```
#include<iostream>
#include<stdlib.h>
#include<conio.h>
using namespace std;
template<class T>class Node
{
public:
   T info;
   Node *next;
   Node(T data)
   {
      info=data;
      next=NULL;
   }

};
template<class T>class SLL
{
```

```
    Node<T> *head,*tail;
    int count;
public:
    SLL()
    {
        head=tail=NULL;
        count=0;
    }
    void InsertAtBeg(T data)
    {
        count++;
        Node<T> *newNode=new
Node<T>(data);
        if(head==NULL)       //When Linked
list is empty
            head=tail=newNode;
        else
            {
```

```
            newNode->next=head;
            head=newNode;
        }
    }
    void InsertAtEnd(T data)
    {
        count++;
        Node<T> *newNode=new
Node<T>(data);
        if(tail==NULL)       //When Linked
list is empty
            head=tail=newNode;
        else
           {
            tail->next=newNode;
            tail=newNode;
        }
    }
```

```
   void InsertAtMid(T data)
   {

       Node<T> *newMid=new
Node<T>(data);
       if(count%2==0)          //Linked list
must have even nodes
       {
           int mid=count/2;
           if(head==NULL )                 //When
linked list is empty
               cout<<"Can't insert data in
between ";
           else
               {
               Node<T> *current=head;
               for(T i=0;i<mid-1;i++)
               {
```

```
            current=current->next;
        }
        newMid->next=current->next;
        current->next=newMid;
        count++;
    }
  }
}
T DelFromMid()
{
  if(head==NULL)
    throw "Linked list is empty";
  else if(head==tail)
  {
    T data=head->info;
    delete head;
    head=tail=NULL;
    return data;
```

```
}
else
   {
    if(count%2!=0)
    {
        T mid=count/2;
        Node<T> *current=head->next;
        Node<T> *temp=head;
        for(int i=1;i<mid;i++)
        {
            temp=current;
            current=current->next;
        }
        temp->next=current->next;
        T data=current->info;
        delete current;
        count--;
        return data;
```

```
            }
            else
                {
                cout<<"Cannot delete the middle
node "<<endl;
                return 0;
            }
        }
    }
    T DelFromBeg()
    {
        count--;
        if(head==NULL)                  //When
linked list is empty
            throw "Linked list is empty ";
        else if(head==tail)             //When
linked list contains single node
        {
```

```
            T data=head->info;
            delete head;
            head=tail=NULL;
            return data;
        }
        else                        //When linked
list contains more than one node
        {
            Node<T> *temp=head;
            T data=head->info;
            head=head->next;
            delete temp;
            return data;
        }
    }
    T Del_From_End()
    {
        count--;
```

```
        if(head==NULL)              //When
linked list is empty
            throw "Linked list is empty ";
        else if(head==tail)             //When
linked list contains single node
        {
            T data=tail->info;
            delete tail;
            head=tail=NULL;
            return data;
        }
        else                        //When linked list
contains more than one node
        {
            T data=tail->info;
            Node<T> *Current_Node=head;
            while(Current_Node->next!=tail)
            {
```

```
Current_Node=Current_Node->next;
        }
        delete tail;
        tail=Current_Node;
        tail->next=NULL;
        return data;
    }
}
void Display()
{
    Node<T> *current=head;
    cout<<"Linked List : ";
    if(current!=NULL)
        while(current!=NULL)
        {
            cout<<current->info<<"  ";
            current=current->next;
```

```
        }
    else
            cout<<"Empty";
    cout<<endl;
  }
  void Count()
  {
    cout<<"\nNo of nodes are "<<count;
  }
  bool Search_Value(T val)
  {
    if(head==NULL)          //When linked
list is empty
        cout<<"Linked List is empty ";
    else                        //When linked
list contains at least one node
    {
        Node<T> *temp=head;
```

```
        while(temp!=NULL)
        {
            if(temp->info==val)
                return true;
            temp=temp->next;
        }
        return false;
    }
}
void Reverse()
{
    if(head==NULL)     //Linked List is
empty
        throw "Linked list is empty. ";

