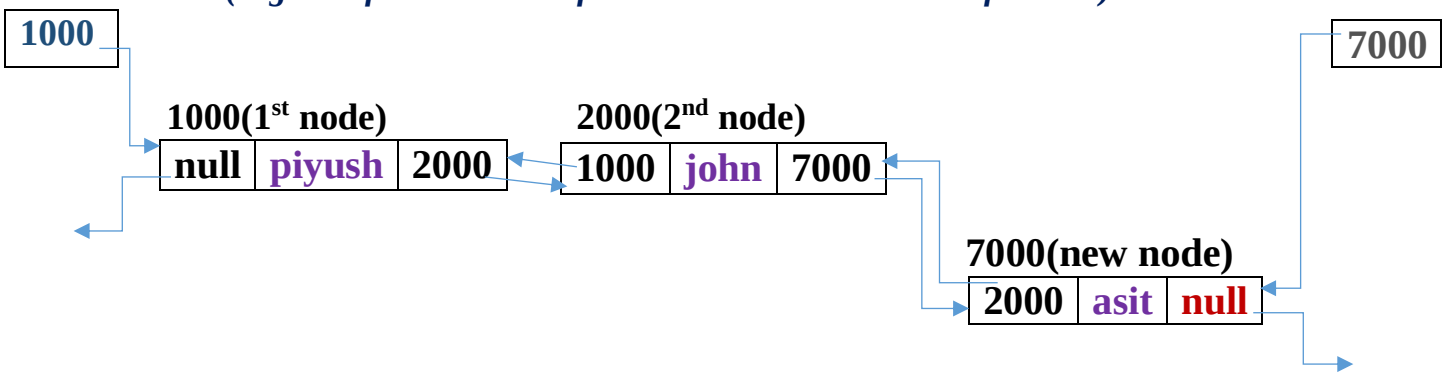
```
        }
    }/*END OF INSERT AT BEGINNING METHOD*/
    public static void insert_at_end(String data)
    {
        DLLNODE newnode=new DLLNODE();
        newnode.sname=data;

        if(start == null)
        {
            start=last=newnode;
            newnode.nextlink=newnode.prevlink=null;
        }
        else
        {
```

start (Fig:15 Before insertion of the new node at back end of the list) last




```
            last.nextlink=newnode;
            newnode.prevlink=last;
            newnode.nextlink=null;
            last=newnode;
```

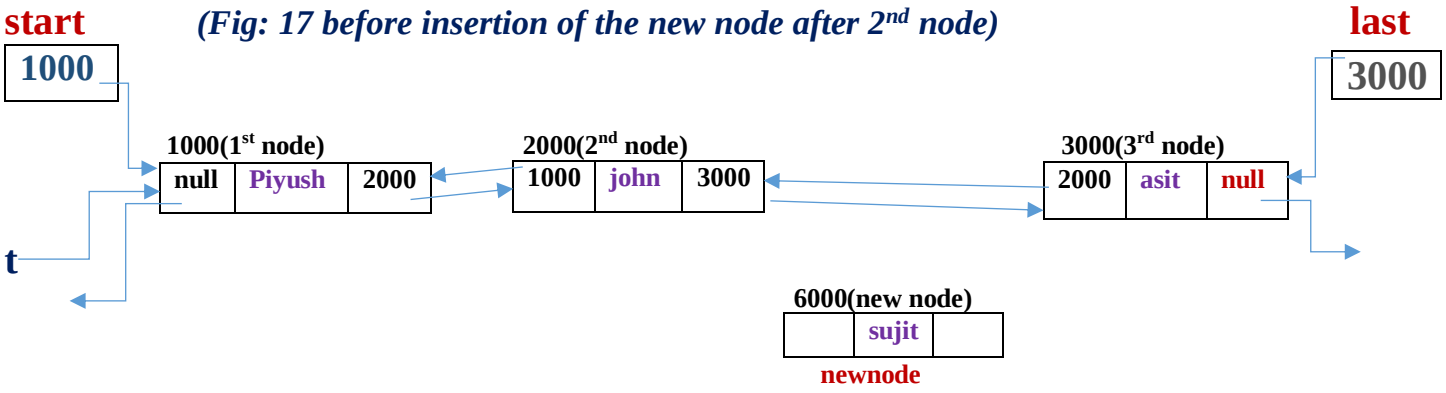
start *(Fig:16 After insertion of the new node at back end of the list)* last




```
        }

    }/*End of insert at back end method*/

/*Insert a new node at any position of the list*/
```

