1.1 Write a C++ program to create two singly linked lists of the same structure and perform the following operations through a menu-driven program :

- **i. Merge and print**
- **ii. Append and print**

➔

```
#include<stdlib.h>
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<malloc.h>
#include<string.h>
#include<assert.h>
#include<ctype.h>
#include<iomanip.h>

struct Cinema
{
        char film[30];
        float boxoffcoll;
        struct Cinema *next;
};

typedef struct Cinema Cin;

class List
{
        private:
                Cin *head,*end;
                int count;
                void sortlist();
        public :
                List()
                {
                        count=0;
                        head=end=NULL;
                }
                        void createlist();
                        void mergelist(List &, List &);
                        void displaylist();
                        void appendlist(List);
};

void List::createlist()
{
        Cin *node;
        char ch;
        head=new Cin();
        assert(head);
```

```
    node=head;

    do
    {
        cout<<"\nFilm " <<++count<<" ?:";
        fflush(stdin);
        cin>>node->film;

        cout<<"\nBox Off. Coll "<<count<<" ?:";
        fflush(stdin);
        cin>>node->boxoffcoll;

        cout<<"\n\nMore film ? y/n:";
        fflush(stdin);
        cin>>ch;

        if(tolower(ch)=='y')
        {
            node->next=new Cin();
            assert(node->next);
            node=node->next;
        }
        else
        {
            node->next=NULL;
            end=node;
        }
    }while(tolower(ch)=='y');
}
void List::sortlist()
{
    Cin *in,*jn,*s;
    s=head;
    float rtemp;
    char ptemp[20];

    for(in=s;in;in=in->next)
    {
        for(jn=in->next;jn;jn=jn->next)
        {
            if(strcmp(in->film,jn->film)>0)
            {
                strcpy(ptemp,in->film);
                strcpy(in->film,jn->film);
                strcpy(jn->film,ptemp);

                rtemp=in->boxoffcoll;
                in->boxoffcoll=jn->boxoffcoll;
                jn->boxoffcoll=rtemp;

            }
        }
    }

}
void List::displaylist()
{
    Cin *node;
    node=head;
    while(node)
    {
        cout<<endl<<node->film<<"\t\t"<<node->boxoffcoll;
        node=node->next;
    }
}

void List::mergelist(List &l1,List &l2)
{
    l1.sortlist();
    l2.sortlist();
    Cin *n1,*n2,*n3,*pnew;
    n1=l1.head;
    n2=l2.head;
    n3=head;
    while(n1 && n2)
    {
        pnew=new Cin();
        assert(pnew);
        pnew->next=NULL;
        if(strcmp(n1->film,n2->film)<0)
        {
            strcpy(pnew->film,n1->film);
            pnew->boxoffcoll=n1->boxoffcoll;
            n1=n1->next;
        }
        else
        {
            strcpy(pnew->film,n2->film);
            pnew->boxoffcoll=n2->boxoffcoll;
            n2=n2->next;
        }
        if(n3!=NULL)
        {
            n3->next=pnew;
            n3=pnew;
        }
        else
        {
            head=pnew;
            n3=head;
        }
    }
    if(n1==NULL && n2!=NULL)
    {
        while(n2)
        {
            pnew=new Cin();
            assert(pnew);
            pnew->next=NULL;
            strcpy(pnew->film,n2->film);
            pnew->boxoffcoll=n2->boxoffcoll;
            n3->next=pnew;
            n3=pnew;
            n2=n2->next;
        }
    }
    else if(n2==NULL && n1!=NULL)
    {
        while(n1)
        {
            pnew=new Cin();
            assert(pnew);
            pnew->next=NULL;
            strcpy(pnew->film,n1->film);
            pnew->boxoffcoll=n1->boxoffcoll;
            n3->next=pnew;
            n3=pnew;
            n1=n1->next;
        }
    }
    else
    {
        end=pnew;
    }
}


void List::appendlist(List X)
{
    Cin *n;
    n=head;
    if(head==NULL)
    {
        head=X.head;
    }
    else
    {
        while(n->next)
        {
            n=n->next;
        }
        n->next=X.head;
    }
```

```
}



int main()
{
        List l1,l2,l3;
        int ch;
        clrscr();
        l1.createlist();
        l2.createlist();
        l3.mergelist(l1,l2);
        cout<<"\n\tPrint List 1 : \n";
        l1.displaylist();
        cout<<"\n\tPrint List 2 : \n";
        l2.displaylist();
        do
        {
                cout<<endl<<"1. Merging List \n2. Append List\n3. Exit"<<endl;
                cout<<"Enter your choice: ";
                cin>>ch;

                switch(ch)
                {
                        case 1:
                                cout<<endl<<"Merge List"<<endl;
                                cout<<"\n\tPrint the merged List 3 : \n";
                                l3.displaylist();
                                break;
                        case 2:
                                cout<<endl<<"Append List"<<endl;
                                l1.appendlist(l2);
                                l1.displaylist();
                                break;
                        case 3:
                                exit(0);
                        default:
                                cout<<"You entered wrong values";
                }


        }while((ch==1) || (ch==2));
        return 0;
}
```