    else if(head==tail)    //Linked list
contains one node
        cout<<"Nothing can be done. ";
```

```
    else        //Linked list contains more
than one node
    {
        Node<T> *prevNode=NULL;
        Node<T> *current=head;
        Node<T> *nextNode=head->next;
        while(current!=NULL)
        {
            current->next=prevNode;
            prevNode=current;
            current=nextNode;
            if(nextNode!=NULL)
                nextNode=nextNode->next;
        }
        tail=head;
        head=prevNode;
    }
```

```
    }
    void InsertAtPos(T data,int pos)
    {
        Node<T> *temp=new Node<T>(data);
        if(pos<=count && pos>0)       //Given
position is not greater than size of linked
list
        {
            if(head==NULL)       //When
Linked list is empty
                throw "Linked List is empty ";
            else                 //When Linked list
contains at least one node
            {
                Node<T> *current=head;
                for(int i=1;i<pos-1;i++)
                {
                    current=current->next;
```

```
            }
            temp->next=current->next;
            current->next=temp;
            count++;
        }
    }
    else
            cout<<"Error";
}

void ConcatFunc(SLL s1,SLL s2)
{
    if(s1.head==NULL)     //When First
linked list is empty
    {
        head=s2.head;
        tail=s2.tail;
    }
```

```
        else if(s2.head==NULL)    //When
Second linked list is empty
        {
            head=s1.head;
            tail=s1.tail;
        }
        else                      //When Both
linked list contain more than one node
        {
            head=s1.head;
            s1.tail=s2.head;
            tail=s2.tail;
        }
    }
    SLL operator+(SLL ob)
    {
        SLL newOb;
```

```
        if(head==NULL)          //When First
linked is empty
        {
            newOb.head=ob.head;
            newOb.tail=ob.tail;
        }
        else if(ob.head==NULL)      //When
Second list is empty
        {
            newOb.head=head;
            newOb.tail=tail;
        }
        else                        //When Both
linked list contains at least one node
        {
            newOb.head=head;
            tail->next=ob.head;
            newOb.tail=ob.tail;
```

```
        }
        return newOb;
    }
    void DelValue(T value)
    {
        if(head==NULL) //case 1 when linked
list is empty
            throw "Linked list is empty ";
        else if(value==head->info) //case 2
when linked list contains single node
            DelFromBeg();
        else if(value==tail->info)
            Del_From_End();
        else
           {
            Node<T> *current=head;
            Node<T> *temp;
```

```
        while(value!=current->info &&
current->next!=NULL)
        {
            temp=current;
            current=current->next;
        }
        if(current->info==value)
        {
            temp->next=current->next;
            delete current;
            count--;
            cout<<"Value deleted : "<<value;
            count--;
        }
        else
                cout<<"VALUE NOT
FOUND ";
    }
```

```
    }
    T DelFromPos(int pos)
    {
        if(head==NULL)                  //when
linked list is empty
            throw "Linked list is empty ";
        else if(head==tail && pos==1)
//when linked list contains single node
        {
            T data=tail->info;
            delete tail;
            head=tail=NULL;
            return data;
        }
        else                            //When Linked
list contains at least one node
        {
            if(pos>0 && pos<=count)
```

```
        {
            Node<T> *current=head->next;
            Node<T> *temp=head;
            for(int i=1;i<pos-1;i++)
            {
                temp=current;
                current=current->next;
            }
            temp->next=current->next;
            T data=current->info;
            delete current;
            count--;
            return data;
        }
        else
                cout<<"Node<T> cannot be
deleted ";
    }
```

```
}
SLL Duplicate()
{
    SLL dup;
    if(head==NULL)
        throw "Linked list is empty ";
    else if(head==tail)
        dup.head=dup.tail=head;
    else
      {
        Node<T> *current=head;
        Node<T> *Duplicate=current;
        dup.head=current;
        while(current->next!=NULL)
        {
            Duplicate->next=current->next;
            Duplicate=Duplicate->next;
