the expression has the value false. Here are some examples:

5 == 5 || 5 == 9    // true because first expression is true
5 > 3 || 5 > 10    // true because first expression is true
5 > 8 || 5 < 10    // true because second expression is true
5 < 8 || 5 > 2    // true because both expressions are true
5 > 8 || 5 < 2    // false because both expressions are false

Because the || has a lower precedence than the relational operators, you don’t need to use parentheses in these expressions. Table 6.1 summarizes how the || operator works.

C++ provides that the || operator is a sequence point. That is, any value changes indicated in the left side take place before the right side is evaluated. For example, consider the following expression:

i++ < 6 || i == j

Suppose i originally has the value 10. By the time the comparison with j takes place, i has the value 11. Also, C++ won’t bother evaluating the expression on the right if the expression on the left is true, for it only takes one true expression to make the whole logical expression true. (The semicolon and the comma operator also are sequence points.)

Listing 6.4 uses the || operator in an if statement to check for both uppercase and lowercase versions of a character. Also, it uses C++’s string concatenation feature (see Chapter 4, "Compound Types") to spread a single string over three lines.

| The Value of expr1 || expr2 | expr1 == true | expr1 == false |
|---------------------|-----------------|---------------|----------------|
| expr2 == true       | true            | true          |
| expr2 == false      | true            | false         |

Listing 6.4 or.cpp

// or.cpp -- use logical OR operator
#include <iostream>