Binary Tree Dynamic Memory Management Functions, Reading

The Destructor

We already have a **clear** function from an earlier reading in this module. The destructor can be written inline, calling clear.

The Copy Constructor

For a simple 3-node system, the logic is like this, after creating a root node:

```
copy original's root contents into new root
if original root has a left child
  create and link a left child node
  copy original's left child contents into new left child
if original root has a right child
  create and link a right child node
  copy original's right child contents into new right child
```

This looks like another recursive solution, because "copy original's left child contents into new left child" does the same thing that the left child that the above does for the root. Considering that the left child will look like a root to *its* descendants, we get this for a recursive loop to copy a tree:

```
void copyTree(Node* p, const Node* copiedTree)
{
  if (!p) return; // to end the loop
  p->key = copiedTree->key;
  p->value = copiedTree->value;
  p->left = p->right = 0;
  if (copiedTree->left) copyTree(p->left = new Node, copiedTree->left);
  if (copiedTree->right) copyTree(p->right = new Node, copiedTree->right);
}
```

To start the process, we create a root node and pass its pointer to the function, along with a pointer to the corresponding node in the copied tree. The first statement ends the recursive loop if there is no node to copy, which should happen when the end of the tree is reached.

The compound function call statements have parameters like **p->left = new Node**, which is allowed by C++. This (1) creates a node, (2) attaches it as a child of "p", and sends that node's memory location to the next cycle of the loop.

The **copyTree** function needs to be a private setter -- the main program has no use for it because it has no access to **Node*** pointers. The copy constructor has to start the process by creating the root and making the first call to start the recursive loop:

```
  rootNode = 0; // in case the copied tree is empty
  if (original.rootNode) copyTree(rootNode = new Node, original.rootNode);
  siz = original.siz;
```

"original" is the constant reference in the copy constructor's parameter list.

The Assignment Operator

After checking for self-assignment (with something like **if (this == &original) return *this;**), the assignment operator does what the destructor and copy constructor do -- same code blocks. It ends by returning a self reference.