cout << "Done now that i = " << i << "\n";
return 0;
}

Here is the output:

Enter the starting countdown value: 4
i = 4
i = 3
i = 2
i = 1
Done now that i = 0

Note that the loop terminates when i reaches 0.

How do relational expressions, such as i < 5, fit into this framework of terminating a loop with a 0 value? Before the bool type was introduced, relational expressions evaluated to 1 if true and 0 if false. Thus, the value of the expression 3 < 5 was 1 and the value of 5 < 5 was 0. Now that C++ has added the bool type, however, relational expressions evaluate to the bool literals true and false instead of 1 and 0. This change doesn't lead to incompatibilities, however, for a C++ program converts true and false to 1 and 0 where integer values are expected, and it converts 0 to false and nonzero to true where bool values are expected.

The for loop is an entry-condition loop. This means the test expression is evaluated before each loop cycle. The loop never executes the loop body when the test expression is false. For example, suppose you rerun the program in Listing 5.2 but give 0 as a starting value. Because the test condition fails the very first time it's evaluated, the loop body never gets executed:

Enter the starting countdown value: 0
Done now that i = 0

This look-before-you-loop attitude can help keep a program out of trouble.

The update-expression is evaluated at the end of the loop, after the body has been executed. Typically, it's used to increase or decrease the value of the variable keeping track of the number of loop cycles. However, it can be any valid C++ expression, as can