COURSE TECHNOLOGY
CENGAGE Learning

Programming Logic and Design
Sixth Edition

Chapter 4
Making Decisions

Objectives

In this chapter, you 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

- **Boolean expression**
  - Value can be only true or false
  - Used in every selection structure

Evaluating Boolean Expressions to Make Comparisons (continued)

- **Dual-alternative** (or **binary**) selection structure
  - Provides an action for each of two possible outcomes

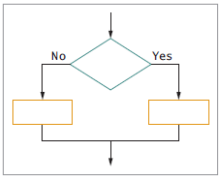


Figure 4-1 The dual-alternative selection structure

Evaluating Boolean Expressions to Make Comparisons (continued)

- **Single-alternative** (or **unary**) selection structure
  - Action is provided for only one outcome
  - **`if-then`**

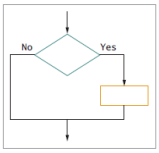
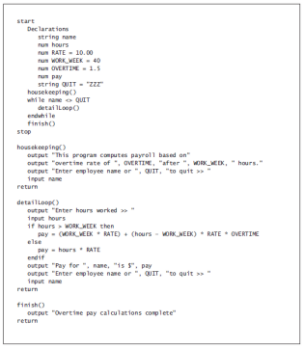


Figure 4-2 The single-alternative selection structure

Figure 4-3 Flowchart and pseudocode for overtime payroll program



```
start
   Declarations
      string name
      num hours
      num RATE = 10.00
      num WORK_WEEK = 40
      num OVERTIME = 1.5
      num pay
      string QUIT = "ZZZ"
   housekeeping()
   while name <> QUIT
      detailLoop()
   endwhile
   finish()
stop

housekeeping()
   output "This program computes payroll based on"
   output "overtime rate of ", OVERTIME, "after ", WORK_WEEK, " hours."
   output "Enter employee name or ", QUIT, "to quit >> "
   input name
return

detailLoop()
   output "Enter hours worked >> "
   input hours
   if hours > WORK_WEEK then
      pay = (WORK_WEEK * RATE) + (hours - WORK_WEEK) * RATE * OVERTIME
   else
      pay = hours * RATE
   endif
   output "Pay for ", name, "is $", pay
   output "Enter employee name or ", QUIT, "to quit >> "
   input name
return

finish()
   output "Overtime pay calculations complete"
return
```

Figure 4-3 Flowchart and pseudocode for overtime payroll program (continued)

Evaluating Boolean Expressions to Make Comparisons (continued)

- `if-then-else` decision
  - **`then` clause**
    - Holds the action or actions that execute when the tested condition in the decision is true
  - **`else` clause**
    - Executes only when the tested condition in the decision is false

Using the Relational Comparison Operators

- **Relational comparison operators**
  - Six types supported by all modern programming languages
  - Binary
  - Two values compared can be either variables or constants
- **Trivial expressions**
  - Will always evaluate to the same result
  - Example: `20 = 20?`

Operator	Name	Discussion
=	Equivalency operator	Evaluates as true when its operands are equivalent. Many languages use a double equal sign (==) to avoid confusion with the assignment operator.
>	Greater-than operator	Evaluates as true when the left operand is greater than the right operand.
<	Less-than operator	Evaluates as true when the left operand is less than the right operand.
>=	Greater-than or equal-to operator	Evaluates as true when the left operand is greater than or equivalent to the right operand.
<=	Less-than or equal-to operator	Evaluates as true when the left operand is less than or equivalent to the right operand.
<>	Not-equal-to operator	Evaluates as true when its operands are not equivalent. Some languages use an exclamation point followed by an equal sign to indicate not equal to (!=). Because the not-equal-to operator differs in the common programming languages, this book will most often spell out "is not equal to" in flowcharts and pseudocode.

Table 4-1 Relational comparisons

Using the Relational Comparison Operators (continued)

- Any logical situation can be expressed with only three types of comparisons: **=** , **>** , and **<**
  - Operators **>=** and **<=** are not necessary but make code more readable
- "Not equal" operator
  - Most confusing of comparisons
  - Most likely to be different in different languages

Using the Relational Comparison Operators (continued)

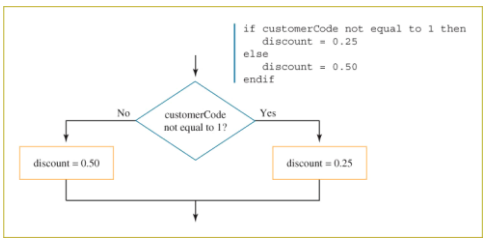


Figure 4-5 Using a negative comparison

Using the Relational Comparison Operators (continued)

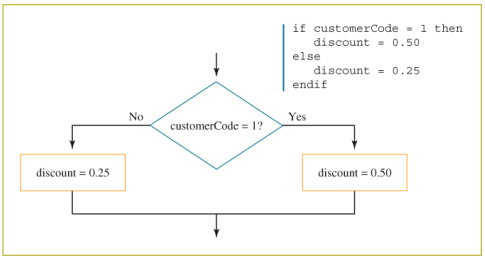


Figure 4-6 Using the positive equivalent of the negative comparison in Figure 4-5

