Chapter 4
Making Decisions

At a Glance

- Overview
- Objectives
- Quick Quizzes
- Additional Resources
- Key Terms
Overview

Chapter 4 introduces relational comparison operators and the use of AND and OR logic. Students will learn how to use these operators and what type of common mistakes they should avoid. Students will also learn to make selections within ranges and to understand the precedence of AND and OR structures.

Chapter Objectives

In this chapter, your students will learn about:

- Evaluating Boolean expressions to make comparisons
- The relational comparison operators
- AND logic
- OR logic
- Making selections within ranges
- Precedence when combining AND and OR operators

Evaluating Boolean Expressions to Make Comparisons

1. Understand the concept of the Boolean expression and discuss how this type of evaluation is “natural” for a computer.

2. Contrast the dual-alternative selection with the single-alternative selection using Figures 4-1 and 4-2.

3. Review the logic for the overtime payroll program seen in Figure 4-3.

   The dual-alternative structure is also known as an if-then-else, while the single-alternative structure is also known as an if-then.

   Understand thoroughly the logic and pseudocode used for the overtime pay calculation. New programming students have a great deal of trouble with algorithm development and often will have statements in an incorrect order.

Using the Relational Comparison Operators

1. Understand the six relational comparison operators (Table 4-1), and these operators are used to build Boolean expressions.

2. Understand the three possible ways to compare two values:
   a. The two values are equal
   b. The first value is greater than the second value
3. Define the term **trivial expression** and review the examples on page 139.

Students new to programming seem to have the most difficulty with the $\leq$ and the $\geq$ symbols and understanding when these operators return a true value.

4. Understand why the $\leq$ and the $\geq$ operators make code more understandable.

5. Understand the need to adjust the Boolean expression and the logic based on the comparison being used.

Students new to programming often have difficulty constructing simple comparison expressions when faced with a problem description.

6. Understand the code readability issues that occur when using a “not equal” test results in a double negative, and describe how to rephrase the question to have a positive test.

Choosing good variable names for Boolean variables and avoiding names that start with “not” because of the potential for double negatives.

**Avoiding a Common Error with Relational Operators**

1. Understand the importance of using the correct relational operator and determining whether to include or exclude the boundary condition.
Quick Quiz 1

1. Many restaurants offer senior meals for persons 55 and older. How might you write a selection question to select all persons eligible for senior meals?
   Answer: If age >= 55

2. When should a single-alternative decision structure be used instead of a dual-alternative structure?
   Answer: when action is to be taken only if the decision expression is true

3. What values can result from evaluating Boolean expressions?
   Answer: True or False

Understanding AND Logic

1. Understand compound conditions, and explain that one type of compound condition is needed when the results of at least two decisions must be true for some action to occur.

2. Understand the concept of an AND decision, in which both tests must evaluate to True, and explain that AND decisions can be constructed using a nested decision.

   Review the concept of nested statements at this point. Remind students that a statement must be completely nested within another statement (no overlap).

3. Understand the nesting of both the flowchart symbols and the pseudocode, and point out that the if statements are nested.

4. Walk through the flowchart in Figure 4-7.

Nesting AND Decisions for Efficiency

1. Understand the analysis one should perform to determine if the order of nesting the decisions will affect performance.

2. Walk through Figure 4-8 to design a nested decision structure that produces a cell phone bill.

3. Some prior knowledge of the nature of the data is required to make an analysis.

   The people who work regularly with data (the users) can usually give you some insight into the nature of the data. This is an important part of understanding the problem that the program is intended to address.

4. In some cases, you may not be able to change the order of the questions in an AND statement due to dependencies.
5. Give the general rule that in an AND decision, first ask the question that is less likely to be true, and discuss how this improves efficiency.

When dealing with very large data sets containing hundreds of thousands or millions of records, asking the correct question first can result in a significant performance improvement.

**Using the AND Operator**

1. Understand the concept of using a conditional AND operator to combine two or more questions in a single statement.

2. All Boolean expressions must be true to return a value of true for the entire statement.

The need to consider efficiency when deciding the order in which the Boolean expressions appear in the statement.

3. Introduce truth tables, and walk through Table 4-2 to describe the truth table for the AND operator.

4. Understand how the computer actually evaluates an expression with a logical AND operator as if it was a nested if statement. Then introduce short-circuit evaluation.

**Avoiding Common Errors in an AND Selection**

1. Understand the mistake of failing to nest the second decision entirely within the first decision in an AND selection, and walk through Figure 4-10 to illustrate.

2. Understand the mistake of using incorrect questions to determine inclusion in a range.

3. Understand the failure to use two complete Boolean expressions when using the AND operator.

Because the computer evaluates the expressions used with the logical AND operator as a set of nested if statements, you must provide two complete Boolean expressions.