1.1 Write a C++ program to create two singly linked lists of the same structure and perform the following operations through a menu-driven program :

iii. Find intersection and print

➔

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
  int data;
  struct node* next;
}SN;

class list
{
    private:
              //SN *head;
    public:

      void push(SN** head_ref, int new_data);


      int isPresent (SN *head, int data);
      SN *getIntersection(SN *head1, SN *head2);
      void printList(SN *node);
};


SN* list::getIntersection(SN *head1, SN *head2)
{
            SN *result=NULL;
           SN *t1=head1;

           while(t1!=NULL)
            {
                if(isPresent(head2,t1->data))
                push(&result, t1->data);
                t1=t1->next;
            }

           return result;
}

void list::push(SN** head_ref, int new_data)
{
           SN *new_node=new SN;
           new_node->data=new_data;
           //link the old list off the new node
           new_node->next=(*head_ref);
           //move the head to point the new node
           (*head_ref)=new_node;
}
```

```
void list::printList(SN *node)
{
         while(node!=NULL)
        {
          cout<<"\t"<<node->data;
          node=node->next;
        }
}

int list::isPresent(SN *head, int data)
{
         SN *t=head;
         while(t!=NULL)
         {
             if(t->data==data)
             return 1;
             t=t->next;

         }
 return 0;
}

void main()
{
// clrscr();
list l1;

 SN* head1=NULL;
 SN* head2=NULL;
 SN* intersecn=NULL;
int i,n,j;
 clrscr();
// int i,n,j;
 cout<<"\nEnter the no. of elements to be inserted in list1: ";
 cin>>n;
 for(i=0;i<n;i++)
 {
         cout<<"\n"<<i+1<<": ";
        cin>>j;
        if(!l1.isPresent(head1,j))
          l1.push(&head1,j);
        else
         {
           cout<<"\n enter another number which is not present in list:\n";
          cout<<i+1<<": ";
        cin>>j;

        l1.push(&head1,j);
       }

 }

/* push(&head1, 20);
 push(&head1,4);
 push(&head1,15);
 push(&head1,10);*/

 l1.push(&head2,10);
 l1.push(&head2,2);
l1.push(&head2,4);
 l1.push(&head2,8);

 intersecn=l1.getIntersection(head1,head2);

 cout<<"\nFirst list is \n";
l1.printList(head1);

 cout<<"\nSecond list is \n";
 l1.printList(head2);

 cout<<"\nIntersection list is\n";
 l1.printList(intersecn);


 getch();

}
```

1.1 Write a C++ program to create two singly linked lists of the same structure and perform the following operations through a menu-driven program :

iv. Find union and print

➔

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
typedef struct node
{
  int data;
  struct node* next;
}SN;

class list
{
    private:
              //SN *head;
    public:
      /*a utility function to insert a node at begining of a linked list*/
      void push(SN** head_ref, int new_data);

      /*a utility function to check if given data is present in a list*/
      int isPresent (SN *head, int data);
      SN *getUnion(SN *head1, SN *head2);

      void printList(SN *node);
};

SN* list::getUnion(SN *head1, SN *head2)
{
        SN *result=NULL;
        SN *t1=head1, *t2=head2;

       //Insert all elements of list1 to he resultlist
      while(t1!=NULL)
       {
           push(&result, t1->data);
          t1=t1->next;
        }

       //insert those elements of list2 which are not presenting list1
      while(t2!=NULL)
       {
   I      f(!isPresent(result,t2->data))
          push(&result,t2->data);
          t2=t2->next;
       }

         return result;
}


void list::push(SN** head_ref, int new_data)
```

```
{
            SN *new_node=new SN;
            new_node->data=new_data;
            //link the old list off the new node
            new_node->next=(*head_ref);
            //move the head to point the new node
            (*head_ref)=new_node;
}

void list::printList(SN *node)
{
             while(node!=NULL)
            {
               cout<<"\t"<<node->data;
               node=node->next;
            }
}

int list::isPresent(SN *head, int data)
{
             SN *t=head;
             while(t!=NULL)
             {
                   if(t->data==data)
                   return 1;
                   t=t->next;

             }
 return 0;
}

void main()
{
// clrscr();
list l1;

 SN* head1=NULL;
 SN* head2=NULL;
SN* uni=NULL;
 int i,n,j;
 clrscr();
// int i,n,j;
 cout<<"\nEnter the no. of elements to be inserted in list1: ";
 cin>>n;
 for(i=0;i<n;i++)
 {
             cout<<"\n"<<i+1<<": ";
            cin>>j;
            if(!l1.isPresent(head1,j))
               l1.push(&head1,j);
            else
             {
                 cout<<"\n enter another number which is not present in list:\n";
                cout<<i+1<<": ";
            cin>>j;

            l1.push(&head1,j);
           }
 }

/* push(&head1, 20);
 push(&head1,4);
 push(&head1,15);
 push(&head1,10);*/

 l1.push(&head2,10);
 l1.push(&head2,2);
l1.push(&head2,4);
 l1.push(&head2,8);

uni=l1.getUnion(head1,head2);

 cout<<"\nFirst list is \n";
l1.printList(head1);

 cout<<"\nSecond list is \n";
 l1.printList(head2);


 cout<<"\nUnion List is\n";
 l1.printList(uni);
 getch();

}
```

