**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 + k – \* L M N S .**

**(Q2) (5 mark)**

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

**12 , 15 , 14 , 22 , 29 , 20**

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 2 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)** |
| **14** |  |  |  |
| **22** |  |  |  |
| **20** |  |  |  |

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

**Algorithm 1: (8 marks)**

You are asked to write a function called **Display\_Uncle\_of\_Leafs** **()**

* Your function will display each leafs and :
  + The uncle of the leaf in the tree (if there is an uncle of the leaf).

Uncle 🡪 means the brother of its parent.

|  |
| --- |
| **90**  80  50  115  180  **399**  **170**  **70**  160  110  500  122  155  130  260  250  200  390  395  400  **600**  **120**  **112**  **450**  **Output will be:**  **70** I’m a leaf without uncle.  **220**  **90** I’m a leaf without uncle.  **112** I’m a leaf & my uncle : **50**  **270**  **120** I’m a leaf & my uncle : **50**  **170** I’m a leaf & my uncle : **250**  **220** I’m a leaf & my uncle : **180**  **270** I’m a leaf & my uncle : **220**  **399** I’m a leaf & my uncle : **200**  **450** I’m a leaf without uncle.  **600** I’m a leaf without uncle.  void Display\_Uncle\_of\_Leafs(CTNode\*pT,CTNode\*pB,CTNode\*pBB)  {  if(pT==NULL)  {  return;  }  Display\_Uncle\_of\_Leafs(pT->pLeft, pT, pB);  if (pT->pLeft == NULL && pT->pRight == NULL)  {  if (pBB != NULL)  {  if (pBB->pRight == pB) //if the parent is on the right of the grandparent  {  if (pBB->pLeft != NULL) //if the left sibling is NOT empty (sibling present)  {  cout << pT->info << ", " << pBB->pLeft->info; //uncle  }  else  {  cout << pT->info << "& " << pBB->pRight->info; //uncle  }    }  }  }  Display\_Uncle\_of\_Leafs(pT->pRight, pT, pB);  } |

**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 a 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 Reverse the values in between 2 values **(V1)** and **(V2)** ,

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

**Assume: (V1 , V2) occurred only once in the list.**

**NOTE: your reversed logic should be address based not info based.**

**e.g. 🡪 V1 = 80 , V2 = 70**

L

Head

**Output will be:**

L

Head

void FindReverse(CNode\* pT, CNode\* pB, int v1, int v2, int& ct, CNode\*& pC, CNode\*& pC2)

{

if (pT == NULL)

{

return;

}

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

{

ct++;

if (ct == 2)

{

pC = pT;

pC2 = pB;

ct--;

return;

}

}

FindReverse(pT->pNext, pT, v1, v2, ct, pC, pC2);

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

{

ct--;

pT->pNext->pNext = pC;

pT->pNext = pC2;

}

if (ct != 0)

{

pT->pNext = pB;

}

}

**Problem 2: (19 marks)**

Write a main function to do the following:

1. Read a Linked List of Binary Search Trees.

Note: Declare the data structure of CListNode.

**4**

**3**

**2**

**1**

**0**

Head

**…**

**…**

**…**

**…**

**…**

**…**

1. Ask the user to enter target value **(TV)** , in the example TV = 73.

Ask the user to select tree by position **(iPos)** , in the example iPos = 4.

Find **(TV)** in the selected tree **(iPos)**

75

72

**R**

**L**

73

815

360

50

20

10

25

8

30

80

35

75

85

84

72

86

**70**

900

850

250

280

90

200

910

800

500

**5**

**3**

**2**

**1**

**0**

**…**

Head

**73**

**R**

**R**

**L**

**R**

**L**

**L**

**R**

**R**

6

**R**

**R**

**R**

**L**

**L**

**L**

**L**

5

**4**

**L**

70

**73**

60

400

***×***

1. Find another tree that includes the (TV) **but** should appear in the **reverse** path.

The tree that satisfy those requirements is at position (1)

No permission to use additional list or array

**R, R, R, L, R, L, R, R (path in tree 4)**

**L, L, L, R, L, R, L, L (path in tree 1)**

1. In the previous 2 trees: cut the subtree of the root that includes (TV) and paste it into a NewListNode.

**In the example**

**1st tree at position 🡪 1 : (TV = 73) at the left subtree of the root (500)**

**2nd tree at position 🡪 4 : (TV = 73) at the right subtree of the root (10)**

Head

**5**

**4**

**3**

**2**

**1**

**0**

**…**

8

5

10

6

800

500

900

850

910

30

20

75

72

**73**

70

60

80

35

400

25

280

90

**73**

75

85

84

72

86

**70**

200

250

73

815

360

50

class CNode

{

public:

CNode\* pNext;

CTNode\* pDown; **!!TAKE CARE OF THE DIFFERENCE**

};

void main()

{

CList L;

CBST T;

int N, m, TV, iPos;

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;

T.Insert(pnT);

}

pnn->pDown = T.pRoot;

L.Attach(pnn);

T.pRoot = NULL;

}

cin >> TV;

cin >> iPos;

CNode\* pPos = L.pHead;

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

{

pPos = pPos->pNext;

}

CNode\* pTrav = L.pHead;

while (pTrav != NULL)

{

CTNode\* pP = pPos->pDown;

CTNode\* pT = pTrav->pDown;

while (pT != NULL && pP != NULL && pP->info != TV)

{

if (TV > pP->info)

{

pP = pP->pRight;

pT = pT->pLeft;

}

else

{

pP = pP->pLeft;

pT = pT->pRight;

}

}

if (pT != NULL && pT->info == TV)

{

break;

}

pTrav = pTrav->pNext;

}

//pTrav is at node 1 and pPos is at node 4

CNode\* pnn = new CNode;

pnn->pNext = NULL;

if (TV > pTrav->pDown->info)

{

pnn->pDown = pTrav->pDown->pRight;

pTrav->pDown->pRight = NULL;

}

else

{

pnn->pDown = pTrav->pDown->pLeft;

pTrav->pDown->pLeft = NULL;

}

L.Attach(pnn);

pnn = new CNode;

pnn->pNext = NULL;

if (TV > pPos->pDown->info)

{

pnn->pDown = pPos->pDown->pRight;

pPos->pDown->pRight = NULL;

}

else

{

pnn->pDown = pPos->pDown->pLeft;

pPos->pDown->pLeft = NULL;

}

L.Attach(pnn);

}