5 == 3 && 4 == 4 // false because first expression is false
5 > 3 && 5 > 10 // false because second expression is false
5 > 8 && 5 < 10 // false because first expression is false
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. Like the || operator, the && operator acts as a sequence point, so the left side is evaluated and any side effects are carried out before the right side is evaluated. If the left side is false, the whole logical expression must be false, so C++ doesn't bother evaluating the right side in that case. Table 6.2 summarizes how the && operator works.

<table>
<thead>
<tr>
<th>The Value of expr1 &amp;&amp; expr2</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr1 == true</td>
</tr>
<tr>
<td>expr2 == true</td>
</tr>
<tr>
<td>expr2 == false</td>
</tr>
</tbody>
</table>

Listing 6.5 shows how to use && to cope with a common situation, terminating a while loop, for two different reasons. In the listing, a while loop reads values into an array. One test (i < ArSize) terminates the loop when the array is full. The second test (temp >= 0) gives the user the option of quitting early by entering a negative number. The && operator lets you combine the two tests into a single condition. The program also uses two if statements, an if else statement, and a for loop, so it demonstrates several topics from this and the preceding chapter.

Listing 6.5 and.cpp

// and.cpp -- use logical AND operator
#include <iostream>
using namespace std;
const int ArSize = 6;
int main()
{