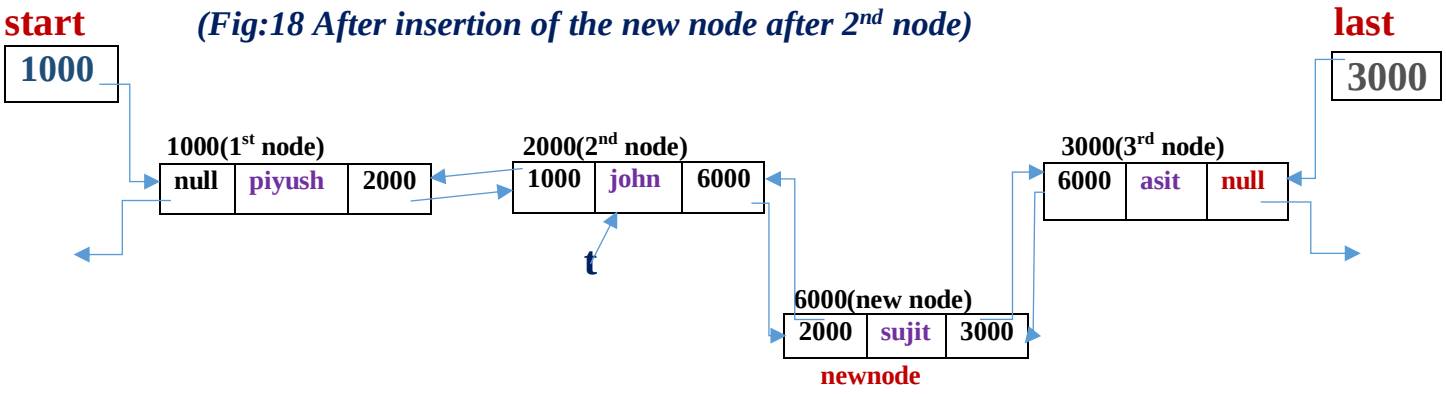
```
public static void insert_at_any_pos(String val  , String key)
{
    DLLNODE newnode=new DLLNODE();
    newnode.sname=val;

    if ( start==null)
    {
        System.out.println("the list is empty.");
        System.out.println("the node after which you want to"
                    + " insert is not present");

        return;
    }
    else
    {    // move to the specific node containing node_info after
        // which new node will be inserted

        DLLNODE temp=start;
        while(temp != null && temp.sname.equals(key)!= true )
        {
                temp=temp.nextlink;
        }

        if ( temp==null )
        {
                System.out.print("\nthe node is not present in the list\n");
                return;
        }
        else if ( last.sname.equals(key) == true)/* if the key is at last node of the list*/
        {
                insert_at_end(key); /*← Same case as insert at back end*/
        }
        else
        {
```

(Fig: 17 before insertion of the new node after 2nd node)

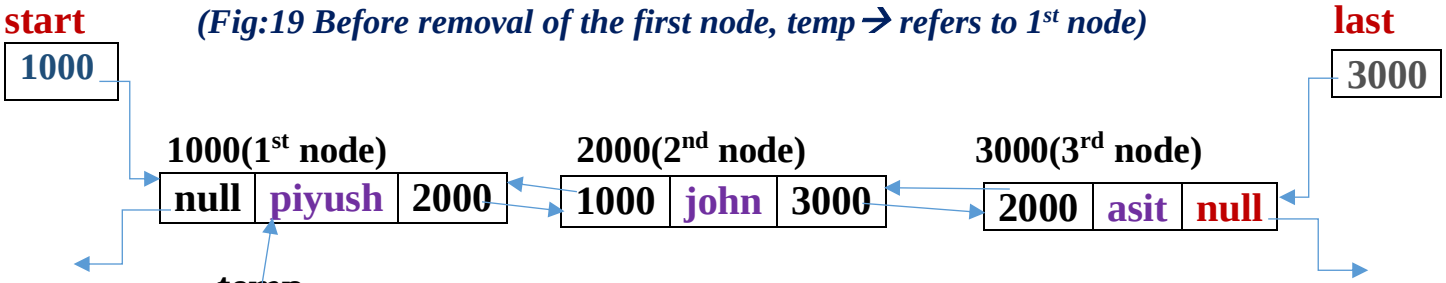



```
            newnode.nextlink=temp.nextlink;

            temp.nextlink.prevlink=newnode;
```

