Binary Tree Code

A simple Linked implementation

```
class TreeNode {

   DataType Data;          //any appropriate type
   TreeNode Left, Right;
}
```

Note: default constructor is TreeNode (). You could define methods like GetData and SetData if desired.

```
public class TreeClass {
   private TreeNode Tree;      //reference to root
   private TreeNode Here ;   //reference to node
                        // in Tree where action occurs
   private TreeNode Parent_of_Here;
                //maintained as a matter of efficiency

public void TreeClass() {                  //constructor
   Tree = null;
   Here = null;
   Parent_of_Here = null;
}
```

```
public TreeNode MakeTree(DataType item) {
                                          //constructor
   TreeNode Temp = new TreeNode;
   Temp.Data = item;
   Temp.Left = null;
   Temp.Right = null;
}

public boolean TreeEmpty() {
       return (Tree == null);
}
```

```
public void SetRight(DataType item) {
                  //add right child to existing node P

   TreeNode P = Here;
   if (P == null) “error handling”     //P must exist
   else if (P.Right != null) “error handling”
                        //It is an error for P to have an
       else {                          //existing right child
          TreeNode Temp = MakeTree (item);
         P.Right = Temp;
       }
}
```

```
public void SetLeft(DataType item) {
                    //add left child to existing node P
   TreeNode P = Here;
   if (P == null) "error handling"    //P must exist
   else if (P.Left != null) "error handling"
                        //It is an error for P to have an
   else {                               //existing left child
      TreeNode Temp = MakeTree (item);
     P.Left = Temp;
   }
} //Note: It would be easy to rewrite SetLeft and SetRight to
   // move an existing child down a level if desired.
```

```
public void TreeCopy (TreeNode Tree) {
   //a copy constructor...takes the existing tree
     Tree and makes a copy of it to initiate a
     new Tree - may require modifying interface
     or ADT
   ... exercise for the student...
}

public DataType TreeDelete
   // to be discussed later
}  //end TreeDelete
```

```
public void TreeSearch (DataType Item) {
            //iterative solution

//Searches tree for value Item.  Result is
  returned in the class variable Here.  Here is
  set to null if the tree is empty or the item is not
  present, otherwise Here is set to point to the
  node containing item

   Here = Tree;       //Start at root of existing tree
   Parent_of_Here = null;
```

```
   if !(Here == null) {  //look thru non-empty tree
      while ((Here != null) && (Here.Data !=
        Item)) {
         Parent_of_Here = Here;
         if (Item < Here.Data) Here = Here.Left
         else Here = Here.Right;
      }  //end while
      if (Here == null) Parent_of_Here = null;
                                    //item not in tree
   }  // end if Here != null
}  //end TreeSearch
```

```
public void TreeInsert
                (DataType Item, TreeNode Root) {
                                    //recursive solution

   if (Root == null) Root = MakeTree(Item);
                                  //insert at root of Tree
   else if (Item < Root.Data) TreeInsert ( Item,
     Root.Left )
   else TreeInsert ( Item, Tree.Right);

}  //end TreeInsert
```

```
public void TreeInsert (DataType Item) {
                                    //iterative solution

   TreeNode Parent;                 //trailing pointer
   TreeNode Curr;                       //Current ptr
   TreeNode Temp = MakeTree(Item);
                          //set up node for insertion
   Curr = Tree;       //Start at root of existing tree
```

```
    if (Curr == null) Curr = Temp;
                                  //insert at root of Tree
    else {
        while (Curr != null) {        //move Curr through
            Parent = Curr;   //tree to identify insert pt
            if (Item < Curr.Data) Curr = Curr.Left
            else Curr = Curr.Right);
        }
        if (Item < Parent.Data) Parent.Left  = Temp
        else Parent.Right = Temp;
    }
}  //end TreeInsert
} //end TreeClass
```

Suppose a user had an instance T of TreeClass and he wanted to insert Bob as the left child of Alice. Assume the variables **BobsName** and **AlicesName** have been correctly defined. The insertion could take place as follows:

```
TreeSearch (AlicesName);            //Set Here to Alice

SetLeft (BobsName);    //Will return an error if Alice
        //has existing left child or if Alice is not in the tree
```

The use of the class variable **Here** allows the user to pass around a pointer without having direct knowledge of it.