**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 M \* Q S.**

**(Q2) (5 mark)**

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

**20 , 31 , 30 , 14 , 29 , 28**

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** |  |  |  |
| **28** |  |  |  |
| **29** |  |  |  |

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

**Algorithm 1: (8 marks)**

You are asked to write a function called **Display\_Good\_Right\_Branch** **()**

* Starting from the root 🡪 if the right branch includes only 1 node that has left child
* Then display this branch, and go to the left child of this node.
* Repeat this process till reach some right branch that includes:
  + more than 1 node with left child.
  + or no nodes with left child.

|  |
| --- |
| 130  600  160  **400**  450  **390**  395  180  170  270  220  200  260  399  250  155  **500**  Good right branch  Has only 1 node has left child  Good right branch  Has only 1 node has left child  Good right branch  Has only 1 node has left child  210  160  230  **bad** right branch  Has 2 nodes with left child  **So, stop your display here.**  **Output will be:**  **130, 155 , 500 , 600**  **160, 400 , 450**  **390, 395, 399**  void Display\_Good\_Right\_Branch()  {  CTNode\* pStart = pRoot;  CTNode\* pTrav = pRoot;  int ct = 0;  while (1)  {  ct = 0;  while (pTrav != NULL)  {  if (pTrav->pLeft != NULL)  {  ct++;  }  pTrav = pTrav->pRight;  }  if (ct > 1)  {  break;  }  if (ct == 1)  {  pTrav = pStart;  while (pTrav != NULL)  {  cout << pTrav->info << " ";  if (pTrav->pLeft != NULL)  {  pStart = pTrav->pLeft;  }  pTrav = pTrav->pRight;  }  }  pTrav = pStart;  cout << endl;  }  } |

**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**

pDown

info

pNext

**{**

**public:**

**int info;**

**CListNode \*pNext;**

**CListNode \*pDown;**

**};**

**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 **cut** all down & **paste** them at the down of the last node.



**e.g.**

|  |  |
| --- | --- |
| L  Head  **Before** | L  Head  **After** |

void Cut\_Paste(CNode\* pT, CNode\* pB, CNode\*& pD)

{

if (pT == NULL)

{

return;

}

if (pD->pDown != NULL)

{

pD = pD->pDown;

Cut\_Paste(pT, pB, pD);

}

else

{

if (pB != NULL)

{

pD->pDown = pB->pDown;

pB->pDown = NULL;

}

pD = pT->pNext;

Cut\_Paste(pT->pNext, pT, pD);

}

}

**Problem 2: (19 marks)**

**class CListNode**

pNext

V2

V1

**{**

**public:**

**int V1;**

**int V2;**

**CTreeNode \*pDown;**

**CListNode \*pNext;**

pDown

**};**

Write a main function to do the following:

1. Read a Linked List of Binary Search Trees.

**4**

**3**

**2**

**1**

**0**

Head

**…**

**…**

**…**

**…**

**…**

**…**

1. Ask the user to select a tree by position (aPos)

the selected tree , [in the example aPos = 1]. Which contains (V1 , V2) = (3 , 2).

(V1 = 3) 🡪 means go to level (3) in this selected tree.

(V2 = 2) 🡪 means go to the 1st (2) leafs only in this level.

Ask the user to select another tree (bPos) : [in the example bPos = 4]

Then connect each founded leaf from (aPos) to a node in (bPos) but which in the same path.

**bPos**

**aPos**

20

10

25

8

50

49

45

75

85

84

72

86

70

900

250

280

90

200

910

800

500

**5**

**3**

**2**

**1**

**0**

**…**

Head

6

5

**4**

47

46

400

**R**

**R**

**L**

**L**

**R**

**R**

**R**

**R**

**Level 3**

**L**

**L**

**L**

**L**

970

7

**Note**: Declare the data structure of CTreeNode.

**Note**: No permission to use additional list or array

1. For each connected node in (bPos)

Display the largest value of its sub-tree.

The largest value for this connected node 🡪 7

**7**

6

250

The largest value for this connected node 🡪 49

46

**49**

45

25

900

47

class CTNode

{

public:

int info;

CTNode\* pLeft;

CTNode\* pRight;

CTNode\* pSide;

};

void FindLeafs(CTNode\*pT,int level,int E,int&ct,CTNode\*pL1,CTNode\*pL2)

{

if (pT == NULL)

{

return;

}

FindLeafs(pT->pLeft, level + 1, E, ct, pL1, pL2);

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

{

ct++;

if (ct == 1 && level == E)

{

pL1 = pT;

}

if (ct == 2 && level == E)

{

pL2 = pT;

}

}

FindLeafs(pT->pRight, level + 1, E, ct, pL1, pL2);

}

void main()

{

CList L;

CBST Tree;

int N, n, apos, bpos;

cin >> N;

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

{

CNode\* pnn = new CNode;

cin >> pnn->v1 >> pnn->v2;

pnn->pNext = NULL;

cin >> n;

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

{

CTNode\* pnT = new CTNode;

cin >> pnT->info;

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

Tree.Insert(pnT);

}

pnn->pDown = Tree.pRoot;

Tree.pRoot = NULL;

L.Attach(pnn);

}

cin >> apos;

cin >> bpos;

CNode\* pTrav = L.pHead;

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

{

pTrav = pTrav->pNext;

}

int v1 = pTrav->v1;

int v2 = pTrav->v2;

CTNode\* pT = pTrav->pDown;

CNode\* pTrav2 = L.pHead;

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

{

pTrav2 = pTrav2->pNext;

}

CTNode\* pT2 = pTrav2->pDown;

CTNode\* pL1 = NULL, \* pL2 = NULL;

int ct = 0;

FindLeafs(pT, 0, v1, ct, pL1, pL2);

pT = pTrav->pDown;

pT2 = pTrav2->pDown;

while (pT != NULL && pT2 != NULL)

{

if (pL1->info < pT->info)

{

pT = pT->pLeft;

pT2 = pT2->pLeft;

}

else

{

pT = pT->pRight;

pT2 = pT2->pRight;

}

}

pL1->pSide = pT2;

pT = pTrav->pDown;

pT2 = pTrav2->pDown;

while (pT != NULL && pT2 != NULL)

{

if (pL2->info < pT->info)

{

pT = pT->pLeft;

pT2 = pT2->pLeft;

}

else

{

pT = pT->pRight;

pT2 = pT2->pRight;

}

}

pL2->pSide = pT2;

pT = pL1->pRight;

pT2 = pL2->pRight;

while (pT->pRight != NULL)

{

pT = pT->pRight;

}

cout << pT->info;

while (pT2->pRight != NULL)

{

pT2 = pT2->pRight;

}

cout << pT2->info;

}