```
/*WAP to implement doubly linked list */
#include<iostream>
#include<cstdlib>
using namespace std;
struct node
{
   int info;
   node *prev,*next;
};
class Dlist
{
   node *START;
public:
   Dlist():START(NULL) {}
   void 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;
         temp->prev=NULL;
         if(START==NULL)
         {
            temp->next=NULL;
            START=temp;
         }
         else
         {
            temp->next=START;
            START->prev=temp;
            START=temp;
         }
      }
   }
   void 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;
      temp->next=NULL;
      if(START==NULL)
      {
         temp->prev=NULL;
         START=temp;
      }
      else
      {
         node *ptr;
         ptr=START;
         while(ptr->next!=NULL)
         {
            ptr=ptr->next;
         }
         ptr->next=temp;
         temp->prev=ptr;
      }
   }

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

      }
```

```
}
void DeleteNodeBeforeGivenData(int data)
{
   if(START->info==data)
      cout<<"\n\nThere is no node before given data.\n\n";
   else if(START->next->info==data)
   {
      node *temp;
      temp=START;
      START=START->next;
      START->prev=NULL;
      cout<<"\n\nThe deleted value of node is :  "<<temp->info<<endl<<endl;
      delete temp;
   }
   else
   {
      int c=0;
      node*ptr;
      ptr=START->next->next;
      while(ptr!=NULL)
      {
         if(ptr->info==data)
         {
            c=1;
            node *temp;
            temp=ptr->prev;
            ptr->prev=ptr->prev->prev;
            temp->prev->next=ptr;
            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 linked list.\n\n";
```

```
    }
}
void DeleteNodeAfterGivenData(int data)
{
    int c=0;
    node *ptr;
    ptr=START;
    while(ptr!=NULL)
    {
        if(ptr->info==data)
        {
            c=1;
            if(ptr->next==NULL)
            {
                cout<<"\n\nThere is no node after given data.\n\n";
                break;
            }
            else
            {
                node *temp;
                temp=ptr->next;
                ptr->next=temp->next;
                if(ptr->next!=NULL)
                    temp->next->prev=ptr;
                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 display()
{
        if(START==NULL)
        {
            cout<<"\n\nThere is no existing linked list.\n\n";
            return false;
        }
        else
        {
            node *temp;
            temp=START;
            cout<<"\n\nElements of linked list are : \n";
            while(temp!=NULL)
            {
                cout<<temp->info<<"\t";
                temp=temp->next;
            }
            cout<<"\n\n";
            return true;
        }
    }
};
int main()
{
    Dlist 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 doubly 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 doubly linked list */
#include<iostream>
#include<cstdlib>
using namespace std;
struct node
{
    int data;
    node* next;
    node* prev;
};
class dlinkedlist
{
    node *head;
public:
    dlinkedlist()
    {
        head = NULL;
    }
    void create_dlinkedlist()
    {
        int val = 0;
        while(val != -1)
        {
            cout<<"\nEnter a value(-1 to stop): ";
            cin>>val;
            if(val != -1)
            {
                node *newNode = new node;
                newNode->data = val;

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

