Q1) linked list implementation in C++

#include<iostream>

using namespace std;

class Node

{

public:

    int data;

    Node \*ref;

    Node(int data)

    {

        this->data = data;

        this->ref = NULL;

    }

};

class LinkedList

{

private:

    Node \*temp;

    Node \*head;

public:

    LinkedList()

    {

        temp = NULL;

        head = NULL;

    }

    void Display()

    {

        int count = 0;

        temp = head;

        if (temp == NULL)

        {

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

        }

        else

        {

            while (temp != NULL)

            {

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

                count++;

                temp = temp->ref;

            }

        }

    }

    void add\_beginning(int data)

    {

        Node \*newNode = new Node(data);

        temp = head;

        newNode->ref = temp;

        temp = newNode;

        head = newNode;

    }

    void add\_end(int data)

    {

        Node \*newNode = new Node(data);

        temp = head;

        if (temp == NULL)

        {

            temp = newNode;

        }

        else

        {

            while (temp != NULL)

            {

                temp = temp->ref;

            }

            temp->ref = newNode;

        }

    }

    void add\_after(int after, int data)

    {

        Node \*newNode = new Node(data);

        int c = 0;

        temp = head;

        while (temp != NULL)

        {

            if (after == temp->data)

            {

                break;

            }

            c++;

        }

        while (c > 1)

        {

            temp = temp->ref;

            c--;

        }

        if (temp == NULL)

        {

            cout << "Element not in list." << endl;

        }

        else

        {

            newNode->ref = temp->ref;

            temp->ref = newNode;

        }

    }

    void del\_beginning()

    {

        temp = head;

        if(temp == NULL)

        {

            cout << "Empty Linked List."  << endl;

        }

        else

        {

            temp = temp -> ref;

        }

    }

    void del\_end()

    {

        temp = head;

        if (temp == NULL)

        {

            cout << "Empty Linked List." << endl;

        }

        else if (temp -> ref == NULL)

        {

            temp = NULL;

        }

        else

        {

            while(temp->ref->ref != NULL)

            {

                temp = temp -> ref;

            }

            temp -> ref = NULL;

        }

    }

    void del\_entry(int value)

    {

        temp = head;

        if( temp == NULL)

        {

            cout << "Empty Linked List." << endl;

        }

        else if( temp -> ref == NULL)

        {

            if(temp -> data == value)

            {

                temp = NULL;

            }

            else

            {

                cout << "Element not found." << endl;

            }

        }

        else

        {

            int c = 0;

            while( temp -> data !=  value)

            {

                c++;

                temp = temp -> ref;

            }

            temp = head;

            while(c > 1)

            {

                temp = temp -> ref;

                c--;

            }

            temp -> ref = temp -> ref -> ref;

        }

    }

};

int main()

{

    LinkedList LL1;

    LL1.add\_beginning(90);

    LL1.add\_beginning(70);

    LL1.add\_beginning(80);

    LL1.add\_beginning(50);

    LL1.del\_end();

    LL1.add\_beginning(10);

    LL1.del\_entry(50);

    LL1.add\_after(10, 65);

    LL1.Display();

    return 0;

}

Q2) Linked List implementation in Python:

*class* Node:  
 *def* \_\_init\_\_(*self*, data):  
 *self*.data = data  
 *self*.ref = *None  
  
  
class* LinkedList:  
 *def* \_\_init\_\_(*self*):  
 *self*.temp = *None  
 self*.head = *None  
  
 def* Display(*self*):  
 count = 0  
 *self*.temp = *self*.head  
 *if self*.temp *is None*:  
 print("Linked List is empty.")  
 *else*:  
 *while self*.temp *is not None*:  
 print(*self*.temp.data, end=" ")  
 count += 1  
 *self*.temp = *self*.temp.ref  
  
 *def* add\_beginning(*self*, data):  
 newNode = Node(data)  
 *self*.temp = *self*.head  
 newNode.ref = *self*.temp *# First node links to Previous first node  
 self*.temp = newNode *# New head pointer is to new first node  
 self*.head = newNode  
  
 *def* add\_end(*self*, data):  
 newNode = Node(data)  
 *self*.temp = *self*.head  
 *if self*.temp *is None*:  
 *self*.temp = newNode  
 *else*:  
 *while self*.temp *is not None*:  
 *self*.temp = *self*.temp.ref *# Now head points to last node.  
 self*.temp.ref = newNode  
  
 *def* add\_after(*self*, data, after):  
 newNode = Node(data)  
 c = 0  
 *self*.temp = *self*.head  
 *while self*.temp *is not None*:  
 *if* after == *self*.temp.data:  
 *break* c += 1  
 *while* c > 1:  
 *self*.temp = *self*.temp.ref  
 c -= 1  
 *if self*.temp *is None*:  
 print("Element not in list.")  
 *else*:  
 newNode.ref = *self*.temp.ref  
 *self*.temp.ref = newNode  
  
 *def* del\_beginning(*self*):  
 *self*.temp = *self*.head  
 *if self*.temp *is None*:  
 print("Empty Linked List.")  
 *elif self*.temp.ref *is None*:  
 *self*.temp = *None  
 else*:  
 *self*.temp = *self*.temp.ref  
  
