**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**) through (**A**). [not accepted to use the pointer **B** in this statement]

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

**True or False :**

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

FALSE

* Make node (**2**) points to node (**4**).

B->pNext=A->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**

**{**

**public:**

**int info;**

**CNode \* pNext;**

**CNode \* pDown;**

**};**

Write  **void Special\_ Display** **( )** function that do the following:

* Your function will display the nodes in Right-Diagonal manner but starting from bottom

**Output:**

72 , 12, 145, 661 , 10 , 810 , 350

15 , 51 , 74 , 111 , 22 , 57

130 , 93, 176 , 51 , 34

11, 26 , 12 , 11

50 , 32 , 24

34 , 56

12

void Special\_Display()

{

CNode\* pTrav1 = pHead;

CNode\* pTrav2 = pHead;

int ct1 = 0, ct2 = 0, z = 0, k = 0;

while (pTrav1 != NULL)

{

pTrav1 = pTrav1->pDown;

ct1++;

}

while (pTrav2 != NULL)

{

pTrav2 = pTrav2->pNext;

ct2++;

}

pTrav1 = pTrav2 = pHead;

for (int i = 0; i < ct1; i++)

{

z = 0, k = 0;

while (z != ct1)

{

pTrav1 = pTrav1->pDown;

z++;

}

while (k != ct2)

{

pTrav2 = pTrav2->pNext;

k++;

}

while (pTrav1 != pTrav2)

{

cout << pTrav1->info;

pTrav1 = pTrav1->pNext;

}

cout << pTrav1->info;

ct1--, ct2--;

}

}

**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**

**{**

**public:**

**int info;**

**CListNode \*pNext;**

**CListNode \*pDown;**

**};**

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 columns (**NC**) in the list. (in below example **NC=10**).

For each column🡪 read **(NR)** nodes to down:

- read node by node till the user enters a node with negative value **(-v).**

In this case make the ***(pNext)*** of this negative node points the node after it by **(+v).**

**e.g.**

* **Cut** and **Paste** 🡪

For each column: Cut the **longest** interval (that starting by negative values),

And past them to new List ***(NewL)*** in ***reverse*** manner

**NOTE:** this step should be in **O(NC).**

#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, newL;

CNode\* pnn, \*pnn1, \*pNeg,\*pMax, \*pB, \*pBmax, \*pEnd;

int NC,NR,v, found=0,max=-9999;

cout << "enter NC \n";

cin >> NC;

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

{

pnn = new CNode;

cin >> pnn->info;

pnn->pNext = NULL;

L.Attach(pnn);

cout << "enter NR \n";

cin >> NR;

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

{

if (pnn->pDown == NULL)

{

pnn1 = new CNode;

cin >> pnn1->info;

pnn1->pNext = NULL;

pnn->pDown = pnn1;

pnn1->pNext = pnn1;

}

else

{

pnn1 = new CNode;

cin >> pnn1->info;

pnn1->pNext = NULL;

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

pnn->pDown->pNext = pnn1; //1st down node,its next points to last node in column

if (pnn1->info < 0)

{

v = -1 \* (pnn1->info);

if (v > max)

{

pBmax = pB;

max = v;

pMax = pnn1; //points to the highest negative node

}

v++;

pNeg = pnn1;

found = 1; //to indicate that it reached negative node

}

if (v != 0 && found == 1) //moves till it points to v+ of -ve num

{

pNeg->pNext = pnn1;

}

v--;

if (v == 0) //reached the last of the interval

{

if (pNeg == pMax)

{

pEnd = pnn1;

}

found = 0;

}

}

pB = pnn1;

}

if (newL.pHead == NULL)

{

newL.pHead = pMax;

newL.pTail = pEnd;

pBmax->pDown = pEnd->pDown;

}

else

{

pEnd->pNext = newL.pHead;

newL.pHead = pMax;

}

}

}