**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 - M + + K W Z + + Y H Q**

**(Q2) (5 mark)**

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

**30 , 41 , 31 , 24 , 46 , 18 , 22 , 15**

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)** |
| **46** |  |  |  |
| **22** |  |  |  |
| **15** |  |  |  |

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

**Algorithm 1: (8 marks)**

You are asked to write a function called **Remove\_The\_first\_ and\_The\_Last\_Leaf** **()**

* Your function will receive remove both:
  + The first leaf in the binary search Tree.
  + The last leaf in the binary search Tree.

|  |
| --- |
| Last Leaf in the Tree  160  110  500  122  155  130  220  270  260  250  200  390  399  395  450  400  **70**  90  80  50  112  120  115  First Leaf in the Tree  **In this example: we have 8 leafs**  **[ 70 , 90 , 112 , 120 , 220 , 270 , 399 450 ]** |

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

{

if (pT == NULL)

return;

CountLeafs(pT->pLeft, ctleafs);

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

{

ctleafs++;

}

CountLeafs(pT->pRight, ctleafs);

}

void RemoveLeafs(CTNode\* pT, CTNode\*pB, int& k, int ctleafs)

{

if (pT == NULL)

return;

RemoveLeafs(pT->pLeft, pT, k, ctleafs);

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

{

k++;

if (k == 1)

{

if (pT->info > pB->info)

{

pB->pRight = NULL;

delete pT;

}

else

{

pB->pLeft = NULL;

delete pT;

}

}

if (k == ctleafs)

{

if (pT->info > pB->info)

{

pB->pRight = NULL;

delete pT;

}

else

{

pB->pLeft = NULL;

delete pT;

}

}

}

RemoveLeafs(pT->pRight, pT, k, ctleafs);

}

void Remove\_The\_first\_and\_The\_Last\_Leaf(CBST& T)

{

int ctleafs = 0;

int k = 0;

CountLeafs(T.pRoot, ctleafs);

RemoveLeafs(T.pRoot, NULL, k, ctleafs);

}

**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 before the given value **(V)** , but in reverse manner.

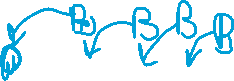
**Assume** that the required **(V)** , will exist in the list only 1 time.



**e.g.**



**V = 70**



L



Head

**Output will be: 60 , 43 , 22 , 30 , 20 , 10 , 80 , 77 .**

void ReverseDisplay(CNode\* pT, CNode\* pB, int v)

{

if (pT == NULL || pT->info == v)

{

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

return;

}

ReverseDisplay(pT->pNext, pT, v);

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

}

**Problem 2: (19 marks)**

Write a main function to do the following:

1. Read a Binary Tree **(T1)**.
2. Find the node which has the longest **Left branch**. (pLongest)

In this example (node [1500] because its left branch with Length = 5 nodes)

**T1**

80

81

82

84

83

86

70

250

280

90

200

800

500

Root

**(3)** Cut the Left Sub-Tree of the **pLongest** , and add it to a Linked List of trees.

**pLongest**

920

940

910

950

1200

**1500**

1900

910

850

900

1100

1000

1350

1180

1300

85

Head

**L**

1200

950

910

920

1180

940

1100

1000

1350

1300

(4) Repeat steps (2 & 3) as much as the user needs.

80

1900

85

81

84

82

83

86

70

1500

900

850

250

280

90

200

910

800

500

Root

**L**

**……………….**

Head

1100

1000

1350

1300

1200

950

910

920

1180

940

84

82

83

80

85

81

(5) Display the average of each tree in the List **(L)**.

(6) Display the lowest node of each tree in the List **(L)**.

void CountLeft(CTNode\* pT, int& max, int ctLeft, CTNode\*& pLongest)

{

if (pT == NULL)

return;

CountLeft(pT->pLeft, max, ctLeft, pLongest);

CTNode\* ptt = pT;

ctLeft = 0;

while (ptt->pLeft != NULL)

{

ctLeft++;

}

if (ctLeft > max)

{

max = ctLeft;

pLongest = pT;

}

ctLeft = 0;

CountLeft(pT->pRight, max, ctLeft, pLongest);

}

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

{

if (pT == NULL)

return;

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

tot += pT->info;

ct++;

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

}

void main()

{

CBST T1;

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;

T1.Insert(pnT);

}

int max = -9999, choice=1;

CTNode\* pLongest = NULL;

while (choice == 1)

{

max = -9999;

pLongest = NULL;

CountLeft(T1.pRoot, max, 0, pLongest);

CNode\* pnn = new CNode;

pnn->pDownL = NULL;

pnn->pDownT = pLongest->pLeft;

pLongest->pLeft = NULL;

L.Attach(pnn);

cout << "Do you want to repeat?\n 1.Yes\n 2.No\n";

cin >> choice;

}

CNode\* pTrav = L.pHead;

CTNode\* pSmallest = NULL;

int avg = 0, tot = 0, ct = 0;

while (pTrav != NULL)

{

tot = 0, ct = 0;

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

avg = tot / ct;

cout << "average: " << avg << endl;

pSmallest = pTrav->pDownT;

while (pSmallest->pLeft != NULL)

{

pSmallest = pSmallest->pLeft;

}

cout << pSmallest->info << endl;

pTrav = pTrav->pNext;

}

}