```
/*WAP to implement circular linked list */
#include<iostream>
#include<cstdlib>
using namespace std;
struct node
{
   int info;
   node *next;
};
class CSlist
{
   node *last;
public:
   CSlist():last(NULL) {}
   void InsertNodeAtBegining(int );
   void InsertNodeAtLast(int );
   void InsertNodeBeforeGivenData(int,int );
   void InsertNodeAfterGivenData(int,int);
   void DeleteFirstNode();
   void DeleteLastNode();
   void DeleteNodeBeforeGivenData(int );
   void DeleteNodeAfterGivenData(int);
   bool display();
};
void CSlist::InsertNodeAtBegining(int val)
{
   node *temp=new node;
   if(temp==NULL)
      cout<<"\n\nFailed to initialize memory for new node\n\n";
   else
   {
      temp->info=val;
      if(last==NULL)
      {
         temp->next=temp;
         last=temp;
      }
```

```
        else
        {
            temp->next=last->next;
            last->next=temp;
        }
    }
}
void CSlist::InsertNodeAtLast(int val)
{
    node *temp=new node;
    if(temp==NULL)
        cout<<"\n\nFailed to initialize memory for new node\n\n";
    else
    {
        temp->info=val;
        if(last==NULL)
        {
            temp->next=temp;
            last=temp;
        }
        else
        {
            temp->next=last->next;
            last->next=temp;
            last=temp;
        }
    }
}
void CSlist::InsertNodeBeforeGivenData(int data,int val)
{
    if(last->next->info==data)
        InsertNodeAtBegining(val);
    else
    {
        int c=0;
        node *ptr;
        ptr=last->next;
```

```
      while(ptr!=last)
      {
         if(ptr->next->info==data)
         {
            c=1;
            node *temp=new node;
            if(temp==NULL)
            {
               cout<<"\n\nFailed to initialize the memory for new node.\n\n";
               break;
            }
            else
            {
               temp->info=val;
               temp->next=ptr->next;
               ptr->next=temp;
               break;
            }
         }
         else
            ptr=ptr->next;
      }
      if(c==0)
         cout<<"\n\nThere is no matching data in linked list.\n\n";
   }
}
void CSlist::InsertNodeAfterGivenData(int data,int val)
{
   if(last->info==data)
      InsertNodeAtLast(val);
   else
   {
      int c=0;
      node *ptr;
      ptr=last->next;
      while(ptr!=last)
      {
         if(ptr->info==data)
         {
            c=1;
            node *temp=new node;
            if(temp==NULL)
            {
               cout<<"\n\nFailed to initialize the memory for new node.\n\n";
               break;
            }
            else
            {
               temp->info=val;
               temp->next=ptr->next;
               ptr->next=temp;
               break;
            }
         }
         else
            ptr=ptr->next;
      }
      if(c==0)
         cout<<"\n\nThere is no matching data in the linked list.\n\n";
   }
}
void CSlist::DeleteFirstNode()
{
   if(last==NULL)
      cout<<"\n\nThere is no existing list.\n\n";
   else if(last->next==last)
   {
      cout<<"\n\nThe deleted value of first node is : "<<last->info<<"\n\n";
      delete last;
      last=NULL;
   }
   else
   {
      node *temp;
```

```
        temp=last->next;
        last->next=last->next->next;
        cout<<"\n\nThe deleted value of node is : "<<temp->info<<"\n\n";
        delete temp;
    }
}
void CSlist::DeleteLastNode()
{
    if(last==NULL)
        cout<<"\n\nThere is no existing list.\n\n";
    else if(last->next==last)
    {
        cout<<"\n\nThe deleted value of last node is : "<<last->info<<"\n\n";
        delete last;
        //cout<<"\n\nThe deleted value of last node is : "<<last->info<<"\n\n";
        last=NULL;
    }
    else
    {
        node *temp;
        temp=last->next;
        while(temp->next!=last)
            temp=temp->next;
        temp->next=last->next;
        cout<<"\n\nThe deleted value of node is : "<<last->info<<"\n\n";
