**Lab Objective:**

Practice using binary search trees to implement a data base program that acts as an address book. This program must be capable of accepting simple entries, storing the entries in a binary tree, displaying entries in order, deleting entries, and updating entries.

A screenshot of a computer

Description automatically generated

Figure 1 – The lab assignment

Please note that it was specifically noted in class that storing and accessing the tree values to file was regarded as extra credit. Note that this could have been done by turning the tree into a series of comma separated values and storing them to file.

How It Works:

I did not use the provided lab functions. I once again built my own. This binary search tree is built out of nodes that store strings as data. For convenience these nodes also store the memory address of the right, left, and previous nodes in the tree.

A screenshot of a computer

Description automatically generated

Figure 2 – The code for the node function:

These nodes are then inserted into the tree. The insert function compares the value of the new node to those of previous nodes, recursively comparing the value of the node to determine if it is smaller, or larger, and traversing the tree in the left or right direction respectively.

A computer screen shot of a program code

Description automatically generated

Figure 3 – The Insert Function – Part 1

A computer screen with text on it

Description automatically generated

Figure 4 – The Insert Function – Part 2

The search function returns the memory address of a specified data value. It does so by traversing the tree until it encounters null pointers. This search function is used in the delete and replace functions to determine the memory addresses of the specified targets for those functions.

A screenshot of a computer program

Description automatically generated

Figure 5 – The search function

Binary search trees have the issue that if there are two children in a node that you wish to delete, then that particular node has issues being deleted – as this would mean the user is attempting to cram three nodes into only two slots (the node being deleted, and the pointer to where the node that replaces the node being deleted goes). Rather than lead to wacky hijinks, my solution was to delete all children below the node being deleted, and insert them back into the tree. This is done with the deleteandinsert function, which deletes a node, and all nodes below it, then inserts them back into the tree. This would not be ideal for large trees as there is likely a faster way – however this way works too!  
A computer screen with text

Description automatically generated

Figure 6 – The delete and insert function.

The deleteandinsert function is paired with the remove function, which identifies the memory address of a target value using the search function, removes its parents references to it, then uses the delete and remove function to delete all of its children, then deletes itself.

A computer screen shot of a program code

Description automatically generated

Figure 7 – The remove function

The replace function also uses the remove function and insert function to remove an existing node, and add a new node into the tree. Alternatively, it would have been easy to program a way to alter the existing data of the node, then resort the tree. However, this way works too.

A computer screen with green text

Description automatically generated

Figure 8 – The replace function.

The display function traverses the tree and displays the content therein. It does so recursively by displaying the smaller left values, the value of the node, then the larger right values.

A computer screen shot of a program code

Description automatically generated

Figure 9 – The display function.

**Test Cases:**

The following test cases display the functionality of the designed binary search tree. It shows insert, displaying, removing, and replacing items in the tree. The tree maintains its order despite the changes to the data.

A computer screen with text and numbers

Description automatically generated

Figure 10 – The code for the test cases for the binary search tree

A screenshot of a computer

Description automatically generated

Figure 11 – The output of the test case code. Showing successful insertion, deletion, and replacement.