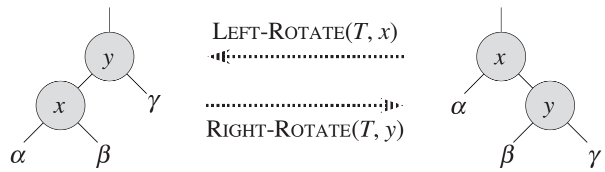

Figure 13.2 The rotation operations on a binary search tree. The operation LEFT-ROTATE(T, x) transforms the configuration of the two nodes on the right into the configuration on the left by changing a constant number of pointers. The inverse operation RIGHT-ROTATE(T, y) transforms the configuration on the left into the configuration on the right. The letters α, β, and γ represent arbitrary subtrees. A rotation operation preserves the binary-search-tree property: the keys in α precede $x.key$, which precedes the keys in β, which precede $y.key$, which precedes the keys in γ.

```
LEFT-ROTATE(T, x)
y = x.right                  // set y
x.right = y.left             // turn y's left subtree into x's right subtree
if y.left ≠ T.nil
    y.left.p = x
y.p = x.p                    // link x's parent to y
if x.p == T.nil
    T.root = y
elseif x == x.p.left
    x.p.left = y
else x.p.right = y
y.left = x                   // put x on y's left
x.p = y
```

Figure 13.3 shows an example of how LEFT-ROTATE modifies a binary search tree. The code for RIGHT-ROTATE is symmetric. Both LEFT-ROTATE and RIGHT-ROTATE run in $O(1)$ time. Only pointers are changed by a rotation; all other attributes in a node remain the same.

Exercises

13.2-1

Write pseudocode for RIGHT-ROTATE.

13.2-2

Argue that in every n-node binary search tree, there are exactly $n - 1$ possible rotations.