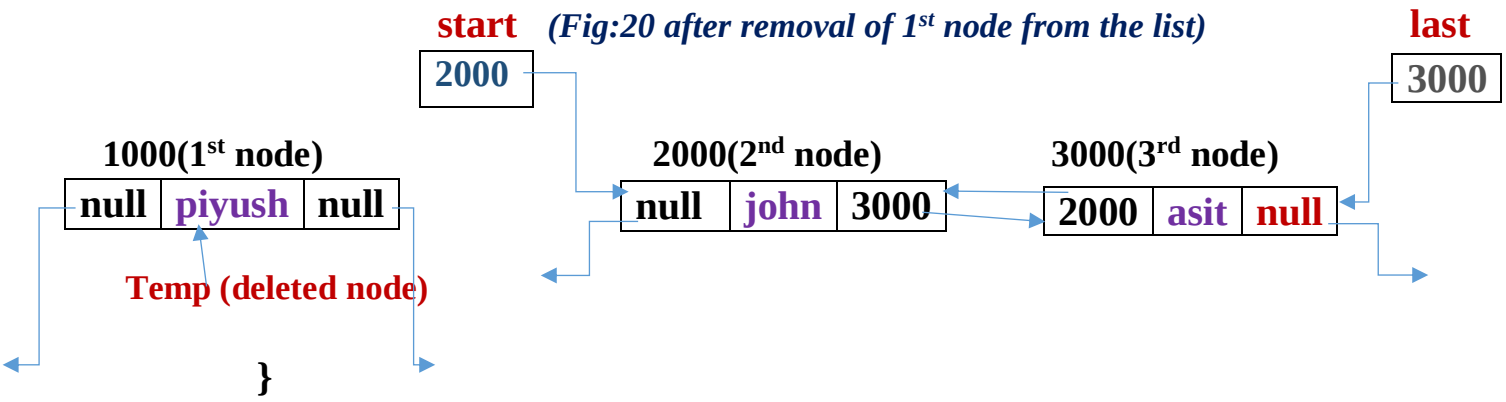
```
                newnode.prevlink=temp;
                temp.nextlink=newnode;
```



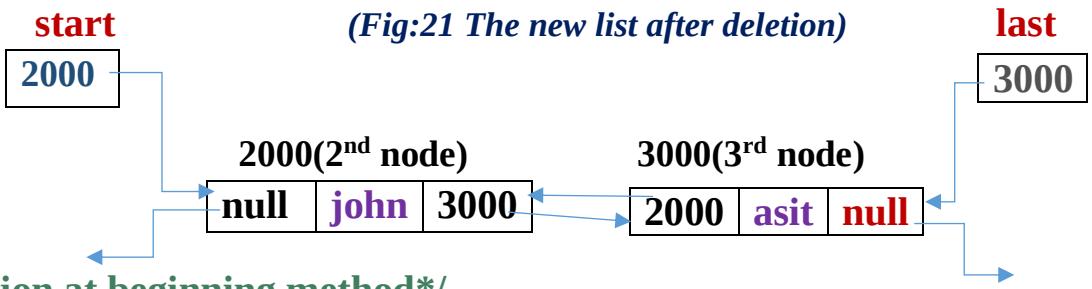

```
            }
        }

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

/*Delete the first node method*/
    public static void delete_at_beg()
    {
        if(start == null)
        {
            System.out.println("list is empty ....");
            return;
        }

        DLLNODE temp=start;

        if ( start.nextlink== null)
        {
            start=last=null;
        }
        else
        {
```



