1. Download the following zipped file. It contains code to implement a binary search tree using a static array.

<https://markbowman.org/231/Lab19.zip>

Be sure to put the data files in the same directory as the project code files. You may not change any of the files or functions, except as instructed.

2. Implement the addition operator, as declared in Tree.h. It should do the following:

* *Declare a local tree temp*
* *Use a loop to copy valid elements from the current tree’s map into temp’s map*
* *Use another loop to put valid elements from the argument tree’s map into temp, using the insert() function.*
* *Return temp*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* addition

\* Addition Operator by Twymun Safford

\* Date Updated: 11-02-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tree tree::operator+(const tree& t)

{

int i;

//add two trees

//declare local tree temp

tree temp;

//loop - copies valid elements from

//current tree map into temp map

for (i = 0; i < TREE\_MAX; i++)

{

if (map[i] != "")

{

temp.insert(map[i]);

}

}

//put valid elements into temp using insert

for (i = 0; i < TREE\_MAX; i++)

{

if (t.map[i] != "")

temp.insert(t.map[i]);

}

//return temp

return temp;

}

3. Test your code with the two data files. Generate an output using the LMR option. Does it show all the items in the correct order?

Text

Description automatically generated

4. Test your code again with the data files switched. Verify that the two LMR ouputs are identical.

Text

Description automatically generated

5. Run the program with Fruit.txt, then School.txt. Generate an output using the Dump option. Save this output. What is the depth of the resulting tree?

Text

Description automatically generated

**The depth is 5 in this case.**

6. Run the program with School.txt, then Fruit.txt. Generate an output using the Dump option. Save this ouput. What is the depth of the resulting tree?

Text

Description automatically generated

A picture containing text

Description automatically generated

**The depth in this case is 8.**

7. Draw a diagram of the resulting trees from steps 6 and 7 to show why the depths are different.

To assist with the process of drawing the trees, I used a resource created by the Computer Science department for the University of San Francisco. I found their Binary Search tree application here: <https://www.cs.usfca.edu/~galles/visualization/BST.html>

It does, however, have a character limit so I had to make do with four characters per node.

The Fruit.txt file as a binary tree is:

**Fruit.txt:**

Diagram, schematic

Description automatically generated

The **School.txt** file as a binary tree is:

Diagram

Description automatically generated

If we select to start with the entries from Fruit.txt to form our initial binary tree and then sort the elements from School.txt into that tree, we get (where labeled with the appropriate node number):

Diagram, schematic

Description automatically generated

If, however, we select to start with the entries from School.txt to form our initial binary tree and then sort the elements from Fruit.txt into that tree, we get (where labeled with the appropriate node number and depth):

Diagram, schematic

Description automatically generated

This should make sense based upon how the dump function is essentially (in columnar form) ‘dumping’ the number associated with each of the keys within the binary search tree. This also demonstrates based on the difference between the two cases that adding elements to a binary search tree is not a commutative process. The resultant binary search tree from the addition of elements into a primary binary search tree is dependent on order.

Code - Tree Public.cpp:

// Tree Functions

#include <iostream>

#include <string>

using namespace std;

#include "Tree.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* tree()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tree::tree()

{

int i;

for (i = 0; i < TREE\_MAX; i++) map[i] = "";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* insert()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool tree::insert(string s)

{

int pos;

// Start at root

pos = 0;

// Loop left/right

while (pos < TREE\_MAX && map[pos] != "")

{

if (s <= map[pos])

pos = left(pos);

else

pos = right(pos);

};

// Fail if past bottom

if (pos >= TREE\_MAX) return false;

// Set node

map[pos] = s;

return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* show()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void tree::show(string arg)

{

if (arg[0] == 'D') dump();

if (arg == "LMR") lmr(0);

if (arg == "RML") rml(0);

if (arg == "MLR") mlr(0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* addition

\* Addition Operator by Twymun Safford

\* Date Updated: 11-02-2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

tree tree::operator+(const tree& t)

{

int i;

//add two trees

//declare local tree temp

tree temp;

//loop - copies valid elements from

//current tree map into temp map

for (i = 0; i < TREE\_MAX; i++)

{

if (map[i] != "")

{

temp.insert(map[i]);

}

}

//put valid elements into temp using insert

for (i = 0; i < TREE\_MAX; i++)

{

if (t.map[i] != "")

temp.insert(t.map[i]);

}

//return temp

return temp;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* depth()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tree::depth()

{

return depth(0) - 1;

}

Extra Credit (5 points)

Change the memory storage in your tree class to dynamic. You will need to change the constructor, and add a destructor. You will also need to create the following operators:

tree(const tree &); // Copy constructor

void operator =(const tree &); // Assignment operator