**Section (I): Tracing Questions (Total: 8 marks)**

**(Q1) (3 mark)**

* Draw the binary expression tree for the following prefix expression.
* Also, write the corresponding postfix, and infix expression.
* **+ + \* \* A B \* \* C D + E F G H**

**(Q2) (5 mark)**

**For exercises (i - iii) use the following values, and draw the hash table for each of them:**

**22 , 25 , 39 , 16 , 45 , 13 , 32**

1. Store the values in a hash table with size 8 cells.
2. Store the values in a hash table with 4 buckets, each bucket contains 2 cells.
3. Store the values in a hash chain table with function (Key % 8).
4. Fill in the following table, showing the number of comparisons needed to find each value:-

|  |  |  |  |
| --- | --- | --- | --- |
|  | **(i)** | **(ii)** | **(iii)** |
| **13** |  |  |  |
| **39** |  |  |  |
| **32** |  |  |  |

**Section (II): Algorithm Questions (Total: 8 marks)**

**Algorithm 1: (8 marks)**

You are asked to write a function called **Remove\_Most\_Left\_ Leaf\_Of\_Target\_Value** **(** **int** **V** **)**

* Your function will receive an integer that represents some Target Value **(V)**.
* You will find **(V)**.
  + If : Left child of (V) was a **leaf** , remove it
  + If : Left child of (V) was **not a leaf** , find the **most left leaf** of (V), and remove it.
  + If : Left child of (V) was NULL, do nothing.

|  |  |
| --- | --- |
| 399  395  399  395  **e.g. V = 115**  450  400  70  90  80  50  112  120  **115**  160  600  110  500  122  155  130  220  270  260  250  200  390 | **e.g. V = 500**  450  400  70  90  80  50  112  120  115  160  600  110  **500**  122  155  130  220  270  260  250  200  390 |

void Remove\_Most\_Left\_Leaf\_Of\_Target\_Value(int v)

{

CTNode\* pTrav = pRoot;

while (pTrav != NULL || pTrav->info != v)

{

if (v > pTrav->info)

{

pTrav = pTrav->pRight;

}

else

{

pTrav = pTrav->pLeft;

}

}

CTNode\* pL = pTrav->pLeft;

CTNode\* pBL = NULL;

if (pTrav->pLeft->pLeft == NULL)

{

pTrav->pLeft = NULL;

delete pL;

}

else

{

while (pL != NULL)

{

if (pL->pLeft != NULL)

{

pBL = pL;

pL= pL->pLeft;

}

else if(pL->pRight!=NULL)

{

pBL = pL;

pL = pL->pRight;

}

else

{

pBL->pLeft = NULL;

delete pL;

}

}

}

}

**Section (III): Problem Solving (Total: 24 marks)**

**Problem 1: (5 marks)**

*In the this question use the following definition for the nodes in the List:*

**class CListNode**

**{**

**public:**

info

pNext

**int info;**

**CListNode \*pNext;**

**};**

**Note:**

* in this problem you have to write just single function

(no permission to write more than 1 function).

* no permission to use loop inside the function.

Write a **recursive** function to do the following:

The function should display the values in between **(V)**.

**Assume** that the required **(V)** , will exist in the list 2 times.

**e.g.**

**V = 70**

L

Head

**Output will be: 20 , 30, 22, 43, 60 .**

void DispBetweenV(CNode\*pT, CNode\*pB, int v, int&found, int&ct)

{

if (pT == NULL)

return;

if (pB->info == v)

{

found = 1;

ct++;

if (ct == 2)

{

cout << pB->info << " ";

found = 0;

}

}

if (found == 1)

{

cout << pT->info << " ";

}

DispBetweenV(pT->pNext,pT, v, found, ct);

}

**Problem 2: (19 marks)**

Write a main function to do the following:

1. Read a Linked List of Binary Trees from the user.

**Note:** you are responsible to declare the CListNode.

**Also:** you are responsible to declare the CTreeNode.

1. For each tree Calculate the average of **Leafs**,

e.g.

