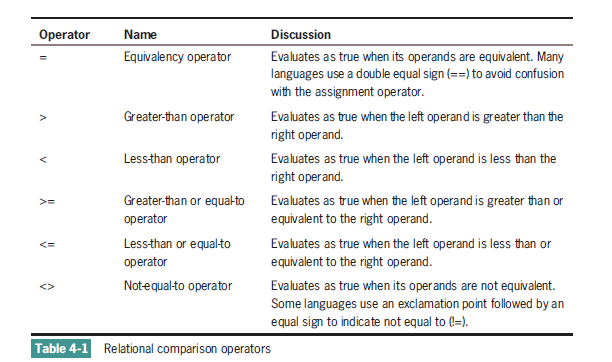
Relational Operators & Selection Structures

Using Relational Comparison Operators

Table 4-1 describes the six relational comparison operators supported by all modern programming languages. Each of these operators is binary—that is, each requires two operands. When you construct an expression using one of the operators described in Table 4-1, the expression evaluates to true or false.

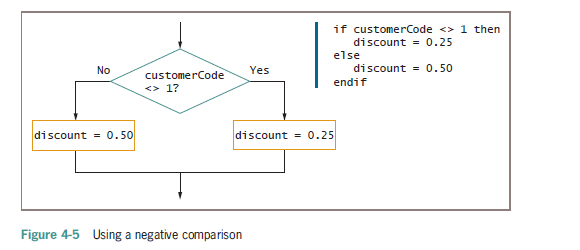


Usually, both operands in a comparison must be the same data type; that is, you can compare numeric values to other numeric values, and text strings to other strings.

In any Boolean expression, the two values compared can be either variables or constants. For example, the expression currentTotal = 100? compares a variable, currentTotal, to a numeric constant, 100. Depending on the currentTotal value, the expression is true or false. In the expression currentTotal = previousTotal?, both values are variables, and the result is also true or false depending on the values stored in each of the two variables. Although it’s legal, you would never use expressions in which you compare two constants—for example,20 = 20? or 30 = 40?. Such expressions are trivial expressions because each will always evaluate to the same result: true for 20 = 20? and false for 30 = 40?.

Any decision can be made using combinations of just three types of comparisons: equal, greater than, and less than. You never need the three additional comparisons (greater than or equal, less than or equal, or not equal), but using them often makes decisions more convenient. For example, assume that you need to issue a 10 percent discount to any customer whose age is 65 or greater, and charge full price to other customers. You can use the greater-than-or-equal-to symbol to write the logic as follows:

if customerAge >= 65 then discount = 0.10  
else  
discount = 0  
endif



In Figure 4-5, if the value of customerCode is equal to 1, the logical flow follows the false  
branch of the selection. If customerCode <> 1 is true, the discount is 0.25; if  
customerCode <> 1 is not true, it means the customerCode is 1, and the discount is  
0.50. Even reading the phrase “if customerCode is not equal to 1 is not true” is awkward.  
Figure 4-6 shows the same decision, this time asked using positive logic. Making the decision based  
on what customerCode is is clearer than trying to determine what customerCode  
not.

