Samuel Boulton

169039695

Assignment binary tree

Storing Data in a Binary Tree

Table of Contents

[Introduction 2](#_Toc29740946)

[How the code works 2](#_Toc29740947)

[Memory 2](#_Toc29740948)

[Conclusion 3](#_Toc29740949)

# Introduction

This document will explain how the code I have created works and the way it uses memory. The document is in sections, you can find your way around the document by using the table of contents to assist you.

# How the code works

Overall the code runs data from a file and displays the sorted binary tree out to the console.

Firstly creating a node and defining its structure with the left and right paths. Then the new node is created with the data information that will later be passed on from reading the file. The new node contains the 4 data[pos]’s, 0 being the module number, 1 being the course name, 2 being the semester and 3 being the year. All this information will be separated line by line within the reading of the file but placed in this order into the node.

Reading the file is the important part as this separated each piece of information that must be taken from the data file. This is done by using strtok to pull the different parts of data separated by a comma (,). Then the module number is the important data to push it left or right of the root. Once the file has been read and data parted it needs to be inserted into the node. Inserting the node into the tree is quite simple as it compares the root node to the other data going into the tree and choosing to put the next node in left or right depends on the module number. Displaying the tree was using if statements and using printf. Travelling through the binary tree and displaying the tree in an order with the directions of where each piece of data is going. Finally int main() wraps everything together and calls the sections in order and displays the sorted tree into the console.

# Memory

Firstly calling a method adds a frame to the stack. The stack has a last in first out, structure. Meaning the main method in the code creates a frame and gets entered into the stack first and will be the last to leave the stack. The main method then sets the TNode \*root to NULL and stores it in the stack under the method of main. Then in the main, the method readingFile() is called which is put on top of the main method in the stack containing the root node inside the method. Then this method goes through all its statements and pushes the information into the stack for example: the integer declared named num will be added. Then the method display\_tree() is added onto the stack; this carries the variable (TNode\* nd) and compared the variable to the statements inside the method.

Once the variable is compared using if statements and finds that the root node is NULL because the data file has been fully read, the recursion process begins. So the display tree will be removed from the stack first because it was the last one to be put into the stack. This then means the program travels to the readingFile() method to remove that process from the stack. The same happens to the main method when the other methods have finished running the return statements. The main method calls the return statement and it is then from the stack. Thereby finishing the program and displaying the output on the console.

# Conclusion

In conclusion I have coded a project that takes the data from a file, sorts into a binary tree with the data and displays the data into the console. Additionally I have created a document that tells you how the code works and the memory it uses.