**Code Reflection week 5**

**Jaden Bryon Knutson**

-------------------------------------------------------------------------------------------------------------

**A brief explanation of the code and its purpose:**

**Description:**

Binary Search Tree

The program: "BinarySearchTree.cpp", is a C++ implementation of a binary search tree data structure. The main function provides a menu-driven interface to interact with the binary search tree including options to load bids from a CSV file, display all bids, find a specific bid, and remove a bid from the tree. Example:

• Option 1 - load bids,

• Option 2 - display all bids,

• Option 3 - find a bid,

• Option 4 - remove a bid,

• Option 9 - and exit the program.

I have added in menu options 5, 6, 7, and 8 to the menu so the user can switch between the two files and switch the ordering method.

• Option 5 - Switches the order to: Pre-order

• Option 6 - Switches the order to: Post-order

• Option 7 - Switches the load directory to: eBid\_Monthly\_Sales.csv

• Option 8 - Switches the load directory to: eBid\_Monthly\_Sales\_Dec\_2016.csv

Once the new directory is switched, the user can reload by entering "1" and can now sort/display the new file.

Most of my work went into:

// FixMe (1): initialize housekeeping variables

// FixMe (2): In order root

// FixMe (3): Post order root

// FixMe (4): Pre order root

// FIXME (5) Implement inserting a bid into the tree

// FIXME (6) Implement removing a bid from the tree

// FIXME (7) Implement searching the tree for a bid

// FIXME (8) Implement inserting a bid into the tree

// FixMe (9): In-order traversal

// FixMe (10): Post-order traversal

// FixMe (11): Pre-order traversal

-------------------------------------------------------------------------------------------------------------

**Brief discussion of your experience in developing it & issues encountered during exercise and approaches to solve them:**

Overall, I feel this week went well. I think I need to slow down and keep a handy copy of assignment templates and questions. That way I will make sure I’m not forgetting something that may have got deleted along the way. I did it with the pre and postOrder and was really confused as to where they had gone. I then got the starting template along with watching the video to make sure I was including the required portions.

-------------------------------------------------------------------------------------------------------------

**A pseudocode description of the code that is clear and understandable and captures accurate logic to translate to the programming language:**

**1. CONSTRUCTOR BinarySearchTree::BinarySearchTree()**

* Initialize root of binary search tree to null.

-------------------------------------------------------------------------------------------------------------------

**2. DESTRUCTOR BinarySearchTree::~BinarySearchTree()**

* Call helper function to delete all nodes in tree starting from root.

-------------------------------------------------------------------------------------------------------------------

**3. DELETE ALL BinarySearchTree::deleteAll(Node\* node)**

* If node is not null:
* Recursively delete all nodes in left subtree of node.
* Recursively delete all nodes in right subtree of node.
* Delete node.

-------------------------------------------------------------------------------------------------------------------

**4. IN ORDER BinarySearchTree::InOrder()**

* Call helper function in Order to perform in-order traversal of tree.

-------------------------------------------------------------------------------------------------------------------

**5. IN ORDER BinarySearchTree::inOrder(Node\* root)**

* If root is not null:
* Create stack for nodes.
* Push nodes onto stack so leftmost node ends up on top.
* While nodes in the stack:
* Pop node from stack and process it.
* Visit right subtree of node.

-------------------------------------------------------------------------------------------------------------------

**6. PRE ORDER BinarySearchTree::PreOrder()**

* Call helper function preOrder to perform preOrder traversal of tree.

-------------------------------------------------------------------------------------------------------------------

**7. PRE ORDER BinarySearchTree::preOrder(Node\* root)**

* If root is not null:
* Create stack for nodes.
* Push nodes onto stack so that each node is processed before its children.
* While nodes in the stack:
* Pop node from stack and process it.

-------------------------------------------------------------------------------------------------------------------

**8. POST ORDER BinarySearchTree::PostOrder()**

* Call helper function postOrder to perform postOrder traversal of tree.

-------------------------------------------------------------------------------------------------------------------

**9. POST ORDER BinarySearchTree::postOrder(Node\* root)**

* If root is not null:
* Create stack for nodes.
* Push nodes onto stack so that each node is processed after its children.
* While nodes in the stack:
* Pop node from stack and process it.

-------------------------------------------------------------------------------------------------------------------

**10. INSERT BinarySearchTree::Insert(Bid bid)**

* Create new node for bid.
* If tree is empty, set root to new node.
* Otherwise find correct spot in tree for new node and insert it.

-------------------------------------------------------------------------------------------------------------------

**11. REMOVE BinarySearchTree::Remove(string bidId)**

* Call helper function removeNode to remove node with given bidId from tree.

-------------------------------------------------------------------------------------------------------------------

**12. REMOVE NODE BinarySearchTree::removeNode(Node\* node, string bidId)**

* If node is not null:
* If bidId is less than node's bidId, remove node from left subtree.
* If bidId is greater than node's bidId, remove node from right subtree.
* If bidId is equal to node's bidId, remove node and rearrange tree to maintain binary search tree property.

-------------------------------------------------------------------------------------------------------------------

**13. SEARCH BinarySearchTree::Search(string bidId)**

* Start at root of tree.
* While current node is not null:
* If bidId is equal to current node's bidId, return bid.
* If bidId is less than current node's bidId, move to left child.
* If bidId is greater than current node's bidId, move to right child.
* If no match found, return empty bid.

-------------------------------------------------------------------------------------------------------------------

**14. ADD NODE BinarySearchTree::addNode(Node\* node, Bid bid)**

* If bidId of current node is greater than bidId of bid to be inserted:
* If left child of current node is null:
* Create new node with bid to be inserted and set it as left child of current node.
* Otherwise:
* Recursively call addNode on left child of current node with the bid to be inserted.
* Otherwise:
* If right child of current node is null:
* Create new node with bid to be inserted and set it as right child of current node.
* Otherwise:
* Recursively call addNode on right child of current node with bid to be inserted.

-------------------------------------------------------------------------------------------------------------------