Derek Caprio, Derek Rodriguez
COP 4530	Project 3
Report and UML diagrams

Report:

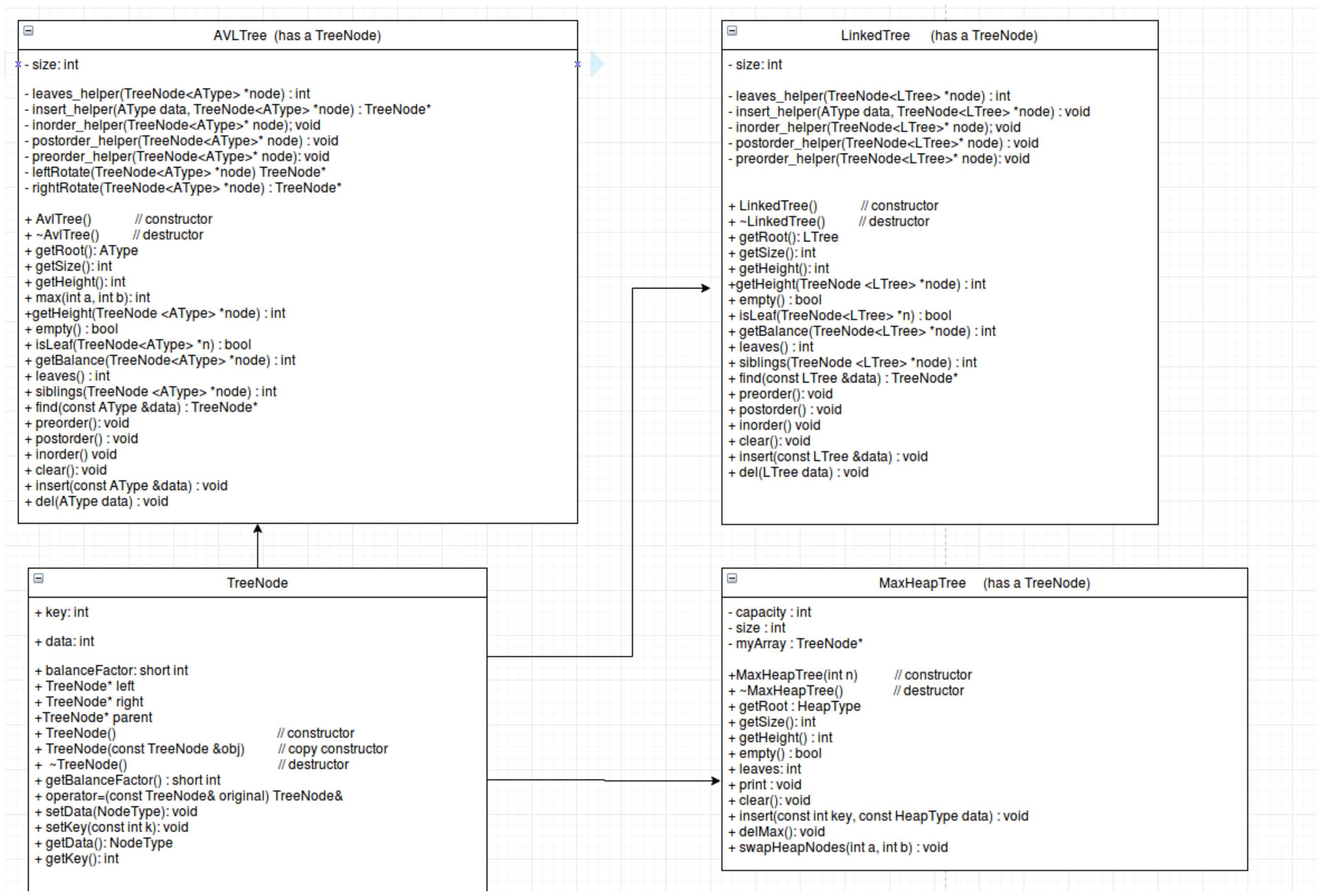
This project implements a binary search tree using the LinkedTree class, an AVL tree using the AVL tree class, and a max-heap using the MaxHeapTree class. All three data structures use objects of the TreeNode class as data members. The TreeNode class has a data member called key which holds the priority value of a node and is used with max heap data structure. It also has pointers to the left child, right child, and parent nodes for use with the linked and AVL trees, as well as a balance factor for use with the AVL tree. Both the linked tree and AVL tree use a linked implementation to build the data structure, while the heap uses a dynamic array based implementation.

