**CS 4280, Section 001, Academic Integrity Statement**

For this project, AI resources can be used for debugging purposes, as long as they are fully disclosed as described herein. ALL OTHER USES OF AI ARE STRICTLY PROHIBITED. Additional non-AI internet resources can be utilized, as long as they are fully disclosed. Furthermore, code written by UMSL students in ***previous*** semesters can be looked at, but never copied.

**IMPORTANT**: Clearly indicate all outside resources utilized and sign below. Failure to cite the use of outside resources will be reported for appropriate disciplinary actions. Note that discussions with other students are encouraged; looking at each other’s code and/or copying – with or without modifications – are unacceptable and will be reported.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I hereby certify that all outside resources utilized, other than suggested textbooks and class materials, are clearly noted in my source code and in the following. The start and finish lines of the affected code are shown using ‘start ORx’ and ‘end ORx’, respectively, where ‘x’ is a unique number. Each value of ‘x’ has a corresponding explanation in this Academic Integrity Statement.

All internet resources include the web address and the date accessed. For each usage of AI, I also include the prompt and code submitted and the output returned.

All other materials I provide for this project submission are my own original work. I hereby certify that I am responsible for each and every line of code that I submit in my source files and I thoroughly understand how the code works to produce the output. I understand that I may be required to meet with the instructor to answer detailed questions about my submitted code.

OR1:

<https://www.w3schools.com/cpp/cpp_files.asp>

* Looking at file handling
* *// variables*
* string file = argv[1];
* string line;
* string chr;
* *// checking if there is contentents in file*
* if (isFileEmpty(file))
* {
* cout << "Error: Missing data" << endl;
* return 1;
* }
* *// checking file is readable*
* *// Check that input data are all character strings with letters, numbers,*
* *// And/or any of the following special characters:*
* *//Assume the strings are separated with any number of standard*
* *//white-space separators (space, tab, newline).*
* ifstream myFile(file);
* if (myFile.is\_open())
* {
* while (getline(myFile, line))
* {
* *// accessing each string that is seperated by white-space separators*
* istringstream strDiv(line);
* while (strDiv >> chr)
* {
* if (stringCharacterCheck(chr))
* {
* cout << chr << endl;
* }
* cout << "Error: Invalid character " << chr << endl;
* exit(1);
* }
* }
* myFile.close();
* }
* else
* {
* cout << file << " couldn't open" << endl;
* exit(1);
* }

OR2:

https://solvepro.co/check-if-a-file-is-empty-in-c/

bool ibool isFileEmpty(const std::string& *filename*) {

std::ifstream file(filename);

return *file*.peek() == std::ifstream::traits\_type::eof();

}

OR3:

<https://www.geeksforgeeks.org/processing-strings-using-stdistringstream/>

* Use of istringstream for reading individual string that have whitespaces and so on
* istringstream strDiv(line);

OR4:

* <https://cplusplus.com/reference/cctype/isalnum/>
* this should work to help check characters in string
* if (isalnum(c))

OR5

* https://cplusplus.com/reference/cctype/isspace
* Help with finding white-space characters
* if (isspace(c))

OR6:

Binary tree code from class at Lindenwood for guidance

#ifndef \_BINARY\_NODE\_HPP\_

#define \_BINARY\_NODE\_HPP\_

// File: AVLNode.hpp

// Author: Grant H

// Date: 3/2023

// Purpose: Definition of an AVL node for an AVL tree

#include <iostream>

**class** BinaryNode

{

**private**:

std::string \_val; // actual value stored in AVL Node

BinaryNode \*\_left; // pointer to node's left child (may be null)

BinaryNode \*\_right; // pointer to node's right child (may be null)

**int** \_height; // height of subtree rooted at this node.

**public**:

// basic constructor for a node. Has no children, so height is 1

BinaryNode(std::string val):\_val(val), \_left(**nullptr**), \_right(**nullptr**), \_height(1) {}

// accessors/modifiers for instance variables.

std::string value() **const** {**return** \_val;}

std::string & value() {**return** \_val;}

BinaryNode\* left() **const** {**return** \_left;}

BinaryNode\* &left() {**return** \_left;}

BinaryNode\* right() **const** {**return** \_right;}

BinaryNode\* &right() {**return** \_right;}

**int** height() **const** {**return** \_height;}

**int** &height() {**return** \_height;}

// is this a leaf node (with no children)?