```
        newNode->prev = ptr;
    }
}
void insert_beg_dlinkedlist(int n)
{
    node*tmp = new node;
    tmp->data = n;
    tmp-> next = head;
    head->prev = tmp;
    tmp->prev = NULL;
    head = tmp;
}
void insert_before_dlinkedlist(int n, int val)
{
    if(head->data == n)
    {
        insert_beg_dlinkedlist(val);
    }
    else
    {
        node *newNode = new node;
        newNode->data = val;
        node *ptr = head;
        while(ptr->data != n)
        {
            ptr = ptr->next;
        }
        newNode->prev = ptr->prev;
        ptr->prev->next = newNode;
        newNode->next = ptr;
        ptr->prev = newNode;
    }
}
void insert_after_dlinkedlist(int n, int val)
{
    node *newNode = new node;
    newNode->data = val;
    node *ptr = head;
    while (ptr->data != n)
    {
        ptr = ptr->next;
    }
    if(ptr->next == NULL)
    {
        newNode->prev = ptr;
        ptr->next = newNode;
        newNode->next = NULL;
    }
    else
    {
        newNode->next = ptr->next;
        ptr->next->prev = newNode;
        newNode->prev = ptr;
        ptr->next = newNode;
    }
}
void delete_beg_dlinkedlist()
{
    if(head == NULL)
    {
        cout<<"\nList is empty!!\n";
    }
    else
    {
        node *ptr = head;
        if(head->next == NULL)
        {
            head = NULL;
        }
        else
        {
            head = head->next;
            head->prev = NULL;
            delete ptr;
        }
    }
}
void delete_end_dlinkedlist()
{
    node *ptr =head;
    while(ptr->next != NULL)
    {
        ptr = ptr->next;
    }
    if(ptr == head)
    {
        cout<<"\nList is empty!!\n";
    }
    else
    {
        ptr->prev->next = NULL;
        delete ptr;
    }
}
void delete_node_dlinkedlist(int n)
{
    node *ptr = head;
    while(ptr->data != n)
    {
        ptr = ptr->next;
    }
    if(ptr == head)
    {
        delete_beg_dlinkedlist();
    }
    else if(ptr->next == NULL)
    {
        delete_end_dlinkedlist();
    }
    else
    {
        ptr->prev->next = ptr->next;
        ptr->next->prev = ptr->prev;
        delete ptr;
    }
}
void delete_before_dlinkedlist(int n)
{
    node *ptr= head;
    if(ptr->data == n)
    {
        cout<<"\nNo node to delete!\n";
    }
    else
    {
        while(ptr->next->data != n)
        {
            ptr = ptr->next;
        }
        if(ptr->data == head->data)
        {
            ptr->next->prev = NULL;
            head = ptr->next;
            delete ptr;
        }
        else
        {
            ptr->prev->next = ptr->next;
            ptr->next->prev = ptr->prev;
            delete ptr;
        }
    }
}
void delete_after_dlinkedlist(int n)
{
    node *ptr= head;
    while(ptr->data != n)
    {
        ptr = ptr->next;
    }
    if(ptr->next == NULL)
    {
        cout<<"\nNO node to delete after\n";
    }
    else
    {
        ptr = ptr->next;
        if(ptr->next == NULL)
        {
            delete_end_dlinkedlist();
        }
        else
        {
            ptr->prev->next = ptr->next;
            ptr->next->prev = ptr->prev;
            delete ptr;
        }
    }
}
void delete_dlinkedlist()
{
    while(head->next != NULL)
    {
        delete_beg_dlinkedlist();
    }
    delete_beg_dlinkedlist();
}
void display_dlinkedlist()
{
    node *ptr = head;
    if(head == NULL)
    {
        cout<<"\nThe list is empty!!"<<endl;
    }
    else
```

```
        {
           cout<<"List:\n";
           while(ptr != NULL)
           {
              cout<<" "<<ptr->data<<" ";
              ptr = ptr->next;
           }
           cout<<endl<<endl;
        }
     }

  };
  int main()
  {
     dlinkedlist listobj;
     int choose;
     do
     {
        cout<<"\n\n1.  Create a doubly linked list."<<endl;
        cout<<"2.  Insert a node at beginningg."<<endl;
        cout<<"3.  Insert a node at end."<<endl;
        cout<<"4.  Insert a node before a given node."<<endl;
        cout<<"5.  Insert a node after a given node."<<endl;
        cout<<"6.  Delete a node at beginning."<<endl;
        cout<<"7.  Delete a node at last."<<endl;
        cout<<"8.  Delete a given node."<<endl;
        cout<<"9.  Delete a node before a given node."<<endl;
        cout<<"10. Delete a node after a given node."<<endl;
        cout<<"11. Delete the entire doubly linked list."<<endl;
        cout<<"12. Exit"<<endl;
        cout<<"\n\n\tChoose an option: ";
        cin>>choose;
        switch (choose)
        {
        case 1:
        {
           listobj.create_dlinkedlist();
```

```
    break;
}
case 2:
{
    int val;
    cout<<"\nenter the number to insert at the beginning: ";
    cin>>val;
    listobj.insert_beg_dlinkedlist(val);
    break;
}

case 3:
{
    int val;
    cout<<"\nenter the number to insert at end: ";
    cin>>val;
    listobj.insert_end_dlinkedlist(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_dlinkedlist(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_dlinkedlist(n,val);
```