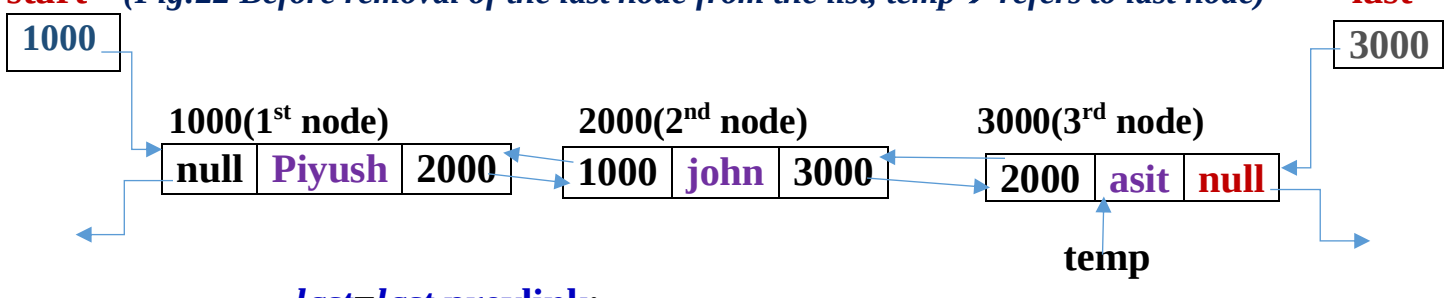

```
            start=start.nextlink;
            start.prevlink=null;
            temp.nextlink=null;
```



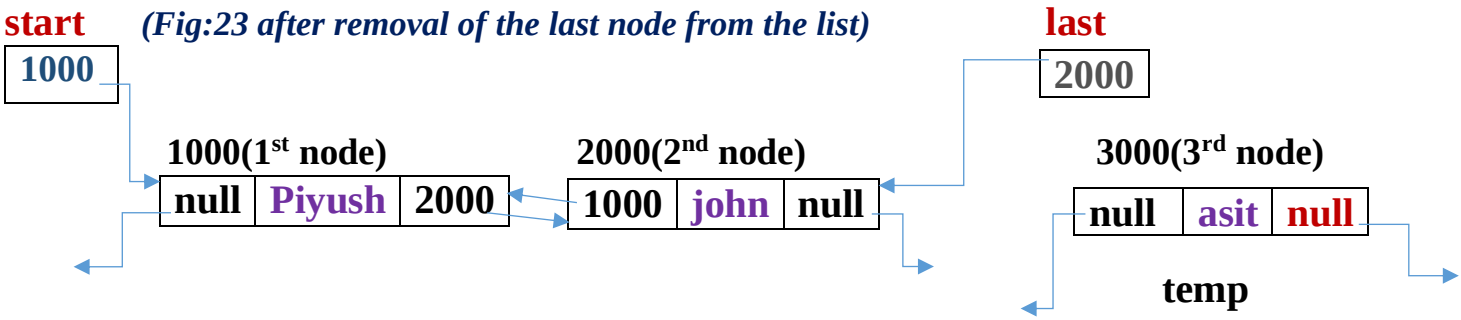

```
        }

        System.out.println("deleted node is:" + temp.sname); /* prints piyush*/
```



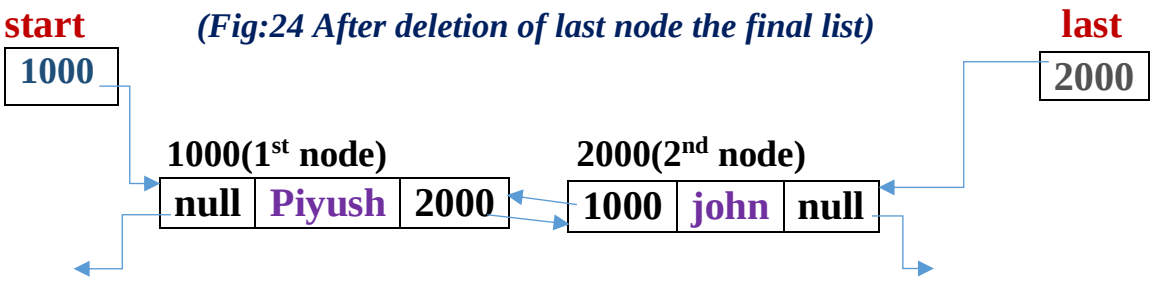

```
    }/*End of deletion at beginning method*/

/*Deletion at back end i.e. deleting the last node of the list*/
    public static void delete_at_back_end()
    {
        if ( start == null)
        {
            System.out.println("the list is empty");
            return;
        }

        DLLNODE temp=last; /*temp refers to last node of the list*/
        if( start.nextlink == null ) /*← if the list contains only one node*/
        {
            start=last=null; /*← the list becomes empty after deletion*/
        }
        else
        {
```




```
            last=last.prevlink;
            last.nextlink=null;
            temp.prevlink=null;
```



