1. Write a main function to do:

* Read 2 linked lists from the user (**L1 , L2**).

Assume that the user will enter same number of Nodes in both (**L1 , L2**).

Head

L1

Head

L2

* Then Remove any node from L2, in case it was equal to the correspondence node in L1.

Head

L2

* Assume that you will not face a deleted node at the first of the list, or at the tail of the list.

#include <iostream>

using namespace std;

class CNode

{

public:

int info;

CNode\* pNext;

};

class CList

{

public:

CNode\* pHead;

CNode\* pTail;

CList()

{

pHead = NULL;

pTail = NULL;

}

void Attach(CNode\* pnn)

{

if (pHead == NULL)

{

pHead = pnn;

pTail = pnn;

}

else

{

pTail->pNext = pnn;

pTail = pnn;

}

}

~CList()

{

CNode\* pTrav = pHead;

while (pHead != NULL)

{

pHead = pTrav->pNext;

pTrav->pNext = NULL;

delete pTrav;

pTrav = pHead;

}

}

};

void main()

{

CList L1;

CList L2;

int N;

CNode\* pnn1;

CNode\* pnn2;

cout << "Enter N \n";

cin >> N;

for (int i = 0; i < N; i++)

{

pnn1 = new CNode;

cout << "enter info list 1\n";

cin >> pnn1->info;

pnn1->pNext = NULL;

L1.Attach(pnn1);

pnn2 = new CNode;

cout << "enter info list 2\n";

cin >> pnn2->info;

pnn2->pNext = NULL;

L2.Attach(pnn2);

}

CNode\* pTrav1 = L1.pHead;

CNode\* pTrav2 = L2.pHead;

CNode\* pB = L2.pHead;

while (pTrav1->pNext!= NULL)

{

if (pTrav1->info == pTrav2->info)

{

pB->pNext = pTrav2->pNext;

pTrav2->pNext = NULL;

delete pTrav2;

pTrav2 = pB->pNext;

}

pTrav1 = pTrav1->pNext;

if (pTrav1->info != pTrav2->info)

{

pB = pTrav2;

pTrav2 = pTrav2->pNext;

}

}

//output

pTrav2 = L2.pHead;

while (pTrav2 != NULL)

{

cout << pTrav2->info << " ";

pTrav2 = pTrav2->pNext;

}

}

**(2)** Write a main function to do:

* Read 2 linked lists from the user (**L1 , L2**).

Assume that the number of nodes in (**L1**) will be even.

L1

Head

L2

Head

Total of nodes between

4 : 7

Total of nodes between

10 : 3

* For each pair of nodes from (**L1**), find the total between them in (**L2**).

#include <iostream>

using namespace std;

class CNode

{

public:

int info;

CNode\* pNext;

};

class CList

{

public:

CNode\* pHead;

CNode\* pTail;

CList()

{

pHead = NULL;

pTail = NULL;

}

void Attach(CNode\* pnn)

{

if (pHead == NULL)

{

pHead = pnn;

pTail = pnn;

}

else

{

pTail->pNext = pnn;

pTail = pnn;

}

}

~CList()

{

CNode\* pTrav = pHead;

while (pHead != NULL)

{

pHead = pTrav->pNext;

pTrav->pNext = NULL;

delete pTrav;

pTrav = pHead;

}

}

};

void main()

{

CList L1;

CList L2;

int N,check=0,tot=0;

CNode\* pnn1;

CNode\* pnn2;

cout << "Enter N L1\n";

cin >> N;

for (int i = 0; i < N; i++)

{

pnn1 = new CNode;

cout << "enter info list 1\n";

cin >> pnn1->info;

pnn1->pNext = NULL;

L1.Attach(pnn1);

}

cout << "Enter N L2\n";

cin >> N;

for (int i = 0; i < N; i++)

{

pnn2 = new CNode;

cout << "enter info list 2\n";

cin >> pnn2->info;

pnn2->pNext = NULL;

L2.Attach(pnn2);

}

CNode\* pTrav1 = L1.pHead, \* pTrav2 = L1.pHead->pNext, \* pTrav3 = L2.pHead, \* pTrav4 = L2.pHead->pNext;

while (pTrav1 != NULL && pTrav2 != NULL && pTrav3 != NULL && pTrav4 != NULL)

{

if (pTrav3->info == pTrav1->info && check==0)

{

check = 1;

tot += pTrav3->info;

}

else

{

if (check == 0) //here so it doesnt move tr3 again once start of range is met

{

pTrav3 = pTrav3->pNext;

pTrav4 = pTrav4->pNext;

}

}

if (pTrav4->info != pTrav2->info)

{

if (check == 1) //here so it doesnt enter else now & ensure tr3 reached start of range first

{

tot += pTrav4->info;

pTrav4 = pTrav4->pNext;

}

}

else

{

tot += pTrav4->info;

cout << "Total between " << pTrav1->info << " and " << pTrav2->info << ": " << tot<<"\n";

pTrav3 = pTrav3->pNext;

pTrav4 = pTrav3->pNext;

pTrav1 = pTrav2->pNext;

pTrav2 = pTrav1->pNext;

tot = 0;

check = 0;

}

}

}

**(3) Write a main function to do the followings:**

* Read 20 Lists from the user.

**L**

Head

**L[0]**

**..**

**..**

**..**

Head

**L[19]**

* For (L[0] & L[19]) cut the intervals that surrounded by negative values. Past the cutting intervals into a new list (NewL).