        delete last;
        last=temp;
    }
}
void CSlist::DeleteNodeBeforeGivenData(int data)
{
    if(last->next->info==data)
        DeleteLastNode();
    else
    {
        int c=0;
        node *ptr,*preptr;
        preptr=ptr=last->next;
        while(ptr->next!=last->next)
        {
            if(ptr->next->info==data)
            {
                c=1;
                preptr->next=ptr->next;
                cout<<"\n\nThe deleted value is : "<<ptr->info<<"\n\n";
                delete ptr;
                break;
            }
            else
            {
                preptr=ptr;
                ptr=ptr->next;
            }
        }
        if(c==0)
            cout<<"\n\nThere is no matching data in the linked list.\n\n";
    }
}
void CSlist::DeleteNodeAfterGivenData(int data)
{
    if(last->info==data)
        DeleteFirstNode();
    else
    {
        int c=0;
        node *ptr;
        ptr=last->next;
        while(ptr!=last)
        {
            if(ptr->info==data)
            {
                c=1;
                if(ptr->next==last)
                {
```

```
                ptr->next=last->next;
                cout<<"\n\nThe deleted value of node is : "<<last->info<<"\n\n";
                delete last;
                last=ptr;
                break;
            }
            else
            {
                node *temp;
                temp=ptr->next;
                ptr->next=ptr->next->next;
                cout<<"\n\nThe deleted value of node is : "<<temp->info<<"\n\n";
                delete temp;
                break;
            }
        }
        else
            ptr=ptr->next;
    }
    if(c==0)
        cout<<"\n\nThere is no matching data in the linked list.\n\n";
  }
}
bool CSlist::display()
{
  if(last==NULL)
  {
    cout<<"\n\nThere is no existing list.\n\n";
    return false;
  }
  else
  {
    node *temp;
    temp=last->next;
    cout<<"\n\nElements of linked list are :\n";
    do
    {
            cout<<temp->info<<"\t";
            temp=temp->next;
        }
        while(temp!=last->next);
        cout<<"\n\n";
    }
}
int main()
{
    CSlist l;
    int choice,data,val;
    while(1)
    {
        cout<<"1. Insert Node at begining\n2. Insert node at last\n3. Insert node
before given data\n4. Insert node after given data\n5. Delete first node\n6.
Delete last node\n7. Delete node before given data\n8. Delete node after given
data\n9. Display circular singly linked list\n10. Exit\nEnter your choice : ";
        cin>>choice;
        switch(choice)
        {
        case 1:
        {
            cout<<"\nEnter the value : ";
            cin>>val;
            l.InsertNodeAtBegining(val);
            break;
        }
        case 2:
        {
            cout<<"\nEnter the value :";
            cin>>val;
            l.InsertNodeAtLast(val);
            break;
        }
        case 3:
        {
            if(l.display())
```

```
          {
             cout<<"\nEnter the value of node before you want to insert new node: 
";
             cin>>data;
             cout<<"\nEnter the value for new node : ";
             cin>>val;
             l.InsertNodeBeforeGivenData(data,val);
          }
          break;
       }
       case 4:
       {
          if(l.display())
          {
             cout<<"\nEnter the value of node after you want to insert new node: ";
             cin>>data;
             cout<<"\nEnter the value for new node : ";
             cin>>val;
             l.InsertNodeAfterGivenData(data,val);
          }
          break;
       }
       case 5:
       {
          l.DeleteFirstNode();
          break;
       }
       case 6:
       {
          l.DeleteLastNode();
          break;
       }
       case 7:
       {
          if(l.display())
          {
             cout<<"\nEnter the value of node who's previous node you want to
delete : ";
             cin>>data;
