1. Download the following zipped file. It contains the code and data files shown. Extract and compile the code in a project.

<https://markbowman.org/231/Lab18.zip>

* Lab18.sln, Lab18.vcxproj, Lab18.vcxproj.filters
* Main.cpp
* Tree.h, Tree Public.cpp, Tree Private.cpp
* Data.txt

1. Be sure to put the data file in the same directory as the project code files. Run the program with Data.txt. What is the depth of the tree? What is the index of the last filled element in the array?

**The depth of the tree is six and the index of the very last filled element in the array is 50.**

Text

Description automatically generated with low confidence

1. Comment out the section in main() that displays the depth and dump of the tree. Remove the /\* \*/ comments for STEP 3, and run the program again. You should see just the root of the tree.

Text

Description automatically generated

1. Modify the tree::lmr() function so that the tree values are displayed in ascending order. The tree::depth() and tree::depth\_recurse() functions may give you examples for your code.

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

\* lmr()

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

void tree::lmr(int pos)

{

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

\*LMR Implementation

\* Twymun Safford

\* Date Updated: 10-28-2021

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

//check if bottom or empty node - return if bottom or empty

if (pos >= TREE\_MAX || map[pos] == "")

{

return;

}

//call recursively for left child branch and lower as it works its way up from bottom node to root

lmr(left(pos));

//call resursively for right child branch and down

lmr(right(pos));

//print (in order) results of LMR binary search

cout << map[pos] << endl;

}

1. Run the program and save the output.

Text

Description automatically generated

1. Replace the /\* \*/ comments in main() for STEP 3, and remove the /\* \*/ comments for STEP 6. Update the tree::show() function, and implement the tree::rml() and tree::mlr() functions.

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

\* rml()

\* RML Implementation by Twymun Safford

\* Date Updated: 10-28-2021

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

void tree::rml(int pos)

{

//check if outside of tree depth or empty and return if so

if (pos >= TREE\_MAX || map[pos] == "")

{

return;

}

//call recursively for right child branch and lower as it works its way up from bottom node to root

rml(right(pos));

//print (in order) results of RML binary search

cout << map[pos] << endl;

//call recursively for left child branch and lower as it works its way up from bottom node to root

rml(left(pos));

}

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

\* mlr()

\* MLR Implementation by Twymun Safford

\* Date Updated: 10-28-2021

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

void tree::mlr(int pos)

{

//outputs the current node first, then the two children (pre-order)

//check if outside of tree depth or empty and return if so

if (pos >= TREE\_MAX || map[pos] == "")

{

return;

}

//print (in order) results of LMR binary search

cout << map[pos] << endl;

//call recursively for left child branch and lower as it works its way up from bottom node to root

mlr(left(pos));

//call recursively for right child branch and lower as it works its way up from bottom node to root

mlr(right(pos));

}

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

\* show()

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

void tree::show(string arg)

{

if (arg == "LMR")

{

lmr(0);

}

if (arg == "RML")

{

rml(0);

}

if (arg == "MLR")

{

mlr(0);

}

}

1. Run your program with each of the three display orders. Save the output for RML and MLR.

**LMR:**

Text

Description automatically generated

**RML:**

**Text

Description automatically generated**

**MLR:**

**Text

Description automatically generated**

Remove the /\* \*/ comments in main for STEP 8. Implement the tree::find() /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* find()

\* Find Function by Twymun Safford

\* Date Updated: 10-28-2021

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

int tree::find(string s)

{

//find a string in the tree by using binary search - start at root node,

//traverse each child, and try to find the string/keyword

int pos = 0; //start at root node

// Loop between both left and right branches as long as pos is less than tree max

// or until after checking every node the string is not found

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

{

//if the string is on a branch where the string is less than the current position. go left

if (s < map[pos])

{

pos = left(pos);

}

//in the other case that the string is greater than current position check right child or leaf

else if (s > map[pos])

{

pos = right(pos);

}

//not found

else

{

return pos;

}

};

// Out of the bounds of tree or not found

if ((pos >= TREE\_MAX) || (map[pos]==""))

{

return TREE\_ERR;

}

//found

return pos;

}

1. Run the program a few more times to demonstrate that the find function works correctly (found positions, and not found). Save your output.

