**Section (I): Tracing Problems (Total: 3 marks)**

*In the following tracing question use the following definition for the nodes in the list:*

**class CListNode**

**{**

**public:**

**int info;**

**CListNode \*pNext;**

**};**

**B**

**A**

**4**

**3**

**1**

**2**

**Write one statement to do the following: -**

* Display the info of node (**4**).

cout<< B->pNext->info;

**True or False :**

* A 🡪 pNext 🡪 pNext 🡪 pNext 🡪 pNext 🡪 pNext 🡪 pNext == A 🡪 pNext 🡪 pNext 🡪 pNext

TRUE

* Make node (**3**) points to node (**4**).

A->pNext = B->pNext;

**Section (II): Algorithms (Total: 7 marks)**

**Algorithm 1: (7 marks)**

*In the following Algorithm question use the following definition for the nodes in the list:*

**class CNode**

**{**

info

**public:**

**int info;**

pNext

**CNode \* pNext;**

**CNode \* pExtra;**

**};**

pExtra

Write  **void Cut\_Paste\_Two\_Largest\_Values** **( )** function that do the following:

* Your function will find the largest 2 values in the list
* Then you have to cut the interval between them & paste them at the end of the list.
* Cut the 4 nodes that precedes the interval (in the above step) & paste it at the first of the list.
* **Note:** Your function should be **O(N)**, O(2n) is not accepted.
* **Note:** we don’t know the order of the 2 largest values (e.g. may be 350 comes before 240).

Head

L

Interval between the 2 Largest values

L’

4 nodes precedes the Interval

Head

#include <iostream>

using namespace std;

class CNode

{

public:

int info;

CNode\* pNext;

CNode\* pExtra;

};

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 Cut\_Paste\_Two\_Largest\_Values(CList& L)

{

CNode\* pTrav = L.pHead;

CNode\* pB = NULL;

int i = 0, pos1 = 0, pos2 = 0, max = -9999,check=0,max1=0,first=0;

CNode\* pMax1=L.pHead, \* pBm1 = L.pHead, \* pMax2 = L.pHead, \* pBm2 = L.pHead;

while (pTrav != NULL)

{

if (check == 0)

{

if (pTrav->info > max && max1==0)

{

max = pTrav->info;

pos1 = i;//to know its position to know which max comes first

pMax1 = pTrav; //points at max

pBm1 = pB; //points at node before max

}

if (pTrav->info > max && pTrav->info != pMax1->info)

{

max = pTrav->info;

pos2 = i;

pMax2 = pTrav;

pBm2 = pB;

}

pB = pTrav;

if (pTrav->pNext != NULL)

{

pTrav = pTrav->pNext; //traverse

pTrav->pExtra = pB; //make the extra of each node point to node before it

i++;

}

else

{

if (max1 == 0)

{

pTrav = L.pHead;

pB = NULL;

i = 0;

max = -9999;

max1 = 1;

}

else

{

check = 1;

pTrav = L.pHead;

}

}

}

else

{

if (pos2 < pos1) //to know the position(node) of the beginning of the 4 previous nodes

{

if (first == 0)

{

pTrav = pMax2;

i = pos2;

first = 1;

}

if (i != pos2 - 4)

{

pTrav = pTrav->pExtra;

i--;

}

}

else

{

if (first == 0)

{

pTrav = pMax1;

i = pos1;

first = 1;

}

if(i != pos1 - 4)

{

pTrav = pTrav->pExtra;

i--;

}

}

}

}

//max = -9999, i=0;

//pTrav = L.pHead, pB = NULL;

//while (pTrav != NULL)

//{

// if (pTrav->info > max && pTrav->info!=pMax1->info)

// {

// max = pTrav->info;

// pos2 = i;

// pMax2 = pTrav;

// pBm2 = pB;

// }

//

// pB = pTrav;

// pTrav = pTrav->pNext;

// pTrav->pExtra = pB;

// i++;

//

//}

//cout << "2\n";

//CNode\* pTrav3 = NULL;

//

//if (pos2 < pos1) //to know the position(node) of the beginning of the 4 previous nodes

//{

// pTrav3 = pMax2;

// i = pos2;

//

// while (i != pos2 - 4)

// {

// pTrav3=pTrav3->pExtra;

// i--;

// }

//}

//else

//{

// pTrav3 = pMax1;

// i = pos1;

//

// while (i != pos1 - 4)

// {

// pTrav3 = pTrav3->pExtra;

// i--;

// }

//}

//cout << "3\n";

if (pos2 < pos1) //to move maximums and nodes between them to the last

{

pBm2->pNext = pMax1->pNext;

pMax1->pNext->pExtra = pBm2;

pMax1->pNext = NULL;

pMax2->pExtra = L.pTail;

L.pTail->pNext = pMax2;

L.pTail = pMax1;