             l.DeleteNodeBeforeGivenData(data);
          }
          break;
       }
       case 8:
       {
          if(l.display())
          {
             cout<<"\nEnter the value of node who's next node you want to delete :
";
             cin>>data;
             l.DeleteNodeAfterGivenData(data);
          }
          break;
       }
       case 9:
       {
          bool a=l.display();
          break;
       }
       default :
          exit(0);
       }
    }
    return 0;
}
```

```
/*WAP to implement circular linked list */
#include<iostream>
#include<cstdlib>
using namespace std;
struct node
```

```
{
   int data;
   node* next;
};
class clinkedlist
{
   node *head;
public:
   clinkedlist()
   {
      head = NULL;
   }
   void create_clinkedlist()
   {
      int val = 0;
      while(val != -1)
      {
         cout<<"\nEnter a value: ";
         cin>>val;
         if(val != -1)
         {
            node *newNode = new node;
            newNode->data = val;

            if(head == NULL)
            {
               head= newNode;
               newNode->next = head;
            }
            else
            {
               node *ptr = head;
               while(ptr->next != head)
               {
                  ptr = ptr->next;
               }
               ptr->next = newNode;
```

```
                newNode->next = head;
            }
        }
    }
}
void insert_end_clinkedlist(int n)
{
    node *ptr = head;
    node *newNode = new node;
    newNode->data = n;
    newNode->next = head;
    if(head == NULL)
    {
        head= newNode;
    }
    else
    {
        while(ptr->next != head)
        {
            ptr = ptr->next;
        }
        ptr->next = newNode;
        newNode->next = head;
    }
}
void insert_beg_clinkedlist(int n)
{
    node *newNode = new node;
    newNode->data = n;
    newNode->next = head;
    node *ptr = head;
    while(ptr->next != head)
    {
        ptr = ptr->next;
    }
    ptr->next = newNode;
    head = newNode;
```

```
}
void insert_before_clinkedlist(int n, int val)
{
   node *newNode = new node;
   newNode->data = val;
   if(head->data == n)
   {
      insert_beg_clinkedlist(val);
   }
   else
   {
      node *ptr = head;
      node *preptr;
      while(ptr->data != n)
      {
         preptr = ptr;
         ptr = ptr->next;
      }
      preptr->next = newNode;
      newNode->next = ptr;
   }
}
void insert_after_clinkedlist(int n, int val)
{
   node *newNode = new node;
   newNode->data = val;

   node *ptr = head;
   while (ptr->data != n)
   {
      ptr = ptr->next;
   }
   if(ptr->next == head)
   {
      ptr->next = newNode;
      newNode->next = head;
   }
```

```
    else
    {
        newNode->next=ptr->next;
        ptr->next = newNode;
    }
}
void delete_beg_clinkedlist()
{
    if(head->next == head)
    {
        head = NULL;
    }
    else
    {
        node *ptr = head;
        node *tmp = head;
        while(ptr->next != head)
        {
            ptr = ptr->next;
        }
        ptr->next = head->next;
        head = head->next;
        delete tmp;
    }
}
void delete_end_clinkedlist()
{
    node *ptr =head;
    node *preptr = ptr;
    while(ptr->next != head)
    {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = head;
    delete ptr;
}
```

```
void delete_node_clinkedlist(int n)
{
   node *ptr = head;
   if(ptr->data == n)
   {
      delete_beg_clinkedlist();
   }
   else
   {
      node*preptr = ptr;
      while(ptr->data != n)
      {
         preptr = ptr;
         ptr = ptr->next;
      }
      preptr->next = ptr->next;
      delete ptr;
   }
}
void delete_after_clinkedlist(int n)
{
   node *ptr= head;
   while(ptr->data != n)
   {
      ptr = ptr->next;
   }
   if(ptr->next == head)
   {
      delete_beg_clinkedlist();
   }
   else
   {
      node *tmp = ptr->next;
      ptr->next = tmp->next;
      delete tmp;
   }
}
```

```
    void delete_clinkedlist()