```
            break;
         }
         case 6:
         {
            listobj.delete_beg_dlinkedlist();
            break;
         }
         case 7:
         {
            listobj.delete_end_dlinkedlist();
            break;
         }
         case 8:
         {
            int n;
            cout<<"\nEnter the node value to delete: ";
            cin>>n;
            listobj.delete_node_dlinkedlist(n);
            break;
         }
         case 9:
         {
            int n;
            cout<<"\nEnter the node value whose preceeding value is to be deleted:
";
            cin>>n;
            listobj.delete_before_dlinkedlist(n);
            break;
         }
         case 10:
         {
            int n;
            cout<<"\nEnter the node value whose succeeding value is to be deleted:
";
            cin>>n;
            listobj.delete_after_dlinkedlist(n);
            break;
```

```
        }
        case 11:
        {
           listobj.delete_dlinkedlist();
           break;
        }
        case 12:
        {
           exit(1);
           break;
        }
        default :
        {
           cout<<"Invalid input";
           break;
        }
        }
        listobj.display_dlinkedlist();
     }
     while (choose!=12);
     return 0;
}


/*WAP to implement doubly linked list */
#include<iostream>
using namespace std;
struct node
{
   int data;
   struct node * next;
   struct node * prev;
};
struct node * start;
struct node * newnode,* temp,* ptr;
void insert_end();
void creation()
```

```
{
        newnode = new node;
        cout<<"Enter the data for the list(insert -1 to end the list): ";
        cin>>newnode->data;
        newnode->prev=NULL;
        newnode->next=NULL;
        if (start==NULL)
        {
           start=newnode;
           temp=newnode;
        }
        else
        {
           temp->next=newnode;
           temp=newnode;
        }
    do{
        insert_end();
    }while (newnode->data!=-1);
}
void insert_end()
{
    newnode=new node;
    cout<<"Enter the data to be stored at the end: "<<endl;
    cin>>newnode->data;
    if (newnode->data!=-1)
    {
        newnode->next=NULL;
        newnode->prev=NULL;
        ptr=start;
        while(ptr->next!=NULL)
        {
           ptr=ptr->next;
        }
        ptr->next=newnode;
        newnode->prev=ptr;
    }
```

```
}
void insert_begin()
{
   newnode = new node;
   cout<<"Enter the data to be inserted at the beginning"<<endl;
   cin>>newnode->data;
   newnode->prev=NULL;
   newnode->next=start;
   start->prev=newnode;
   start=newnode;
}
void insert_afternode()
{
   int val;
   newnode = new node;
   cout<<"Enter after which value you want to insert: "<<endl;
   cin>>val;
   cout<<"Enter the new data you want to insert: "<<endl;
   cin>>newnode->data;
   ptr=start;
   while(ptr->data!=val)
   {
      ptr=ptr->next;
      if(ptr==NULL)
      {
         cout<<"error data not found";
      }
   }
   newnode->next=ptr->next;
   newnode->prev=ptr;
   ptr->next->prev=newnode;
   ptr->next=newnode;
}
void insert_beforenode()
{
   int val;
   newnode = new node;
```

```
    cout<<"Enter before which value you want to insert: "<<endl;
    cin>>val;
    cout<<"Enter the new data you want to insert: "<<endl;
    cin>>newnode->data;
    ptr=start;
    while(ptr->data!=val)
    {
        ptr=ptr->next;
        if(ptr==NULL)
        {
            cout<<"error data not found";
        }
    }
    newnode->next=ptr;
    newnode->prev=ptr->prev;
    ptr->prev->next=newnode;
    ptr->prev=newnode;
}
void del_end()
{
    ptr=start;
    while(ptr->next!=NULL)
    {
        ptr=ptr->next;
    }
    cout<<"The deleted value is: "<<ptr->data;
    ptr->prev->next=NULL;
    delete ptr;
}
void del_begin()
{
    ptr=start->next;
    delete start;
    start=ptr;
    ptr->prev=NULL;
}
void del_node()
```

