**Pseudo Code [Handwritten]**

**Code: [Paste your code here]**

#include<iostream>

using namespace **std**;

class **Node**{

    public:

    int data;

**Node**\* next;

**Node**(int val)

    {

        data = val;

        next = nullptr;

    }

};

*//u need to keep the head private*

class **LinkedList**{

    private:

**Node**\* head;

*// head = nullptr which means initially wed have 0 nodes*

    public:

**LinkedList**(){

        head = nullptr;

    }

    void **insert\_at\_beginning**(int data)

    {

**Node**\* temp = new **Node**(data);

        temp -> next = head;

        head = temp;

    }

    void **insert\_at\_end**(int data)

    {

**Node**\* new\_node = new **Node**(data);

        if(head == nullptr){

            head = new\_node;

            return;

        }

**Node**\* temp = head;

        while(temp -> next!=nullptr){

            temp = temp -> next;

        }

        temp -> next = new\_node;

    }

    void **insert\_at\_between**(int data, int pos)

    {

        if(pos<0){

            cout **<<** "entered invalid position!" ;

            return;

        }

        if(pos==0){

**insert\_at\_beginning**(data);

            return;

        }

**Node**\* temp = head;

        int count = 0;

        while(count < pos - 1 && temp -> next != nullptr)

        {

            temp = temp -> next;

            count++;

        }

        if(temp ->next == nullptr){

            cout **<<** "end of list" ;

            return;

        }

**Node**\* newNode = new **Node**(data);

        newNode -> next = temp -> next;

        temp -> next = newNode;

    }

    void **display**()

    {

        if(head == nullptr){

            cout **<<** "The linked list does not exist" **<<** **endl**;

            return;

        }

**Node**\* temp = head;

        while(temp!=nullptr){

            cout **<<** temp -> data**<<**" ";

            temp = temp -> next;

        }

    }

    bool **search**(int key)

    {

**Node**\* temp = head;

        while(temp!=nullptr){

            if(temp -> data == key){

                return true;

            }

            temp = temp -> next;

        }

        return false;

    }

*//1. Count the number of Nodes in the LinkedList*

    int **count**(int key)

    {

**Node**\* temp = head;

        int count = 0;

        while(temp!= nullptr){

            if(temp -> data == key){

                count++;

            }

            temp = temp->next;

        }

        return count;

    }

*//2. Count the number of Odd and Even number of elements in the LinkedList*

    void **even\_odd\_count**()

    {

**Node**\* temp = head;

        int even = 0;

        int odd = 0;

        while(temp!= nullptr){

            if(temp -> data % 2 == 0){

                even++;

            }

            else{

                odd++;

            }

            temp = temp->next;

        }

        cout **<<** "Even elements: " **<<** even **<<** **endl**;

        cout **<<** "Odd elements: " **<<** odd **<<** **endl**;

        return;

    }

*//3. Split the given linkedlist into two sub linkedlists such that one linkedlist has odd elements while other has even*

    void **OddEven\_subLL**()

    {

        if(head==nullptr){

            cout**<<**"Linked List is empty"**<<endl**;

            return;

        }

**LinkedList** odd;

**LinkedList** even;

**Node**\* temp=head;

        while (temp!=nullptr)

        {

            if(temp->data%2==0){

                even.**insert\_at\_end**(temp->data);

            }

            else{

                odd.**insert\_at\_end**(temp->data);

            }

            temp=temp->next;

        }

        cout**<<**"Odd LL is"**<<endl**;

        odd.**display**();

        cout **<<** **endl**;

        cout**<<**"even LL is"**<<endl**;

        even.**display**();

        return;

    }

    void **delete\_node**(int key)

    {

**Node**\* temp = head;

        if(head == nullptr){

            cout **<<** "Linked List doesn't exist";

            return;

        }

*// If the node to be deleted is the head node*

        if (temp->data == key) {

            head = temp->next; *// Move head to the next node*

            delete temp; *// Delete the old head*

            return;

        }

*// Traverse the list to find the node to delete*

**Node**\* prev = nullptr;

        while (temp != nullptr && temp->data != key) {

            prev = temp;

            temp = temp->next;

        }

*// If the key was not found in the list*

        if (temp == nullptr) {

            cout **<<** "Node with value " **<<** key **<<** " not found in the list." **<<** **endl**;

            return;

        }

*// Unlink the node from the list*

        prev->next = temp->next;

        delete temp;

    }

};

int **main**(){

**LinkedList** list;

    list.**insert\_at\_end**(50);

    list.**insert\_at\_end**(33);

    list.**insert\_at\_end**(20);

    list.**insert\_at\_end**(10);

    list.**insert\_at\_end**(40);

    list.**display**();

    int key = 20;

    cout **<<** **endl**;

    int occurence = list.**count**(key) ;

    cout **<<** occurence **<<** **endl**;

    cout **<<** **endl**;

    list.**even\_odd\_count**();

    list.**OddEven\_subLL**();

    return 0;

}

**Output: [Paste your output here]**

**POST-LAB Exercise [Handwritten-Write the code]**