**Understanding OR Logic**

1. Understand the concept of the OR decision, in which only one of the conditions must be true to produce a true result.

2. If the first condition is true in an OR decision, there is no need to test the second condition.
In some programming logic, the compiler will not even evaluate the second condition if the first condition is true. This is called short-circuiting.

3. Walk through the sample application shown in Figure 4-11.

**Writing OR Decisions for Efficiency**

1. Walk through Figure 4-12, and understand how the efficiency is affected by the order in which the questions are asked in an OR statement.

2. The question more likely to be true will eliminate the need to ask the second question.

**Using the OR Operator**

1. Understand the concept of using a conditional OR operator to combine two or more questions in a single statement.

2. Walk through Table 4-3 to present the truth table for the OR operator.

3. Only one of the Boolean expressions must be true to return a value of true for the entire statement.

The need to consider efficiency when deciding the order in which the Boolean expressions appear in the statement.

**Avoiding Common Errors in an OR Selection**

1. Understand the error of creating an OR selection that is unstructured, as shown in Figure 4-14.

2. Understand the problems that casual use of the word AND can cause when the logic requires an OR decision.

3. The mistake of incorrectly setting the boundary conditions on an OR decision, as in Figure 4-15, and the correct logic in Figure 4-16.

**Making Selections within Ranges**

1. Understand the concept of a range check by presenting the example on pages 160-161 and in Figures 4-19 and 4-20.

**Avoiding Common Errors Using Range Checks**

1. Understand the error of testing values (using Figure 4-21) that lead to dead or unreachable paths.
2. Understand the error of asking unnecessary questions and the two possible conditions that might cause this.

**Understanding Precedence When Combining AND and OR Selections**

1. You can combine many AND and OR operators in an expression.

2. Understand the use of multiple AND operators when multiple conditions must all be true.

3. Contrast the use of multiple OR operators when only one of multiple conditions must be true.

| Be careful about the casual use of AND and OR in English when determining the actual conditions that must be met. |

4. Understand the precedence of the AND operator when combined with the OR operator in a single statement and the use of parentheses to force the OR expression to be evaluated first.

5. Using too many AND and OR operators in a single statement will make the code less readable.

**Quick Quiz 2**

1. Which logical operator requires that only one condition be true to produce a true result?  
   Answer: OR

2. Which logical operator requires that both conditions be true to produce a true result?  
   Answer: AND

3. In an OR decision, you should first ask the question that is ____ (more/less) likely to be true.  
   Answer: more

4. In an AND decision, you should first ask the question that is ____ (more/less) likely to be true.  
   Answer: less

5. When AND and OR operators are combined in the same statement, which has precedence?  
   Answer: AND
Additional Resources

1. An article about truth tables:
   www.math.csusb.edu/notes/logic/lognot/node1.html

2. A Wikipedia article about logical operators:
   http://en.wikipedia.org/wiki/Boolean_operators

3. A brief biography of George Boole:
   www.sjsu.edu/depts/Museum/boole.html

Key Terms

- **AND decision** – two conditions must both be true for an action to take place.
- **Boolean expression** – represents only one of two states, usually expressed as true or false.
- **Cascading if statement** – a series of nested if statements.
- **Compound condition** – constructed when you need to ask multiple questions before determining an outcome.
- **Conditional AND operator** (or, more simply, an **AND operator**) – a symbol that you use to combine decisions so that two (or more) conditions must be true for an action to occur.
- **Conditional OR operator** (or, more simply, an **OR operator**) – a symbol that you use to combine decisions when any one condition can be true for an action to occur.
- **Dead or unreachable path** – a logical path that can never be traveled.
- **else clause** – holds the action or actions that execute only when the Boolean expression in the decision is false.
- **if-then decision structure** – action is taken only when the Boolean expression in the decision is true.
- **Logical NOT operator** – a symbol that reverses the meaning of a Boolean expression.
- **Nested decision (nested if)** – a decision “inside of” another decision.
- **OR decision** – contains two (or more) decisions; if at least one condition is met, the resulting action takes place.
- **Precedence** – determines order of evaluation for operands.
- **Range check** – compare a variable to a series of values that marks the limiting ends of ranges.
- **Relational comparison operators (relational operators or comparison operators)** – the symbols that express Boolean comparisons. Examples include =, >, <, >=, <=, and <>.
- **Short-circuit evaluation** – a logical feature in which expressions in each part of a larger expression are evaluated only as far as necessary to determine the final outcome.
- **then clause** – holds the action that results when the Boolean expression in the decision is true.
- **Trivial expression** – one that always evaluates to the same value.
- **Truth tables** – diagrams used in mathematics and logic to help describe the truth of an entire expression based on the truth of its parts.