```
{
   int val;
   cout<<"Enter the value of node which you want to delete: "<<endl;
   cin>>val;
   ptr=start;
   while(ptr->data!=val)
   {
      ptr=ptr->next;
   }
   ptr->prev->next=ptr->next;
   ptr->next->prev=ptr->prev;
   delete ptr;
}
void del_after ()
{
   int val;
   cout<<"Enter the value of node after which you want to delete: "<<endl;
   cin>>val;
   ptr=start;
   while(ptr->data!=val)
   {
      ptr=ptr->next;
   }
   temp=ptr->next;
   ptr->next=temp->next;
   temp->next->prev=ptr;
   delete temp;
}
void display_list()
{
   ptr=start;
   cout<<"\n\nThe list is: "<<endl;
   cout<<"\t\t\t"<<ptr->data<<endl;
   while(ptr->next!=NULL)
   {
      ptr=ptr->next;
      if (ptr->data==-1)
```

```
            break;
        cout<<"\t\t\t"<<ptr->data<<endl;
    }
}
int main()
{
    start=NULL;
    int choice=0,c=0;
    while(choice!=10){
        c++;
        cout<<"\n\nyour Choice please: "<<endl;
        if (c==1){cout<<"0-Creating a new list "<<endl;}
        cout<<"1-Inserting in front of list "<<endl;
        cout<<"2-Inserting at the end "<<endl;
        cout<<"3-Inserting after value a node "<<endl;
        cout<<"4-Inserting before value a node"<<endl;
        cout<<"5-Delete from front "<<endl;
        cout<<"6-Delete from the last "<<endl;
        cout<<"7-Delete a node"<<endl;
        cout<<"8-Delete after value a node"<<endl;
        cout<<"10-Exit.\n"<<endl;
        cout<<"\t\tyour choice: ";
        cin>>choice;
        switch (choice){
        case 0:
            creation();
            break;
        case 1:
            insert_begin();
            break;
        case 2:
            insert_end();
            break;
        case 3:
            insert_afternode();
            break;
        case 4:
            insert_beforenode();
            break;
        case 5:
            del_begin();
            break;
        case 6:
            del_end();
            break;
        case 7:
            del_node();
            break;
        case 8:
            del_after();
            break;
        }
        display_list();
    }
    cout<<"THANK YOU";
}
```

```
/*WAP to implement doubly linked list */
#include<iostream>
using namespace std;
class linkList
{
   struct Node
   {
      Node *prev;
      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;
      while(p!=NULL)
      {
         q=p;
         p=p->next;
         delete q;
      }
      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;
   if(head==NULL)
   {
      ins(new_data);
   }
   else
   {
      p=new Node;
      p->prev=NULL;
      p->data= new_data;
      p->next=head;
      head->prev=p;
      head=p;
   }
}
void ins(int new_data)     //insert at the last
{
   nodeptr p;
   nodeptr ptr=head;
   if(head==NULL)
   {
      p=new Node;
      p->prev=NULL;
      p->next=NULL;
      p->data=new_data;
      head=p;
   }
   else
   {
      while(ptr->next!=NULL)
      {
         ptr=ptr->next;
      }
      p=new Node;
      p->prev=ptr;
      ptr->next=p;
      p->data =new_data;
      p->next =NULL;
   }
}
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==NULL)
      {
         cout<<"error data not found";
         return;//exit(1);
      }
   }
   p=new Node;
   p->prev=ptr;
   p->next =ptr->next;
   ptr->next=p;
   ptr->next->prev=p;
   p->data =new_data;
}
void ins_bef(int old_data,int new_data)    //insert before certain data
{
   nodeptr p;
   nodeptr ptr=head;
   if(ptr->data==old_data)
   {
      push(new_data);
   }
   else
   {
      while(ptr->data!=old_data)
      {
         ptr=ptr->next;
         if(ptr==NULL)
         {
            cout<<"error data not found";
            return;//exit(1);
         }
```

