



Programming Logic and Design

Ninth Edition

Chapter 4

Making Decisions



Objectives

In this chapter, you will learn about:

- The selection structure
- The relational comparison operators
- AND logic
- OR logic
- NOT logic
- Making selections within ranges
- Precedence when combining AND and OR operators
- The case structure



The Selection Structure

- Boolean expressions can be only true or false
- Every computer decision yields a true-or-false, yes-or-no, 1-or-0 result
- Used in every selection structure

The Selection Structure (continued -1)

- Dual-alternative (or binary) selection structure
 - Provides an action for each possible outcome

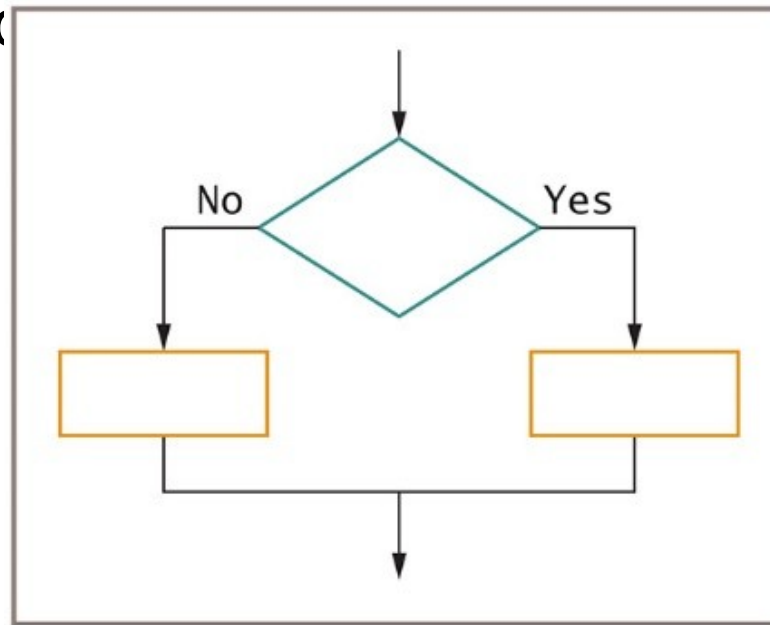


Figure 4-1 The dual-alternative selection structure

The Selection Structure (continued -2)

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

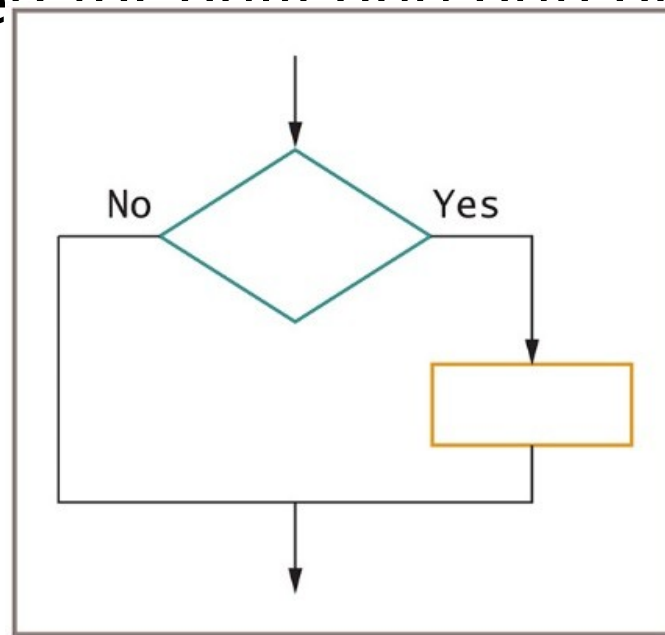


Figure 4-2 The single-alternative selection structure

The Selection Structure (continued -3)