    {
        while(head != NULL)
        {
            delete_beg_clinkedlist();
        }
    }
    void display_clinkedlist()
    {
        node *ptr = head;
        if(head == NULL)
        {
            cout<<"\nThe list is empty!!"<<endl;
        }
        else
        {
            cout<<endl<<endl<<"Head: "<<head->data<<endl;
            while(ptr->next != head)
            {
                cout<<" "<<ptr->data<<" ";
                ptr = ptr->next;
            }
            cout<<" "<<ptr->data<<" ";
            cout<<endl<<endl;
        }
    }
};
int main()
{
    clinkedlist listobj;
    int choose;
    do
    {
        cout<<"\n\n1. Create a linked list."<<endl;
        cout<<"2. Insert at beginningg."<<endl;
        cout<<"3. Insert at end."<<endl;
        cout<<"4. Insert before a node in linked list."<<endl;
```

```
cout<<"5. Insert after a node in linked list."<<endl;
cout<<"6. Delete beginning of a linked list."<<endl;
cout<<"7. Delete end of a linked list."<<endl;
cout<<"8. Delete a node of a linked list."<<endl;
cout<<"9. Delete after a node of a linked list."<<endl;
cout<<"10. Delete a linked list."<<endl;
cout<<"11.Exit"<<endl;
cout<<"\n\nChoose an option: ";
cin>>choose;
switch (choose)
{
case 1:
{
   listobj.create_clinkedlist();
   break;
}
case 2:
{
   int val;
   cout<<"\nenter the number to insert at the beginning: ";
   cin>>val;
   listobj.insert_beg_clinkedlist(val);
   break;
}
case 3:
{
   int val;
   cout<<"\nenter the number to insert at end: ";
   cin>>val;
   listobj.insert_end_clinkedlist(val);
   break;
}
case 4:
{
   int n,val;
   cout<<"\nEnter the the node value whose predecessor is to be added: ";
   cin>>n;
```

```
    cout<<"Enter the number to insert: ";
    cin>>val;
    listobj.insert_before_clinkedlist(n,val);
    break;
}
case 5:
{
    int n,val;
    cout<<"\nEnter the the node value whose successor is to be added: ";
    cin>>n;
    cout<<"Enter the number to insert: ";
    cin>>val;
    listobj.insert_after_clinkedlist(n,val);
    break;
}
case 6:
{
    listobj.delete_beg_clinkedlist();
    break;
}
case 7:
{
    listobj.delete_end_clinkedlist();
    break;
}
case 8:
{
    int n;
    cout<<"\nEnter the node value to delete: ";
    cin>>n;
    listobj.delete_node_clinkedlist(n);
    break;
}
case 9:
{
    int n;
```

```
         cout<<"\nEnter the node value whose succeeding value is to be deleted:
";
         cin>>n;
         listobj.delete_after_clinkedlist(n);
         break;
      }
      case 10:
      {
         listobj.delete_clinkedlist();
         break;
      }
      case 11:
      {
         exit(1);
         break;
      }
      default :
      {
         cout<<"Invalid input";
         break;
      }
      }
      listobj.display_clinkedlist();
   }
   while (choose!=11);
   return 0;
}



/*WAP to implement circular linked list */
#include<iostream>
using namespace std;
class linkList
{
   struct Node
   {
        int data;
        Node *next;
    };
    typedef struct Node* nodeptr;
    nodeptr head;
public:
    linkList()
    {
        head=NULL;
    }
    void del_list()
    {
        if(head!=NULL)
        {
            nodeptr p,q;
            p=head;
            do
            {
                q=p;
                p=p->next;
                delete q;
            }
            while(p!=head);
            head=NULL;
        }
    }
    void create()
    {
        nodeptr ptr=head;
        int val=0;
        cout<<"insert and end with -1"<<endl;
        cin>>val;
        while(val!=-1)
        {
            ins(val);
            cin>>val;
        }
```

```
}
void push(int new_data)
{
   nodeptr p,ptr;
   p=new Node;
   p->data= new_data;
   p->next=head;
   ptr=head;
   if(head==NULL)
   {
      ins(new_data);
   }
   else
   {
      while(ptr->next!=head)
      {
         ptr=ptr->next;
      }
      head=p;
      ptr->next=head;
   }
}
void ins(int new_data)
{
