Task-1:
Create a double link list and perform the mentioned tasks.
i. Insert a new node at the end of the list.
ii. Insert a new node at the beginning of the list.
iii. Insert a new node at a given position.
iv. Delete any node.
v. Print the complete double link list.

```
#include<iostream>
using namespace std;
class node{
	public:
		int data;
		node* next;
		node* prev;
		node(){
			data=0;
			prev=next=NULL;
		}
};
class DLL{
	public:
		node *head=NULL;
		node *tail=NULL;
		node *last;
		//insert at end
		void insertatend(int d){
			node* newnode=new node;
			newnode->data=d;
			newnode->next=NULL;
			if(head==NULL){
				newnode->prev=NULL;
				head=newnode;
				tail=newnode;
			}
			else{
				node* temp=head;
				while(temp->next!=NULL){
					temp=temp->next;
				}
				newnode->prev=tail;
				tail->next=newnode;
				tail=newnode;
			}
		}
		//insert at head
		void insertatfront(int d){
			node* temp=head;
```

```
            node* newnode=new node();
            newnode->data=d;
            newnode->next=NULL;
            if(head==NULL){
                  head=newnode;
                  newnode->next=NULL;
                  newnode->prev=NULL;
                  tail=newnode;
            }
            else{
                  newnode->next=temp;
                  temp->prev=newnode;
                  newnode->prev=NULL;
                  head=newnode;
            }
      }
      //insert at any position
      void insertatanypos(int d,int k){
            node* temp=head;
            node* temp2;
            int count=1;
            node* newnode=new node();
            newnode->data=d;
            newnode->next=NULL;
            if(head==NULL){
                  cout<<"Linked list is empty.\n";
            }
            else{
                  while(temp->next!=NULL){
                        if(count==k-1){
                              temp2=temp->next;
                              temp->next=newnode;
                              newnode->prev=temp;
                              newnode->next=temp2;
                              temp2->prev=newnode;
                        }
                        count++;
                        temp=temp->next;
                  }
            }
      }
  node* nodeexists(int k){
node* ptr=NULL;
node* temp = head;

while (temp != NULL) {
  if (temp->data== k) {
    ptr =temp;
  }
```

```
    temp=temp->next;
  }
  return ptr;
}
    void delatany(int k) {
  node* temp=nodeexists(k);
  if (temp==NULL) {
    cout <<"No node exists with key value:\n" <<k<< endl;
      }
      else {
    if (head->data==k){
      head=head->next;
      cout << "Node UNLINKED with keys value :\n" << k << endl;
          }
          else {
      node* nextNode = temp -> next;
      node* prevNode = temp -> prev;
      // deleting at the end
      if (nextNode == NULL) {

        prevNode -> next = NULL;
        cout << "Node Deleted at the END" << endl;
      }
      //deleting in between
      else {
        prevNode -> next = nextNode;
        nextNode -> prev = prevNode;
        cout << "Node Deleted in Between" << endl;
      }
    }
  }
    }
          void display(){
                node *temp;
                temp=head;
                while(temp!=NULL){
                      cout<<temp->data<<" ";
                      last=temp;
                      temp=temp->next;
                }
                cout<<endl;
          }
          //reverse display
          void reversedisplay(){
                while(last!=NULL){
                      cout<<last->data<<" ";
                      last=last->prev;
                }
                cout<<endl;
```

```
            }
};
int main(){
      DLL obj;
      cout<<"Insertion at end:\n";
      obj.insertatend(2);
      obj.insertatend(4);
//    obj1.insertatend(6);
//    obj1.insertatend(8);
//    obj1.insertatend(10);
//    cout<<"List L\n";
      obj.display();
      cout<<endl;
      cout<<"Insertion at begining:\n";
      obj.insertatfront(8);
      obj.insertatfront(9);
      obj.display();
      cout<<endl;
      cout<<"Insertion at given position:\n";
      obj.insertatanypos(7,2);
      obj.display();
      cout<<endl;
      cout<<"Deleting by value--";
      obj.delatany(8);
      obj.display();
      cout<<endl;
      cout<<"Printing list:\n";
      obj.display();
}
```

Task-2:
Create two doubly link lists, say L and M . List L should be containing all even elements from 2 to 10, and list
M should contain all odd elements from 1 to 9. Create a new list N by concatenating lists L and M.

