Daniel Loranger

CS-300

Week 5 assignment

* Code Reflection – Having just learned about the Binary Search Trees, this class module went quite smoothly as I have also recently had practice with dealing with search trees in a previous course, so everything was quite clear what was needed. The overall flow of the code was fairly intuitive, with the main portion of the code that I stumbled on was in the function RemoveNode(). My intuition for how nodes is removed is still not well developed, so getting the code flow right took some time and required referencing some other implementations.
* Additionally, I still struggle with recursive work, and referenced some recursive functions from stackoverflow (linked in the code) that I used for the destructor functionality.
* Deleting the node with 2 children also does not make sense to my why you would find the smallest node and bring it to the current node, but it must work as the program works as expected.
* This is also the second time having using co-pilot to help with code development and I found in a few places that it led me in the wrong direction. This is to be expected, but overall, the feature seems to offer a lot of value in increasing coding efficiency and entering of comments around existing code.
* Pseudo-code

For brevity, existing static functions for testing are not detailed, only the functions that are edited are covered with below pseudo code.

* + Declare variables
  + Includes
  + Struct Bid
    - bidId: string
    - title: string
    - fund: string
    - amount: double
    - Bid() { amount = 0.0}
  + Struct Node
    - bid: Bid
    - left : Node pointer
    - right: node pointer
    - Node(): constructor
    - Node(bid): initializer
  + BinaryTreeSearch: Class
    - -root: Node\*
    - -addNode
    - -inOrder
    - -postOrder
    - -preOrder
    - -ClearNodes
    - -removeNode
    - +BinarySearchTree() // constructor
    - +~BinarySearchTree() // destructor
    - +InOrder
    - +PostOrder
    - +PreOrder
    - +Insert
    - +Remove
    - +Search
  + BinarySearchTree::BinarySearchTree() // constructor
    - Root = null
  + BinarySearchTree::~BinarySearchTree() //destructor
    - clearNodes(root)
  + BinarySearchTree::ClearNodes //recursive deletion of nodes
    - Recursively drill down the tree left and right to delete the tree of nodes.
  + BinarySearchTree::InOrder()
    - inOrder();
  + BinarySearchTree::PostOrder()
    - postOrder()
  + BinarySearchTree::PreOrder()
    - preOrder()
  + BinarySearchTree::Insert(Bid bid)
    - If root is empty, new bid becomes root
    - Else – add bid to root tree
  + BinarySearchTree::Remove(string bidId)
    - removeNode();
  + BinarySearchTree::Search(string bidId)
    - While not an empty node
      * If node is a match
        + Return node
      * Else
        + If bid is smaller – traverse left
        + Else traverse right
    - Return empty bid
  + BinarySearchTree::addNode(Node\* node, Bid bid)
    - If current node is larger than bid
      * If left is null bid becomes left
      * Else recurse left tree
    - Else //current is smaller than bid
      * If right is null, bid becomes right
      * Else recurse the right tree
  + BinarySearchTree::inOrder(Node\* node)
    - if node is not null
      * inOrder(left)
      * print bid
      * inOrder(right)
  + BinarySearchTree::postOrder(Node\* node)
    - If node is not null
      * postOrder(left)
      * postOrder(right)
      * print bid
  + BinarySearchTree::preOrder(Node\* node)
    - If node is not null
      * Print bid
      * postOrder(left)
      * postOrder(right)
  + BinarySearchTree::removeNode(Node\* node, string bidId)
    - If node is null, return node
    - If bidId < node bidID
      * removeNode(left) //Recurse left
    - else if bidId > node bidId
      * removeNode(right) //recurse right
    - else //bid is even
      * if no children delete node
      * else
        + if child only left

delete current node, update left pointer

* + - * + else if child only right

delete current node, update right pointer

* + - * + else //multiple children

traverse left to find smallest value

copy smallest data to the current node

remove the smallest node

* + - return node