**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 K – W + Z \* – + E M N Q**

**(Q2) (5 mark)**

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

**22 , 16 , 15, 14 , 31 , 11 , 25**

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 % 3).
4. Fill in the following table, showing the number of comparisons needed to find each value:-

|  |  |  |  |
| --- | --- | --- | --- |
|  | **(i)** | **(ii)** | **(iii)** |
| **15** |  |  |  |
| **14** |  |  |  |
| **25** |  |  |  |

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

**Algorithm 1: (8 marks)**

You are asked to write a function called

Void **Find\_And\_Disp\_N\_Nodes\_of\_Right\_Branch** **(** **int** **V** **, int N)**

* Your function will receive an integer **(V)** that represents some value.
* You will find the node which contains (**V**).
* Then your function should
  + display the 1st **(N)** nodes of the right branch of (V)
  + then display also the last **(N)** nodes of the right branch of (V).

**e.g. V = 200 , N = 3**

273

277

98

94

95

360

330

275

800

25

50

65

60

100

**250**

750

650

70

700

600

400

80

500

90

Root

**200**

**Output will be :**

250

270

280

335

350

380

**270**

**380**

**350**

**335**

320

285

**280**

void Find\_And\_Disp\_N\_Nodes\_of\_Right\_Branch(int v, int N)

{

CTNode\* pT = pRoot;

CTNode\* pV = NULL;

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

{

if (v > pT->info)

{

pT = pT->pRight;

}

else

{

pT = pT->pLeft;

}

}

pV = pT;

int len = 0, ct = 0;

while (pT->pRight != NULL)

{

pT = pT->pRight;

len++;

}

pT = pV;

while (pT->pRight != NULL)

{

if (ct < N)

{

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

}

if (ct >= (len - N + 1))

{

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

}

pT = pT->pRight;

ct++;

}

}

**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:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 2 | 7 | 4 | 12 | 2 |

L

14

12

13

8

7

6

11

10

9

4

3

5

2

1

0

Head

* Each pair of cells in the array carries the following information
  + The 1st one is the required **position** in the List (L).
  + The 2nd one is the **length** of nodes to be displayed.
* The size of the array will be one of the passed parameters to your function.
  + **Assume** the size of the array will be even.

**Output will be:**

**70, 20, 43, 60, 50, 68, 40, 15.**

void DispSpecial(CNode\* pT, int array[], int size, CNode\*& pTrav, int& j, int& k, int& ct)

{

if (pT == NULL)

return;

if (ct == array[j])

{

pTrav = pT;

for (int z = 0; z < array[k]; z++)

{

cout << pTrav->info << " ";

pTrav = pTrav->pNext;

}

j += 2;

k += 2;

}

ct++;

DispSpecial(pT->pNext, array, size, pTrav, j, k, 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.

1. Calculate the total of all leafs in the 1st tree. (name it **Tot1**)

in the example **Tot1** = 310. [20 + 140 +150 = 310]

Head

**65**

60

90

85

**150**

110

145

122

135

130

**103**

100

**95**

**140**

**20**

**200**

80

70

**60**

**50**

55

1. Find another tree such that the total of its leafs equals to **Tot1**.

in the example the 3rd tree is the matched tree that satisfy the required

**Total of leafs of the 3rd tree** = 310. [50 + 60 +200 = 310]

1. Display the leafs of the matched tree.

e.g

50 , 60 , 200

1. Copy the non-leafs nodes from the 1st tree.

And paste it into a new node in the list.

Head

1. Copy the non-leafs nodes from the matched tree.

And paste it into another new node in the list.

**60**

**50**

55

70

**200**

80

**95**

**103**

100

90

85

**65**

60

**140**

**150**

110

145

122

135

130

**20**

The list after steps (5 & 6)

Head

70

80

55

55

70

**200**

80

**60**

**50**

**65**

60

**95**

**103**

100

90

85

110

145

122

135

130

**140**

**150**

110

145

122

135

130

**20**

class CNode

{

public:

CNode\* pNext;

CTNode\* pDownT;

};

void TotalLeafs(CTNode\* pT, int& tot1)

{

if (pT == NULL)

return;

TotalLeafs(pT->pLeft, tot1);

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

{

tot1 += pT->info;

}

TotalLeafs(pT->pRight, tot1);

}

void CopyNodes(CTNode\* pT,CBST Tree)

{

if (pT == NULL)

return;

if (pT->pLeft != NULL || pT->pRight != NULL)

{

CTNode\* pnT = new CTNode;

pnT->info = pT->info;

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

Tree.Insert(pnT);

}

CopyNodes(pT->pLeft, Tree);

CopyNodes(pT->pRight, Tree);

}

void DisplayLeafs(CTNode\* pT)

{

if (pT == NULL)

return;

DisplayLeafs(pT->pLeft);

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

{

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

}

DisplayLeafs(pT->pRight);

}

void main()

{

CBST T;

CList L;

int N, m;

int tot1 = 0, tot2 = 0;

cin >> N;

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

{

CNode\* pnn = new CNode;

pnn->pNext = NULL;

pnn->pDownT = 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;

CNode\* pt = L.pHead;

for (int i = 0; pTrav != NULL; i++)

{

tot1 = 0;

TotalLeafs(pTrav->pDownT, tot1);

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

{

pt = L.pHead;

tot2 = 0;

if (i == 0)

{

CBST Tree;

CNode\* pnn = new CNode;

pnn->pNext = NULL;

pnn->pDownT = NULL;

CopyNodes(pTrav->pDownT, Tree);

pnn->pDownT = Tree.pRoot;

L.Attach(pnn);

}

if (i != j)

{

TotalLeafs(pt->pDownT, tot2);

if (tot1 == tot2)

{

DisplayLeafs(pt->pDownT);

CBST Tree;

CNode\* pnn = new CNode;

pnn->pNext = NULL;

pnn->pDownT = NULL;

CopyNodes(pt->pDownT, Tree);

pnn->pDownT = Tree.pRoot;

L.Attach(pnn);

}

}

}

pTrav = pTrav->pNext;

}

}