```
#include<iostream>
using namespace std;
class node{
        public:
                int data;
                node* next;
                node* prev;
                node(){
                        data=0;
                        prev=next=NULL;
```

```
		}
};
class DLL{
	public:
		node *head=NULL;
		node *tail=NULL;
		node *last;
		//insert at end
		void insertatend(int d){
			node* newnode=new node;
			newnode->data=d;
			newnode->next=NULL;
			if(head==NULL){
				newnode->prev=NULL;
				head=newnode;
				tail=newnode;
			}
			else{
				node* temp=head;
				while(temp->next!=NULL){
					temp=temp->next;
				}
				newnode->prev=tail;
				tail->next=newnode;
				tail=newnode;
			}
		}
		//insert at head
		void insertatfront(int d){
			node* temp=head;
			node* newnode=new node();
			newnode->data=d;
			newnode->next=NULL;
			if(head==NULL){
				head=newnode;
				newnode->next=NULL;
				newnode->prev=NULL;
				tail=newnode;
			}
			else{
				newnode->next=temp;
				temp->prev=newnode;
				newnode->prev=NULL;
				head=newnode;
```

```
                }
        }
        //insert at any position
        void insertatanypos(int d,int k){
                node* temp=head;
                node* temp2;
                int count=1;
                node* newnode=new node();
                newnode->data=d;
                newnode->next=NULL;
                if(head==NULL){
                        cout<<"Linked list is empty.\n";
                }
                else{
                        while(temp->next!=NULL){
                                if(count==k-1){
                                        temp2=temp->next;
                                        temp->next=newnode;
                                        newnode->prev=temp;
                                        newnode->next=temp2;
                                        temp2->prev=newnode;
                                }
                                count++;
                                temp=temp->next;
                        }
                }
        }
    node* nodeexists(int k){
  node* ptr=NULL;
  node* temp = head;

  while (temp != NULL) {
    if (temp->data== k) {
      ptr =temp;
    }
    temp=temp->next;
  }
  return ptr;
}
    void delatany(int k) {
  node* temp=nodeexists(k);
  if (temp==NULL) {
    cout <<"No node exists with key value:\n" <<k<< endl;
          }
```

```
        else {
  if (head->data==k){
    head=head->next;
    cout << "Node UNLINKED with keys value :\n" << k << endl;
                }
                else {
    node* nextNode = temp -> next;
    node* prevNode = temp -> prev;
    // deleting at the end
    if (nextNode == NULL) {

      prevNode -> next = NULL;
      cout << "Node Deleted at the END" << endl;
    }
    //deleting in between
    else {
      prevNode -> next = nextNode;
      nextNode -> prev = prevNode;
      cout << "Node Deleted in Between" << endl;
    }
  }
}
      }
      //function to get head
            node* gethead(){
            return head;
      }
      //function to reverse lists
      node* reverseList(node* head){
if (head->next == NULL)
   return head;
node* rest = reverseList(head->next);
head->next->next = head;
head->next = NULL;
return rest;
      }
      //functions to merge two lists
      node* concatenate(node* a,node* b){
a = reverseList(a);
b = reverseList(b);
node* head = NULL;
node* temp;
while (a!=NULL && b!= NULL) {
   if (a->data>=b->data) {
```

```
        temp = a->next;
        a->next = head;
        head = a;
        a = temp;
    }
    else {
        temp=b->next;
        b->next=head;
        head=b;
        b=temp;
    }
}
while (a!=NULL) {
    temp = a->next;
    a->next = head;
    head = a;
    a = temp;
}
while (b != NULL) {
    temp = b->next;
    b->next = head;
    head = b;
    b = temp;
}
temp=head;
node* last1;
        while(temp!=NULL){
                cout<<temp->data<<" ";
                last1=temp;
                temp=temp->next;
        }
        cout<<endl;
