This is a two part project. In the first part you will construct a binary search tree and node class, with some basic functions. In the second part, you will expand the functionality of the binary search tree. Test your program with the code files below:

<https://markbowman.org/231/Program05.zip>

You may not change the Main.cpp file. You will need to create the Tree.h, Tree.cpp, Node.h, and Node.cpp files.

Tree Class

The binary search tree class should have the following data elements:

* Root Pointer to the root node

The binary search tree class should have the following functions:

* Constructor Initialize the empty tree
* Destructor Delete all the nodes in the tree
* Insert Insert a new value into the tree
* Show Display all the values
* Find Return true/false if the string is found in the tree

Node Class

The node class should have the following data elements:

* Value Stored value
* Left, Right Pointers to children nodes

The node class should have the following functions:

* Constructor Create an empty node
* Destructor Delete all children of this node
* Put Output the value
* LMR Output the current node and all children values in LMR order

The value and children pointers should be private to the node class. It may help to declare the tree class a friend to the node class.

What to Hand In

Hand in a copy of each code file (don’t include Main.cpp), and runs with both input files to demonstrate that your program and functions work. Sample runs are shown on the next page.

Sample Runs

|  |  |  |
| --- | --- | --- |
| Enter file name: ***People.txt***  LMR Order  --------------  Alisa  Ann  Carlos  David  Frank  John  Kathy  Lailee  Lisa  Mark  Michael  Richard  Rochelle  Steve  Susan  Enter value to find: ***Frank***  Find: Frank found! |  | Enter file name: ***People.txt***  LMR Order  --------------  Alisa  Ann  Carlos  David  Frank  John  Kathy  Lailee  Lisa  Mark  Michael  Richard  Rochelle  Steve  Susan  Enter value to find: ***Carlos***  Find: Carlos found! |
|  |  |  |
| Enter file name: ***Numbers.txt***  LMR Order  --------------  Eight  Five  Four  Nine  One  Seven  Six  Ten  Three  Two  Enter value to find: ***Two***  Find: Two found! |  | Enter file name: ***Numbers.txt***  LMR Order  --------------  Eight  Five  Four  Nine  One  Seven  Six  Ten  Three  Two  Enter value to find: ***Eleventeen***  Find: Eleventeen not found |

Code:

**Tree.h:**

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

\* Tree.h

\* Written by Twymun K. Safford

\* Last Updated: 11/8/2021

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

#ifndef TREE\_H

#define TREE\_H

//#include <string>

//#include <fstream>

#include "Node.h"

class tree

{

public:

tree(); // public tree constructor

~tree(); // public ~tree deconstructor

void insert(string value); // public insert

void show(string order, ostream& out); // public show

bool find(string value); // public find

private:

node\* root; // private node\* root

bool finding(string value, node\* root); // private bool finding

node\* inserting(string value, node\* root); // private node\* inserting

};

#endif

**Tree.cpp:**

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

\* Tree.cpp

\* Written by Twymun K. Safford

\* Last Updated: 11/8/2021

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

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#include "Tree.h"

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

\* Constructor

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

tree::tree()

{

root = NULL;

}

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

\* Destructor

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

tree::~tree()

{

if (root != NULL)

{

delete root;

}

}

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

\* insert

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

void tree::insert(string value)

{

// update the value of root post insertion

root = inserting(value, root);

}

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

\* node\* inserting

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

node\* tree::inserting(string value, node\* root)

//helper function that helps with inserting values into tree

{

node\* newNode;

// if root is null insert the value here

if (root == NULL)

{

newNode = new node(value);

// Return newNode

return newNode;

}

// if input value is less than root's value, insert in left subtree

else if (root->value > value)

{

root->left = inserting(value, root->left);

}

// if input value is greater than root's value, insert in right subtree

else

{

root->right = inserting(value, root->right);

}

// return root

return root;

}

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

\* show()

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

void tree::show(string order, ostream& out)

{

// if order is LMR

if (order == "LMR")

{

// display the node in LMR order

if (root != NULL)

{

root->lmr(out);

}

}

}

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

\* finding(value, node\* root)

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

bool tree::finding(string value, node\* root)

//helper function that assesses if a value exists -> pass to find

{

// if root is null, the value is not found, return false

if (root == NULL)

{

return false;

}

// if the root's value equals input value, return true

if (root->value == value)

{

return true;

}

// if input value is less than root's value, search left subtree

else if (value < root->value)

{

return finding(value, root->left);

}

// if the input value is more than root's value, search right subtree

else

{

return finding(value, root->right);

}

}

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

\* find(string value)

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

bool tree::find(string value)

{

// return finding value, root

return finding(value, root);

}

**Node.h:**

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

\* Node.h

\* Written by Twymun K. Safford

\* Last Updated: 11/8/2021

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

#ifndef NODE\_H

#define NODE\_H

//#include <string>

//#include <fstream>

//using namespace std;

class tree;

class node

{

friend tree; // declare tree as a friend class to node class

public:

node(string val); // public node(string val) constructor

~node(); // public ~node() deconstructor

void put(ostream& out); // void put(ostream& out)

void lmr(ostream& out); // void lmr(ostream& out)

private:

string value; // private string value

node\* left; // private node\* left

node\* right; // private node\* right

};

typedef node\* node\_ptr;

#endif

**Node.cpp:**

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

\* Node.cpp

\* Written by Twymun K. Safford

\* Last Updated: 11/8/2021

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

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#include "Node.h"

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

\* constructor

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

node::node(string val)

{

value = val;

left = NULL;

right = NULL;

}

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

\* destructor

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

node::~node()

{

// if left value present, delete left value

if (left != NULL)

{

delete left;

}

// if left value present, delete left value

if (right != NULL)

{

delete right;

}

}

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

\* put()

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

void node::put(ostream& out)

{

// output the value to given ostream

out << value << endl;

}

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

\* lmr()

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

void node::lmr(ostream& out)

{

if (left != NULL)

{

left->lmr(out);

}

put(out);

if (right != NULL)

{

right->lmr(out);

}

}

**Sample Runs:**

**People.txt:**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Numbers.txt:**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**