**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 .**

**(Q2) (5 mark)**

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

**41 , 60 , 42 , 43 , 65 , 45 , 25 , 18**

1. Store the values in a hash table with size 8 cells.
2. Store the values in a hash table with 3 buckets, each bucket contains 3 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)** |
| **65** |  |  |  |
| **25** |  |  |  |
| **18** |  |  |  |

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

**Algorithm 1: (8 marks)**

You are asked to write a function called **Remove\_The\_Middle\_Leaf** **()**

* Your function will remove :
  + The leaf at the middle of all leafs.

|  |
| --- |
| 600  160  110  500  122  155  130  **220**  270  260  250  200  390  399  395  450  400  **70**  90  80  50  112  115  120  **We have 9 leafs in this tree >> [ 70 , 90 , 112 , 120 , 220 , 270 , 399 , 450 , 600 ]** |

void CountLeafs(CTNode\* pT, int& ct)

{

if (pT == NULL)

{

return;

}

CountLeafs(pT->pLeft, ct);

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

{

ct++;

}

CountLeafs(pT->pRight, ct);

}

void FindLeaf(CTNode\* pT, CTNode\*& pMid,CTNode\*pB, CTNode\*&pMB, int middle, int& k)

{

if (pT == NULL)

return;

FindLeaf(pT->pLeft, pMid, pT, pMB, middle, k);

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

{

k++;

}

if (k == middle)

{

pMB = pB;

pMid = pT;

}

FindLeaf(pT->pRight, pMid, pT, pMB, middle, k);

}

void Remove\_The\_Middle\_Leaf()

{

CTNode\* pTrav = pRoot;

int ct = 0;

CTNode\* pMid = NULL;

CTNode\* pB = NULL;

CTNode\* pMB = NULL;

CountLeafs(pTrav, ct);

int middle = (ct + 1) / 2; //odd number of leafs

pTrav = pRoot;

int k = 0;

FindLeaf(pTrav, pMid, pB, pMB, middle, k);

if (pMB->info > pMid->info)

{

pMB->pLeft = NULL;

}

else

{

pMB->pRight = NULL;

}

delete pMid;

}

**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 2 values **(V1)** and **(V2)** , but in reverse manner.

**NOTE: we don’t know if (V1) will come before or after (V2) in the list.**

**e.g. 🡪 V1 = 80 , V2 = 70**

L

Head

**Output will be: 60 , 43 , 22 , 30 , 20 , 10**

void ReverseDisplay(CNode\* pT, int v1, int v2, int& ct)

{

if (pT == NULL)

{

return;

}

if (pT->info == v1 || pT->info == v2)

{

ct++;

if (ct == 2)

{

ct--;

return;

}

}

ReverseDisplay(pT->pNext, v1, v2, ct);

if (pT->info == v1 || pT->info == v2)

{

ct--;

}

if (ct != 0)

{

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

}

} **Problem 2: (19 marks)**

Write a main function to do the following:

1. Read a Binary Tree.

84

83

86

70

250

400

90

200

800

500

Root

**380**

The **Largest** Value is (380)

of the Sub-Tree of the target value at **Level 9**

e.g.

**Target** Value 270 at **Level 5**

**Level 10**

**Level 9**

**Level 8**

**Level 7**

**Level 6**

**Level 5**

**Level 4**

**Level 0**

**Level 1**

**Level 2**

**Level 3**

360

340

300

290

**270**

265

260

910

850

900

85

1. Ask the user to enter target value **(TV)** , in the above example TV = 270.
2. Findthe (TV), and calculateits level (iLevelTarget) .

In the above example**: iLevelTarget = 5**

1. In the Sub-tree of the target value 🡪 Find the largest value (380), and calculate its level (iLevelLargest)

In the above example**: iLevelLargest = 9**

1. Calculate the difference between the 2 levels, and call it **(OUR\_DIFF).**

Head

In the above example 🡪 OUR\_DIFF = 9 - 5 = 4.

1. Read linked list of trees.

800

950

90

100

2700

200

25

20

700

200

100

**270**

23

**270**

250

860

850

550

**270**

**950**

**700**

520

480

460

300

500

280

400

590

550

580

**TRUE iLevelTarget = 3**

**iLevelLargest = 7**

difference = 4

**600**

**………………**

**………….**

**FALSE iLevelTarget = 3**

**iLevelLargest = 6**

difference = 3

140

990

900

850

**Note:** declare the data structure of your CListNode.

1300

2200

**TRUE iLevelTarget = 2**

**iLevelLargest = 6**

difference = 4

1. Find how many trees that satisfy the following:
   * The tree includes the target value **(TV).**
   * The difference between the **level** of the **target**

And the **level** of the **largest** value in its sub-tree is equal to **(OUR\_DIFF).**

In the above example**: there is only 2 trees stratify the required conditions.**

class CNode

{

public:

CNode\* pNext;

CTNode\* pDown;

};

void FindTarget(CTNode\* pT, int TV, int&iLevelTarget, int level, CTNode\*&pTarget)

{

if (pT == NULL)

return;

FindTarget(pT->pLeft, TV, iLevelTarget, level + 1, pTarget);

if (pT->info == TV)

{

iLevelTarget = level;

pTarget = pT;

}

FindTarget(pT->pRight, TV, iLevelTarget, level + 1, pTarget);

}

void main()

{

CBST T;

CList L;

int N;

cin >> N;

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

{

CTNode\* pnT = new CTNode;

cin >> pnT->info;

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

T.Insert(pnT);

}

int TV, iLevelTarget = 0, iLevelLargest = 0, level = 0;

CTNode\* pTarget = NULL;

CTNode\* pTrav = T.pRoot;

cin >> TV;

FindTarget(pTrav, TV, iLevelTarget, level, pTarget);

pTrav = pTarget;

iLevelLargest = iLevelTarget; //to start counting after target level

while (pTrav->pRight != NULL)

{

pTrav = pTrav->pRight;

iLevelLargest++;

}

int OUR\_DIFF = iLevelLargest - iLevelTarget;

int m;

CBST Tree;

cin >> N;

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

{

CNode\* pnn = new CNode;

pnn->pNext = NULL;

pnn->pDown = NULL;

cin >> m;

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

{

CTNode\* pnT = new CTNode;

cin >> pnT->info;

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

Tree.Insert(pnT);

}

pnn->pDown = Tree.pRoot;

L.Attach(pnn);

Tree.pRoot = NULL;

}

int ctTrue = 0;

CNode\* pLTrav = L.pHead;

int newDiff = 0;

while (pLTrav != NULL)

{

iLevelTarget = 0, iLevelLargest = 0, level = 0, newDiff = 0;

pTarget = NULL;

pTrav = pLTrav->pDown;

FindTarget(pTrav, TV, iLevelTarget, level, pTarget);

pTrav = pTarget;

iLevelLargest = iLevelTarget; //to start counting after target level

while (pTrav->pRight != NULL)

{

pTrav = pTrav->pRight;

iLevelLargest++;

}

newDiff = iLevelLargest - iLevelTarget;

if (OUR\_DIFF == newDiff)

{

ctTrue++;

}

pLTrav = pLTrav->pNext;

}

}