**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 K M N \* P + Q Z**

**(Q2) (5 mark)**

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

**46 , 57 , 47 , 40 , 62 , 34 , 38 , 31**

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** |  |  |  |
| **38** |  |  |  |
| **31** |  |  |  |

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

**Algorithm 1: (8 marks)**

You are asked to write a function called **Repeated\_Display** **(** **int** **V** **)**

* Your function will receive an integer that represents some Target Value **(V)**.
  + Display its path.
  + Display its path except the 1st node.
  + Display its path except the 2nd node.
  + And so on.

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

void Repeated\_Display(int v)

{

CTNode\* pTrav = pRoot;

int ct = 0;

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

{

pTrav = pRoot;

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

{

if (v > pTrav->info)

{

pTrav = pTrav->pRight;

}

else

{

pTrav = pTrav->pLeft;

}

}

while (pTrav != NULL && pTrav->info != v)

{

if (i == 0)

{

ct++;

}

cout << pTrav->info;

if (v > pTrav->info)

{

pTrav = pTrav->pRight;

}

else

{

pTrav = pTrav->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 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 search about 2 contiguous repeated nodes,

Finally the function will:

* Move the nodes **before** the 2 founded cells to be **after**.
* Move the nodes **after** the 2 founded cells to be **before**.

**Note:** you have no permission to use another List, or any other data structures.

**e.g.**

Head

L

Head

**Output will be:**

L ‘

void Swap(CNode\* pT, CNode\* pB, CNode\* pBB, CNode\*& pCat1, CNode\*& pCat2, int& check, CList&L)

{

if (pT == NULL && check == 1) //check might be unnecessary

{

return;

}

//if no tail present

//if (pT == NULL)

//{

// pMyTail=pB;

// return;

//}

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

{

pCat1 = pBB;

pCat2 = pT->pNext;

pT->pNext = L.pHead;

L.pHead = pCat2;

L.pTail->pNext = pCat1->pNext;

//pMyTail->pNext=pCat1->pNext;

pCat1->pNext = NULL;

L. pTail = pCat1;

//pMyTail=pCat1;

check = 1;

}

Swap(pT->pNext, pT, pB, pCat1, pCat2, check, L);

}

**Problem 2: (19 marks)**

Write a main function to do the following:

**L**

1. Read the required data structure **(L)** according to the following protocol:

Head

* + Ask the user to determine the number of rows he needs to enter (**nR**). 🡪 in the example nR = 5
  + Ask the user about the size of the row (**nC**) assume all rows are in the same size. 🡪 in example nC = 6
  + **In the last row**: you have to read some binary search tree from the user for each node, and points to it by down pointer of the node.

60

30

50

82

85

95

80

90

20

70

130

100

90

95

120

81

84

82

83

80

85

230

240

250

310

300

80

200

1200

950

940

1300

330

320

88

**Notes:**

* Each node in the Listnode , will accept either (1 or 0) from the user.

**L**

Head

* The 1st node in each row will points to the 1st node in the following row.
* You are responsible to declare the data structure for the ListNode.

1. Ask the user to select a row (iR), and then display the trees that required (each 1 means display the tree under this node) by this row.

e.g. ( iR = 3 🡪 [80, 82 , 85 , 90 , 95] 🡪 [30 , 50 , 60] )

82

85

95

80

90

60

30

50

330

230

320

240

250

310

300

80

200

95

90

100

80

85

81

84

82

83

130

120

20

70

940

1300

1200

950



88

1. Ask the user to select a row (iR), Copy the right branches of the required trees of (iR), **but** in **reverse** manner to new list (**NL**).

e.g. ( iR = 4)

82

85

95

80

90

60

30

50

330

230

320

240

250

310

300

80

200

88

95

90

100

80

85

81

84

82

83

130

120

20

70

940

1300

1200

950

**NL**

pHead

class CNode

{

public:

int info;

CNode\* pNext;

CTNode\* pDownT;

CNode\* pDownL;

};

void Display(CTNode\* pT)

{

if (pT == NULL)

return;

Display(pT->pLeft);

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

Display(pT->pRight);

}

void CopyReverse(CTNode\* pT, CList& NL)

{

while (pT != NULL)

{

CNode\* pnL = new CNode;

pnL->info = pT->info;

pnL->pNext = NULL;

if (NL.pHead == NULL)

{

NL.pHead = pnL;

NL.pTail = pnL;

}

else

{

pnL->pNext = NL.pHead;

NL.pHead = pnL;

}

pT = pT->pRight;

}

}

void main()

{

CList L;

CBST T;

CNode\* pMyTail = NULL, \* pCurrent = NULL, \* pFirst = NULL;

int nr, nc,info;

cin >> nr;

cin >> nc;

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

{

pCurrent = NULL;

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

{

CNode\* pnn = new CNode;

cin >> info;

if (info == 1 || info == 0)

{

pnn->info = info;

}

pnn->pNext = NULL;

pnn->pDownT = NULL;

if (j == 0)

{

if (i > 0)

{

pFirst->pDownL = pnn;

pFirst = NULL;

}

pFirst = pnn;

pCurrent = pFirst;

}

else

{

pCurrent->pNext = pnn;

pCurrent = pnn;

}

}

}

CNode\* pTrav = pFirst;

int m;

while (pTrav != NULL)

{

cin >> m;

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

{

CTNode\* pnT = new CTNode;

cin >> pnT->info;

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

T.Insert(pnT);

}

pTrav->pDownT = T.pRoot;

T.pRoot = NULL;

pTrav = pTrav->pNext;

}

int iR;

cin >> iR;

CNode\* pT = pFirst;

pTrav = L.pHead;

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

{

pTrav = pTrav->pDownL;

}

while (pTrav != NULL)

{

if (pTrav->info == 1)

{

Display(pT->pDownT);

cout << endl;

}

pT = pT->pNext;

pTrav = pTrav->pNext;

}

CList NL;

cin >> iR;

pT = pFirst;

pTrav = L.pHead;

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

{

pTrav = pTrav->pDownL;

}

while (pTrav != NULL)

{

if (pTrav->info == 1)

{

CopyReverse(pT->pDownT, NL);

}

pT = pT->pNext;

pTrav = pTrav->pNext;

}

}