   nodeptr p;
   nodeptr ptr=head;
   if(head==NULL)
   {
      p= new Node;
      p->data= new_data;
      p->next=p;
      head=p;
   }
   else
   {
      while(ptr->next!=head)
      {
            ptr=ptr->next;
        }
        p=new Node;
        ptr->next=p;
        p->data =new_data;
        p->next =head;
    }
}
void ins_after(int old_data,int new_data)
{
    nodeptr p;
    nodeptr ptr=head;
    while(ptr->data!=old_data)
    {
        ptr=ptr->next;
        if(ptr==head)
        {
            cout<<"error data not found";
            return;
        }
    }
    p=new Node;
    p->next =ptr->next;
    ptr->next=p;
    p->data =new_data;
}
void ins_bef(int old_data,int new_data)
{
    nodeptr p,preptr;
    nodeptr ptr=head;
    if(ptr->data==old_data)
    {
        push(new_data);
    }
    else
    {
        while(ptr->data!=old_data)
```

```
            {
                preptr=ptr;
                ptr=ptr->next;
                if(ptr==head)
                {
                    cout<<"error data not found";
                    return ;
                }
            }
            p=new Node;
            p->next =ptr;
            preptr->next=p;
            p->data =new_data;
        }
    }
    void pop()
    {
        nodeptr ptr=head;
        nodeptr p;
        if(head->next==head)
        {
            delete ptr;
            head=NULL;
        }
        else
        {
            p=head;
            while(p->next!=head)
            {
                p=p->next;
            }
            p->next=ptr->next;
        }
        if(head!=NULL)
        {
            head=ptr->next;
            delete ptr;
        }
    }
    void del_data(int old_data)
    {
        nodeptr ptr=head;
        nodeptr preptr=ptr;
        if(head!=NULL && head->data!=old_data)
        {
            while(ptr->data!=old_data)
            {
                preptr=ptr;
                ptr=ptr->next;
                if(ptr==head)
                {
                    cout<<"data not found"<<endl;
                    return;
                }
            }
            preptr->next=ptr->next;
            delete ptr;
        }
        else if(head->data==old_data)
        {
            pop();
        }
    }
    void del_last()
    {
        if(head!=NULL)
        {
            nodeptr ptr=head;
            nodeptr preptr=ptr;
            if(head->next==head)
            {
                del_list();
            }
            else if(head!=NULL)
            {
                while(ptr->next!=head)
                {
                    preptr=ptr;
                    ptr=ptr->next;
                }
                preptr->next=head;
                delete ptr;
            }
        }
    }
    void del_after(int old_data)
    {
        nodeptr ptr=head;
        nodeptr preptr=ptr;
        ptr=ptr->next;
        if(head!=NULL)
        {
            while(preptr->data!=old_data)
            {
                preptr=ptr;
                ptr=ptr->next;
                if(ptr==head)
                {
                    cout<<"data not found"<<endl;
                    return;
                }
            }
            preptr->next=ptr->next;
            delete ptr;
        }
    }
    void display()
    {
        nodeptr p=head;
        if(head==NULL)
        {
            cout<<"\t\tEmpty"<<endl;
        }
        else
        {
            cout<<"\taddress"<<"\t\t\tdata"<<"\t\tnext"<<endl;
            do
            {
                cout<<"\t"<<p<<"\t\t"<<p->data<<"\t\t"<<p->next<<endl;
                p=p->next;
            }
            while(p!=head);
        }
    }
};
int main()
{
    linkList li;
    int x,a;
    int choice=0;
    while(choice!=10)
    {
        cout<<"\n\nYour Choice please: "<<endl;
        cout<<"0-Create "<<endl;
        cout<<"1-Inserting infront of list "<<endl;
        cout<<"2-Inserting at the end "<<endl;
        cout<<"3-Inserting after value a "<<endl;