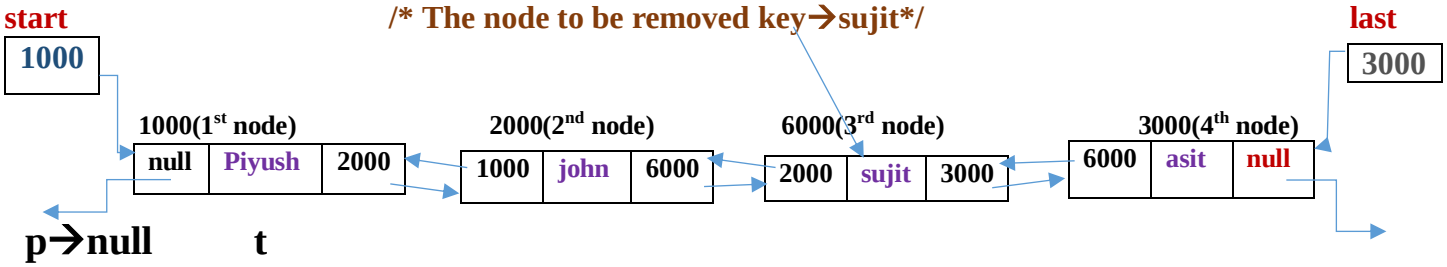

```
        }

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




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

/*Deletion of a node at any position of the list*/
    public static void delete_at_any_pos(String key)
    {
        if ( start==null )
        {
            System.out.println("the list is empty");
            return ;
        }

        if( start.sname.equals(key) == true)/*if the key is at first node*/
        {
            delete_at_beg(); /*←same as deletion at front end case i.e. 1st node deletion*/
        }
        else if( last.sname.equals(key) == true)/*if the key is at last node*/
        {
           delete_at_back_end(); /*←same case as deletion at back end*/
        }
        Else
         {    /* if the key is at any other node other than 1st and last node*/
            DLLNODE p=null;
            DLLNODE t=start;
/*move from the first node until you don't get the key node and reach at end of the list*/
            while( t != null && t.sname.equals(key) != true )
            {       p=t;      /*← stores the current value of  t  */
                    t = t.nextlink;   /*← move t  to the next node*/
```

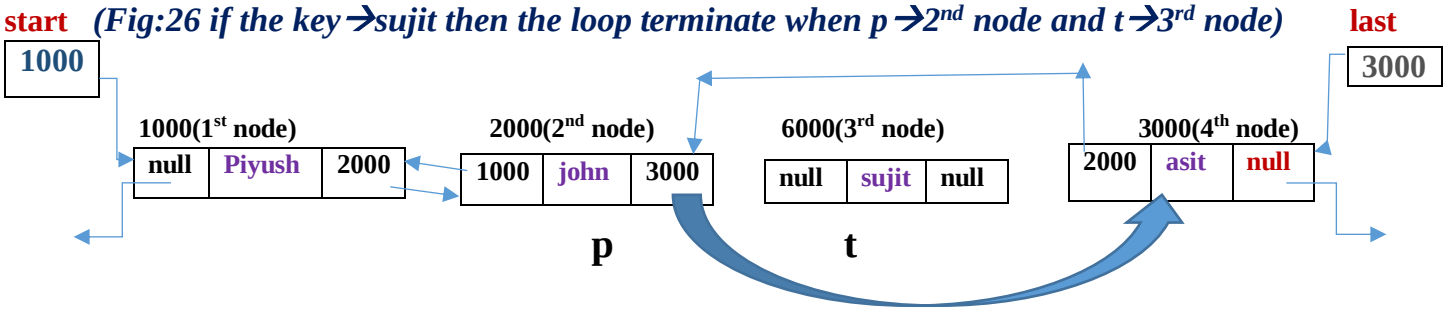
```
        }/*the loop terminates if either we reach at end of the list or we get the key
node*/
        if ( t == null )    /*←if key is not present then we reach at end of the list*/
        {
              System.out.println("The node to be deleted  is not present");
              return;
        }
        else
        {
```

(Fig:25 if the key→sujit then the loop terminate when p→2nd node and t→3rd node)



