Group 1

12/7/24

Rivers are Trees Design Document

This program implements a binary tree data structure to store, traverse, and manage information about water body features like rivers and dams on the Columbia River. Each node in the tree represents a feature and contains metadata such as its name, type, and additional descriptive information. Users can interact with the tree through commands like adding, deleting, searching, and traversing nodes.

Program Use: To interact with the Binary tree, users can explore and modify the tree dynamically through various options. The program will start by asking the user to pick a method of traversal and it will output the tree in the corresponding order. Then, the program lists the options for the user and what each option does.

Data Storage: Our data for this assignment is added to our binary tree in main through our add() and addat() methods which will be explained more later. Each function takes the name of the water feature, the direction for it to be placed on the tree, and the metadata for each item as parameters.

Node: A node represents an individual element in the binary tree. It holds 3 strings and 3 Node Pointers as data. The 3 strings are the name of the element, the direction for it to go on the tree, and the metadata corresponding to the element. The 3 Node pointers are The left and right children as well as the parent node.

Binary Tree: The binary tree class manages tree structures, performs operations, and facilitates user interactions. It maintains a pointer to the root node and provides methods for constructing and destructing the tree, ensuring proper memory management. Key features include tree management methods such as add, delete, search, and traversal.

> **Constructor/Destructor:** BinaryTree() initializes a new tree with an empty root and ~BinaryTree() deletes all nodes in the tree to prevent memory leaks.
>
> **Tree Management:** The add() method adds a new node to the tree. It has parameters name, type and metadata. The addAt() method adds a new node at a specific position, and

it has the parameters Node* start, Node* parent, name, type and metadata. This allows for the tree to be successfully updated by adding at specific spots. The deleteNode() method deletes a node with its parameter name and removes the subtree of that node. Finally, the deleteTree() method recursively deletes the tree.

Traversal: The BinaryTree class provides several traversal methods to explore its structure. The inorder method performs an in-order traversal, visiting nodes in the order of Left, Root, Right, while the preorder method follows a pre-order traversal pattern of Root, Left, Right. Similarly, the postorder method conducts a post-order traversal, visiting nodes in the order of Left, Right, Root. Additionally, the traversetree() method allows the user to manually explore the options of the tree using various traversal types or directional inputs.

Search: The searchNode() method allows the user to search for a specific node by name. It has parameters Node* current and a constant variable, name.

Display: For display, the display() method outputs the entire tree structure, and the printTree() method provides a recursive, representation of the tree's contents, making it easier to understand its organization. It has parameters Node* node, indent, and a boolean value last. This allowed us to change the spacing in the display due to the indent being incremented.

Below is what each user input does:

“inorder”: Performs an in-order traversal (Left, Root, Right) of the entire tree, starting from the root and outputs the nodes in ascending order based on their placement in the binary tree.

“postorder”: Performs a post-order traversal (Left, Right, Root) of the tree.

“preorder”: Performs a pre-order traversal (Root, Left, Right) of the tree, starting from the root.

1: Moves the current pointer to the right child of the current node. If the right child is nullptr, notify the user they’ve reached the end of the right branch.

2: Moves the current pointer to the left child of the current node. If the left child is nullptr, notify the user they've reached the end of the left branch.

3: Moves the current pointer to the parent node of the current node. If the current node is the root, notify the user they are already at the root.

4: Prompts the user to add a new node to the tree at the current location. The user must provide the name, type ("left" or "right"), and metadata for the new node. Adds the node if the position is valid.

5: Deletes the current node along with its subtree. If the current node doesn't exist, notify the user.

6: Prompts the user to enter the name of a node to search for in the tree. Then performs a recursive search starting from the root and displays the found node's details, or notifies if the node doesn't exist.

-1(Or any other input or invalid input): Ends the traversal session and exits the interactive loop. Any input not listed above results in an "Invalid input" message, and the traversal session is terminated.