**Public class Tree{**

**Public Tree()**

Initialize root to null

//end constructor

**Public void preorderIt()**

Instantiate new LinkyStack object called nodeStack

Push root onto nodeStack

While(nodeStack is not empty)

Declare Node current and set to president popped off stack

Display current

If (current’s right child is not null)

Push current’s right child onto nodeStack

//end if

If(current’s left child is not null)

Push current’s left child onto nodeStack

//end if

//end while

//end preorderIt()

**Public void inorderIt()**

Instantiate new LinkyStack called nodeStack

Declare Node current and set to root

Declare Node temp

While(nodeStack is not empty or current is not null)

If(current is not null)

Push current onto nodeStack

Set current to current’s left child

Else

Set temp to president popped off stack

Display temp

Set current to temp’s right child

//end if

//end while

//end inorderIt()

**Public void postOrderIt()**

Instantiate new LinkyStack called nodeStack

Declare Node current and set to root

Push current onto the nodeStack

While(nodeStack is not empty)

Declare Node next and set to node on top of stack

Declare flag Boolean finishedSubtrees (true if next’s right child or left child equals current)

Declare flag Boolean isLeaf (true if next’s left or right child equals null)

If(finishedSubtrees or isLeaf is true)

Pop Node off of nodeStack

Display next

Set current to next

Else

If(next’s rightChild is not null)

Push next’s right child onto nodeStack

//end if

If(next’s leftChild is not null)

Push next’s left child onto nodeStack

//end if

//end if

//end while

//end postorderIt()

**Public void preorderRec(Node localRoot)**

If(localRoot is not null)

Display Node localRoot

Call preorderRec(localRoot.leftChild)

Call preorderRec(localRoot.rightChild)

//end if

//end preorderRec

**Public void inorderRec(Node localRoot)**

If(localRoot is not null)

Call inorderRec(localRoot.leftChilld)

Display Node localRoot

Call inorderRec(localRoot.rightChild)

//end if

//end inorderRec()

**Public void postorderRec(Node localRoot)**

If(localRoot is not null)

Call postorderRec(localRoot.leftChild)

Call postorderRec(localRoot.rightChild)

Display Node localRoot

//end if

//end postOrderRec()

**Public void insert(String last, String party, String num, String term)**

Instantiate new Node called newNode

Set newNode’s last to last

Set newNode’s party to party

Set newNode’s number to num

Set newNode’s term to term

If(root is null)

Set root to newNode

Else

Declare Node current and set to root

Declare Node parent

While(true)

Set parent to current

If( new last comes before current’s last)

Set current to current’s left child

If (current is null)

Set parent’s leftChild to newNode

Return;

//end if

Else

Set current to current’s right child

If (current is null)

Set parent’s rightChild to newNode

Return;

//end if

//end if

//end while

//end if

//end insert()

**Public Boolean delete(String key)**

Declare Node current and set to root

Declare Node parent and set to root

Declare Boolean isLeftChild and set to true

While (current’s last name does not equal the key)

Set parent to current

If (key comes before current’s last)

Set isLeftChild to true

Set current to current’s leftChild

Else

Set isLeftChild to false

Set current to current’s rightChild

//end if

If(current is null)

Return false

//end if

//end while

If(current’s left child is null and current’s right child is null)

If(isLeftChild is true)

Set parent’s left child to null

Else

Set parent’s right child to null

//end if

Else if (current’s right child is null)

If(current equals root)

Set root to current’s left child

Else if (isLeftChild is true)

Set parent’s left child to current’s left child

Else

Set parent’s right child to current’s left child

//end if

Else if (current’s left child is null)

If (current equals root)

Set root to current’s right child

Else if (isLeftChild is true)

Set parent’s left child to current’s right child

Else

Set parent’s right child to current’s right child

//end if

Else

Declare node successor and set to Node returned by getSuccessor(current)

If (current equals root)

Set root to successor

Else if(isLeftChild is true)

Set parent’s left child to successor

Else

Set parent’s right child to successor

//end if

Set successor’s left child to current’s left child

//end if

Return true

//end delete()

**Private Node getSuccessor (Node delNode)**

Declare Node successorParent and set to delNode

Declare Node successor and set to delNode

Declare Node current and set to delNode’s right child

While(current is not null)

Set successorParent to successor

Set successor to current

Set current to current’s left child

//end while

If(successor is not delNode’s right child)

Set successorParent’s left child to successor’s right child

Set successor’s right child to delNode’s right child

//end if

Return successor

//end getSuccessor()

**Public Node getRoot()**

Return root

//end getRoot()

//end Tree class