**Section (I): Tracing Questions (Total: 10 marks)**

**(Q1) (5 mark)**

* Draw the binary expression tree for the following prefix expression.
* Also, write the corresponding postfix, and infix expression.
* **+ - \* A + - B C K M – D E**

**(Q2) (5 mark)**

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

**12 , 18 , 29 , 20 , 30 , 11 , 38**

1. Store the values in a hash table with size 10 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 % 3).
4. Fill in the following table, showing the number of comparisons needed to find each value:-

|  |  |  |  |
| --- | --- | --- | --- |
|  | **(i)** | **(ii)** | **(iii)** |
| **11** |  |  |  |
| **30** |  |  |  |
| **38** |  |  |  |

**Section (II): Algorithm Questions (Total: 14 marks)**

**Algorithm 1: (14 marks)**

You are asked to write a function called **Display\_RepeatedPath ( int V )**

* Your function will find the value **(V).**
  + You will display the path till reach **(V).**
  + Then starting from **(V)**, walk the same path again and display its nodes.
* e.g. V = 400

**output : 500, 200, 450, 210, 215, 400**

**400, 300, 380, 310, 320, 350**

**350**

**320**

**310**

**300**

**380**

**400**

460

950

215

73

75

85

84

72

86

70

900

850

210

450

90

200

910

800

500

380

300

**400**

**500**

850

90

910

800

73

75

85

84

72

86

70

900

**210**

**450**

**200**

**215**

460

950

350

320

310

void Display\_RepeatedPath(int v)

{

CTNode\* pTrav = pRoot;

CTNode\* pTrav2 = NULL;

while (pTrav != NULL)

{

if (pTrav->info == v)

{

break;

}

if (v < pTrav->info)

{

pTrav = pTrav->pLeft;

cout << pTrav << endl;

}

else

{

pTrav = pTrav->pRight;

cout << pTrav << endl;

}

}

pTrav2 = pTrav; //pTrav2 on node that carries v

pTrav = pRoot;

while (pTrav->info != v)

{

if (v < pTrav->info)

{

pTrav = pTrav->pLeft;

pTrav2 = pTrav2->pLeft;

cout << pTrav2 << endl;

}

else

{

pTrav = pTrav->pRight;

pTrav2 = pTrav2->pRight;

cout << pTrav2 << endl;

}

}

}

**Section (III): Problem Solving (Total: 36 marks)**

**Problem 1: (10 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 copy the last **(N)** nodes and paste them at the first of the list.

**e.g. 🡪 N = 3**

L

Head

**The list after calling of your function will be:**

L

Head

void CopyPaste(CNode\* pT, int& N,CList&L)

{

if (pT == NULL)

{

return;

}

CopyPaste(pT->pNext, N, L);

if (N > 0)

{

CNode\* pnn = new CNode;

pnn->info = pT->info;

pnn->pNext = NULL;

pnn->pNext = L.pHead;

L.pHead = pnn;

N--;

}

}

**Problem 2: (26 marks)**

Write a main function to do the following:

1. Read a Binary Search Tree.
2. Find the Smallest Node (call it **pSmallest**)

e.g. Smallest 🡪 50

**pSmallest**

430

560

450

550

500

755

400

Root

10250

10600

10400

10200

920

15000

1200

950

970

1180

980

1100

1000

1700

10000

720

680

700

750

**50**

900

850

793

794

780

790

910

800

795

1. Copy the subtree of the pSmallest to a linked list of trees (LL), **but** during the copy add (10) to each node in the subtree in both the original one and the one that copied.

**pHead**

**LL**

690

440

570

460

560

510

410

730

710

765

760

430

560

450

550

500

755

400

720

680

700

750

1. Repeate steps (2&3) till there is no subtree for the pSmallest**.**

440

570

460

560

510

765

410

920

10250

10600

10400

15000

730

690

710

760

**50**

10000

970

1180

980

1100

1000

10200

1700

1200

950

900

850

793

794

780

790

910

800

795

Root

**pSmallest**

570

460

560

510

410

730

710

765

760

440

430

560

450

550

500

755

400

720

680

700

750

**pHead**

**LL**

580

450

470

570

520

440

570

460

560

510

690

450

580

470

570

520

765

410

920

10250

10600

10400

15000

730

690

710

760

**50**

10000

970

1180

980

1100

1000

10200

1700

1200

950

900

850

793

794

780

790

910

800

795

Root

1. Ask the user to select subtree from (LL) and display each node in this subtree & display each original value**.**

**e.g.**

**selected subtree 2nd one.**

**Then your display will be:**

**450 was 430**

**470 was 450**

**520 was 500**

**570 was 550**

**580 was 560**

void VisitAll(CTNode\* pT,CBST&T)

{

if (pT == NULL)

{

return;

}

VisitAll(pT->pLeft, T);

pT->info += 10;

CTNode\* pnn = new CTNode;

pnn->info = pT->info;

pnn->pLeft = pnn->pRight = NULL;

T.Insert(pnn);

VisitAll(pT->pRight, T);

}

void Disp(CTNode\*pT,int index)

{

if (pT == NULL)

{

return;

}

Disp(pT->pLeft, index);

cout << pT->info << "was" << pT->info - (index \* 10);

Disp(pT->pRight, index);

}

void main()

{

CBST CTree;

CList L;

int N;

cout << "Enter N\n";

cin >> N;

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

{

CTNode\* pnn = new CTNode;

cin >> pnn->info;

pnn->pLeft = pnn->pRight = NULL;

CTree.Insert(pnn);

}

CTNode\* pMin = NULL;

while (pMin != NULL)

{

CTNode\* pTrav = CTree.pRoot;

while (pTrav->pLeft != NULL)

{

pTrav = pTrav->pLeft;

}

pMin = pTrav; //found min

pTrav = pTrav->pRight; //to get subtree

CBST T2;

VisitAll(pTrav, T2);

CNode\* pList = new CNode;

pList->pNext = NULL;

pList->pExtra = T2.pRoot;

T2.pRoot = NULL;

L.Attach(pList);

pMin = pMin->pRight; //to start looking for new min in the subtree

}

int index;

cin >> index;

CNode\* pTrav = L.pHead;

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

{

pTrav = pTrav->pNext;

}

Disp(pTrav->pExtra, index);

}