        cout<<"In descending order:\n";
        while(last1!=head){
                                cout<<last1->data<<" ";
                                last1=last1->prev;
                        }
        cout<<endl;

         return head;
}


                void display(){
```

```
                node *temp;
                temp=head;
                while(temp!=NULL){
                        cout<<temp->data<<" ";
                        last=temp;
                        temp=temp->next;
                }
                cout<<endl;
        }
        //reverse display
        void reversedisplay(){
                while(last!=NULL){
                        cout<<last->data<<" ";
                        last=last->prev;
                }
                cout<<endl;
        }
};
int main(){
        DLL obj1;
        DLL obj2;
        obj1.insertatend(2);
        obj1.insertatend(4);
        obj1.insertatend(6);
        obj1.insertatend(8);
        obj1.insertatend(10);
        cout<<"List L\n";
        obj1.display();
        obj2.insertatend(1);
        obj2.insertatend(3);
        obj2.insertatend(5);
        obj2.insertatend(7);
        obj2.insertatend(9);
        cout<<"List M\n";
        obj2.display();
        obj1.gethead();
        obj2.gethead();
        DLL obj3;
        cout<<"Merging L and M with sorting:\n";
        obj3.concatenate(obj1.gethead(),obj2.gethead());

        //obj.reversedisplay();
//      obj.insertatfront(8);
//      obj.insertatfront(9);
```

```
//	obj.display();
//	obj.insertatanypos(7,2);
//	obj.display();
//	obj.delatany(2);
//	obj.display();

}
```

Task-3:
Using the above-created list N, sort the contents of list N is descending order.

```
#include<iostream>
using namespace std;
class node{
	public:
		int data;
		node* next;
		node* prev;
		node(){
			data=0;
			prev=next=NULL;
		}
};
class DLL{
	public:
		node *head=NULL;
		node *tail=NULL;
		node *last;
		//insert at end
		void insertatend(int d){
			node* newnode=new node;
			newnode->data=d;
			newnode->next=NULL;
			if(head==NULL){
				newnode->prev=NULL;
				head=newnode;
				tail=newnode;
			}
			else{
				node* temp=head;
				while(temp->next!=NULL){
					temp=temp->next;
```

```
                }
                newnode->prev=tail;
                tail->next=newnode;
                tail=newnode;
        }
}
//insert at head
void insertatfront(int d){
        node* temp=head;
        node* newnode=new node();
        newnode->data=d;
        newnode->next=NULL;
        if(head==NULL){
                head=newnode;
                newnode->next=NULL;
                newnode->prev=NULL;
                tail=newnode;
        }
        else{
                newnode->next=temp;
                temp->prev=newnode;
                newnode->prev=NULL;
                head=newnode;
        }
}
//insert at any position
void insertatanypos(int d,int k){
        node* temp=head;
        node* temp2;
        int count=1;
        node* newnode=new node();
        newnode->data=d;
        newnode->next=NULL;
        if(head==NULL){
                cout<<"Linked list is empty.\n";
        }
        else{
                while(temp->next!=NULL){
                        if(count==k-1){
                                temp2=temp->next;
                                temp->next=newnode;
                                newnode->prev=temp;
                                newnode->next=temp2;
                                temp2->prev=newnode;
```

```
                                        }
                                        count++;
                                        temp=temp->next;
                                    }
                                }
                            }
        node* nodeexists(int k){
  node* ptr=NULL;
  node* temp = head;

  while (temp != NULL) {
    if (temp->data== k) {
      ptr =temp;
    }
    temp=temp->next;
  }
  return ptr;
}
        void delatany(int k) {
  node* temp=nodeexists(k);
  if (temp==NULL) {
