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

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

**class CListNode**

pNext

info

pBack

**{**

**public:**

**int info;**

**CListNode \*pNext;**

**CListNode \*pBack;**

**};**

**A**

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

* Display the info of node (3).

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

* True or false?

A 🡪 pBack🡪 pBack == A

TRUE

* Make the “pNext” of Node(2) points to the Node (3).

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

**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**

pUp

**{**

**public:**

pNext

**int info;**

**CNode \*pNext;**

**CNode \*pUp;**

**};**

info

*Void* ***Remove*** *( int iRow, int iCol, int count )*

* *Your function 3 integers :*
  + *iRaw , represents the required row.*
  + *iCol , represents the required column.*
  + *count , number of nodes to be removed.*

**e.g.**

Remove ( 2 , 4 , 3 )

H

After the Remove

H

void Remove(int iRow, int iCol, int count)

{

CNode\* pTrav = pHead, \* pBc, \* pBr;

//to get the pBc

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

{

pTrav = pTrav->pUp;

}

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

{

pBc = pTrav;

pTrav = pTrav->pNext;

}

//to get the pBr

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

{

pTrav = pTrav->pNext;

}

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

{

pBr = pTrav;

pTrav = pTrav->pUp;

}

//pTrav is at the 1st node to be removed

for (int i = 0; i < count-1 && pTrav!=NULL; i++)

{

pBc->pNext = pTrav->pNext;

pTrav->pNext = NULL;

pBr->pUp = pTrav->pUp;

pTrav->pUp = NULL;

/\*if i have to delete it then delete pTrav then pTrav=pBc->pNext\*/

if (pBr->pNext != NULL)

{

pBr = pBr->pNext;

}

pTrav = pTrav->pNext;

}

}

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

**Problem 1: (10 marks)**

info

**class CNode**

pNext

**{**

**public:**

**int info;**

**CNode \* pNext;**

pDown

**CNode \* pDown;**

**};**

Write a main function to do:

* + Read 2 Lists from the user.
  + For the 1st list find the first 2 odd values,
    - Then make the second odd points to the first odd (using the pDown pointer)
    - Make the last node of the list (regardless it is odd or not) points to the second odd (using the pDown pointer)
  + For the 2nd list find the first 2 odd values,
    - Then make the second odd points to the first odd (using the pDown pointer)
    - Make the last node of the list (regardless it is odd or not) points to the second odd (using the pDown pointer)

L1

Head

Head

L2

* **Copy** the 2 odd nodes from (L2) and push them at the head of (L1),

also make the 1st odd in (L1) points to the new coming odds

**Make this step in O(1) .**

L1

Head

L2

Head

void main()

{

CList L1, L2;

CNode\* pnn,\*pTrav, \*p1Odd1,\*p1Odd2, \*p2Odd1, \*p2Odd2;

p1Odd1 = p1Odd2 = p2Odd1 = p2Odd2 = NULL;

int N,check=0;

cout << "enter N for l1 \n";

cin >> N;

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

{

pnn = new CNode;

cout << "enter info pnn \n";

cin >> pnn->info;

pnn->pNext = NULL;

pnn->pDown = NULL;

L1.Attach(pnn);

}

cout << "enter N for l2 \n";

cin >> N;

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

{

pnn = new CNode;

cout << "enter info pnn \n";

cin >> pnn->info;

pnn->pNext = NULL;

pnn->pDown = NULL;

L2.Attach(pnn);

}

pTrav = L1.pHead;

while (pTrav != NULL)

{

if (pTrav->info % 2 != 0)

{

if (check == 0)

{

p1Odd1 = pTrav;

check = 1;

}

else

{

p1Odd2 = pTrav;

break;

}

}

pTrav = pTrav->pNext;

}

p1Odd2->pDown = p1Odd1;

L1.pTail->pDown = p1Odd2;

check = 0;

pTrav = L2.pHead;

while (pTrav != NULL)

{

if (pTrav->info % 2 != 0)

{

if (check == 0)

{

p2Odd1 = pTrav;

check = 1;

}

else

{

p2Odd2 = pTrav;

break;

}

}

pTrav = pTrav->pNext;

}

p2Odd2->pDown = p2Odd1;

L2.pTail->pDown = p2Odd2;

//first node copied

pnn = new CNode;

pnn->info = p2Odd1->info;

pnn->pNext = L1.pHead;

L1.pHead = pnn;

//second node copied

pnn = new CNode;

pnn->info = p2Odd2->info;

pnn->pNext = L1.pHead->pNext;

L1.pHead->pNext = pnn;

pnn->pDown = L1.pHead; //to make down of 2nd node point to 1st node

/\*or L1.pHead->pNext->pDown=L1.pHead;\*/

p1Odd1->pDown = L1.pHead->pNext; //to make the down of 1st odd in l1 point to new odds

/\*or p1Odd1->pDown = pnn\*/

}