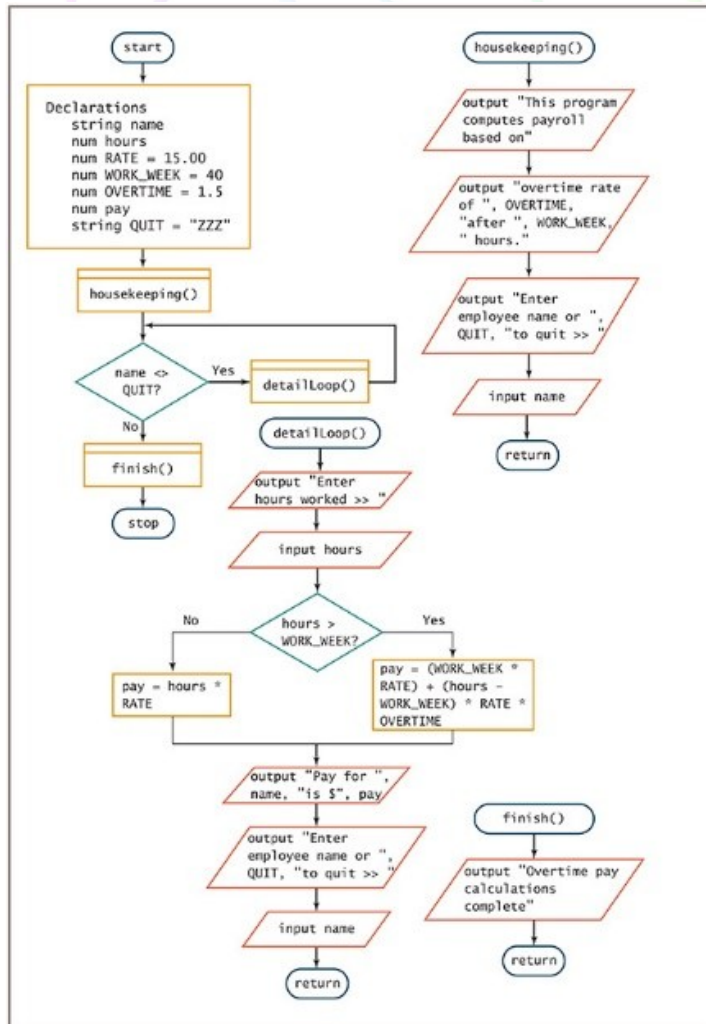


Figure 4-3 Flowchart and pseudocode for overtime payroll program (continues)

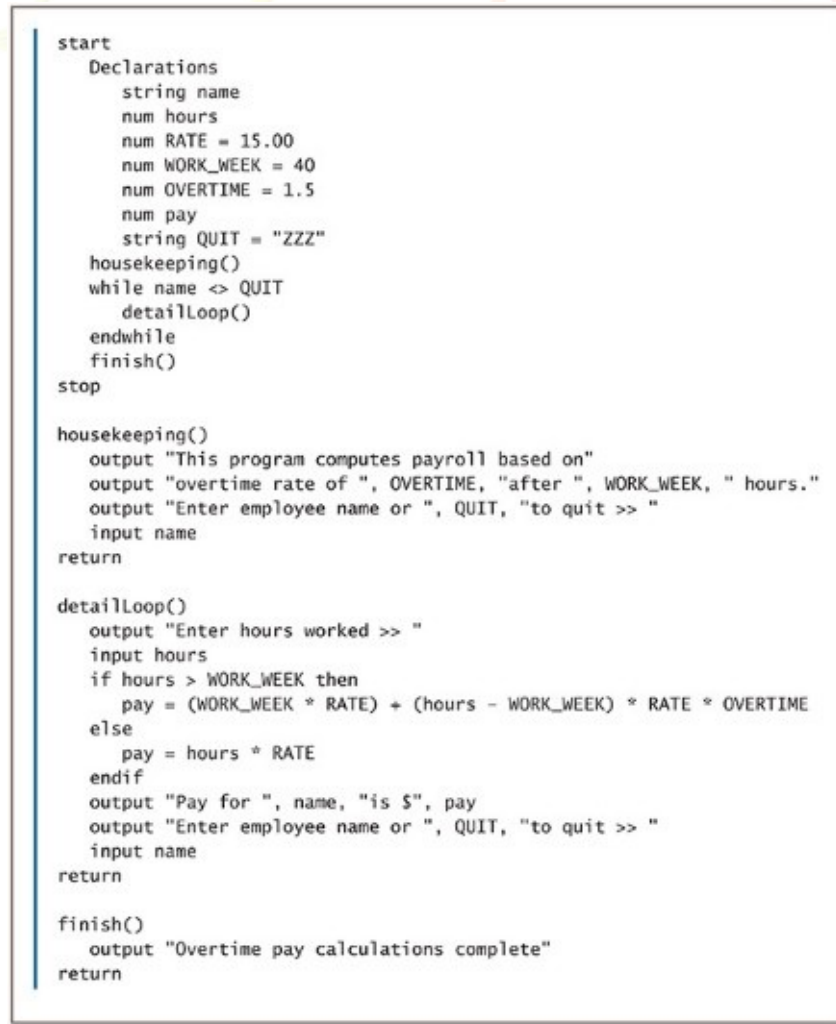


Figure 4-3 Flowchart and pseudocode for overtime payroll program

The Selection Structure (continued -4)