        cout<<"4-Inserting before value a "<<endl;
        cout<<"5-Delete from front "<<endl;
        cout<<"6-Delete from the last "<<endl;
        cout<<"7-Delete value a"<<endl;
        cout<<"8-Delete after value a"<<endl;
        cout<<"9-Delete all list"<<endl;
        cout<<"10-Exit\n\n"<<endl;
        cout<<"\nYour choice: ";
        cin>>choice;
        cout<<"\tBEFORE LIST";
        li.display();
```

```
switch (choice)
{
case 0:
   li.create();
   break;
case 1:
   cout<<"enter data to insert: ";
   cin>>x;
   li.push(x);
   break;
case 2:
   cout<<"enter data to insert: ";
   cin>>x;
   li.ins(x);
   break;
case 3:
   cout<<"insert after: ";
   cin>>a;
   cout<<"enter data to insert: ";
   cin>>x;
   li.ins_after(a,x);
   break;
case 4:
   cout<<"insert before: ";
   cin>>a;
   cout<<"enter data to insert: ";
   cin>>x;
   li.ins_bef(a,x);
   break;
case 5:
   li.pop();
   break;
case 6:
   li.del_last();
   break;
case 7:
   cout<<"delete value: ";
```

```
          cin>>a;
          li.del_data(a);
          break;
       case 8:
          cout<<"Delete after: ";
          cin>>a;
          li.del_after(a);
          break;
       case 9:
       case 10:
          li.del_list();
          cout<<"this list is deleted!!"<<endl;
          break;
       }
       cout<<"\tAFTER LIST\n";
       li.display();
    }
    return 0;
}


/*WAP to implement circular linked list */
#include<iostream>
using namespace std;
class linkList
{
    struct Node
    {
       int data;
       Node *next;
    };
    typedef struct Node* nodeptr;
    nodeptr head;
public:
    linkList()     //constructor
    {
       head=NULL;
```

```
}
void del_list()     //delete whole list
{
   if(head!=NULL)
   {
      nodeptr p,q;
      p=head;
      do
      {
         q=p;
         p=p->next;
         delete q;
      }
      while(p!=head);
      head=NULL;
   }
}
void create()           // create linked list having some data
{
   nodeptr ptr=head;
   int val=0;
   cout<<"insert and end with -1"<<endl;
   cin>>val;

   while(val!=-1)
   {
      ins(val);
      cin>>val;
   }
}
void push(int new_data)    // insert at the front
{
   nodeptr p,ptr;
   p=new Node;
   p->data= new_data;
   p->next=head;
   ptr=head;
```

```
    if(head==NULL)
    {
        ins(new_data);
    }
    else
    {
        while(ptr->next!=head)      // for changing next of last
        {
            ptr=ptr->next;
        }
        head=p;
        ptr->next=head;
    }
}
void ins(int new_data)     //insert at the last
{
    nodeptr p;
    nodeptr ptr=head;
    if(head==NULL)
    {
        p= new Node;
        p->data= new_data;
        p->next=p;
        head=p;
    }
    else
    {
        while(ptr->next!=head)
        {
            ptr=ptr->next;
        }
        p=new Node;
        ptr->next=p;
        p->data =new_data;
        p->next =head;
    }
}
```

```
void ins_after(int old_data,int new_data)    //insert after certain data
{
   nodeptr p;
   nodeptr ptr=head;
   while(ptr->data!=old_data)
   {
      ptr=ptr->next;
      if(ptr==head)
      {
         cout<<"error data not found";
         return;
      }
   }
   p=new Node;
   p->next =ptr->next;
   ptr->next=p;
   p->data =new_data;
}
void ins_bef(int old_data,int new_data)    //insert before certain data
{
   nodeptr p,preptr;
   nodeptr ptr=head;
   if(ptr->data==old_data)
   {
      push(new_data);
   }
   else
   {
      while(ptr->data!=old_data)
      {
         preptr=ptr;
         ptr=ptr->next;
         if(ptr==head)
         {
            cout<<"error data not found";
            return ;
         }
```

```
            }
            p=new Node;
            p->next =ptr;
            preptr->next=p;
            p->data =new_data;
        }
    }
    void pop()           // delete from the front
    {
        nodeptr ptr=head;
        nodeptr p;
        if(head->next==head)
        {