**1st tree’ Average**  = 99. [20 + 90 + 97 + 140 + 150 = 497/ 5]

**2nd tree’ Average** = 88. [65+ 95 + 105 = 265 / 3]

**3rd tree’ Average** =112. [50+ 60 + 90 + 250 = 450 / 4]

Head

**65**

60

90

85

**150**

145

135

130

**90**

**97**

80

95

88

**105**

100

**95**

**140**

**20**

99

80

**90**

70

**60**

**50**

55

**250**

1. For each tree :
   * Search in the other trees about some node equals to the average of the tree
   * And, connect the founded node to the first leaf.
   * And, catch this node into your ListNode.

**88**

**90**

**20**

**60**

**50**

55

70

**99**

80

**65**

60

**95**

**105**

100

90

85

**140**

**150**

80

145

135

130

Head

**90**

**97**

95

**250**

Avg = 112

Avg = 88

Avg = 88

Avg = 88

Avg = 99

1. Connect the leafs in each tree

**250**

**90**

**20**

**60**

**50**

55

70

99

80

**65**

60

**95**

**105**

100

90

85

**140**

**150**

80

145

88

135

130

Head

**90**

**97**

95

1. Traverse your list and display the founded nodes , and its leafs.

e.g.

**65**

**95**

**105**

88

99

**90**

**97**

**20**

**140**

**150**

class CNode

{

public:

CNode\* pNext;

CTNode\* pDownT;

CTNode\* pAvg;

};

class CTNode

{

public:

int info;

CTNode\* pLeft;

CTNode\* pRight;

CTNode\* pSide;

};

void CalcAvg(CTNode\* pT, int& tot, int& ct)

{

if (pT == NULL)

return;

CalcAvg(pT->pLeft, tot, ct);

if (pT->pLeft == NULL && pT->pRight == NULL)

{

tot += pT->info;

ct++;

}

CalcAvg(pT->pRight, tot, ct);

}

void FirstLeaf(CTNode\* pT, int& k, CTNode\*&pFirst)

{

if (pT == NULL || k == 1)

return;

FirstLeaf(pT->pLeft, k , pFirst);

if (pT->pLeft == NULL && pT->pRight == NULL)

{

k++;

}

if (k == 1)

{

pFirst = pT;

}

FirstLeaf(pT->pLeft, k , pFirst);

}

void ConnectLeafs(CTNode\* pT, CTNode\*&pPrev)

{

if (pT == NULL)

return;

ConnectLeafs(pT->pLeft,pPrev);

if (pT->pLeft == NULL && pT->pRight == NULL)

{

if (pPrev == NULL)

{

pPrev = pT;

}

else

{

pPrev->pSide = pT;

pPrev = pT;

}

}

ConnectLeafs(pT->pRight,pPrev);

}

void main()

{

CBST T;

CList L;

int N, m, tot = 0, ct = 0;

cin >> N;

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

{

CNode\* pnn = new CNode;

pnn->pNext = NULL;

pnn->pDownT = NULL;

pnn->pAvg = NULL;

cin >> m;

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

{

CTNode\* pnT = new CTNode;

cin >> pnT->info;

pnT->pLeft = pnT->pRight = NULL;

T.Insert(pnT);

}

pnn->pDownT = T.pRoot;

L.Attach(pnn);

T.pRoot = NULL;

}

CNode\* pTrav = L.pHead;

CTNode\* pFirst = NULL;

int avg;

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

{

tot = 0, ct = 0;

CalcAvg(pTrav->pDownT, tot, ct);

avg = tot / ct;

int k = 0;

pFirst = NULL;

FirstLeaf(pTrav->pDownT, k, pFirst);

CNode\* pT = L.pHead;

for (int j = 0; pT != NULL; j++)

{

pT = L.pHead;

CTNode\* pD = NULL;

if (j != i)

{

pD = pT->pDownT;

while (pD != NULL)

{

if (avg > pD->info)

{

pD = pD->pRight;

}

else

{

pD = pD->pLeft;

}

}

pT->pAvg = pD;

pD->pSide = pFirst;

}

}

pTrav = pTrav->pNext;

}

pTrav = L.pHead;

CTNode\* pPrev = NULL;

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

{

ConnectLeafs(pTrav->pDownT,pPrev);

pTrav = pTrav->pNext;

}

pTrav = L.pHead;

CTNode\* ptt = NULL;

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

{

ptt = pTrav->pAvg;

while (ptt != NULL)

{

cout << ptt->info << " ";

ptt = ptt->pSide;

}

pTrav = pTrav->pNext;

}

}