**Sample Cases – Known Names:**

Text

Description automatically generated

Text

Description automatically generated

**Searching for Not Found:**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Final Code:**

**Tree.h:**

// Tree Declarations

#define TREE\_MAX 1024

#define TREE\_ERR -1

class tree

{ public:

tree(); // Constructor

bool insert(string); // Insert string into tree

void dump(); // Dump contents of array

int depth(); // Total depth

void show(string); // Display in order

int find(string); //find a string in the tree

private:

string map[TREE\_MAX]; // Data array

int left(int pos); // Left child

int right(int pos); // Right child

int depth\_recurse(int pos); // Recursive depth function

void lmr(int pos); // Recursive LMR display

void rml(int pos); // Recursive LMR display

void mlr(int pos); // Recursive LMR display

};

**TreePublic.cpp:**

// Tree Functions

#include <iostream>

#include <iomanip>

#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;

}

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

\* dump()

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

void tree::dump()

{

int i, n;

// Get max position

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

if (map[i] > "")

n = i;

// Display up to max

cout << std::left;

for (i = 0; i <= n; i++)

{

cout << setw(4) << i << map[i] << endl;

}

}

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

\* show()

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

void tree::show(string arg)

{

if (arg == "LMR")

{

lmr(0);

}

if (arg == "RML")

{

rml(0);

}

if (arg == "MLR")

{

mlr(0);

}

}

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

\* depth()

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

int tree::depth()

{

return depth\_recurse(0);

}

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

\* find()

\* Find Function by Twymun Safford

\* Date Updated: 10-28-2021

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

int tree::find(string s)

{

//find a string in the tree by using binary search - start at root node,

//traverse each child, and try to find the string/keyword

int pos = 0; //start at root node

// Loop between both left and right branches as long as pos is less than tree max

// or until after checking every node the string is not found

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

{

//if the string is on a branch where the string is less than the current position. go left

if (s < map[pos])

{

pos = left(pos);

}

//in the other case that the string is greater than current position check right child or leaf

else if (s > map[pos])

{

pos = right(pos);

}

//not found

else

{

return pos;

}

};

// Out of the bounds of tree or not found

if ((pos >= TREE\_MAX) || (map[pos]==""))

{

return TREE\_ERR;

}

//found

return pos;

}

**TreePrivate.cpp:**

// Tree Functions

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

#include "Tree.h"

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

\* left()

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

int tree::left(int pos)

{

return pos \* 2 + 1;

}

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

\* right()

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

int tree::right(int pos)

{

return (pos + 1) \* 2;

}

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

\* depth()

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

int tree::depth\_recurse(int pos)

{

int l, r;

// Return if bottom or empty node

if (pos >= TREE\_MAX)

{

return 0;

}

if (map[pos] == "")

{

return 0;

}

// Left,Right

l = depth\_recurse(left(pos));

r = depth\_recurse(right(pos));

// Max of both sides + middle

return max(l, r) + 1;

}

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

\* lmr()

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

void tree::lmr(int pos)

{

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

\*LMR Implementation

\* Twymun Safford

\* Date Updated: 10-28-2021

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

//check if outside of tree depth or empty and return if so

if (pos >= TREE\_MAX || map[pos] == "")

{

return;

}

//call recursively for left child branch and lower as it works its way up from bottom node to root

lmr(left(pos));

//print (in order) results of LMR binary search

cout << map[pos] << endl;

//call resursively for right child branch and down

lmr(right(pos));

}

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

\* rml()

\* RML Implementation by Twymun Safford

\* Date Updated: 10-28-2021

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

void tree::rml(int pos)

{

//check if outside of tree depth or empty and return if so

if (pos >= TREE\_MAX || map[pos] == "")

{

return;

}

//call recursively for right child branch and lower as it works its way up from bottom node to root

rml(right(pos));

//print (in order) results of RML binary search

cout << map[pos] << endl;

//call recursively for left child branch and lower as it works its way up from bottom node to root

rml(left(pos));

}

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

\* mlr()

\* MLR Implementation by Twymun Safford

\* Date Updated: 10-28-2021

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

void tree::mlr(int pos)

{

//outputs the current node first, then the two children (pre-order)

//check if outside of tree depth or empty and return if so

if (pos >= TREE\_MAX || map[pos] == "")

{

return;

}

//print (in order) results of LMR binary search

cout << map[pos] << endl;

//call recursively for left child branch and lower as it works its way up from bottom node to root

mlr(left(pos));

//call recursively for right child branch and lower as it works its way up from bottom node to root

mlr(right(pos));

}