Project Report - P03

Project Number: P03

Author Information:

- Nathan Wiley: nwiley@uco.edu
- Colton Cox: ccox60@uco.edu

Struct TreeNode

The TreeNode struct has a templatized definition in *TreeNode.h* which includes the boolean data member "deleted", which serves as a flag to mark deleted nodes in the tree without actually removing those nodes.

Class LazyBinarySearchTree

The LazyBinarySearchTree class, defined in *LazyBinarySearchTree.h* implements the binary search tree data structure with lazy deletion, meaning that nodes are never actually removed from the tree, only flagged as deleted.

Insert()

The `insert()` function verifies that the data being inserted is within the allowed range (1-99), and then utilizes the recursive function `search_and_insert()` to traverse the tree and insert the node according to the standard BST constraints. If a node with a matching key value is found in the tree but is flagged as *deleted*, it is simply unflagged. If a node is found with a matching key value that is *not deleted*, the function returns false.

Remove()

The `remove()` function utilizes the recursive function `search_and_delete()` to traverse the tree (similarly to `search_and_insert()`) until a node with a matching key value is found. The matching node is then flagged as deleted, (regardless of whether or not it was already marked as deleted) and the function returns true. If no node is found with a matching key value, the function returns false.

findMin()

The `findMin()` function finds the smallest *non-deleted* key value in the tree, by traversing the leftmost branch of the tree, and pushing each of it's nodes onto a stack until the end of the branch is reached. Because of the binary search tree structure and the order in which the nodes are pushed onto the stack, the smallest value will be on the top of the stack. Nodes are popped from the top of the stack until the first non-deleted node is found, and it's value is returned.

findMax()

The `findMax()` function operates in the same way as `findMin()` with the exception that it traverses the rightmost branch of the tree in order to find the largest value. In all other ways it works exactly the same as findMin().

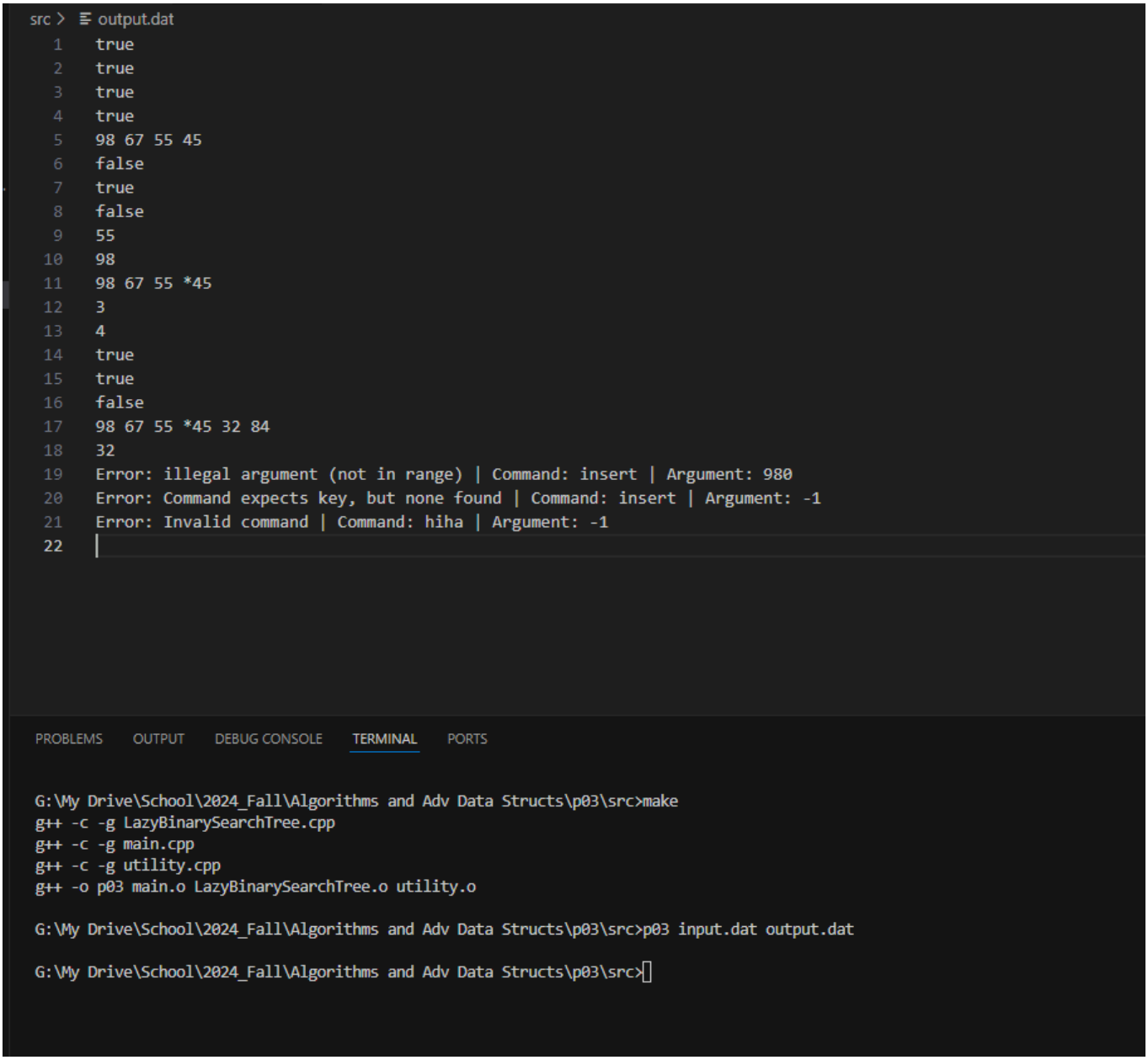
print()

The `print()` function uses two helper functions: `recursive_preorder()` and `lazy_print()`. First, `print()` checks if the tree is empty. If not, it calls `recursive_preorder()`, passing the tree's root, and an empty string by reference. `recursive_preorder()` traverses the tree and calls the `lazy_print()` function on each node, passing node and the string. `lazy_print()` then appends the string with the node's key value if it is not flagged as deleted, or an " * " if it is.

Input File

The input file is read line by line, with each line being parsed and tokenized, delimited by the ":" character. Once each line in the file is split into an array of individual tokens, the length of the array and the first word determine whether it is a valid command, and if so, whether it takes an argument. Based on this, the function `process_command()` is called, which takes as it's argument a reference to the tree, an output file stream , the string command, and optionally the string argument. `process_command` then matches the appropriate member function of the tree to the command/argument, and then outputs the result to the output file stream.

Test Run Screenshot



Given the following input file:

src > input.dat

```
insert:98
insert:67
insert:55
insert:45
print
remove:84
remove:45
contains:45
findMin
findMax
print
height
size
insert:84
insert:32
insert:32
print
findMin
insert:980
insert
hiha
```