**NewL**

Head

* Also for (L[1] & L[18]) cut the intervals that surrounded by negative values. Past the cutting intervals into the same (NewL).
* And so on repeat the same step (Cut & Past) for the reminder Lists.

#include <iostream>

using namespace std;

class CNode

{

public:

int info;

CNode\* pNext;

};

class CList

{

public:

CNode\* pHead;

CNode\* pTail;

CList()

{

pHead = NULL;

pTail = NULL;

}

void Attach(CNode\* pnn)

{

if (pHead == NULL)

{

pHead = pnn;

pTail = pnn;

}

else

{

pTail->pNext = pnn;

pTail = pnn;

}

}

~CList()

{

CNode\* pTrav = pHead;

while (pHead != NULL)

{

pHead = pTrav->pNext;

pTrav->pNext = NULL;

delete pTrav;

pTrav = pHead;

}

}

};

void main()

{

CList L[20];

CList newL;

CNode\* pnn;

CNode\* pTrav1, \* pB1, \* pF1, \* pF2;

CNode\* pTrav2, \* pB2;

int N, check1 = 0, check2 = 0;

for (int j = 0; j < 20; j++)

{

cout << "Enter N for list " << j + 1 << "\n";

cin >> N;

for (int i = 0; i < N; i++)

{

pnn = new CNode;

cout << "enter info list\n";

cin >> pnn->info;

pnn->pNext = NULL;

L[j].Attach(pnn);

}

}

for (int j = 0; j < 10; j++)

{

pTrav1 = L[j].pHead, pB1 = L[j].pHead, pF1 = L[j].pHead;

pTrav2 = L[19 - j].pHead, pB2 = L[19 - j].pHead, pF2 = L[19 - j].pHead;

check1 = 0, check2 = 0;

while (pF1->pNext != NULL)

{

if (pTrav1->info > 0)

{

pB1 = pTrav1;

pTrav1 = pTrav1->pNext;

pF1 = pTrav1->pNext;

}

else

{

check1 = 1;

}

if (pF1->info > 0 && check1 == 1)

{

pF1 = pF1->pNext;

}

else if(check1==1)

{

break;

}

}

while (pF2->pNext != NULL)

{

if (pTrav2->info > 0)

{

pB2 = pTrav2;

pTrav2 = pTrav2->pNext;

pF2 = pTrav2->pNext;

}

else

{

check2 = 1;

}

if (pF2->info > 0 && check2 == 1)

{

pF2 = pF2->pNext;

}

else if (check2 == 1)

{

break;

}

}

if (newL.pHead == NULL)

{

newL.pHead = pTrav1;

pB1->pNext = pF1->pNext;

pF1->pNext = pTrav2;

pB2->pNext = pF2->pNext;

pF2->pNext = NULL;

newL.pTail = pF2;

}

else

{

newL.pTail->pNext = pTrav1;

pB1->pNext = pF1->pNext;

pF1->pNext = pTrav2;

pB2->pNext = pF2->pNext;

pF2->pNext = NULL;

newL.pTail = pF2;

}

}

//output

CNode\* pOut = newL.pHead;

while (pOut != NULL)

{

cout << pOut->info << " ";

pOut = pOut->pNext;

}

}

(4) Write a main function to do:

* Read 30 Lists from the user.

H

H

H

H

Find the (-1) in the first List

H

H

H

H

Cut the correspondence node from the next list& past it into the first List

* repeat this task for (3rd & 4th lists) & (5th & 6th ) and so on.

#include <iostream>

using namespace std;

class CNode

{

public:

int info;

CNode\* pNext;

};

class CList

{

public:

CNode\* pHead;

CNode\* pTail;

CList()

{

pHead = NULL;

pTail = NULL;

}

void Attach(CNode\* pnn)

{

if (pHead == NULL)

{

pHead = pnn;

pTail = pnn;

}

else

{

pTail->pNext = pnn;

pTail = pnn;

}

}

~CList()

{

CNode\* pTrav = pHead;

while (pHead != NULL)

{

pHead = pTrav->pNext;

pTrav->pNext = NULL;

delete pTrav;

pTrav = pHead;

}

}

};

void main()

{

CList L[30];

CNode\* pnn;

CNode\* pTrav1, \* pB1;

CNode\* pTrav2, \* pB2;

int N;

for (int j = 0; j < 30; j++)

{

cout << "Enter N for list " << j + 1 << "\n";

cin >> N;

for (int i = 0; i < N; i++)

{

pnn = new CNode;

cout << "enter info list\n";

cin >> pnn->info;

pnn->pNext = NULL;

L[j].Attach(pnn);

}

}

for (int j = 0; j < 30; j += 2)

{

pTrav1 = L[j].pHead, pB1 = L[j].pHead;

pTrav2 = L[j + 1].pHead, pB2 = L[j + 1].pHead;

while (pTrav1 != NULL)

{

if (pTrav1->info != -1)

{

pB1 = pTrav1;

pB2 = pTrav2;

pTrav1 = pTrav1->pNext;

pTrav2 = pTrav2->pNext;

}

else

{

break;

}

}

pB2 = pTrav2->pNext;

pTrav2->pNext = pTrav1->pNext;

pTrav1->pNext = pTrav2;

}

//output

CNode\* pOut = L[0].pHead;

while (pOut != NULL)

{

cout << pOut->info << " ";

pOut = pOut->pNext;

}

}