1.3 Write a C++ program to create a singly linked list for any structure and perform the following operations through a menu-driven program :

i. **Insert**
ii. **Sort**
iii. **Delete**
iv. **Count**
v. **Display**
vi. **Print reverse**
vii. **Search**

➔

```
#include<stdlib.h>
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<malloc.h>
#include<string.h>
#include<assert.h>
#include<ctype.h>
#include<iomanip.h>

struct Mausam
{
        char city[20];
        float tapmaan;
        struct Mausam *next;
};

typedef struct Mausam M;

class List
{
        private:
                M *head,*end;
                int count;
                void reverse(M *);
        public :
                List()
                {
                        count=0;
                        head=end=NULL;
                }
                        void createlist();
                        void insertlist();
                        void sortlist();
                        void deletelist(char *);
                        int countlist();
                        void displaylist();
                        void printRev();
                        M * searchlist(char *);
};

void List::printRev()
{
        reverse(head);
}

void List::reverse(M *h)
{
        if(h)
        {
                reverse(h->next);
                cout<<endl<<h->city<<" ";
                cout<<h->tapmaan;
                cout<<endl;
        }
        else
                return;
}

void List::deletelist(char *k)
{
        M *ppre,*ploc;
        int f;
        ppre=ploc=head;

        while(ploc)
        {
                if(strcmp(head->city,k)==0)
                {
                        head=head->next;
                        free(ploc);
                        count--;
                        break;
                }
                else if(strcmp(ploc->city,k)==0)
                {
                        ppre->next=ploc->next;
                        free(ploc);
                        count--;
                        break;
                }
                else
                {
                        ppre=ploc;
                        ploc=ploc->next;
                }
        }
}
void List::createlist()
{
        M *node;
        char ch;
        head=new M();
        assert(head);
        node=head;

        do
        {
                cout<<"\nCity " <<++count<<" ?:";
                fflush(stdin);
                cin>>node->city;
                cout<<"\nTapmaan "<<count<<" ?:";
                fflush(stdin);
                cin>>node->tapmaan;

                cout<<"\n\nMore city ? y/n:";
                fflush(stdin);
                cin>>ch;

                if(tolower(ch)=='y')
                {
                        node->next=new M();
                        assert(node->next);
                        node=node->next;
                }
                else
                {
                        node->next=NULL;
                        end=node;
                }
        }while(tolower(ch)=='y');
}
void List::insertlist()
{
        M *node;
        node=head;
        while(node->next)
        {
                node=node->next;
        }
        node->next=new M();
        node=node->next;
        cout<<endl<<"Enter new city to be inserted";
        cin>>node->city;
        cout<<endl<<"Its Tapmaan ?";
        cin>>node->tapmaan;
        node->next=NULL;
}
void List::sortlist()
{
        M *in,*jn,*s;
        s=head;
        float rtemp;
        char ptemp[20];

        for(in=s;in;in=in->next)
        {
                for(jn=in->next;jn;jn=jn->next)
                {
                        if(strcmp(in->city,jn->city)>0)
                        {
                            strcpy(ptemp,in->city);
                            strcpy(in->city,jn->city);
                            strcpy(jn->city,ptemp);

                            rtemp=in->tapmaan;
                            in->tapmaan=jn->tapmaan;
                            jn->tapmaan=rtemp;

                        }
```

```
		}
	}

}
int List::countlist()
{
	M *n;
	count=0;
	n=head;
	while(n)
	{
		count++;
		n=n->next;
	}
	return count;
}
M * List::searchlist(char *c)
{
	M *n=head;
	while(n)
	{
		if(strcmp(n->city,c)==0)
			return n;
		else
			n=n->next;
	}
	return NULL;
}
void List::displaylist()
{
	M *node;
	node=head;
	while(node)
	{
		cout<<endl<<node->city<<"\t"<<node->tapmaan;
		node=node->next;
	}
}


int main()
{
	List l1,l2;
	int c,ch;
	clrscr();
	l1.createlist();
	l1.displaylist();
	M *x;
	char key[20];
	do
	{
		cout<<endl<<endl<<endl<<"1. Insert"<<endl<<"2. Sort"<<endl<<"3.
		Delete"<<endl<<"4. Count"<<endl<<"5. Display"<<endl<<"6. Print
		Reverse"<<endl<<"7. Search"<<endl<<"8. Exit"<<endl;
		cout<<endl<<"Enter your choice";
		cin>>ch;

		switch(ch)
		{
```

```
            case 1:
                l1.insertlist();
                l1.displaylist();
                break;
            case 2:
                cout<<endl<<endl<<"After Sorting";
                l1.sortlist();
                l1.displaylist();
                break;
            case 3:
                cout<<endl<<"Enter a city to be deleted";
                cin>>key;
                l1.deletelist(key);
                l1.displaylist();
                break;
            case 4:
                c=l1.countlist();
                cout<<endl<<endl<<"The number of nodes list has are "<<c;
                break;
            case 5:
                l1.displaylist();
                break;
            case 6:
                l1.printRev();
                break;
            case 7:
                cout<<endl<<"Enter a city name to be searched";
                cin>>key;
                x=l1.searchlist(key);
                if(x==NULL)
                    cout<<"City not found";
                else
                    cout<<"City found"<<endl<<"Its tapmaan is "<<x-
>tapmaan;

                break;
            case 8:
                exit(0);
                break;
            default:
                cout<<endl<<"You entered wrong number";
        }
    }while(ch<9);

    return 0;
```