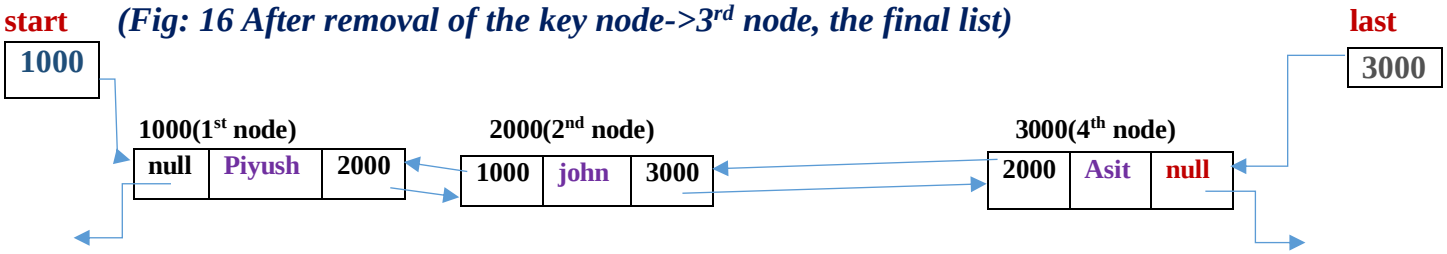

```
              p.nextlink=t.nextlink;
              t.nextlink.prevlink=p;
             t.nextlink=t.prevlink=null;
```

(Fig:26 if the key→sujit then the loop terminate when p→2nd node and t→3rd node)



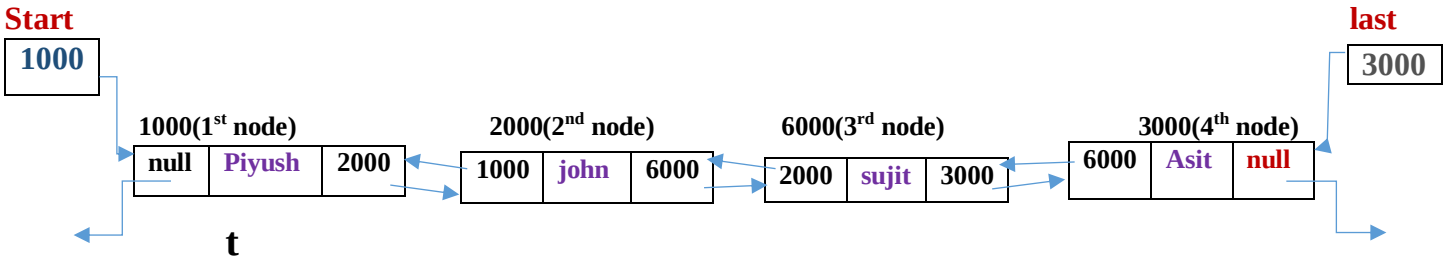

```
        }
        System.out.println("The deleted node is:" + t.sname);
    }
```

(Fig: 16 After removal of the key node->3rd node, the final list)

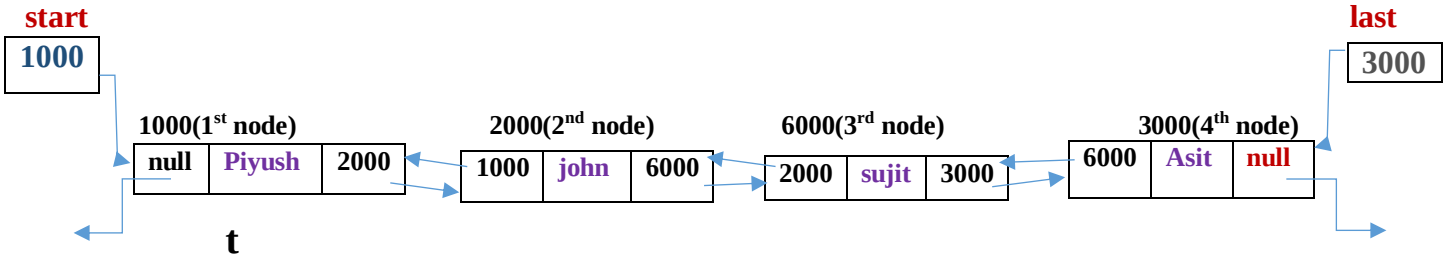



```
}/*End of deletion at any position of the list*/

/*Searching a key node from a doubly lined list using linear search algorithm*/
    public static void linear_search(String key) /*key is the name to be searched*/
    {
       if(start==null)
       {
           System.out.println("the  list is empty");
           return;
       }
```

```
        else
        {   (Fig: 28 suppose key→sujit which we want to search)
```