**bool** isLeaf() **const** {**return** !\_left && !\_right;}

// print out node value and heights of children

std::ostream& print(std::ostream &os) **const**;

// insert value into subtree rooted at this node

BinaryNode\* insert(std::string newVal, BinaryNode \*intoSubTree);

**bool** find(std::string val, BinaryNode \*node);

// overload << to display this node (not entire subtree rooted at node)

**friend** std::ostream& **operator**<<(std::ostream &os, **const** BinaryNode &n);

};

#endif

// File: BnaryNode.cpp

// Author: Grant H

// Date: 3/2023

// Purpose: Implementation of an AVL node for a BinaryNode

#include "BinaryNode.hpp"

#include <ostream>

**using** **namespace** std;

**int** getHeight(BinaryNode \*n)

{

**if** (!n)

{

**return** 0;

}

**else**

{

**return** n->height();

}

}

BinaryNode\*

BinaryNode::insert(string val, BinaryNode \*intoSubTree)

{

**if** (intoSubTree == **nullptr**)

{

BinaryNode \*newRoot= **new** BinaryNode(val);

// newRoot->\_height= intoSubTree->\_height + 1;

**return** newRoot;

}

**if** (val < intoSubTree->\_val)

{

intoSubTree->\_left= insert(val, intoSubTree->\_left);

intoSubTree->\_left->\_height= (intoSubTree->\_height + 1);

}

**if** (val > intoSubTree->\_val)

{

intoSubTree->\_right= insert(val, intoSubTree->\_right);

intoSubTree->\_right->\_height= (intoSubTree->\_height + 1);

}

**return** intoSubTree;

}

**bool**

BinaryNode::find(string val, BinaryNode \*node)

{

**bool** found= **true**;

**if** (node->\_left->\_val != val)

{

found= **false**;

}

**if** (node->\_right->\_val != val)

{

found= **false**;

}

**return** found;

}

ostream&

BinaryNode::print(ostream &os) **const**

{

os << \_val << " (" << getHeight(\_left) << "<->" << getHeight(\_right) << ")";

**return** os;

}

ostream& **operator**<<(ostream &os, **const** BinaryNode &n)

{

**return** n.print(os);

}

OR8:

* <https://cplusplus.com/doc/tutorial/files/>
* Writing to a file
* ostream writeFile("output.txt", ios::app);

OR9:

* https://stackoverflow.com/questions/17032970/clear-data-inside-text-file-in-c
* ofstream fileToClear("output.txt", ios::out | ios::trunc);
* fileToClear.close();

OR10:

* <https://cplusplus.com/reference/queue/queue/>
* Shows use of queues in c++
* *// variables*
* queue<node\_t\*> nodeQueue;
* if (node == nullptr) *// checking if there is a root*
* return;
* *// there is node(s)*
* nodeQueue.push(node);
* while (!nodeQueue.empty())
* {
* node\_t \*currentNode = nodeQueue.front();
* nodeQueue.pop(); *// removing current node from queue*
* cout << currentNode->\_charInString << ": ";
* for (int i = 0; i < currentNode->strings.size(); i++)
* {
* cout << currentNode->strings[i] << " ";
* }
* cout << endl;
* *// adding left and right nodes*
* if (currentNode->\_left)
* nodeQueue.push(currentNode->\_left);
* if (currentNode->\_right)
* nodeQueue.push(currentNode->\_right);
* }

OR11:

* Using stacks also for traversing
* <https://cplusplus.com/reference/stack/stack/>
* void Traversals::traversePreOrder(node\_t \*node)
* {
* *// variables*
* stack<node\_t\*> nodeStack;
* if (node == nullptr) *// checking if there is a root*
* return;
* *// there is node(s)*
* nodeStack.push(node);
* while (!nodeStack.empty())
* {
* node\_t \*currentNode = nodeStack.top();
* nodeStack.pop(); *// removing current node from stack*
* cout << currentNode->\_charInString << ": ";
* for (size\_t i = 0; i < currentNode->strings.size(); i++)
* {
* cout << currentNode->strings[i] << " ";
* }
* cout << endl;
* *// adding left and right nodes but right first in preorder*
* if (currentNode->\_right)
* nodeStack.push(currentNode->\_right);
* if (currentNode->\_left)
* nodeStack.push(currentNode->\_left);
* }
* }
* void Traversals::traversePostOrder(node\_t \*node)
* {
* *// variables*
* stack<node\_t\*> preOrderStack;
* stack<node\_t\*> postOrderStack;
* if (node == nullptr) *// checking if there is a root*
* return;
* *// there is node(s)*
* preOrderStack.push(node);
* *// getting nodes from tree*
* while (!preOrderStack.empty())
* {
* node\_t \*currentNode = preOrderStack.top();
* preOrderStack.pop(); *// removing current node from stack*
* postOrderStack.push(currentNode);
* *// access left/right nodes*
* if (currentNode->\_left)
* preOrderStack.push(currentNode->\_left);
* if (currentNode->\_right)
* preOrderStack.push(currentNode->\_right);
* }
* *// print the nodes in post order*
* while (!postOrderStack.empty())
* {
* node\_t \*currentPostOrderNode = postOrderStack.top(); *// getting first node*
* postOrderStack.pop();
* cout << currentPostOrderNode->\_charInString << ": ";
* for (size\_t i = 0; i < currentPostOrderNode->strings.size(); i++)
* {
* cout << currentPostOrderNode->strings[i] << " ";
* }
* cout << endl;
* }
* }