1.4 Write the properties of a linked list and depict it with a simple diagram and write the algorithms for the following operations for a singly linked list:

1) Insert

2) Delete (May 11)

3)Search (May 08)

4) Count (May 08, Dec 08, May 09)

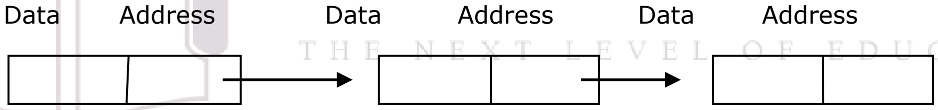
5) Append (Dec 08, May 09, May 11)

6) Sort

7) Display

8) Reverse

1) In Singly Linked list one data element at a time can be dynamically allotted till all the data are entered in fragments and each fragment called a node is linked through pointer addresses, pointing to their succeeding node.
2) If a huge disk file is to be loaded in memory can be loaded one record at a time through single dynamic self referential data structures, linked through pointer in the form of a list.
3) In singly link list addition and deletion of a record is possible.
4) Singly link list does not need to be declared at the beginning but it can be dynamically allotted.



Algorithm of Insert:-

```
Algorithm insertNode( val pList <head pointer>,
                      Val pPre<node pointer>,
                      Val datain <data Type>)
```

Insert data into a new node in the linked list

Pre: pList is a pointer to a valid list head structre.

pPre is a pointer to the data's logical predecessor.

Datain contains data to be inserted

Post : data have been inserted in sequence

Return : true if successful ,false if memory overflow

```
1. Allocate(pNew)
2. If(memory overflow)
       return false
3. pNew->data=dataIn
4. if(pPre null)
```

```
                Adding before first node or to empty list
              1.  pNew->link=pList->head
              2.  pList->head=pNew
5.  else
                Adding in middle or at end
               1.  pNew->link=pPre->link
               2.  pPre->link=pNew
6.  pList->count=pList->count+1
7.  return true
end insertNode
```

Algorithm of Delete:-

```
Algorithm deleteNode ( val pList <head pointer>,
                       Val pPre <node pointer>,
                       Val pLoc <node pointer>,
                       ref dataout <dataType>)
Deletes data from a linked list and returns it to calling module

Pre:   pList is a pointer to a valid list head structure
         pPre is a pointer to predecessor node
        pLoc is a pointer to node to be deleted
        dataOut address of variable to receive data
  Post : data have been deleted and returned to caller
 1.  dataOut = pLoc->data
 2.  if(pPre null)
               Deleting first no
           1.  pList->head = pLoc ->link
       else
                 Deleting other nodes
                1.pPre ->link = pLoc ->link
 4. pList->count = pList ->count-1
 5. release (pLoc)
 6. return
```