    cout <<"No node exists with key value:\n" <<k<< endl;
          }
          else {
    if (head->data==k){
      head=head->next;
      cout << "Node UNLINKED with keys value :\n" << k << endl;
                }
                else {
      node* nextNode = temp -> next;
      node* prevNode = temp -> prev;
      // deleting at the end
      if (nextNode == NULL) {

        prevNode -> next = NULL;
        cout << "Node Deleted at the END" << endl;
      }
      //deleting in between
      else {
        prevNode -> next = nextNode;
        nextNode -> prev = prevNode;
        cout << "Node Deleted in Between" << endl;
      }
    }
```

```
    }
        }
        //function to get head
                node* gethead(){
                return head;
        }
        //function to reverse lists
        node* reverseList(node* head){
    if (head->next == NULL)
       return head;
    node* rest = reverseList(head->next);
    head->next->next = head;
    head->next = NULL;
    return rest;
        }
        //functions to merge two lists
        node* concatenate(node* a,node* b){
//    a = reverseList(a);
//    b = reverseList(b);
    node* res = NULL;

    if (a==NULL && b==NULL) return NULL;
    while (a!=NULL && b!= NULL) {
       if (a->data<=b->data) {
          node *temp = a->next;
          a->next = res;
          res = a;
          a = temp;
       }
       else {
          node *temp = b->next;
          b->next = res;
          res = b;
          b = temp;
       }
    }
    while (a!=NULL) {
       node *temp = a->next;
       a->next = res;
       res = a;
       a = temp;
    }
    while (b!=NULL) {
    node *temp = b->next;
```

```
        b->next = res;
        res = b;
        b = temp;
}
    node* temp=res;
          while(temp!=NULL){
                    cout<<temp->data<<" ";
                    temp=temp->next;
          }
                    cout<<endl;
          return res;
   }
//   void printList(node* node){
//    while (node!=NULL)
//    {
//        cout <<node->data<< " ";
//        node=node->next;
//    }
//}


                    void display(){
                              node *temp;
                              temp=head;
                              while(temp!=NULL){
                                        cout<<temp->data<<" ";
                                        last=temp;
                                        temp=temp->next;
                              }
                              cout<<endl;
                    }
                    //reverse display
                    void reversedisplay(){
                              while(last!=NULL){
                                        cout<<last->data<<" ";
                                        last=last->prev;
                              }
                              cout<<endl;
                    }
};
int main(){
          DLL obj1;
          DLL obj2;
          obj1.insertatend(2);
```

```
        obj1.insertatend(4);
        obj1.insertatend(6);
        obj1.insertatend(8);
        obj1.insertatend(10);
        cout<<"List L\n";
        obj1.display();
        obj2.insertatend(1);
        obj2.insertatend(3);
        obj2.insertatend(5);
        obj2.insertatend(7);
        obj2.insertatend(9);
        cout<<"List M\n";
        obj2.display();
        obj1.gethead();
        obj2.gethead();
        DLL obj3;
        cout<<"Merging L and M with sorting:\n";
        obj3.concatenate(obj1.gethead(),obj2.gethead());
        //printList(res);
        //obj.reversedisplay();
//      obj.insertatfront(8);
//      obj.insertatfront(9);
//      obj.display();
//      obj.insertatanypos(7,2);
//      obj.display();
//      obj.delatany(2);
//      obj.display();

}
```

Task-4:
Create a circular link list and perform the mentioned tasks.

i. Insert a new node at the end of the list.
ii. Insert a new node at the beginning of the list.
iii. Insert a new node at a given position.
iv. Delete any node.
v. Print the complete circular link list.