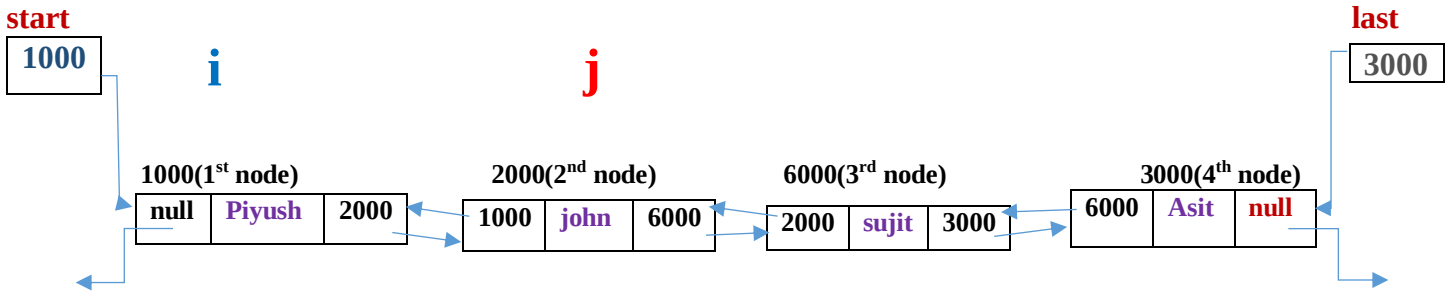



```
            boolean flag=false; /*set flag→false to indicate we have not found the key node*/
            DLLNODE t=start; /*← t starts from 1st node*/
            while( t!=null ) /*← while…loop continues until end of the list is not reached*/
            {
               if( t.sname.equals(key) == true) /*←if the key is found at current node t*/
               {
                  flag=true; /*←set flag→true to indicate search is successful*/
                  break;   /*←key node found, so immediately terminate the loop*/
               }
               t= t.nextlink; /*← if t is not the key node, then move to next node*/
            } /* End of while…loop*/
            if(flag==true) /*←After the loop if flag→true means we have found the key node*/
              System.out.println("the key node is present in the list");
            else
             System.out.println("the key node is not present in the list");
        }
    }/*End of linear search method*/

/*Update the information of a key node by a new value if present in the list*/
/*key is the value of the node which we want to update , new_val is new value of the node */

        public static void update_node(String key , String new_val)
        {
            if(start==null)
            {
               System.out.println("the  list is empty");
               return;
             }
             else
             {                  (Fig:29 The original list we have created)
```




```
/*Search the key node using linear search, if present then update it by new_val*/
                    boolean flag=false; /*set flag→false to assume key is not present*/
```

```
                DLLNODE t=start; /*→start from 1st node using ref. variable t*/
            while( t!=null ) /*move from one to another node until end of the list is not reached*/
                {
                        if( t.sname.equals(key)== true) /*←if key is found at node t*/
                        {
                            t.sname=new_val; /*←assign the new_val to info part of node t*/
                            flag=true; /*set flag→true to indicate successful update*/
                            break; /*←immediately terminate the loop*/
                        }
                        t= t.nextlink; /*←if  t  is not the key node then move to next node*/
                 }
                if(flag==true)
                        System.out.println("the key node got updated");
                else
                        System.out.println("the node is not present");
            }
        }/*End of Update node method*/

/*sorting a doubly lined list using bubble sort algorithm*/
        public static void sort_list()
        {
            if(start==null)
            {
                 System.out.println("the  list is empty");
                return;
            }
            else
            {       for(DLLNODE i=start ; i.nextlink != null ; i=i.nextlink)
                    {
                        for(DLLNODE j=i.nextlink; j!=null; j=j.nextlink )
                        {
                            if( j.sname.charAt(0) < i.sname.charAt(0))
                            {
                                String temp=j.sname;
                                j.sname=i.sname;
                                i.sname=temp;
                            }
                         }/*End of inner for…loop */
                     }/*End of outer for…loop*/
```



(Fig: 30 The original list we have create before sorting)

```
            }/*end of  else clause*/

        }/*END OF SORTING METHOD*/


} /*END OF DOUBLY_LL_DEMO CLASS*/
```