OR12:

* <https://en.cppreference.com/w/cpp/types/size_t>
* */\**
* *Author: Grant Hughes*
* *Created: Febuary 11rd, 2025*
* *Program that traverses the binary tree*
* *\*/*
* #include <iostream>
* #include "BinaryNode.hpp"
* #include "BuildTree.hpp"
* #include "Traversals.hpp"
* #include <stack>
* using namespace std;
* void Traversals::traverseLevelOrder(node\_t \*node)
* {
* *// variables*
* queue<node\_t\*> nodeQueue;
* if (node == nullptr) *// checking if there is a root*
* return;
* *// there is node(s)*
* nodeQueue.push(node);
* while (!nodeQueue.empty())
* {
* node\_t \*currentNode = nodeQueue.front();
* nodeQueue.pop(); *// removing current node from queue*
* cout << currentNode->\_charInString << ": ";
* for (size\_t i = 0; i < currentNode->strings.size(); i++)
* {
* cout << currentNode->strings[i] << " ";
* }
* cout << endl;
* *// adding left and right nodes but left first in level order*
* if (currentNode->\_left)
* nodeQueue.push(currentNode->\_left);
* if (currentNode->\_right)
* nodeQueue.push(currentNode->\_right);
* }
* }
* void Traversals::traversePreOrder(node\_t \*node)
* {
* *// variables*
* stack<node\_t\*> nodeStack;
* if (node == nullptr) *// checking if there is a root*
* return;
* *// there is node(s)*
* nodeStack.push(node);
* while (!nodeStack.empty())
* {
* node\_t \*currentNode = nodeStack.top();
* nodeStack.pop(); *// removing current node from stack*
* cout << currentNode->\_charInString << ": ";
* for (size\_t i = 0; i < currentNode->strings.size(); i++)
* {
* cout << currentNode->strings[i] << " ";
* }
* cout << endl;
* *// adding left and right nodes but right first in preorder*
* if (currentNode->\_right)
* nodeStack.push(currentNode->\_right);
* if (currentNode->\_left)
* nodeStack.push(currentNode->\_left);
* }
* }
* void Traversals::traversePostOrder(node\_t \*node)
* {
* *// variables*
* stack<node\_t\*> preOrderStack;
* stack<node\_t\*> postOrderStack;
* if (node == nullptr) *// checking if there is a root*
* return;
* *// there is node(s)*
* preOrderStack.push(node);
* *// getting nodes from tree*
* while (!preOrderStack.empty())
* {
* node\_t \*currentNode = preOrderStack.top();
* preOrderStack.pop(); *// removing current node from stack*
* postOrderStack.push(currentNode);
* *// access left/right nodes*
* if (currentNode->\_left)
* preOrderStack.push(currentNode->\_left);
* if (currentNode->\_right)
* preOrderStack.push(currentNode->\_right);
* }
* *// print the nodes in post order*
* while (!postOrderStack.empty())
* {
* node\_t \*currentPostOrderNode = postOrderStack.top(); *// getting first node*
* postOrderStack.pop();
* cout << currentPostOrderNode->\_charInString << ": ";
* for (size\_t i = 0; i < currentPostOrderNode->strings.size(); i++)
* {
* cout << currentPostOrderNode->strings[i] << " ";
* }
* cout << endl;
* }
* }

*Project #*: **\_\_P0\_\_**

Grant Hughes 1/31/2025

*Signature (typed name is fine) Date*