Algorithm of Search:-

```
algorithm searchList (val pList <head pointer>,
                      ref pPre <node pointer>,
                      ref pLoc <node pointer>,
                      val target <key Type>)
Searches list and passes back address of node containing target and its logical predecessor
Pre : pList is apointer to a linked list head structure
      pPre is a pointer variable to receive predecessor
      pLoc is a pointer variable to receive predecessor
      target is the key being sought
Post : pLoc points to first node with equal or greater key
         -or- null if target > key of last node
         pPre points to largest node smaller than key
         -or- null if target < key of first node
Return : true if found, false if not found

   1.  pPre = null
   2.  pLoc  = pList -> head
   3.  loop(pLoc not null AND target >pLoc ->data.key)
        1.  pPre = pLoc
        2.  pLoc = pLoc->link
     if (pLoc null)
      1.  found=false
     else
      1.  if (target equal pLoc->data.key)
           1.  found = true
           2.  else
                1.  found = false
         return found
end searchList
```

Algorithm of Count :-

```
algorithm listCount( val pList <head pointer>)
Returns integer representing number of nodes in list
Pre : pList is a pointer to a valid list head structure
Return : count for number of nodes in list
   1.  return ( pList ->count)
end listCount
```

Algorithm of Append:-

```
algorithm append ( val pList1<head pointer>, val pList2 <head pointer>)
This algorithm appends the second list at the end of the first
Pre : pList1 and pList2 are pointers to valid lists
Post : Second list appended to nth first list
      1.  if(pList1 -> count zero)
            1.  pList1->head = pList2->head
           else
              1.pLoc = pList1->head
             2. loop(pLoc ->link not null)
                  1. pLoc=pLoc->link
             3. pLoc->link = pList2->head
          3. pList1->count = pList1 ->count + pList2->count
           4. return
End append
```

Algorithm of Sort:

```
 Algorithm sortlist()

              It will sort  the elements of doubly linked list

Pre:     front is pointer to a linked list structure pointer.
                    Ploc is a pointer variable to receive current node
                    Target is the key being sought.
                    Inode and jnode are pointers of structure.
Ploc:               ploc points to first node with equal or greater key
                    Or  NULL  if target > key of last node
Temp:  is a temporary pointer

                1.            ploc=front
                2.           loop( inode equal to ploc and inode not equal to NULL)
            1.   loop( jnode equal to ploc and jnode not equal to NULL)
                    1.  if ( inode->data.key > jnode->data.key)
                              1.temp->data.key=inode->data.key
                              2.inode->data.key=jnode->data.key
                              3.jnode->data.key=temp->data.key
             3.          return
End sortlist
```

Algorithm of Display:

Algorithm printlist()

It will print the elements of doubly linked list

Front : is pointer of structure
Temp: is a temporary pointer

```
1.      temp=front
2.      loop(temp is not NULL)
        1. print temp->data.key
        2. temp=temp->forward
3 .     return
End printlist
```

Algorithm of Reverse:

algorithm reverseList(pList <end pointer>)

It will print the list elements in reverse order.

Pre: pList is a pointer to a linked list head structure.
Post: List will get print in reverse order.

```
1       if(pList->count is zero)
        1       success=false
2       else
        1       pLoc=pList->end
        2       loop(pLoc not null)
                1       print pLoc->data
                2       pLoc=pLoc->backward
        3       return success
3       return success
end reverseList
```