            current=current->next;
```

```
        }
        dup.tail=Duplicate;
        return dup;
    }
}
void DelAll()
{
    if(head==NULL)
        throw "Linked list is empty ";
    else if(head==tail)
    {
        delete head;
        head=tail=NULL;
    }
    else
      {
        Node<T> *temp=head;
        Node<T> *current=head;
```

```
        while(temp->next!=NULL)
        {
            temp=current->next;
            delete current;
            count--;
            current=temp;
        }
        delete temp;
        head=tail=NULL;
    }
}
void SumLL()
{
    int sum=0;
    if(head==NULL)
    {
        cout<<"SUM : "<<sum<<endl;
        throw "Linked list is empty ";
```

```
        }
        else if(head==tail)
        {
            sum=sum+head->info;
            cout<<"SUM : "<<sum<<endl;
        }
        else
           {
            Node<T> *current=head;
            while(current->next!=NULL)
            {
                sum+=current->info;
                current=current->next;
            }
            sum+=current->info;
            cout<<"SUM of all elements of
Linked list : "<<sum;
        }
```

```
}
void DelAltNode()
{
    if(head==NULL)
        throw "linked list is empty ";
    else if(count==2)
    {
        Node<T> *temp=head->next;
        T data=temp->info;
        head->next=NULL;
        delete temp;
    }
    else
      {
        Node<T> *temp=head;
        Node<T> *current=head->next;
        while(current!=NULL)
        {
```

```
            temp->next=current->next;
            delete current;
            count--;
            if(temp->next!=NULL)
               temp=temp->next;
            current=temp->next;
         }
      }
   }
   int ListEqual(SLL s)
   {
      int flag=0;
      if(s.head==NULL || head==NULL)
         throw "One of the linked list is
empty ";
      else
         {
          if(s.count==count)
```

```
{
    Node<T> *temp1=s.head;
    Node<T> *temp2=head;
    while(temp1!=NULL)
    {
        if(temp1->info==temp2->info)
        {
            flag=flag+1;
            temp1=temp1->next;
            temp2=temp2->next;
        }
        else
                {
            flag=0;
            break;
        }

    }
```

```
            if(flag==count)
               return 1;
            else
                     return 0;
         }
         return 0;
      }

   }
   SLL operator=(SLL s)
   {
      int flag=0;
      if(s.head==NULL || head==NULL)
         throw "One of the linked list is
empty ";
      else
         {
         if(s.count==count)
```

```
{
    Node<T> *temp1=s.head;
    Node<T> *temp2=head;
    while(temp1!=NULL)
    {
        if(temp1->info==temp2->info)
        {
            flag=flag+1;
            temp1=temp1->next;
            temp2=temp2->next;
        }
        else
                {
            flag=0;
            break;
        }
    }
}
```

```
    }
  }
  void menu()
  {
      int ch;
      cout<<"MENU";
      cout<<"\n1.Insert node at the
beginning. ";
      cout<<"\n2.Insert node at the end. ";
      cout<<"\n3.Insert node in the mid. ";
      cout<<"\n4.Display Linked list.";
      cout<<"\n5.Delete node from the
beginning. ";
      cout<<"\n6.Delete node from the
End. ";
      cout<<"\n7.Delete node from the
middle. ";
```

```
        cout<<"\n8.Count the number of
Nodes. ";
        cout<<"\n9.Search Value. ";
        cout<<"\n10.Reverse of Linked list. ";
        cout<<"\n11.Insert node at given
position. ";
        cout<<"\n12.Delete a particular node
at kth position. ";
        cout<<"\n13.Delete a particular
element node. ";
        cout<<"\n14.Delete/Free all the
nodes of the linked list. ";
        cout<<"\n15.Sum of the elements of
Linked list. ";
        cout<<"\n16.Delete alternate nodes
of the linked list starting with node 2. ";
        cout<<"\n17.Go back to menu ";
        choice();
```

```
}
void choice()
{
    T value,n,K;
    int p,ch;
    bool k;
    cout<<"\nEnter your choice : ";
    cin>>ch;
    char c='Y';
    try
      {
       switch(ch)
       {

            case 1:    cout<<"Enter the data
to be inserted : ";
                cin>>value;
                InsertAtBeg(value);
```

```
                break;

            case 2:    cout<<"Enter the data
to be inserted ";
                cin>>value;
                InsertAtEnd(value);
                break;

            case 3:    cout<<"Enter the data
to be inserted ";
                cin>>value;
