#include <iostream>

using namespace std;

class Node {

private:

int Number;

Node\* Next;

Node\* Prior;

public:

Node() {

this->Number = 0;

this->Next = nullptr;

this->Prior = nullptr;

}

Node(const int& n);

const int& GetNumber() const {

return Number;

}

const Node\* GetNext()const {

return Next;

}

const Node\* GetPrior()const {

return Prior;

}

friend class LinkedList;

};

Node::Node(const int& n) {

this->Number = n;

this->Next = nullptr;

this->Prior = nullptr;

}

class LinkedList {

private:

Node\* Head;

Node\* Tail;

public:

LinkedList() {

Head = nullptr;

Tail = nullptr;

}

void Clear();//清除堆空间

~LinkedList() { Clear(); }//析构函数

const Node\* GetHead()const { return Head; }

Node\* Add(const int& Data);//插入在尾部

//Node\* InsertFrist(const int& Data);//插入在第一个

const int GetCount();//计总数

const int GetThisCount(Node\* This);

int CheckMaxPlace();

void Combine(LinkedList& L1, LinkedList& L2);//链表递增合并到L1，将L2归空

void AntiCombine(LinkedList& L1, LinkedList& L2);//链表递减合并到L1，将L2归空

void Antitone(LinkedList& L1);//链表反序

void DeleteAB(LinkedList& L1,int Mink,int Maxk);

LinkedList &DifferenceSet(const LinkedList& L1,const LinkedList& L2);//求L1和L2差集

//const Node\* operator[] (int index)const;

//Node\* operator[](int index);

//Node\* Insert(const int& Data,int index);//插入在index处

};

void LinkedList::Clear() {

while (Head) {

Node\* newHead = Head->Next;

delete Head;

Head = newHead;

}

}

Node\* LinkedList::Add(const int& Data) {

Node\* newNode = new Node(Data);

if (!Tail)

Head = Tail = newNode;

else

{

Tail->Next = newNode;

newNode->Prior = Tail;

Tail = newNode;

}

return newNode;

}

void LinkedList::DeleteAB(LinkedList& L1, int Mink, int Maxk) {

Node\* Count = L1.Head;

do {

if (Count == L1.Head && Count->Number<Maxk && Count->Number>Mink) {

if (Count->Next) {

L1.Head = Count->Next;

delete Count;

Count = L1.Head;

}

else {

delete Count;

L1.Head = nullptr;

Count = nullptr;

}

}

else {

if (Count!=nullptr&&Count->Number<Maxk && Count->Number>Mink) {

Count->Prior->Next = Count->Next;

Count->Next->Prior = Count->Prior;

Node\* Temp = Count->Next;

delete Count;

Count = Temp;

}

else {

Count = Count->Next;

}

}

} while (Count != nullptr);

}

ostream& operator << (ostream& o, const LinkedList& list)

{

if (list.GetHead() != nullptr) {

for (const Node\* p = list.GetHead();

p;

p = p->GetNext())

{

o << "[" << p->GetNumber() << "]";

if (p->GetNext())

o << " -> ";

}

}

else {

o << "This list is empty.";

}

return o;

}

int main()

{

LinkedList Mylist1;

Mylist1.Add(1);

Mylist1.Add(2);

Mylist1.Add(7);

Mylist1.Add(16);

Mylist1.Add(22);

cout << "Mylist1 is" << endl;

cout << Mylist1 << endl;

Mylist1.DeleteAB(Mylist1, 2, 9);//删除大于2小于9的数

cout <<"Delete from 2 to 9"<<endl << "Mylist1 is" << endl << Mylist1 << endl;

Mylist1.DeleteAB(Mylist1, 0, 10);//删除大于0小于10的数

cout << "Delete from 0 to 10" << endl << "Mylist1 is" << endl << Mylist1 << endl;

Mylist1.DeleteAB(Mylist1, 0, 90);//删除大于0小于90的数

cout << "Delete from 0 to 90" << endl << "Mylist1 is" << endl<< Mylist1 << endl;

}

