**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;**

**};**

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

* Display the info of node (**1**) .

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

**True or False :**

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

TRUE

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

A->pNext->pNext=A;

**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**

**{**

**public:**

**int info;**

**CNode \* pNext;**

**CNode \* pDown;**

**};**

Write  **void Attach\_And\_Cut\_When** **( CNode \*pNN )** function that do the following:

* Your function will attach the (pNN) normally at the end of the list.
* **But** :

when the # of nodes becomes 4 🡪 you have to **cut** the 1st half of the list & **past** it to down.

when the # of nodes becomes 6 🡪 you have to cut the 1st half of the list & past it to down.

when the # of nodes becomes 8 🡪 you have to cut the 1st half of the list & past it to down.

when the # of nodes becomes 10 🡪 you have to cut the 1st half of the list & past it to down.

when the # of nodes becomes 12 🡪 you have to cut the 1st half of the list & past it to down.

And so on…

* **Note:** There is a third pointer (pExtra) in your list additional to the pHead & pTail.
* **Note:** there are 2 integer values in your list.
* Your function should be in **O(1)**

e.g.

* **Attach…** (15) , **Attach…** (70) , **Attach…** (33) , **Attach…** (10)

* **Attach…** (40) , **Attach…** (60) , **Attach…** (50) , **Attach…** (25)
* **Attach…** (60) , **Attach…** (12) , **Attach…** (39) , **Attach…** (13) , **Attach…** (28)
* **Attach…** (11) , **Attach…** (31) , **Attach…** (14) , **Attach…** (84) , **Attach…** (91) , **Attach…** (53)

void Attach\_And\_Cut\_When(CNode\* pnn)

{

if (pHead == NULL)

{

v1 = 4;

v2 = 1;

pHead = pnn;

pHead->pNext = NULL;

pExtra = pHead;

}

else

{

if (v2 < v1) //make nodes build till v1 reach v4

{

v2++;

pTail->pNext = pnn;

pTail = pnn;

pTail->pNext = NULL;

}

else if (v2 == v1) //reached the condition where to start breaking

{

v1 += 2; //once at 4 then 6 then 8 then 10 and so on....

v2 /= 2; //once at 2 then 3 then 4 then 5

pExtra = pExtra->pNext; //I want pExtra to move one step

pExtra->pNext->pDown = pHead;

pHead = pExtra->pNext;

pExtra->pNext = NULL;

pExtra = pTail;

}

}

}

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

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

**class CNode**

**{**

**public:**

**int info;**

**CNode \* pNext;**

**CNode \* pDown;**

**};**

Write a main function to do:

* + Read 20 Lists from the user.

(Assume all lists with the same length)

* + Connect each node carries odd value to its down node
* But in case that the down node also carries odd value.
* Copy any column that was fully connected to another List **(NL)**.

#include <iostream>

using namespace std;

class CNode

{

public:

int info;

CNode\* pNext;

CNode\* pDown;

};

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], newL;

CNode\* pnn, \* pTrav1, \* pTrav2;

int N,ct=0,pos=0;

cout << "Enter N \n";

cin >> N;

for (int i = 0; i < 19; i++)

{

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

{

pnn = new CNode;

cin >> pnn->info;

pnn->pNext = NULL;

pnn->pDown = NULL;

L[i].Attach(pnn);

}

}

for (int i = 0; i < 19; i++)

{

pTrav1 = L[i].pHead;

pTrav2 = L[i + 1].pHead;

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

{

if (pTrav1->info % 2 != 0 && pTrav2->info % 2 != 0)

{

pTrav1->pDown = pTrav2;

}

pTrav1 = pTrav1->pNext;

pTrav2 = pTrav2->pNext;

}

}

pTrav1 = L[0].pHead;

pTrav2 = L[0].pHead;

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

{

pTrav1 = pTrav2;

while (pTrav1->pDown != NULL)

{

pTrav1 = pTrav1->pDown;

ct++;

}

if (ct == 20)

{

break;

}

pTrav2 = pTrav2->pNext;

}

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

{

pnn = new CNode;

pnn->info = pTrav2->info;

pnn->pNext = NULL;

newL.Attach(pnn);

pTrav2 = pTrav2->pDown;

}

}