```
#include <iostream>
using namespace std;
```

```
class node {
        public:

   int data;
   node* next;
   node* prev;
};

class cll {
  public:
   node* head;
  public:

   cll(){
     head = NULL;
   }
   void athead(int val)
   {

      node* newNode = new node();


     newNode->data = val;


     newNode->next = NULL;
     newNode->prev = NULL;


  if(head == NULL) {
   head = newNode;
   newNode->next = head;
   newNode->prev = head;
  } else {


   node* temp = head;
   while(temp->next != head)
     temp = temp->next;


   temp->next = newNode;
   newNode->prev = temp;
   newNode->next = head;
```

```
  head->prev = newNode;
  head = newNode;
 }
}
 //--------------------------
   void attail(int val)
   {

 node* newNode = new node();


 newNode->data = val;


 newNode->next = NULL;
 newNode->prev = NULL;


 if(head == NULL) {
  head = newNode;
  newNode->next = head;
  newNode->prev = head;
 } else {


  node* temp = head;
  while(temp->next != head)
    temp = temp->next;


  temp->next = newNode;
  newNode->next = head;
  newNode->prev = temp;
  head->prev = newNode;
 }
         }
         //-------------------------------------------
         void inserany(int pos,int ele)
  {

 node* newnode = new node();
 newnode->data = ele;
 newnode->next = NULL;
 newnode->prev = NULL;
```

```
  if (pos == 1) {


  newnode->next = head;
  head->prev = newnode;
  head = newnode;
} else {


  node* temp = head;
  for(int i = 1; i < pos-1; i++) {
    if(temp != NULL) {
      temp = temp->next;
    }
  }


  if(temp != NULL) {
    newnode->next = temp->next;
    newnode->prev = temp;
    temp->next = newnode;
    if(newnode->next != NULL)
      newnode->next->prev = newnode;
  } else {


    cout<<"\nThe previous node is null.";
  }
}
        }
        //-----------------------------------




  //display the content of the list
  void show() {
    node* temp = head;
    if(temp != NULL) {
```

```
    while(true) {
      cout<<temp->data<<" ";
      temp = temp->next;
      if(temp == head)
        break;
    }
    cout<<endl;
  } else {
    cout<<"The list is empty.\n";
  }
}
        //-------------------------------------

                //-------------------------------------

        //-------------------------------------
        void deldup(){
                node* a=head;
                node* temp=head->next;
                if(head->data==temp->data){
                        while(a->next!=head){
                                a->next=temp;
                                temp->prev=a;
                                head=temp;
                        }
                        temp=temp->next;
                        a=a->next;

                }
                else{
                        while(temp->next!=head){
                                node* b=temp->next;
                                if(temp->data==a->data){
                                        a->next=b;
                                        b->prev=a;
                                }
                                temp=temp->next;
                                a=a->next;
                        }
                }
        }
        void delany(int position){
        node*temp=new node();
        node*pre;
```

```
        node*curr=head;
        for(int i=1;i<position;i++)
        {
                pre=curr;
                curr=curr->next;
                curr->prev=pre;
                  }
                  temp=curr->next;
                  pre->next=temp;
                  temp->prev=pre;
                  curr->prev=NULL;
                  curr->next=NULL;
                  delete curr;
          }
        //-------------------------

};

// test the code
int main() {
        cll c;
        cout<<"at head :  ";
        c.athead(1);
        //c.athead(1);
        c.athead(3);
        c.athead(5);
        c.athead(7);
        c.athead(7);
        c.show();
        cout<<"at tail : ";
        c.attail(9);
        c.attail(9);
        //c.attail(11);
        c.show();
        cout<<"insert 4 at position 3 : ";
        c.inserany(3,4);
        c.show();
        cout<<"delete element at position 4 : ";
        c.delany(4);
        c.show();
//      cout<<"Removing duplication:\n";
//      c.deldup();
//      c.show();
```

}

Task-5: Break the above-created circular linked list into two halves.

Task-6:
Create a circular Double link list and perform the mentioned tasks.
i. Insert two new nodes at the end of the list with the same data.
ii. Insert two new nodes at the beginning of the list with the same data..
iii. Insert a new node at a given position.
iv. Delete any node.
v. Print the complete circular double link list.