            delete ptr;
            head=NULL;
        }
        else
        {
            p=head;
            while(p->next!=head)
            {
                p=p->next;
            }
            p->next=ptr->next;
        }
        if(head!=NULL)
        {
            head=ptr->next;
            delete ptr;
        }
    }
    void del_data(int old_data)     // delete the specified data
    {
        nodeptr ptr=head;
        nodeptr preptr=ptr;
        if(head!=NULL && head->data!=old_data)
        {
```

```
      while(ptr->data!=old_data)
      {
         preptr=ptr;
         ptr=ptr->next;
         if(ptr==head)
         {
            cout<<"data not found"<<endl;
            return;
         }
      }
      preptr->next=ptr->next;
      delete ptr;
   }
   else if(head->data==old_data)
   {
      pop();
   }
}
void del_last()      //delete the last data
{
   if(head!=NULL)
   {
      nodeptr ptr=head;
      nodeptr preptr=ptr;
      if(head->next==head)
      {
         del_list();
      }
      else if(head!=NULL)
      {
         while(ptr->next!=head)
         {
            preptr=ptr;
            ptr=ptr->next;
         }
         preptr->next=head;
         delete ptr;
      }
   }
}
void del_after(int old_data)     // delete data after the specified data
{
   nodeptr ptr=head;
   nodeptr preptr=ptr;
   ptr=ptr->next;
   if(head!=NULL)
   {
      while(preptr->data!=old_data)
      {
         preptr=ptr;
         ptr=ptr->next;
         if(ptr==head)
         {
            cout<<"data not found"<<endl;
            return;
         }
      }
      preptr->next=ptr->next;
      delete ptr;
   }
}
void display()     // display the list
{
   nodeptr p=head;
   cout<<"\n\t================X==============="<<endl;
   if(head==NULL)
   {
      cout<<"\t\tEmpty"<<endl;
   }
   else
   {
      cout<<"\taddress"<<"\t\tdata"<<"\tnext"<<endl;
      do
      {
            cout<<"\t"<<p<<"\t"<<p->data<<"\t"<<p->next<<endl;
            p=p->next;
         }
         while(p!=head);
      }
      cout<<"\tthats it"<<endl;
      cout<<"\t=================X================\n"<<endl;
   }
};
int main()
{
   linkList li;
   int x,a;
   int choice=0;
   while(choice!=10)
   {
      cout<<"\n\nyour Choice please: "<<endl;
      cout<<"0-create "<<endl;
      cout<<"1-inserting infront of list "<<endl;
      cout<<"2-inserting at the end "<<endl;
      cout<<"3-inserting after value a "<<endl;
      cout<<"4-inserting before value a "<<endl;
      cout<<"5-delete from front "<<endl;
      cout<<"6-delete from the last "<<endl;
      cout<<"7-delete value a"<<endl;
      cout<<"8-delete after value a"<<endl;
      cout<<"9-delete all list"<<endl;
      cout<<"10-Exit\n\n"<<endl;
      cout<<"\t\tyour choice: ";
      cin>>choice;
      system("CLS");
      cout<<"\tBEFORE LIST";
      li.display();
      switch (choice)
      {
      case 0:
         li.create();
```

```
    break;
case 1:
    cout<<"enter data to insert: ";
    cin>>x;
    li.push(x);
    break;
case 2:
    cout<<"enter data to insert: ";
    cin>>x;
    li.ins(x);
    break;
case 3:
    cout<<"insert after: ";
    cin>>a;
    cout<<"enter data to insert: ";
    cin>>x;
    li.ins_after(a,x);
    break;
case 4:
    cout<<"insert before: ";
    cin>>a;
    cout<<"enter data to insert: ";
    cin>>x;
    li.ins_bef(a,x);
    break;
case 5:
    li.pop();
    break;
case 6:
    li.del_last();
    break;
case 7:
    cout<<"delete value: ";
    cin>>a;
    li.del_data(a);
    break;
case 8:
```

```
            cout<<"Delete after: ";
            cin>>a;
            li.del_after(a);
            break;
        case 9:
        case 10:
            li.del_list();
            cout<<"this list is deleted!!"<<endl;
            break;
        }
        cout<<"\tAFTER LIST";
        li.display();
    }
    cout<<"\n============X==========="<<endl;
    cout<<"\t THANK YOU "<<endl;
    return 0;
}
```