**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 N**

**(Q2) (5 mark)**

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

**18 , 50 , 28, 21 , 27 , 17 , 20 , 32**

1. Store the values in a hash table with size 10 cells.
2. Store the values in a hash table with 4 buckets, each bucket contains 3 cells.
3. Store the values in a hash chain table with function (Key % 4).
4. Fill in the following table, showing the number of comparisons needed to find each value:-

|  |  |  |  |
| --- | --- | --- | --- |
|  | **(i)** | **(ii)** | **(iii)** |
| **20** |  |  |  |
| **32** |  |  |  |
| **17** |  |  |  |

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

**Algorithm 1: (8 marks)**

You are asked to write a function called **Special\_Display** **(**  **)**

* Starting from the root , your function will display the current node & its 2 children.
* Then your function will go to the child which includes odd value.
* And repeat this task till reach :
  + Node that hasn’t 2 children
  + Or , the 2 children hasn’t odd value

**Note :** If the 2 children are odds, then go to the right child.

85

80

255

**285**

90

**211**

800

500

Root

**Output:**

500, 211, 800 🡪 Go to 211 (odd)

211, 90, 285 🡪 Go to 285 (odd)

285, 255, 405 🡪 Go to 405 (both are odd)

405, 305, 410 🡪 Go to 305 (odd)

305, 291, 320 🡪 Go to 291 (odd)

291, 286, 295 🡪 Go to 295 (odd)

293

**295**

286

**291**

320

**405**

**305**

410

970

1180

980

1100

1000

1350

1300

1200

950

900

1400

1500

void Special\_Display()

{

CTNode\* pT = pRoot;

while (1)

{

cout << pT->info << " ";

cout << pT->pLeft->info << " ";

cout << pT->pRight->info << " ";

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

{

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

{

pT = pT->pLeft;

}

else if (pT->pRight->info % 2 != 0)

{

pT = pT->pRight;

}

else

{

break;

}

}

}

}

**///////////DR WAY**

void Special\_Display()

{

CTNode\* pT = pRoot;

while (pT->pLeft != NULL && pT->pRight != NULL)

{

cout << pT->info << " ";

if (pT->pLeft != NULL)

{

cout << pT->pLeft->info << " ";

}

if (pT->pRight != NULL)

{

cout << pT->pRight->info << " ";

}

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

{

pT = pT->pRight;

}

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

{

pT = pT->pLeft;

}

else

{

break;

}

}

}

**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;**

**};**

Write a **recursive** function to do:

check that the 4th quarter is a mirror to the 1st quarter.

**Assume:** that the length (number of nodes = 8 in the example) of the list is given to your function as a parameter.

1st quarter

4th quarter

Head

**The output in this example will be 🡪 TRUE**

void Check(CNode\* pT, CNode\*pK, int N, int i, int&check)

{

if (pT == NULL)

return;

Check(pT->pNext, pK, N, i + 1, check);

if (i > N - N / 4)

{

if (pT->info != pK->info)

{

check = 1;

}

pK = pK->pNext;

}

}**Problem 2: (19 marks)**

Write a main function to do the following:

**2**

**4**

**4**

1. Read 2 Binary Trees (**T1 , T2**).
2. Find the largest **leaf** in (T1).
3. Find the lowest **leaf** in (T2).

**240**

230

220

285

**600**

**980**

**505**

**540**

650

590

530

T2

580

**520**

290

950

510

300

820

400

Root

**60**

**655**

T1

660

**85**

**75**

70

80

510

90

920

100

Root

1330

210

200

1530

1500

1400

1340

**1390**

1350

**Largest Leaf**

**Lowest Leaf**

1. Copy the path of the largest leaf to and the path of the lowest leaf to a linked list (but one node from the 1st path & one node from the 2nd path).

**9**

**240**

230

220

210

285

200

**600**

**980**

**505**

**540**

650

590

530

T2

580

**520**

290

950

510

300

820

400

Root

**655**

T1

660

**85**

**75**

70

80

1330

510

90

920

100

Root

**1390**

1350

1340

1400

1500

1530

**60**

L

Head

void FindLowest(CTNode\*pT,CTNode\*&pLowest,int k, int&maxHeight)

{

if (pT == NULL)

return;

FindLowest(pT->pLeft, pLowest, k + 1, maxHeight);

if (k > maxHeight && pT->pLeft!=NULL && pT->pRight!=NULL)

{

maxHeight = k;

pLowest = pT;

}

FindLowest(pT->pRight, pLowest, k + 1, maxHeight);

}

void FindLargest(CTNode\* pT, CTNode\*& pLargest, int&maxValue)

{

if (pT == NULL)

return;

FindLargest(pT->pLeft, pLargest, maxValue);

if (pT->info > maxValue && pT->pLeft != NULL && pT->pRight != NULL)

{

maxValue = pT->info;

pLargest = pT;

}

FindLargest(pT->pRight, pLargest, maxValue);

}

void main()

{

CBST T1;

CBST T2;

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);

}

cin >> N;

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

{

CTNode\* pnT = new CTNode;

cin >> pnT->info;

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

T2.Insert(pnT);

}

CTNode\* pLargest = T1.pRoot;

int maxValue = -9999;

FindLargest(T1.pRoot, pLargest, maxValue);

CTNode\* pLowest = T2.pRoot;

int maxHeight = 0, k=0, f=0;

FindLowest(T2.pRoot, pLowest, k, maxHeight);

CTNode\* pTrav1 = T1.pRoot;

CTNode\* pTrav2 = T2.pRoot;

CList L;

while (pTrav1 != NULL || pTrav2 != NULL)

{

if (f == 0)

{

if (pLargest->info > pTrav1->info)

{

CNode\* pnn = new CNode;

pnn->info = pTrav1->info;

pnn->pNext = NULL;

L.Attach(pnn);

pTrav1 = pTrav1->pRight;

}

else

{

CNode\* pnn = new CNode;

pnn->info = pTrav1->info;

pnn->pNext = NULL;

L.Attach(pnn);

pTrav1 = pTrav1->pLeft;

}

f = 1;

}

else

{

if (pLowest->info > pTrav2->info)

{

CNode\* pnn = new CNode;

pnn->info = pTrav2->info;

pnn->pNext = NULL;

L.Attach(pnn);

pTrav2 = pTrav2->pRight;

}

else

{

CNode\* pnn = new CNode;

pnn->info = pTrav2->info;

pnn->pNext = NULL;

L.Attach(pnn);

pTrav2 = pTrav2->pLeft;

}

f = 0;

}

}

}