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

Cout<< B->pNext->pNext->info;

**True or False :**

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

TRUE

* 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 \* pExtra;**

**};**

Write  **CNode \* Cut\_ Flip\_And\_Return** **( int Val )** function that do the following:

* Your function will find the largest **region** that starting by (val) and ending by (val).
* remove the founded region.
* Flip the removed region based on the middle node of (Val)s
* Return the removed nodes.
* Your function should be in **O(N) 🡪 Not accepted O(2N).**

e.g.

**Cut\_ Flip\_And\_Return** ( 10 )

**After the remove and flip**

CNode\* Cut\_Flip\_And\_Return(int val)

{

CNode\* pB = NULL,\* pTrav = pHead, \*pMid=NULL, \*pF=NULL, \* pLast = NULL, \* pBval = NULL, \* p1 = NULL, \* p2 = NULL;

int ct = 0;

while (pTrav != NULL)

{

if (pTrav->pNext->info == val)

{

ct++;

if (ct == 1)

{

pF = pTrav; //before the first val node

}

if (ct > 1)

{

pBval->pExtra = pTrav; //every before val node will have its extra pointing to the before of next val node

}

if (ct == 3)

{

pMid = pBval; //assume at first that if they are just 3 nodes then the mid will be the second node which in this case is the pBval before moving it

}

if (ct > 3 && ct % 2 != 0)

{

pMid = pMid->pExtra; //if more than three and its an odd num then obv the mid will be the next val node

}

pLast = pTrav; //before last val node

pBval = pTrav; //before trav

}

pB = pTrav;

pTrav = pTrav->pNext;

}

pLast = pLast->pNext; //last val node

p1 = pF->pNext;

//first region before mid

p2 = pMid->pNext->pNext;

//region after mid

pF->pNext = pLast->pNext;

//remove

//flip

pLast->pNext = pMid->pNext;

pMid->pNext->pNext = p1;

pMid->pNext = NULL;

}

**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 nodes (**N**) in the 1st (3 columns) in the list.

(in the below example **N=12**) **ASSUME**: that **N** will be always divisible by 3).

- You will read (N) nodes for the 1st (3 columns).

- then you will read (N-3) nodes for the 2nd (3 columns).

- and so on. **Note :** you will stop when reach zero nodes.

**e.g.**

* Check if the Right-Bottom borders of your List is mirror or not.



#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;

CNode\* pnn, \*pnn1, \*pPrev, \*pC;

int N,z=0,ct=0,check=0;

pnn = pnn1 = pPrev = pC = NULL;

cout << "enter n of 3 columns \n";

cin >> N;

while (N != 0)

{

z = 0;

ct++;

if (z != 3)

{

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

{

if (j == 0)

{

pnn = new CNode;

cout << "enter info for pnn \n";

cin >> pnn->info;

pnn->pNext = NULL;

pnn->pDown = NULL;

L.Attach(pnn);

}

else

{

if (pnn->pDown == NULL)

{

pnn1 = new CNode;

cout << "enter info for pnn1 \n";

cin >> pnn1->info;

pnn1->pNext = NULL;

pnn->pDown = pnn1;

pnn1->pNext = pnn1;

}

else

{

pnn1 = new CNode;

cout << "enter info for pnn1 \n";

cin >> pnn1->info;

pnn1->pNext = NULL;

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

pnn->pDown->pNext = pnn1; //next of first pnn1 node points to last node

}

}

if (j == N - 3 && z==2 && check==0)

{

pC = pnn1;

}

if (check == 1 && z == 0)

{

pnn1->pNext = pC;

}

}

pnn1->pNext = pPrev;

pPrev = pnn1;

z++;

}

N -= 3;

if (ct % 2 != 0)

{

check = 1;

}

else

{

check = 0;

}

}

ct = 0;

//to be able to point at the "1" node in the second set of 3s

CNode\* pBf = L.pHead;

CNode\* pAf = L.pHead;

while (pBf != NULL)

{

if (ct != (N / 2)-1)

{

pBf = pBf->pNext;

ct++;

}

else

{

break;

}

}

ct = 0;

while (pAf != NULL)

{

if (ct != (N / 2))

{

pAf = pAf->pNext;

ct++;

}

else

{

break;

}

}

while (pAf != NULL)

{

pBf = pBf->pDown;

pAf = pAf->pDown;

}

//new pAf will point at last node horizontally

pAf = L.pTail;

z = 0;

int bad = 0;

while (pAf->pNext !=NULL && pAf->pDown!=NULL) //as long as there is a node below or beside it then the L shape is not down

{

if (pBf->info != pAf->info)

{

bad = 1;

break;

}

else

{

if (pAf->pDown != NULL && pBf->pDown != NULL)

{

pBf = pBf->pDown;

pAf = pAf->pDown;

z = 0;

}

else

{

if (pAf->pNext != NULL && pBf->pNext != NULL)

{

pBf = pBf->pNext;

pAf = pAf->pNext;

}

}

}

}

}