The linked tree and the AVL tree share most of the same function definitions. When inserting into either data structure, the node to insert begins at the root and recursively moves to the left or right subtree depending on whether the new node's data is less than or greater than the data at the current node until it finds where it should insert itself. The difference with the AVL tree is that after this insertion is completed, balance factors are checked and the nodes are rotated as needed.

The max heap inserts the data according to its key value, with the higher keys being higher in the tree. The node is first inserted at the bottom and then bubbled-up the tree until the new node's index is greater than the index of its children.

The way we divided this project was that Derek C worked mostly on designing the heap while Derek R worked mostly on designing the trees. After this, fixing and debugging the code was shared across all parts of the project. This project took approximately 15 hours each.

[UML DIAGRAMS ON NEXT PAGE]


AVLTree (has a TreeNode)
- size: int
- leaves_helper(TreeNode<AType> *node) : int
- insert_helper(AType data, TreeNode<AType> *node) : TreeNode*
- inorder_helper(TreeNode<AType>* node); void
- postorder_helper(TreeNode<AType>* node) : void
- preorder_helper(TreeNode<AType>* node): void
- leftRotate(TreeNode<AType> *node) TreeNode*
- rightRotate(TreeNode<AType> *node) : TreeNode*
+ AvlTree() // constructor
+ ~AvlTree() // destructor
+ getRoot(): AType
+ getSize(): int
+ getHeight(): int
+ max(int a, int b): int
+getHeight(TreeNode <AType> *node) : int
+ empty() : bool
+ isLeaf(TreeNode<AType> *n) : bool
+ getBalance(TreeNode<AType> *node) : int
+ leaves() : int
+ siblings(TreeNode <AType> *node) : int
+ find(const AType &data) : TreeNode*
+ preorder(): void
+ postorder() : void
+ inorder() void
+ clear(): void
+ insert(const AType &data) : void
+ del(AType data) : void
LinkedTree (has a TreeNode)
- size: int
- leaves_helper(TreeNode<LTree> *node) : int
- insert_helper(AType data, TreeNode<LTree> *node) : void
- inorder_helper(TreeNode<LTree>* node); void
- postorder_helper(TreeNode<LTree>* node) : void
- preorder_helper(TreeNode<LTree>* node): void
+ LinkedTree() // constructor
+ ~LinkedTree() // destructor
+ getRoot(): LTree
+ getSize(): int
+ getHeight(): int
+getHeight(TreeNode <LTree> *node) : int
+ empty() : bool
+ isLeaf(TreeNode<LTree> *n) : bool
+ getBalance(TreeNode<LTree> *node) : int
+ leaves() : int
+ siblings(TreeNode <LTree> *node) : int
+ find(const LTree &data) : TreeNode*
+ preorder(): void
+ postorder() : void
+ inorder() void
+ clear(): void
+ insert(const LTree &data) : void
+ del(LTree data) : void
TreeNode
+ key: int
+ data: int
+ balanceFactor: short int
+ TreeNode* left
+ TreeNode* right
+TreeNode* parent
+ TreeNode() // constructor
+ TreeNode(const TreeNode &obj) // copy constructor
+ ~TreeNode() // destructor
+ getBalanceFactor() : short int
+ operator=(const TreeNode& original) TreeNode&
+ setData(NodeType): void
+ setKey(const int k): void
+ getData(): NodeType
+ getKey(): int
MaxHeapTree (has a TreeNode)
- capacity : int
- size : int
- myArray : TreeNode*
+MaxHeapTree(int n) // constructor
+ ~MaxHeapTree() // destructor
+ getRoot : HeapType
+ getSize(): int
+ getHeight() : int
+ empty() : bool
+ leaves: int
+ print : void
+ clear(): void
+ insert(const int key, const HeapType data) : void
+ delMax(): void
+ swapHeapNodes(int a, int b) : void