 *def* del\_end(*self*):  
 *self*.temp = *self*.head  
 *if self*.temp *is None*:  
 print("Empty Linked List.")  
 *elif self*.temp.ref *is None*:  
 *self*.temp = *None # One element in LL.  
  
 else*:  
 *while self*.temp.ref.ref *is not None*:  
 *self*.temp = *self*.temp.ref  
 *self*.temp.ref = *None  
  
 def* del\_entry(*self*, data):  
 *self*.temp = *self*.head  
 *if self*.temp *is None*:  
 print("Empty Linked List.")  
 *elif self*.temp.ref *is None*:  
 *if self*.temp.data == data:  
 *self*.temp = *None  
 else*:  
 print("Element Not Found.")  
 *else*:  
 c = 0  
 *while self*.temp.data != data:  
 c += 1  
 *self*.temp = *self*.temp.ref *# Now, points to the entry to delete  
 self*.temp = *self*.head *# c will count the index, and temp again sent to starting.  
 while* c > 1:  
 *self*.temp = *self*.temp.ref *# c - 1 times loop, i.e. we found the entry right before the required entry.* c -= 1  
 *self*.temp.ref = *self*.temp.ref.ref  
  
  
LL1 = LinkedList()  
LL1.add\_beginning(90)  
LL1.add\_beginning(70)  
LL1.add\_beginning(80)  
LL1.del\_end()  
LL1.add\_beginning(50)  
LL1.add\_beginning(10)  
LL1.del\_entry(50)  
LL1.add\_after(100, 10)  
LL1.Display()

Q3) Search for 7 and print the index:

*class* Node:  
 *def* \_\_init\_\_(*self*, data):  
 *self*.ref = *None  
 self*.data = data  
  
  
*class* LinkedList:  
 *def* \_\_init\_\_(*self*):  
 *self*.head = *None  
 self*.pointer = *None  
  
 def* Display(*self*):  
 *self*.pointer = *self*.head  
 *if self*.pointer *is None*:  
 print("Empty Linked List.")  
 *else*:  
 *while self*.pointer *is not None*:  
 print(*self*.pointer.data, end=" ")  
 *self*.pointer = *self*.pointer.ref  
  
 *def* addStart(*self*, data):  
 newNode = Node(data)  
 *self*.pointer = *self*.head  
 newNode.ref = *self*.pointer  
 *self*.pointer = newNode  
 *self*.head = newNode  
  
 *def* search(*self*, data):  
 *self*.pointer = *self*.head  
 count = 1  
 *while* data != *self*.pointer.data:  
 count += 1  
 *self*.pointer = *self*.pointer.ref  
 print("(Starting from 1)", data, "found at index ", count)  
  
  
LL = LinkedList()  
LL.addStart(3)  
LL.addStart(2)  
LL.addStart(8)  
LL.addStart(3)  
LL.addStart(7)  
LL.addStart(5)  
LL.addStart(1)  
LL.search(int(input("To search: ")))  
LL.Display()

Q4) Delete all numbers > 25:

*class* Node:  
 *def* \_\_init\_\_(*self*, data):  
 *self*.ref = *None  
 self*.data = data  
  
  
*class* LinkedList:  
 *def* \_\_init\_\_(*self*):  
 *self*.head = *None  
 self*.pointer = *None  
 self*.tail = *None  
  
 def* Display(*self*):  
 *self*.pointer = *self*.head  
 *if self*.pointer *is None*:  
 print("Empty Linked List.")  
 *else*:  
 *while self*.pointer *is not None*:  
 print(*self*.pointer.data, end=" ")  
 *self*.pointer = *self*.pointer.ref  
  
 *def* addStart(*self*, data):  
 newNode = Node(data)  
 *self*.pointer = *self*.head  
 newNode.ref = *self*.pointer  
 *self*.pointer = newNode  
 *self*.head = newNode  
  
 *def* delete(*self*, data):  
  
 *self*.pointer = *self*.head  
 count = 0  
 *if self*.pointer *is None*:  
 *return  
  
 if self*.pointer.data == data:  
 *self*.head = *self*.pointer.ref  
 *return  
 while* data != *self*.pointer.data *and self*.pointer *is not None*:  
 count += 1  
 *self*.pointer = *self*.pointer.ref  
 *self*.pointer = *self*.head  
 *while* count > 1:  
 *self*.pointer = *self*.pointer.ref  
 count -= 1  
 *self*.pointer.ref = *self*.pointer.ref.ref  
  
 *def* grtThan25(*self*):  
 *self*.pointer = *self*.head  
 *while self*.pointer.ref *is not None*:  
 *if self*.pointer.data > 25:  
 *self*.delete(*self*.pointer.data)  
 *self*.pointer = *self*.pointer.ref  
  
  
LL = LinkedList()  
n = int(input("Number of Entries: "))  
*for* i *in* range(n):  
 LL.addStart(int(input("Enter number between 1 & 50: ")))  
LL.grtThan25()  
LL.Display()