Avoiding a Common Error with Relational Operators

- Common errors
  - Using the wrong operator
  - Missing the boundary or limit required for a selection

Understanding AND Logic

- **Compound condition**
  - Asks multiple questions before an outcome is determined
- **AND decision**
  - Requires that both of two tests evaluate to true
  - Requires a **nested decision** (**nested `if`**)
- Using **nested `if`** statements
  - Second selection structure is contained entirely within one side of first structure
  - `else` clause paired with last `if`

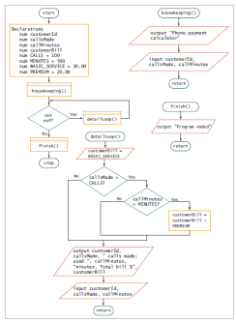


Figure 4-7 Flowchart and pseudocode for cell phone billing program


```
start
   Declarations
      num customerId
      num callsMade
      num callMinutes
      num customerBill
      num CALLS = 100
      num MINUTES = 500
      num BASIC_SERVICE = 30.00
      num PREMIUM = 20.00
   housekeeping()
   while not eof
      detailLoop()
   endwhile
   finish()
stop

housekeeping()
   output "Phone payment calculator"
   input customerId, callsMade, callMinutes
return

detailLoop()
   customerBill = BASIC_SERVICE
   if callsMade > CALLS then
      if callMinutes > MINUTES then
         customerBill = customerBill + PREMIUM
      endif
   endif
   output customerId, callsMade, " calls made; used ",
      callMinutes, " minutes. Total bill $", customerBill
   input customerId, callsMade, callMinutes
return

finish()
   output "Program ended"
return
```

Figure 4-7 Flowchart and pseudocode for cell phone billing program (continued)

Nesting AND Decisions for Efficiency

- When nesting decisions
  - Either selection can come first
- Performance time can be improved by asking questions in the proper order
- In an AND decision, first ask the question that is less likely to be true
  - Eliminates as many instances of the second decision as possible
  - Speeds up processing time

Using the AND Operator

- **Conditional AND operator**
  - Ask two or more questions in a single comparison
  - Each Boolean expression must be true for entire expression to evaluate to true
- **Truth tables**
  - Describe the truth of an entire expression based on the truth of its parts
- **Short-circuit evaluation**
  - Expression evaluated only as far as necessary to determine truth

Using the AND Operator (continued)

x	y	x AND y
True	True	True
True	False	False
False	True	False
False	False	False

Table 4-2 Truth table for the AND operator

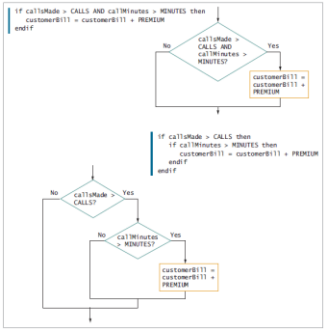


Figure 4-9 Using an AND operator and the logic behind it

Avoiding Common Errors in an AND Selection

- Second decision must be made entirely within the first decision
- In most programming languages, logical AND is a binary operator
  - Requires complete Boolean expression on both sides

Understanding OR Logic

- **OR decision**
  - Take action when one or the other of two conditions is true
- Example
  - "Are you free for dinner Friday or Saturday?"

Writing OR Decisions for Efficiency

- May ask either question first
  - Both produce the same output but vary widely in number of questions asked
- If first question is true, no need to ask second
- In an OR decision, first ask the question that is more likely to be true
  - Eliminates as many repetitions as possible of second decision

Using the OR Operator

- **Conditional OR operator**
  - Ask two or more questions in a single comparison
- Only one Boolean expression in an OR selection must be true to produce a result of true
- Question placed first will be asked first
  - Consider efficiency
- Computer can ask only one question at a time

Using the OR Operator(continued)

x	y	x OR y
True	True	True
True	False	True
False	True	True
False	False	False

Table 4-3 Truth table for the OR operator

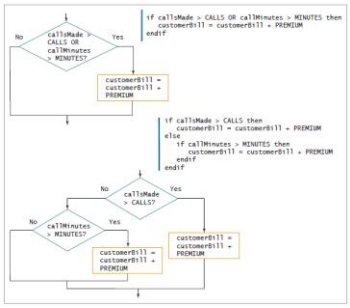


Figure 4-13 Using an OR operator and the logic behind it

Avoiding Common Errors in an OR Selection

- Second question must be self-contained structure with one entry and exit point
- Request for A and B in English often translates to a request for A or B logically
  - Example
    - "Give a bonus to anyone who has sold at least three items and to anyone who has sold $2000"
    - "Give a bonus to anyone who has sold at least three items or $2000"

Avoiding Common Errors in an OR Selection (continued)

