Pearson New International Edition

Starting Out with Programming Logic and Design

Tony Gaddis

Third Edition

ALWAYS LEARNING™

PEARSON®

Pearson New International Edition

Starting Out with Programming
Logic and Design
Tony Gaddis
Third Edition

PEARSON®

Program 8

```
// Constants for the TV prices
Constant Real MODEL_100_PRICE = 199.99
Constant Real MODEL_200_PRICE = 269.99
Constant Real MODEL_300_PRICE = 349.99

// Constants for the TV sizes
Constant Integer MODEL_100_SIZE = 24
Constant Integer MODEL_200_SIZE = 27
Constant Integer MODEL_300_SIZE = 32

// Variable for the model number
Declare Integer modelNumber

// Get the model number.
Display "Which TV are you interested in?"
Display "The 100, 200, or 300?"
Input modelNumber

// Display the price and size.
Select modelNumber
   Case 100:
      Display "Price: $", MODEL_100_PRICE
      Display "Size: ", MODEL_100_SIZE
   Case 200:
      Display "Price: $", MODEL_200_PRICE
      Display "Size: ", MODEL_200_SIZE
   Case 300:
      Display "Price $", MODEL_300_PRICE
      Display "Size: ", MODEL_300_SIZE
   Default:
      Display "Invalid model number"
End Select
```

Program Output (with Input Shown in Bold)

```
Which TV are you interested in?
The 100, 200, or 300?
100 [Enter]
Price: $199.99
Size: 24
```

Program Output (with Input Shown in Bold)

```
Which TV are you interested in?
The 100, 200, or 300?
200 [Enter]
Price: $269.99
Size: 27
```

Program Output (with Input Shown in Bold)

```
Which TV are you interested in?
The 100, 200, or 300?
300 [Enter]
Price: $349.99
Size: 32
```

Program Output (with Input Shown in Bold)

```
Which TV are you interested in?
The 100, 200, or 300?
500 [Enter]
Invalid model number
```

NOTE: The details of writing a case structure differ from one language to another. Because of the specific rules that each language uses for writing case structures, you might not be able to use the case structure for every multiple alternative decision. In such an event, you can use the `If-Then-Else If` statement or a nested decision structure.

Checkpoint

17 What is a multiple alternative decision structure?

18 How do you write a multiple alternative decision structure in pseudocode?

19 What does the case structure test, in order to determine which set of statements to execute?

20 You need to write a multiple alternative decision structure, but the language you are using will not allow you to perform the test you need in a `Select Case` statement. What can you do to achieve the same results?

6 Logical Operators

CONCEPT: The logical `AND` operator and the logical `OR` operator allow you to connect multiple Boolean expressions to create a compound expression. The logical `NOT` operator reverses the truth of a Boolean expression.

Programming languages provide a set of operators known as *logical operators,* which you can use to create complex Boolean expressions. Table 3 describes these operators.

Table 4 shows examples of several compound Boolean expressions that use logical operators.

Table 3 Logical operators

Operator	Meaning
`AND`	The `AND` operator connects two Boolean expressions into one compound expression. Both subexpressions must be true for the compound expression to be true.
`OR`	The `OR` operator connects two Boolean expressions into one compound expression. One or both subexpressions must be true for the compound expression to be true. It is only necessary for one of the subexpressions to be true, and it does not matter which.
`NOT`	The `NOT` operator is a unary operator, meaning it works with only one operand. The operand must be a Boolean expression. The `NOT` operator reverses the truth of its operand. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

Table 4 Compound Boolean expressions using logical operators

Expression	Meaning
`x > y AND a < b`	Is `x` greater than `y` `AND` is `a` less than `b`?
`x == y OR x == z`	Is `x` equal to `y` OR is `x` equal to `z`?
`NOT (x > y)`	Is the expression `x > y` NOT true?

NOTE: In many languages, most notably C, C++, and Java, the `AND` operator is written as `&&`, the `OR` operator is written as `||`, and the `NOT` operator is written as `!`.

The AND Operator

The `AND` operator takes two Boolean expressions as operands and creates a compound Boolean expression that is true only when both subexpressions are true. The following is an example of an `If-Then` statement that uses the `AND` operator:

```
If temperature < 20 AND minutes > 12 Then
   Display "The temperature is in the danger zone."
End If
```

In this statement, the two Boolean expressions `temperature < 20` and `minutes > 12` are combined into a compound expression. The `Display` statement will be executed only if `temperature` is less than 20 AND `minutes` is greater than 12. If either of the Boolean subexpressions is false, the compound expression is false and the message is not displayed.

Table 5 shows a truth table for the `AND` operator. The truth table lists expressions showing all the possible combinations of true and false connected with the `AND` operator. The resulting values of the expressions are also shown.

Table 5 Truth table for the AND operator

Expression	Value of the Expression
true AND false	false
false AND true	false
false AND false	false
true AND true	true

As the table shows, both sides of the AND operator must be true for the operator to return a true value.

