**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;**

**};**

**A**

**1**

**2**

**4**

**3**

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

* Display the info of node (**2**).

Cout<<A->pNext->pNext->info;

* Make node (**2**) points to node (**3**).

A->pNext->pNext->pNext =A;

**True or False :**

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

TRUE

**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**

**{**

pUpRight

**public:**

pUpLeft

**int info;**

pNext

**CNode \*pNext;**

info

**CNode \*pUpLeft;**

**CNode \*pUpRight;**

**CNode \*pExtra;**

**};**

pExtra

Write  **CNode \*Pop\_Left\_and\_Right\_Intervals** **( int target )** function that do the following:

* Your function will receive an integer that represents a target value to be found **O(K)**.
* Cut the intervals (both Left & right), connect them, and return them. **O(1)**.

**Note :**

* Assume the target node will **not point** to the head **nor** to the tail.

in our example target node (20) , points to [5 not a head & 8 not a tail] .

* Assume all nodes in the intervals has empty **pUpLeft , & pUpRight** pointers.

In our example all nodes in the intervals [ 5, 30, 60, 2 , 8] has empty **pUp(**s)

e.g.

**Pop\_Left\_and\_Right\_Intervals** ( 20 )

pHead

pHead

**K**

After the function

The returned

pointer

CNode\* Pop\_Left\_and\_Right\_Intervals(int target)

{

CNode\* pTrav = pHead, \* pB = NULL;

while (pTrav->info != target)

{

pB = pTrav;

pTrav = pTrav->pNext;

pTrav->pExtra = pB;

}

pTrav->pUpLeft->pExtra->pNext = pTrav;

pTrav->pUpLeft->pExtra->pNext = NULL;

pTrav->pNext = pTrav->pUpRight->pNext;

pB->pNext = pTrav->pNext;

pTrav->pUpRight->pNext = NULL;

pB = pTrav->pUpLeft;//make pB point to the 1st node in the removed part

pTrav->pUpLeft = pTrav->pUpRight = NULL;

return pB;

}

**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**

info

**{**

**public:**

pNext

**int info;**

**CListNode \*pNext;**

**CListNode \*pDown;**

**};**

pDown

Write a main function to do:

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

- in 1st column read 1 down node.

- in 2nd column read 2 down nodes.

- in 3rd column read 3 down nodes.

- and so on.

**e.g.**

L

Head

Tail

* Ask the user to enter a target value (T),
  + In those columns that include the (T), check the reminder cells in those columns make mirror or not.

e.g.

T = 91

Tail

Head

L

**In our example:**

**30 , 25, 10, 14, 14, 10 , 25 , 30**

**Yes it is mirror**

void main()

{

CList L;

CNode\* pnn, \* pnn1;

int N, t;

cout << "enter N \n";

cin >> N;

for (int i = 1; i <= N; i++)

{

pnn = new CNode;

cout << "enter info pnn \n";

cin >> pnn->info;

pnn->pNext = NULL;

pnn->pDown = NULL;

L.Attach(pnn);

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

{

pnn1 = new CNode;

cout << "enter info pnn1 \n";

cin >> pnn1->info;

pnn1->pNext = NULL;

pnn1->pDown = NULL;

if (pnn1->pDown == NULL)

{

pnn->pDown = pnn1;

pnn->pDown->pNext = pnn1;

}

else

{

pnn->pDown->pNext->pDown = pnn1;

pnn->pDown->pNext = pnn1;

}

}

}

cout << "enter target value \n";

cin >> t;

CStack S, S2;

CNode\* pTrav = L.pHead;

int ct = 0;

while (pTrav != NULL)

{

pnn = pTrav;

while (pnn != NULL)

{

if (pnn->info == t)

{

break;

}

pnn = pnn->pDown;

}

if (pnn != NULL)

{

pnn = pnn->pDown;

while (pnn != NULL)

{

pnn1 = new CNode;

pnn1->info = pnn->info;

pnn1->pNext = NULL;

S.push(pnn1); //stack containing all nodes that are below t in the WHOLE list

ct++; //count the total num of nodes in the WHOLE list that are below the t

pnn = pnn->pDown;

}

}

pTrav = pTrav->pNext;

}

for (int i = 0; i < ct / 2; i++) //to go through only half of the nodes in stack1 & put them in stack2

{

pnn1 = S.pop();

S2.push(pnn1);

}

if (ct % 2 != 0) //delete the odd one ??not sure

{

pnn1 = S.pop();

delete pnn1;

}

int check = 0;

while (S2.pHead != NULL)

{

CNode\* p1 = S.pop();

CNode\* p2 = S2.pop();

if (p1->info != p2->info)

{

check = 1;

}

delete p1;

delete p2;

}

if (check == 1)

{

cout << "not mirror";

}

else

{

cout << "mirror";

}

}