Doubly Linked List Programs

```
#include<iostream.h>
#include<conio.h>
#include<ctype.h>
#include<assert.h>
typedef struct DBList
{
        struct DBList *prev;
        int data;
        struct DBList *next;
}ll;
class List
{
        private:
                ll *head;
                int count;
        public:
                List();
                void createList();
                void sort();
                void delete();
                int count();
                void display();
                void printReverse();
                int search();
                void append(List,List);
                void merge(List,List);
                void unionList(List,List);
                void intersect(List,List);
};
List::List()
{
        head=NULL;
        count=0;
}
void List::createList()
{
        ll *node,*prev;
        char ch;
        head=new ll();
        assert(head);
        node=prev=head;
        node->prev=NULL;
        do
        {
                cout<<"\n\tEnter a Number->";
                cin>>node->data;
                count++;
                cout<<"\n\tContinue ?(y/n)";
                cin>>ch;
                if(tolower(ch)=='y')
                {
                        node->next=new ll();
```

```
            assert(node->next);
            node=node->next;
            node->prev=prev;
            prev=node;
        }
        else
        {
            node->next=NULL;
        }
    }while(tolower(ch)=='y');
}
void List::sort()
{
    ll *i,*j;
    i=j=head;
    int temp;
    for(i=head;i;i=i->next)
    {
        for(j=i->next;j;j=j->next)
        {
            if(i->data>j->data)
            {
                temp=i->data;
                i->data=j->data;
                j->data=temp;
            }
        }
    }
}
int List::count()
{
    return count;
}
void List::delete(int  data)
{
    ll *ppre,*node;
    ppre=node=head;
    while(node->next||(node->data==data))
    {
        node=node->next;
    }
    node->prev->next=node->next;
    node->next->prev=node->prev;
    delete node;
    ppre->next=NULL;
    count--;
    printf("\nData deleted");
}
void List::display()
{
    cout<<"\n\tList Contains";
    cout<<"\n\t-------------";
    ll *node=head;
```

```
        while(node)
        {
                cout<<"\n\t"<<node->data;
                node=node->next;
        }
}
void List::printReverse()
{
        ll *node=head;
        while(node->next)
        {
                node=node->next;
        }
        cout<<"\n\tList Contains(Reverse)";
        cout<<"\n\t-------------";
        while(node)
        {
                cout<<endl<<"\t"<<node->data;
                node=node->prev;
                }
}
int List::search()
{
        ll *node=head;
        int num=0;
        cout<<"\n\tEnter Number to search->";
        cin>>num;
        while(node)
        {
                if(node->data==num)
                {
                        return 1;
                }
                node=node->next;
        }
        return 0;
}
void List::append(List l)
{
        ll* node=head;
        if(head==NULL)
        {
                head=l.head;
                count=l.count;
        }
        else
        {
                while(node->next)
                {
                        node=node->next;
                }
                node->next=l.head;
                l.head->prev=node;
                count=count+l.count;
        }
}
void List::merge(List l1)
{
        ll *n1,*n2,*prev,*node;
        n1=head;
        n2=l1.head;
        while(n1->next)
        {
                n1=n1->next;
        }
        n1->next=n2;
        n2->prev=n1;
        if(n1==NULL && n2!=NULL)
        {
                n1=n2;
        }
        else if(n2==NULL && n1!=NULL)
        {
                n2=n1;
        }
}
void List::unionList(List l1,List l2)
{
        ll *n1,*n2,*n3,*pnew.*prev;
        n1=l1.head;
        n2=l2.head;
        n3=NULL;
        prev=NULL;
        while(n1&&n2)
        {
                if(n3!=NULL)
                {
                        if(n1->data < n2->data)
                        {
                                if(n3->data!=n1->data)
                                {
                                        pnew=new ll;
                                        assert(pnew);
                                        pnew->next=NULL;
                                        pnew->data=n1->data;
                                        n1=n1->next;
                                        n3->next=pnew;
                                        n3->prev=prev;
                                        n3=pnew;
                                        prev=pnew;
                                }
                                else
                                {
                                        n1=n1->next;
                                }
                        }
                        else if(n2->data<n1->data)
                        {
```

```
            if(n3->data!=n2->data)
            {
                pnew=new ll;
                assert(pnew);
                pnew->next=NULL;
                pnew->data=n2->data;
                n2=n2->next;
                n3->next=pnew;
                n3->prev=prev;
                n3=pnew;
                prev=pnew;
            }
            else
            {
                n2=n2->next;
            }
        }
    }
    else
    {
        pnew=new ll;
        assert(pnew);
        pnew->next=NULL;
        pnew->prev=NULL;
        if(n1->data<n2->data)
        {
            pnew->data=n1->data;
            n1=n1->next;
        }
        else
        {
            pnew->data=n2->data;
            n2=n2->next;
        }
        n3=head=pnew;
        prev=pnew;
    }
}
if(n1==NULL && n2!=NULL)
{
    while(n2)
    {
        if(n3->data!=n2->data)
        {
            pnew=new ll();
            assert(pnew);
            pnew->next=NULL;
            pnew->data=n2->data;
            n2=n2->next;
            n3->next=pnew;
            n3->prev=prev;
            n3=pnew;
            prev=pnew;
            }
            else
            {
                n2=n2->next;
