**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 / K M W.**

**(Q2) (5 mark)**

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

**38 , 57 , 39 , 40 , 62 , 42 , 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)** |
| **62** |  |  |  |
| **22** |  |  |  |
| **15** |  |  |  |

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

**Algorithm 1: (8 marks)**

You are asked to write a function called **First\_\_Left\_Display** **(** **int** **V** **)**

* Your function will receive an integer that represents some Target Value **(V)**.
* You will find **(V)**.
  + .then display its path, but starting from the 1st left turn.

|  |  |
| --- | --- |
| **e.g. V = 220**  450  400  70  90  80  50  112  120  115  160  600  110  500  122  155  130  220  270  260  250  200  390 | **---------------------------------------------------------**  **160, 400, 390, 200, 250, 220**  **---------------------------------------------------------** |

void First\_\_Left\_Display(int v)

{

CTNode\* pTrav = pRoot;

int check = 0;

while (pTrav != NULL)

{

if (v > pTrav->info)

{

pTrav = pTrav->pRight;

}

else

{

check = 1;

pTrav = pTrav->pLeft;

}

if (check == 1)

{

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

}

}

}

**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 2nd half of 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

8 values before the (V)

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

**Another example**

**e.g. 🡪 V = 30**

L

4 values before the (V)

Head

**Output will be: 20 , 10**

void ReverseDisp(CNode\* pT, int v)

{

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

return;

ReverseDisp(pT->pNext, v);

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

280

90

200

800

500

Root

**Lowest Node at Level 3**

**Level 7**

**Level 6**

**Level 5**

**Level 4**

**Level 3**

**Level 2**

**Level 1**

**Level 0**

910

850

900

85

1. Findthe lowest node in the tree , and calculate its level (iLevel) .

In the above example**: iLevel = 3**

1. Attach this tree to a list **but** at the node of position (iLevel).

**99**

**5**

**4**

**3**

**2**

**1**

**0**

**…**

Head

**70**

900

850

250

280

90

200

910

800

500

85

84

83

86

**Note1**: you have to declare the data structure of the CListNode

**Note2**: at first , you have to create 100 empty nodes in your List , before attaching any tree to it.

1. Repeat steps ( 1: 3) as much as the user needs.

**Note3**: in case if any new tree needs to attached to node in the List that already carries tree 🡪 then don't attach this tree to the List (just neglect it).

**5**

**…**

**4**

500

**3**

**2**

**1**

**0**

Head

85

84

83

86

**70**

900

850

250

280

90

200

910

800

90

100

150

**70**

130

140

600

300

400

600

**10**

120

118

110

115

90

**20**

117

119

1. Display the count of nodes for each tree in the List.

In the example:

3 , 0, 0, 14, 0 , 16, ….

class CNode

{

public:

CNode\* pNext;

CTNode\* pDownT;

};

void CountAll(CTNode\* pT, int& ctNodes)

{

if (pT == NULL)

return;

CountAll(pT->pLeft, ctNodes);

ctNodes++;

CountAll(pT->pRight, ctNodes);

}

void main()

{

CBST T;

CList L;

for (int i = 0; i < 100; i++)

{

CNode\* pnn = new CNode;

pnn->pNext = NULL;

pnn->pDownT = NULL;

L.Attach(pnn);

}

int N, choice = 1, iLevel = 0, ctNodes = 0;

CTNode\* pTrav = NULL;

CNode\* pTravL = NULL;

while (choice == 1)

{

cin >> N;

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

{

CTNode\* pnT = new CTNode;

cin >> pnT->info;

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

T.Insert(pnT);

}

pTrav = T.pRoot;

iLevel = 0;

while (pTrav->pLeft != NULL)

{

pTrav = pTrav->pLeft;

iLevel++;

}

pTravL = L.pHead;

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

{

pTravL = pTravL->pNext;

}

if (pTravL->pDownT == NULL)

{

pTravL->pDownT = T.pRoot;

T.pRoot = NULL;

}

cout << "Do you want to create and attach another tree?\n 1.Yes\n 2.No";

cin >> choice;

}

pTravL = L.pHead;

while (pTravL != NULL)

{

ctNodes = 0;

if (pTravL->pDownT != NULL)

{

CountAll(pTravL->pDownT, ctNodes);

cout << ctNodes << " ";

}

else

{

cout << 0 << " ";

}

pTravL = pTravL->pNext;

}

}