/// Andrew Souza

/// Comp 210 -- Spring 2024

/// Trees Assignment

#include <iostream>

#include <string>

#include <vector>

using namespace std;

// 16.11.5

/\* Reimplement the BinarySearchTree class to use a vector<string> elements

\* instead of Node objects to store the nodes. Store the root element in

\* elements[0] and the children of elements[i] in elements[2 \* i] and

\* elements[2 \* i + 1].

\* If one of these elements is the empty string or the index is not valid,

\* then the corresponding child does not exist. \*/

class BST\_Vector {

private:

vector<string> Tree;

unsigned int index;

public:

void insert(string node);

string search(string node);

BST\_Vector() {

Tree.resize(8);

index = 1; // We will use 1 for our index rather than 0

};

};

void BST\_Vector::insert(string node) {

index = 1;

while (!Tree[index].empty()) {

if (node < Tree[index]) {

index = index \* 2;

} else if (node > Tree[index]) {

index = index \* 2 + 1;

}

if (index >= Tree.size()) {

Tree.resize(Tree.size() \* 2); // Resize function when the index goes out of bounds

}

}

Tree[index] = node;

}

string BST\_Vector::search(string node) {

index = 1;

string error = "String not found.";

while (index < Tree.size()) {

if (Tree[index] == node) {

return "Found " + node + " at index: " + to\_string(index);

} else if (node < Tree[index]) {

index = index \* 2;

} else if (node > Tree[index]) {

index = index \* 2 + 1;

}

}

return error;

}

int main() {

BST\_Vector myTree;

myTree.insert("G");

myTree.insert("D");

myTree.insert("Z");

myTree.insert("E");

myTree.insert("Y");

myTree.insert("A");

myTree.insert("M");

myTree.insert("B");

myTree.insert("I");

cout << myTree.search("A") << "\n";

cout << myTree.search("B") << "\n";

cout << myTree.search("C") << "\n";

cout << myTree.search("D") << "\n";

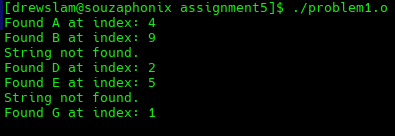
cout << myTree.search("E") << "\n";

cout << myTree.search("F") << "\n";

cout << myTree.search("G") << "\n";

cout << endl;

return 0;

}