```
#include <iostream>
using namespace std;


class node {
        public:

    int data;
    node* next;
    node* prev;
};

class cll {
  public:
    node* head;
  public:

    cll(){
      head = NULL;
    }
    void athead(int val)
    {

       node* newNode = new node();
```

```
    newNode->data = val;


    newNode->next = NULL;
    newNode->prev = NULL;


  if(head == NULL) {
    head = newNode;
    newNode->next = head;
    newNode->prev = head;
  } else {


    node* temp = head;
    while(temp->next != head)
      temp = temp->next;


    temp->next = newNode;
    newNode->prev = temp;
    newNode->next = head;
    head->prev = newNode;
    head = newNode;
  }
}
  //--------------------------
     void attail(int val)
     {

  node* newNode = new node();


  newNode->data = val;


  newNode->next = NULL;
  newNode->prev = NULL;


  if(head == NULL) {
    head = newNode;
    newNode->next = head;
    newNode->prev = head;
```

```
} else {


  node* temp = head;
  while(temp->next != head)
    temp = temp->next;


  temp->next = newNode;
  newNode->next = head;
  newNode->prev = temp;
  head->prev = newNode;
}
          }
          //-------------------------------------------
          void inserany(int pos,int ele)
  {

node* newnode = new node();
newnode->data = ele;
newnode->next = NULL;
newnode->prev = NULL;



  if (pos == 1) {


  newnode->next = head;
  head->prev = newnode;
  head = newnode;
} else {


  node* temp = head;
  for(int i = 1; i < pos-1; i++) {
    if(temp != NULL) {
      temp = temp->next;
    }
  }


  if(temp != NULL) {
    newnode->next = temp->next;
```

```
    newnode->prev = temp;
    temp->next = newnode;
    if(newnode->next != NULL)
      newnode->next->prev = newnode;
  } else {


    cout<<"\nThe previous node is null.";
  }
}
        }
        //-----------------------------------




  //display the content of the list
  void show() {
    node* temp = head;
    if(temp != NULL) {

      while(true) {
        cout<<temp->data<<" ";
        temp = temp->next;
        if(temp == head)
          break;
      }
      cout<<endl;
    } else {
      cout<<"The list is empty.\n";
    }
  }
        //-----------------------------------

                //---------------------------------------

        //---------------------------------------
        void deldup(){
                node* a=head;
                node* temp=head->next;
                if(head->data==temp->data){
                        while(a->next!=head){
                                a->next=temp;
```

```
                        temp->prev=a;
                        head=temp;
                    }
                    temp=temp->next;
                    a=a->next;

                }
                else{
                    while(temp->next!=head){
                        node* b=temp->next;
                        if(temp->data==a->data){
                            a->next=b;
                            b->prev=a;
                        }
                        temp=temp->next;
                        a=a->next;
                    }
                }
            }
            void delany(int position){
            node*temp=new node();
            node*pre;
            node*curr=head;
            for(int i=1;i<position;i++)
            {
                pre=curr;
                curr=curr->next;
                curr->prev=pre;
                  }
                  temp=curr->next;
                  pre->next=temp;
                  temp->prev=pre;
                  curr->prev=NULL;
                  curr->next=NULL;
                  delete curr;
              }
            //-------------------------

};

// test the code
int main() {
            cll c;
            cout<<"at head :  ";
```

```
        c.athead(1);
        //c.athead(1);
        c.athead(3);
        c.athead(5);
        c.athead(7);
//      c.athead(7);
        c.show();
        cout<<"at tail : ";
        c.attail(9);
        c.attail(9);
        //c.attail(11);
        c.show();
        cout<<"insert 4 at position 3 : ";
        c.inserany(3,4);
        c.show();
        cout<<"delete element at position 4 : ";
        c.delany(4);
        c.show();
        cout<<"Removing duplication:\n";
        c.deldup();


        c.show();

}
```

Task-7:
Remove duplicates from the above created Doubly Circular Linked list.