}

else

{

pBm1->pNext = pMax2->pNext;

pMax2->pNext->pExtra = pBm1;

pMax2->pNext = NULL;

pMax1->pExtra = L.pTail;

L.pTail->pNext = pMax1;

L.pTail = pMax2;

}

if (pos2 < pos1) //to move the 4 previous nodes to the beginning of list

{

pBm2->pNext->pExtra = pTrav->pExtra;

pTrav->pExtra->pNext = pBm2->pNext;

pTrav->pExtra = NULL;

pBm2->pNext = L.pHead;

L.pHead = pTrav;

}

else

{

pBm1->pNext->pExtra = pTrav->pExtra;

pTrav->pExtra->pNext = pBm1->pNext;

pTrav->pExtra = NULL;

pBm1->pNext = L.pHead;

L.pHead = pTrav;

}

}

void main()

{

CList L;

CNode\* pnn;

cout << "Enter N \n";

int N;

cin >> N;

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

{

pnn = new CNode;

cout << "enter info\n";

cin >> pnn->info;

pnn->pNext = NULL;

L.Attach(pnn);

}

Cut\_Paste\_Two\_Largest\_Values(L);

//output

CNode\* pTrav = L.pHead;

while (pTrav != NULL)

{

cout << pTrav->info << " ";

pTrav = pTrav->pNext;

}

}

**Section (III): Problem Solving (Total: 10 marks)**

**Problem 1: (10 marks)**

*In the following question use the following definition for the nodes in the list:*

**class CListNode**

pUp1

pUp2

**{**

**public:**

pNext

**int info;**

**CListNode \*pNext;**

**CListNode \*pUp1;**

**CListNode \*pUp2;**

**};**

info

Write a main function to do:

* Read a Linked List **(L)** from the user, but according to the following protocol:

- the user will determine the number of nodes (**NC**) in the 1st row in the list. (in below example **NC=7**).

For each column🡪 according to the info inside the node create UP nodes.

* Also connect each top node to another one in the next column but in its level.

**e.g.**

L

Head

* Starting from the 1st node, check each isolated region if it is mirror or not.

Head

L

**Logic when traced is CORRECT, but does NOT RUN FULLY due to problem with pB**

#include <iostream>

using namespace std;

class CNode

{

public:

int info;

CNode\* pNext;

CNode\* pUp1;

CNode\* pUp2;

};

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()

{

int NC;

CList L;

CNode\* pnn, \*pnn1,\*pnn2, \*pB=L.pHead , \*pF;

int prev = 0, check=0;

pnn = pnn1 = pnn2 = pF = NULL;

cout << "enter nc \n";

cin >> NC;

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

{

//read horizontally

pnn = new CNode; //pnn points at horizontal node

cout << "enter pnn info \n";

cin >> pnn->info;

pnn->pNext = NULL;

pnn->pUp1 = NULL;

pnn->pUp2 = NULL;

L.Attach(pnn);

for (int j = 0; j < pnn->info; j++)

{

//read vertically

//pnn1 points at vertical node on left

//pnn2 points at vertical node on right

if (pnn->pUp1 == NULL && pnn->pUp2 == NULL)

{

pnn1 = new CNode;

cout << "enter pnn1 info \n";

cin >> pnn1->info;

pnn1->pNext = pnn1;

pnn->pUp1 = pnn1;

pnn2 = new CNode;

cout << "enter pnn2 info \n";

cin >> pnn2->info;

pnn2->pNext = pnn2;

pnn->pUp2 = pnn2;

}

else

{

pnn1 = new CNode;

cout << "enter pnn1 info \n";

cin >> pnn1->info;

pnn1->pNext = pnn1;

pnn->pUp1->pNext->pUp1 = pnn1;

pnn->pUp1->pNext->pUp2 = NULL;

pnn->pUp1->pNext = pnn1; //pNext of first node up points at last node up

pnn2 = new CNode;

cout << "enter pnn2 info \n";

cin >> pnn2->info;

pnn2->pNext = pnn2;

pnn->pUp2->pNext->pUp1 = pnn2;

pnn->pUp1->pNext->pUp2 = NULL;

pnn->pUp2->pNext = pnn2; //pNext of first node up points at last node up

}

if (i%2!=0)

{ //only after each other node

if (j == prev - 1)

{

pF = pnn1;

pB->pUp2->pNext->pNext = pF;

}

}

}

pnn1->pNext = pnn2;

prev = pnn->info;

CNode\* pTrav1 = pB->pUp1, \* pTrav2 = pF;

if (i % 2 != 0)

{

for (int z = 0; z < pB->info; z++)

{

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

{

check = 1;

break;

}

else

{

pTrav1 = pTrav1->pUp1;

pTrav2 = pTrav2->pUp1;

}

}

if (check == 1)

{

cout << "not mirror";

}

else

{

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

{

cout << "mirror";

}

else

{

cout << "not mirror";

}

}

}

pB = pnn; //prev node horizontally eg:6 if you are at 10 node

}

}