The OR Operator

The OR operator takes two Boolean expressions as operands and creates a compound Boolean expression that is true when either of the subexpressions is true. The following is an example of an If-Then statement that uses the OR operator:

```
If temperature < 20 OR temperature > 100 Then
   Display "The temperature is in the danger zone."
End If
```

The Display statement will execute only if temperature is less than 20 OR temperature is greater than 100. If either subexpression is true, the compound expression is true. Table 6 shows a truth table for the OR operator.

Table 6 Truth table for the OR operator

Expression	Value of the Expression
true OR false	true
false OR true	true
false OR false	false
true OR true	true

All it takes for an OR expression to be true is for one side of the OR operator to be true. It doesn't matter if the other side is false or true.

Short-Circuit Evaluation

In many languages both the AND and OR operators perform *short-circuit evaluation*. Here's how it works with the AND operator: If the expression on the left side of the AND operator is false, the expression on the right side will not be checked. Because the compound expression will be false if only one of the subexpressions is false, it would waste CPU time to check the remaining expression. So, when the AND operator finds that the expression on its left is false, it short-circuits and does not evaluate the expression on its right.

Here's how short-circuit evaluation works with the OR operator: If the expression on the left side of the OR operator is true, the expression on the right side will not be checked. Because it is only necessary for one of the expressions to be true, it would waste CPU time to check the remaining expression.

The NOT Operator

The `NOT` operator is a unary operator that takes a Boolean expression as its operand and reverses its logical value. In other words, if the expression is true, the `NOT` operator returns false, and if the expression is false, the `NOT` operator returns true. The following is an `If-Then` statement using the `NOT` operator:

```
If NOT(temperature > 100) Then
   Display "This is below the maximum temperature."
End If
```

First, the expression (`temperature > 100`) is tested and a value of either true or false is the result. Then the `NOT` operator is applied to that value. If the expression (`temperature > 100`) is true, the `NOT` operator returns false. If the expression (`temperature > 100`) is false, the `NOT` operator returns true. The previous code is equivalent to asking: "Is the temperature not greater than 100?"

NOTE: In this example, we have put parentheses around the expression `temperature > 100`. The reason for this is that, in many languages, the `NOT` operator has higher precedence than the relational operators. Suppose we wrote the expression as follows:

```
NOT temperature > 100
```

In many languages this expression would not work correctly because the `NOT` operator would be applied to the `temperature` variable, not the expression `temperature > 100`. To make sure that the operator is applied to the expression, we enclose it in parentheses.

Table 7 shows a truth table for the `NOT` operator.

Table 7 Truth table for the `NOT` operator

Expression	Value of the Expression
`NOT` true	false
`NOT` false	true

The Loan Qualifier Program Revisited

In some situations the `AND` operator can be used to simplify nested decision structures. For example, recall that the loan qualifier program in Program 5 uses the following nested `If-Then-Else` statements:

```
If salary >= 30000 Then
    If yearsOnJob >= 2 Then
       Display "You qualify for the loan."
    Else
       Display "You must have been on your current"
       Display "job for at least two years to qualify."
    End If
Else
    Display "You must earn at least $30,000"
    Display "per year to qualify."
End If
```

The purpose of this decision structure is to determine that a person's salary is at least $30,000 and that he or she has been at his or her current job for at least two years. Program 9 shows a way to perform a similar task with simpler code.

Program 9

```
 1 // Declare variables
 2 Declare Real salary, yearsOnJob
 3
 4 // Get the annual salary.
 5 Display "Enter your annual salary."
 6 Input salary
 7
 8 // Get the number of years on the current job.
 9 Display "Enter the number of years on your ",
10         "current job."
11 Input yearsOnJob
12
13 // Determine whether the user qualifies.
14 If salary >= 30000 AND yearsOnJob >= 2 Then
15    Display "You qualify for the loan."
16 Else
17    Display "You do not qualify for this loan."
18 End If
```

Program Output (with Input Shown in Bold)

```
Enter your annual salary.
35000 [Enter]
Enter the number of years on your current job.
1 [Enter]
You do not qualify for this loan.
```

Program Output (with Input Shown in Bold)

```
Enter your annual salary.
25000 [Enter]
Enter the number of years on your current job.
5 [Enter]
You do not qualify for this loan.
```

Program Output (with Input Shown in Bold)

```
Enter your annual salary.
35000 [Enter]
Enter the number of years on your current job.
5 [Enter]
You qualify for the loan.
```

The `If-Then-Else` statement in lines 14 through 18 tests the compound expression `salary >= 30000 AND yearsOnJob >= 2`. If both subexpressions are true, the compound expression is true and the message "You qualify for the loan" is displayed. If either of the subexpressions is false, the compound expression is false and the message "You do not qualify for this loan" is displayed.