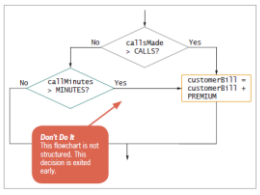


Figure 4-14 Unstructured flowchart for determining customer cell phone bill

Avoiding Common Errors in an OR Selection (continued)

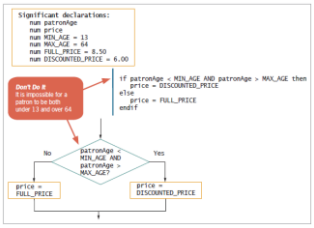


Figure 4-15 Incorrect logic that attempts to provide a discount for young and old movie patrons

Avoiding Common Errors in an OR Selection (continued)

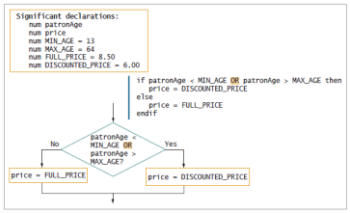


Figure 4-16 Correct logic that provides a discount for young and old movie patrons

Avoiding Common Errors in an OR Selection (continued)

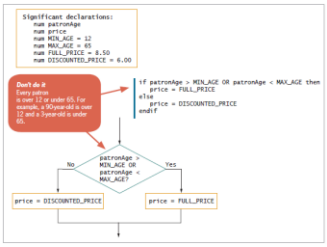


Figure 4-17 Incorrect logic that attempts to charge full price for patrons over 12 and under 65

Avoiding Common Errors in an OR Selection (continued)

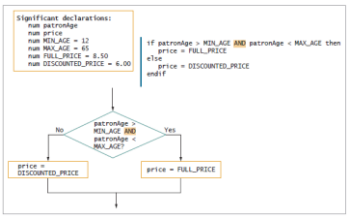


Figure 4-18 Correct logic that charges full price for patrons over 12 and under 65

Making Selections within Ranges

- Range check
  - Compare a variable to a series of values between limits
- Use the lowest or highest value in each range
- Adjust the question logic when using highest versus lowest values
- Should end points of the range be included?
  - Yes: use >= or <=
  - No: use < or >

Making Selections within Ranges (continued)

Items Ordered	Discount Rate (%)
0 to 10	0
11 to 24	10
25 to 50	15
51 or more	20

Figure 4-19 Discount rates based on items ordered

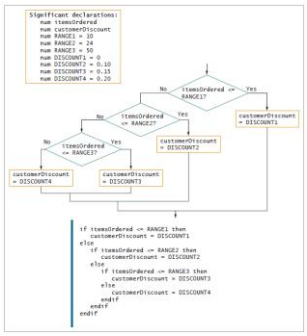


Figure 4-20 Flowchart and pseudocode of logic that selects correct discount based on items

Avoiding Common Errors When Using Range Checks

- Avoid **dead** or **unreachable paths**
  - Don't check for values that can never occur
  - Requires some prior knowledge of the data
- Never ask a question if there is only one possible outcome
- Avoid asking a question when the logic has already determined the outcome

Understanding Precedence When Combining AND and OR Selections

- Combine multiple AND and OR operators in an expression
- When multiple conditions must all be true, use multiple ANDs

```
if score1 >= 75 AND score2 >= 75 AND
score 3 >= 75 then
   classGrade = "Pass"
else
   classGrade = "Fail"
endif
```


Understanding Precedence When Combining AND and OR Selections (continued)

- When only one of multiple conditions must be true, use multiple ORs

```
if score1 >= 75 OR score2 >= 75 OR
score3 >= 75 then
   classGrade = "Pass"
else
   classGrade = "Fail"
endif
```


Understanding Precedence When Combining AND and OR Selections (continued)

- When AND and OR operators are combined in the same statement, AND operators are evaluated first

```
if age <= 12 OR age >= 65 AND rating = "G"
```

- Use parentheses to correct logic and force evaluations to occur in the order desired

```
if (age <= 12 OR age >= 65) AND rating = "G"
```


Understanding Precedence When Combining AND and OR Selections (continued)

- Mixing AND and OR operators makes logic more complicated
- Can avoid mixing AND and OR decisions by nesting if statements

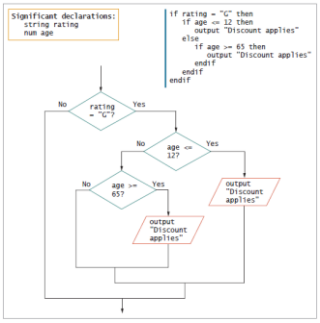


Figure 4-23 Nested decisions that determine movie patron discount

Summary

- Decisions involve evaluating Boolean expressions
- Use relational operators to compare values
- AND decision requires that both conditions be true to produce a true result
- In an AND decision, first ask the question that is less likely to be true
- OR decision requires that either of the conditions be true to produce a true result

Summary (continued)

- In an OR decision, first ask the question that is more likely to be true
- For a range check:
  - Make comparisons with the highest or lowest values in each range
  - Eliminate unnecessary or previously answered questions