**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 \* G H + K M.**

**(Q2) (5 mark)**

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

**4 , 7 , 6 , 14 , 21 , 12**

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)** |
| **6** |  |  |  |
| **14** |  |  |  |
| **12** |  |  |  |

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

**Algorithm 1: (8 marks)**

You are asked to write a function called **Find\_And\_Display\_1st\_Half**  **(int v )**

* Your function will find the value (V).
  + Then Display the nodes in its path , BUT only in the 1st half of its height.
* e.g. 275 at level 6

so , display all nodes in its path BUT till level 3

**output : 500, 200, 280**

Level 9

Level 8

Level 7

Level 1

Level 6

**500**

Level 3

Level 2

Level 5

Level 4

**275**

850

90

910

800

73

75

85

84

72

86

70

900

**250**

**280**

**200**

**260**

void Find\_And\_Display\_1st\_Half(int v)

{

CTNode\* pT = pRoot;

int ct = 1;

while (pT != NULL)

{

if (pT->info == v)

{

ct++;

break;

}

if (v > pT->info)

{

pT = pT->pRight;

ct++;

}

else

{

pT = pT->pLeft;

ct++;

}

}

ct /= 2;

pT = pRoot;

while (ct > 0)

{

if (v > pT->info)

{

cout << pT->info;

ct--;

pT = pT->pRight;

}

else

{

cout << pT->info;

ct--;

pT = pT->pLeft;

}

}

}

**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 Display **(N)** nodes and neglect **(N)** nodes, and so on.

**e.g. 🡪 N = 3**

L

Head

**Output will be: 77, 80, 10, 43, 60, 70**

void DispSpecial(CNode\* pT, int& ct, int N, int& i)

{

if (pT == NULL)

{

return;

}

ct++;

if (ct <= N)

{

cout << pT->info;

DispSpecial(pT->pNext, ct, N, i);

}

else

{

i++;

if (i == N)

{

i = 0;

ct = 0;

}

DispSpecial(pT->pNext, ct, N, i);

}

}

**dr WAY**

void DoIt(CNode\* pT, int i, int N, int& F)

{

if (pT == NULL)

{

return;

}

if (i % N == 0)

{

if (i % (2 \* N) == 0) //reached the pos (twice 3)

{

F = 1;

}

else

{

F = 0;

}

}

if (F == 1)

{

cout << pT->info;

}

DoIt(pT, i + 1, N, F);

}

**Problem 2: (19 marks)**

Write a main function to do the following:

1. Read a Binary Search Tree.
2. Find the Largest Node & the smallest Node.

e.g. Largest 🡪 1500

Smallest 🡪 80

84

81

83

85

**80**

**1500**

975

1180

980

1100

1000

1351

1300

1200

950

900

850

250

280

90

200

910

800

500

Root

1. In the Sub-Tree of the Largest value: Count the odd leafs.

84

81

83

85

80

**1500**

**975**

**1180**

980

1100

1000

**1351**

1300

1200

950

900

850

250

280

90

200

910

800

500

Root

3 Leafs, but only 2 are odds

1. In the Sub-Tree of the Smallest value: Count the odd leafs.

**84**

**81**

83

85

**80**

1500

975

1180

980

1100

1000

1351

1300

1200

950

900

850

250

280

90

200

910

800

500

Root

2 Leafs, but only 1 is odd

1. For the Largest Node 🡪
   * Create embedded List that carries copy of the odd leafs in its sub-tree.
   * Create another embedded List that carries copy of the even leafs in its sub-tree.

**NOTE**: you are responsible to declare the CTreeNode ‘ Data structure.

84

81

83

85

80

**975**

**1180**

980

1100

1000

**1351**

1300

1200

950

900

850

250

280

90

200

910

800

500

Root

**1500**

1. For the Smallest Node 🡪
   * Create embedded List that carries copy of the odd leafs in its sub-tree.
   * Create another embedded List that carries copy of the even leafs in its sub-tree.

**84**

**81**

83

85

**80**

1500

975

1180

980

1100

1000

1351

1300

1200

950

900

850

250

280

90

200

910

800

500

Root

class CTNode

{

public:

int info;

CTNode\* pLeft;

CTNode\* pRight;

CNode\* pEven;

CNode\* pOdd;

};

void CountOddLeafs(CTNode\* pT, int& ct)

{

if (pT == NULL)

{

return;

}

CountOddLeafs(pT->pLeft, ct);

if (pT->pLeft == NULL && pT->pRight == NULL && pT->info % 2 != 0)

{

ct++;

}

CountOddLeafs(pT->pRight, ct);

}

void FindLeafs(CTNode\* pT, CBST& T, CList& LEven, CList& LOdd)

{

if (pT == NULL)

{

return;

}

FindLeafs(pT->pLeft, T, LEven, LOdd);

CNode\* pnn = new CNode;

pnn->info = pT->info;

pnn->pNext = NULL;

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

{

if (pT->info % 2 == 0)

{

LEven.Attach(pnn);

}

else

{

LOdd.Attach(pnn);

}

}

FindLeafs(pT->pRight, T, LEven, LOdd);

}

void main()

{

CBST T;

CList LEven;

CList LOdd;

int N;

cin >> N;

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

{

CTNode\* pnT = new CTNode;

cin >> pnT->info;

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

pnT->pEven = pnT->pOdd = NULL;

T.Insert(pnT);

}

CTNode\* pLargest = T.pRoot;

CTNode\* pSmallest = T.pRoot;

while (pLargest->pRight != NULL)

{

pLargest = pLargest->pRight;

}

while (pSmallest->pLeft != NULL)

{

pSmallest = pSmallest->pLeft;

}

int ct = 0;

CountOddLeafs(pLargest, ct);

cout << "largest: " << ct << endl;

ct = 0;

CountOddLeafs(pSmallest, ct);

cout << "smallest: " << ct << endl;

FindLeafs(pLargest, T, LEven, LOdd);

pLargest->pEven = LEven.pHead;

LEven.pHead = NULL;

pLargest->pOdd = LOdd.pHead;

LOdd.pHead = NULL;

FindLeafs(pSmallest, T, LEven, LOdd);

pSmallest->pEven = LEven.pHead;

LEven.pHead = NULL;

pSmallest->pOdd = LOdd.pHead;

LOdd.pHead = NULL;

}