**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 followings:-**

* Display the info of node (1).

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

* True or false?

A 🡪 pNext 🡪 pNext == A 🡪 pBack 🡪 pNext

TRUE

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

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

**Section (II): Algorithms (Total: 8 marks)**

**Algorithm 1: (8 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* ***Display\_in\_Reverse*** *( char \*path )*

* *Your function will receive a string that includes directions of the required nodes.*
* *Your function should display those nodes* ***but*** *in* ***a reverse*** *manner.*

**Note:** ‘U’ means go up.

‘R’ means go Right.

There is no down & left directions.

**Note: your function should be in O(N)**

O(2N) accepted as ≈ O(N)

But O(N2) ≠ O(N)

**e.g.**

path

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| U | R | R | U | R | R | \0 | ,, | .. | .. | .. | .. |

**O( N )**

**The required nodes only**

**N = 7 nodes in this example**

O( M )

Means all nodes in the list

M = 26 nodes in this example

H

**Output:**

**19, 38, 34, 25, 33, 22, 17**

void Display\_in\_Reverse(char\* path)

{

CNode\* pTrav = pHead, \*pD;

CStack S;

CNode\* pnn;

for (int i = 0; path[i] != '\0'; i++)

{

if (i == 0)

{

pnn = new CNode;

pnn->info = pTrav->info;

S.push(pnn);

}

if (path[i] == 'U')

{

pTrav = pTrav->pUp;

pnn = new CNode;

pnn->info = pTrav->info;

S.push(pnn);

}

if (path[i] == 'R')

{

pTrav = pTrav->pNext;

pnn = new CNode;

pnn->info = pTrav->info;

S.push(pnn);

}

}

pTrav = S.pHead;

while (pTrav != NULL)

{

pD = S.pop();

cout << pD->info;

delete pD;

}

}

**Section (III): Problem Solving (Total: 9 marks)**

**Problem 1: (9 marks)**

pDown

info

**class CNode**

pNext

**{**

**public:**

**int info;**

**CNode \* pNext;**

**CNode \* pDown;**

**};**

Write a main function to do:

* + Read 30 Lists from the user.

(Assume all lists with the same length)

* During your read, make any zero-node points to its down node in the next List.
* Make the down nodes points to the previous nodes.
* **O(N)** for each List

Head

Head

Head

L

0

1

2

Head

4

3

Head

* Ask the user to select a list (by index).

e.g. List # : 3

* Find its zero-region
  + Cut them & its corresponding down nodes
  + Paste them into a New List.

And make each node in the new list points to the previous node using its (pDown) pointer.

.

Before the cut & paste

As example: if the selected list was List # 3.

3

2

1

0

Head

Head

Head

Head

Head

L

4

After the cut & paste

NewList

Head

Head

Head

Head

Head

Head

L

4

3

2

1

0

void main()

{

CList L[30], newL;

CNode\* pnn, \* pTrav1, \* pB=NULL, \* pTrav2;

int N, num;

cout << "enter n \n";

cin >> N;

for (int i = 0; i < 30; i++)

{

if (i != 0)

{

pTrav1 = L[i - 1].pHead; //traverse in list before current list

}

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

{

pnn = new CNode;

cout << "enter pnn info \n";

cin >> pnn->info;

pnn->pNext = NULL;

pnn->pDown = NULL;

L[i].Attach(pnn);

if (j == 0)

{

pTrav2 = L[i].pHead; //traverse in current list

}

if (i != 0 && j != 0)

{

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

{

pTrav1->pDown = pTrav2;

pTrav2->pDown = pB;

}

pTrav1 = pTrav1->pNext;

pB = pTrav2;

pTrav2 = pTrav2->pNext;

}

}

}

cout << "select a list \n";

cin >> num;

pTrav1 = L[num].pHead;

pB = NULL;

int i=0, ct = 0, pos=0;

while (pTrav1 != NULL)

{

i++;

if (pTrav1->info == 0)

{

ct++;

if (newL.pHead == NULL)

{

newL.pHead = pTrav1;

newL.pTail = pTrav1;

pB->pNext = pTrav1->pNext;

newL.pTail->pNext = NULL;

pTrav1 = pB;

pos = i;

}

else

{

pB->pNext = pTrav1->pNext;

newL.pTail->pNext = pTrav1;

newL.pTail->pNext->pDown = newL.pTail; //prev node

newL.pTail = pTrav1; //then move tail

newL.pTail->pNext = NULL;

pTrav1 = pB;

}

}

pB = pTrav1;

pTrav1 = pTrav1->pNext;

if (ct != 0 && pTrav1->info != 0)

{

break; //to lessen the num of useless loops

}

}

pTrav2 = L[num+1].pHead;

pB = NULL;

i = 0;

while (pTrav2 != NULL)

{

i++;

while (i == pos && ct != 0)

{

pB->pNext = pTrav2->pNext;

newL.pTail->pNext = pTrav2;

newL.pTail->pNext->pDown = newL.pTail; //prev node

newL.pTail = pTrav2; //then move tail

newL.pTail->pNext = NULL;

pTrav2 = pB;

ct--;

pB = pTrav2;

pTrav2 = pTrav2->pNext;

}

if (ct == 0)

{

break;

}

}

}