                InsertAtMid(value);
                break;

            case 4: Display();
                break;

            case 5:    K=DelFromBeg();
```

```
                cout<<"Value Deleted
"<<K<<endl;
                break;

            case 6: K=Del_From_End();
                cout<<"Value Deleted :
"<<K<<endl;
                break;
            case 7: K=DelFromMid();
                cout<<"Value Deleted from
the middle : "<<K;
                break;
            case 8: Count();
                break;

            case 9: cout<<"Enter no. to be
searched ";
                cin>>n;
```

```
                k=Search_Value(n);
                if(k)
                    cout<<"Value Found ";
                else
                                cout<<"Value not
found ";
                break;

            case 10: Reverse();
                break;

            case 11: cout<<"Enter data : ";
                cin>>value;
                cout<<"Enter position : ";
                cin>>p;
                InsertAtPos(value,p);
                break;
```

```
            case 12: cout<<"Enter position :
";
                cin>>p;
                K=DelFromPos(p);
                cout<<"Value Deleted
"<<K<<endl;
                break;

            case 13: cout<<"Enter data : ";
                cin>>value;
                DelValue(value);
                break;
            case 14: DelAll();
                break;
            case 15:SumLL();
                break;
            case 16: DelAltNode();
                break;
```

```
            case 17: menu();
                break;
            default:cout<<"Wrong Input";
        }


    }
    catch(const char *msg)
    {
        cout<<msg<<endl;
    }
    cout<<"\nDo you want to
continue(Y/N) : ";
    cin>>c;
    if(c=='y' || c=='Y')
        choice();
    else
```

```
                cout<<"\nExiting this program
and going for performing concatenation.
";
   }
};
template<class T>class Concat
{
   SLL<T> ob1,ob2,ob3,Dup;
public:
   void option()
   {
      cout<<"\n\nOPTIONS FOR
CONCATENATE ";
      cout<<"\n1.Insert At Beg for Linked
list 1 . ";
      cout<<"\n2.Insert At end for Linked
List 2. ";
```

```
        cout<<"\n3.Insert At Beg for Linked
list 1 . ";
        cout<<"\n4.Insert At end for Linked
List 2. ";
        cout<<"\n5.Concatenate two linked
list using function. ";
        cout<<"\n6.Concatenate two linked
list using operator +. ";
        cout<<"\n7.Display Linked list 1 .";
        cout<<"\n8.Display Linked list 2 .";
        cout<<"\n9.Duplicate Linked list 1.";
        cout<<"\n10.Display Duplicate Linked
list.";
        cout<<"\n11.Compare two Linked lists
without using = operator.";
        cout<<"\n12.Compare two Linked lists
using = operator.";
        cout<<"\n13.Go Back to options ";
```

```
    cout<<"\n14.Exit. ";
    LinkedList();
}
void LinkedList()
{
    T val,var;
    int ch;
    char sel='Y';
    cout<<"\nEnter your choice : ";
    cin>>ch;
    switch(ch)
    {
        case 1: cout<<"Enter Data  : ";
            cin>>val;
            ob1.InsertAtBeg(val);
            break;
        case 2: cout<<"Enter Data  : ";
            cin>>val;
```

```
        ob1.InsertAtEnd(val);
        break;
    case 3: cout<<"Enter Data : ";
        cin>>val;
        ob2.InsertAtBeg(val);
        break;
    case 4: cout<<"Enter Data : ";
        cin>>val;
        ob2.InsertAtEnd(val);
        break;
    case 5: ob3.ConcatFunc(ob1,ob2);
        ob3.Display();
        break;
    case 6: ob1=ob1+ob2;
        ob1.Display();
        break;
    case 7: ob1.Display();
        break;
```

```
            case 8: ob2.Display();
                break;
            case 9: Dup=ob1.Duplicate();
                break;
            case 10: Dup.Display();
                break;
            case 11: var=ob1.ListEqual(ob2);
                cout<<"Linked list equal or not :
"<<var;
                break;
            case 12: ob1=ob2;

            case 13: option();
                break;
            case 14: exit(0);
            default: cout<<"Wrong input
provided ";
        }
```

```
        cout<<"\nDo you want to Continue
this(Y/N) : ";
        cin>>sel;
        if(sel=='y' || sel=='Y')
           LinkedList();
        else
           {
            cout<<"Exiting the program ";
        }


    }
};
int main()
{
    SLL<int> ob;
    SLL<float> ob2;
    Concat<int> C;
    ob.menu();
```

```
    ob2.menu();
    C.option();
    return 0;
}
```