```
#include <iostream>
using namespace std;


class node {
        public:

   int data;
   node* next;
   node* prev;
};

class cll {
  public:
```

```
  node* head;
 public:

  cll(){
    head = NULL;
  }
  void athead(int val)
  {

     node* newNode = new node();


    newNode->data = val;


    newNode->next = NULL;
    newNode->prev = NULL;


 if(head == NULL) {
  head = newNode;
  newNode->next = head;
  newNode->prev = head;
 } else {


  node* temp = head;
  while(temp->next != head)
    temp = temp->next;


  temp->next = newNode;
  newNode->prev = temp;
  newNode->next = head;
  head->prev = newNode;
  head = newNode;
 }
}
 //---------------------------
    void attail(int val)
    {

 node* newNode = new node();
```

```
newNode->data = val;


newNode->next = NULL;
newNode->prev = NULL;


if(head == NULL) {
  head = newNode;
  newNode->next = head;
  newNode->prev = head;
} else {


  node* temp = head;
  while(temp->next != head)
    temp = temp->next;


  temp->next = newNode;
  newNode->next = head;
  newNode->prev = temp;
  head->prev = newNode;
}
          }
          //-------------------------------------------
          void inserany(int pos,int ele)
  {

node* newnode = new node();
newnode->data = ele;
newnode->next = NULL;
newnode->prev = NULL;



  if (pos == 1) {


  newnode->next = head;
  head->prev = newnode;
  head = newnode;
} else {
```

```
  node* temp = head;
  for(int i = 1; i < pos-1; i++) {
    if(temp != NULL) {
      temp = temp->next;
    }
  }


  if(temp != NULL) {
    newnode->next = temp->next;
    newnode->prev = temp;
    temp->next = newnode;
    if(newnode->next != NULL)
      newnode->next->prev = newnode;
  } else {


    cout<<"\nThe previous node is null.";
  }
}
        }
        //-----------------------------------




  //display the content of the list
  void show() {
    node* temp = head;
    if(temp != NULL) {

      while(true) {
        cout<<temp->data<<" ";
        temp = temp->next;
        if(temp == head)
          break;
      }
      cout<<endl;
    } else {
      cout<<"The list is empty.\n";
    }
```

```
}
	//-----------------------------------

		//---------------------------------------

	//---------------------------------------
	void deldup(){
		node* a=head;
		node* temp=head->next;
		if(head->data==temp->data){
			while(a->next!=head){
				a->next=temp;
				temp->prev=a;
				head=temp;
			}
			temp=temp->next;
			a=a->next;

		}
		else{
			while(temp->next!=head){
				node* b=temp->next;
				if(temp->data==a->data){
					a->next=b;
					b->prev=a;
				}
				temp=temp->next;
				a=a->next;
			}
		}
	}
	void delany(int position){
	node*temp=new node();
	node*pre;
	node*curr=head;
	for(int i=1;i<position;i++)
	{
		pre=curr;
		curr=curr->next;
		curr->prev=pre;
		  }
		  temp=curr->next;
		  pre->next=temp;
		  temp->prev=pre;
```

```
                curr->prev=NULL;
                curr->next=NULL;
                delete curr;
            }
        //-------------------------

};

// test the code
int main() {
        cll c;
        cout<<"at head :  ";
        c.athead(1);
//      c.athead(1);
        c.athead(3);
        c.athead(5);
        c.athead(7);
        c.show();
        cout<<"at tail : ";
        c.attail(9);
        c.attail(11);
        c.show();
        cout<<"insert 4 at position 3 : ";
        c.inserany(3,4);
        c.show();
        cout<<"delete element at position 4 : ";
        c.delany(4);
        c.show();
//      cout<<"Removing duplication:\n";
//      c.deldup();
//      c.show();

}
```