            }
        }
    }
    else if(n2==NULL && n1!=NULL)
    {
        while(n1)
        {
            if(n3->data!=n1->data)
            {
                pnew=new ll();
                assert(pnew);
                pnew->next=NULL;
                pnew->data=n1->data;
                n1=n1->next;
                n3->next=pnew;
                n3->prev=prev;
                n3=pnew;
                prev=pnew;
            }
            else
            {
                n1=n1->next;
            }
        }
    }
}
void List::intersect(List l1,List l2)
{
    ll *n1,*n2,*n3,*pnew,*prev;
    n1=l1.head;
    n2=l2.head;
    n3=pnew=head;
    prev=NULL;
    int repeat1,repeat2,ctrl1,ctrl2;
    if(n1)
    {
        ctrl1=n1->data;
        n1=n1->next;
    }
    if(n2)
    {
        ctrl2=n2->data;
        n2=n2->next;
    }
    repeat1=repeat2=0;
    while(n1 && n2)
    {
        while(n1 && n1->data==ctrl1)
        {
                repeat1=1;
                n1=n1->next;
        }
        while(n2 && n2->data==ctrl2)
        {
                repeat2=1;
                n2=n2->next;
        }
        if(repeat1==1)
        {
                pnew=new ll;
                assert(pnew);
                pnew->next=NULL;
                pnew->data=ctrl1;
                if(n3)
                {
                        n3->next=pnew;
                        n3=pnew;
                }
                else
                {
                        n3=head=pnew;
                }
                n3->prev=prev;
                prev=pnew;
                repeat1=0;
        }
        if(repeat2==1)
        {
                pnew=new ll;
                assert(pnew);
                pnew->next=NULL;
                pnew->data=ctrl2;
                if(n3)
                {
                        n3->next=pnew;
                        n3=pnew;
                }
                else
                {
                        n3=head=pnew;
                }
                n3->prev=prev;
                prev=pnew;
                repeat2=0;
        }
        if(n1)
        {
                ctrl1=n1->data;
                n1=n1->next;
        }
        if(n2)//same for n2
        {
            ctrl2=n2->data;
            n2=n2->next;
        }
    }
    if( !n1&& n2==NULL)
    {
        repeat1=0;
        while(n1)
        {
            if(ctrl1==n1->data)
            {
                repeat1=1;
                n1=n1->next;
                continue;
            }
            else if(repeat1==1)
            {
                pnew=new ll;
                assert(pnew);
                pnew->next=NULL;
                pnew->data=ctrl1;
                ctrl1=n1->data;
                n1=n1->next;
                pnew->prev=prev;
                prev=n1;
                repeat1=0;
            }
            else
            {
                n1=n1->next;
                repeat1=0;
            }
        }
    }
    else if( n1&& n2!=NULL)
    {
        repeat2=0;
        while(n2)
        {
            if(ctrl2==n2->data)
            {
                repeat2=1;
                n2=n2->next;
                continue;
            }
            else if(repeat2==1)
            {
                pnew=new ll;
                assert(pnew);
                pnew->next=NULL;
                pnew->data=ctrl2;
                ctrl2=n2->data;
                n2=n2->next;
```

```
                                        pnew->prev=prev;
                                        prev=n2;
                                        repeat2=0;
                                }
                                else
                                {
                                        n2=n2->next;
                                        repeat2=0;
                                }
                        }
                }
        }


        int main()
        {
                clrscr();
                List l1, l2, l3;
                l1. createList();
                l2. createList();
                int val, del;
                do
                {
                        cout<<"\n\n1-Insert\n2-Sort\n3-Delete\n4-Count\n5-Display\n6-rint
                        Reverse\n7-Search\n8-Append 2 List\n9-Merge\n10-nion\n11-
                        Intersection\n12-Quit\n\n";
                        fflush(stdin);
                        cin>>val;
                        switch(val)
                        {
                                case 1:l.display();
                                        break;
                                case 2:l.sort();
                                        break;
                                case 3:l.append(l1);
                                        break;
                                case 4:l.createList();
                                        break;
                                case 5:cout<<"\nNo of elements : "<<l.count();
                                        break;
                                case 6:   cout<<"\nEnter the element you want to delete";
                                        cin>>del;
                                        l.delete(del);
                                        break;
                                case 7: l.printReverse();
                                        break;
                                case 8:cout<< "\nAppend List";
                                        l.append(l1);
                                        break;
                                case 9: cout<< "\nMerging 2 List";
                                        l.merge(l1);
                                        break;
                                        ase 10:cout<<"\nUnion Of 2 List";
                l3.unionList(l1,l2);
                l3.display();
                break;
            case 10:cout<< "\nIntersection of 2 Lists";
                l3.intersect(l1,l2);
                l3.display();
                break;
            case 12:break;
            default: printf("\nEnter correct value");
                  break;
        }
    }while(val!=12);
        getch();
        return 0;
}
```