```
        }
        p=new Node;
        p->prev=ptr->prev;
        p->next =ptr;
        ptr->prev=p;
        p->prev->next=p;
        p->data =new_data;
    }
}
void pop()           // delete from the front
{
    nodeptr ptr=head;
    if(head!=NULL)
    {
        head=ptr->next;
        head->prev=NULL;
        delete ptr;
    }
    else
    {
        cout<<"Empty"<<endl;
    }
}
void del_data(int old_data)     // delete the specified data
{
    nodeptr ptr=head;
    //nodeptr preptr=ptr;
    if(head!=NULL)
    {
        while(ptr->data!=old_data)
        {
            ptr=ptr->next;
            if(ptr==NULL)
            {
                cout<<"data not found"<<endl;
                return;//exit(1);
            }
        }
        ptr->prev->next=ptr->next;
        ptr->next->prev=ptr->prev;
        delete ptr;
    }
}
void del_last()      //delete the last data
{
    if(head!=NULL)
    {
        nodeptr ptr=head;
        //nodeptr preptr=ptr;
        if(head!=NULL)
        {
            while(ptr->next!=NULL)
            {
                ptr=ptr->next;
            }
            ptr->prev->next=NULL;
            delete ptr;
        }
    }

}
void del_after(int old_data)     // delete data after the specified data
{
    nodeptr ptr=head,p;
    //nodeptr preptr=ptr;
    if(head!=NULL)
    {
        while(ptr->data!=old_data)
        {
            ptr=ptr->next;
            if(ptr==NULL)
            {
                cout<<"data not found"<<endl;
                return;//exit(1);
            }
        }
        p=ptr->next;
        ptr->next=p->next;
        p->next->prev=ptr;
        delete p;
    }
}
void del_before(int old_data)     // delete data before the specified data
{
    nodeptr ptr=head,p;
    //nodeptr preptr=ptr;
    if(head!=NULL)
    {
        while(ptr->data!=old_data)
        {
            ptr=ptr->next;
            if(ptr==NULL)
            {
                cout<<"data not found"<<endl;
                return;//exit(1);
            }
        }
        p=ptr->prev;
        ptr->prev=p->prev;
        p->prev->next=ptr;
        delete p;
    }
}
void display()
{
    nodeptr p=head;
    cout<<"\n\t===============X==============="<<endl;
    cout<<"\taddress"<<"\t\tdata"<<"\tnext"<<endl;
    while(p!=NULL)
    {
        cout<<"\t"<<p<<"\t"<<p->data<<"\t"<<p->next<<endl;
```

```
         p=p->next;
      }
      if(head==NULL)
      {
         cout<<"\tEmpty"<<endl;
      }
      cout<<"\tthats it"<<endl;
      cout<<"\t==================X=================\n"<<endl;
   }
   void displaydetail()      // display the list
   {
      nodeptr p=head;

cout<<"\n\t==========================X==========================="<<
endl;
      //cout<<"\tprev\t\taddress\t\tdata\tnext"<<endl;
      while(p!=NULL)
      {

         cout<<"\tpre: "<<p->prev<<"\t\tadd: "<<p<<"\t\tval: "<<p-
>data<<"\t\tnex: "<<p->next<<endl;
         p=p->next;
      }
      if(head==NULL)
      {
         cout<<"\t\t\tEmpty"<<endl;
      }
      cout<<"\t\t\tthats it"<<endl;

cout<<"\t=========================X===========================\n"<<
endl;
   }
};
int main()
{
   linkList li;
   int x,a;
```

```
    int choice=0;
    while(choice!=12)
    {
       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 before value a"<<endl;
       cout<<"10-delete all list"<<endl;
       cout<<"11-display detailed"<<endl;
       cout<<"12-Exit\n"<<endl;
       cout<<"*** auto display doesnt show previous address so try option 11 to
see it***"<<endl;
       cout<<"\t\tyour choice: ";
       cin>>choice;
       system("CLS");
       if(choice<10)
       {
          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<<"Inserting after: ";
   cin>>a;
   cout<<"enter data to insert: ";
   cin>>x;
   li.ins_after(a,x);
   break;
case 4:
   cout<<"inserting 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:
   cout<<"Delete before : ";
```

```
            cin>>a;
            li.del_before(a);
            break;
        case 10:
        case 12:
            li.del_list();
            cout<<"this list is deleted!!\n"<<endl;
            break;
        case 11:
            li.displaydetail();

        }
        if(choice<10)
        {
            cout<<"\tAFTER LIST";
            li.display();
        }
    }
    cout<<"\n===========X==========="<<endl;
    cout<<"\t THANK YOU "<<endl;
    return 0;
}
```