Algorithms for Doubly Linked List

Algorithm for Insertion:

algorithm insertDbl(ref list <metadat>,val dataIn <datatype>)

This algorithm inserts data into a doubly linked list.

Pre : list is metadata structure to valid list
dataIn: contains the data to be inserted
Post : the data have been inserted in sequence
Return:<integer> 0:failed-dynamic memory overflow
1:successful
2:failed-duplicate key presented

```
1       if(full list)
        1       return 0
2       end if
Locate insertion point in list
3       found=searchList(list,pPre,pSucc,dataIn.key)
4       if(not found)
        1       allocate(pNew)
        2       pNew->data=dataIn
        3       if(pPre is null)
                inserting before first node or into empty list
                1. pNew->back = NULL
                2. pnew->fore = list.head
*               3. if(pNew->fore not NULL)
*                       1. pNew->fore->back = pNew
                4. list.head = pNew
*               5. return 1
        4       else
                inserting into middle or end of list
                1       pNew->fore=pPre->fore
                2       pNew->back=pPre
        5       end if
        Test for insert into null list or at end of list
        6       if(pPre->fore is null)
                inserting at end of list-set rear pointer
                1       list.rear=pNew
        7       else
                inserting in middle of list-point successor to new
                1       pSucc->back=pNew
        8       end if
        9       pPre->fore=pNew
        10      list.count=list.count+1
        11      return 1
5       end if
        DUPLICATE DATA.KEY ALREADY EXISTS
6       RETURN 2
end insertDbl
```

Algorithm To sort:

```
 Algorithm sortlist()
              It will sort  the elements of doubly linked list

Pre:    front is pointer to a linked list structure pointer.
                   Ploc is a pointer variable to receive current node
                   Target is the key being sought.
                   Inode and jnode are pointers of structure.
Ploc:              ploc points to first node with equal or greater key
                   Or  NULL  if target > key of last node
Temp:              is a temporary pointer

               1.           ploc=front
               2.          loop( inode equal to ploc and inode not equal to NULL)
          1.   loop( jnode equal to ploc and jnode not equal to NULL)
                    1.  if ( inode->data.key > jnode->data.key)
                              1.temp->data.key=inode->data.key
                              2.inode->data.key=jnode->data.key
                              3.jnode->data.key=temp->data.key
             3.          return
End sortlist
```

Algorithm for Deletion:

```
algorithm deleteDbl(ref list <metadat>, val pDlt <node pointer>)

This algorithm deletes a node from doubly linked list

Pre:    list is metadata structure to a valid list
       pDlt is a pointer to the node to be deleted
Post:  node deleted
1        if(pDlt null)
         1        abort(Impossible condition in delete double)
2        end if
3        list.count=list.count-1
4        if(pDlt->back not null)
         point predecessor to successor
         1        pPred=pDlt->back
         2        pPred->fore=pDlt->fore
5        else
         update head pointer
         1        list.head=pDlt->fore
6        end if
7        if(pDlt->fore not null)
         point successor to predessor
         1        pSucc=pDlt->fore
         2        pSucc->back=pDlt->back
8        else
         point rear to predecessor
         1        list.rear=pDlt->back
9        end if
10       recycle(pDlt)
11       return
end deleteDbl
```

Algorithm to count the elements:
Algorithm count()

It will count the elements of doubly linked list

Front : is pointer of structure
Temp: is a temporary pointer

```
1. temp= front
2.  loop(temp is NOT NULL)
        1.count=count +1
        2. temp=temp->forward
3. return count
End count
```

Algorithm to print the list:
Algorithm printlist()

It will print the elements of doubly linked list

Front : is pointer of structure
Temp: is a temporary pointer

```
1.      temp=front
2.      loop(temp is not NULL)
        1. print temp->data.key
        2. temp=temp->forward
3 .     return
End printlist
```

Algorithm to print in Reverse Order:
algorithm reverseList(pList <end pointer>)

It will print the list elements in reverse order.

Pre: pList is a pointer to a linked list head structure.
Post: List will get print in reverse order.

```
1       if(pList->count is zero)
        1       success=false
2       else
        1       pLoc=pList->end
        2       loop(pLoc not null)
                1       print pLoc->data
                2       pLoc=pLoc->backward
        3       return success
3       return success
end reverseList
```

Algorithm to Search:

Algorithm searchlist(val plist<object pointer> ,val target <key type>)

Searches list for a node and passes back address of nodes containing target.

Pre: plist is pointer to a linked list structure pointer.
Ploc is a pointer variable to receive current node
Target is the key being sought.
Post: ploc points to first node with equal or greater key
Or NULL if target > key of last node
Return: true if found , false if not found.

```
1. Ploc=plist->head
2. Loop( ploc not NULL AND target not equal ploc->data.key)
        1. Ploc=ploc->forward
3.    if (ploc  NULL)
       1.found=false
4.   else
        1. if(target equal ploc->data.key)
                1.found=true
        2. else
                1. found=false
5.   return found
END searchlist
```