- `if-then-else` decision
 - **`if-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 Relational Comparison Operators

- **Relational comparison operators**
 - Six types supported by all modern programming languages
 - Two values compared can be either variables or constants but must be same data type
- **Trivial expressions**
 - Will always evaluate to the same result
 - Examples:
 - $20 = 20$? TRUE
 - $30 = 40$? FALSE

Using Relational Comparison Operators (continued -1)

- **Relational comparison operators**
 - Equivalency operator: **=**
 - Evaluates as true when its operands are equivalent
 - 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

Using Relational Comparison Operators (continued -2)

- **Relational comparison operators**
 - Greater-than-or-equal-to operator: \geq
 - Evaluates as true when the left operand is greater than or equivalent to the right operand
 - Less-than-or-equal-to operator: \leq
 - Evaluates as true when the left operand is less than or equivalent to the right operand
 - Not-equal-to operator: \neq
 - Evaluates as true when its operands are not equivalent

Using Relational Comparison Operators (continued -3)

- Any decision can be made with only three types of comparisons: =, >, and <
- “Not equal” operator <>
 - Involves thinking in double negatives
 - Best to restrict usage to “if without an else”—that is, only take action when some comparison is false

Using Relational Comparison Operators (continued -4)

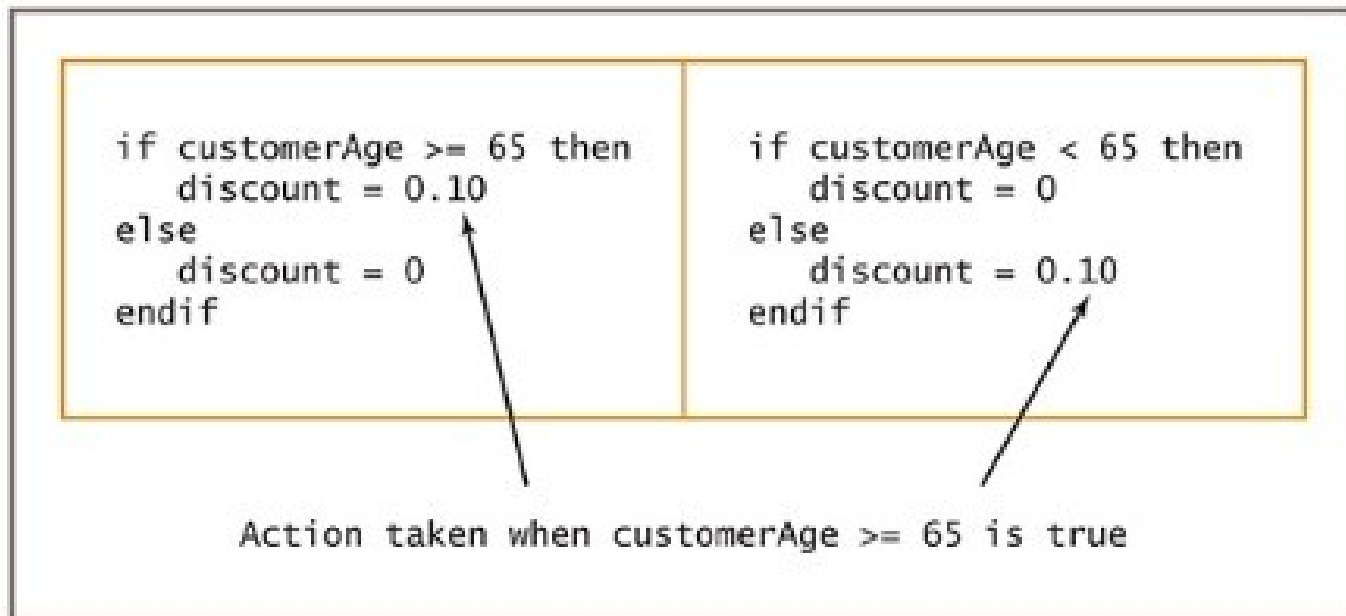


Figure 4-5 Identical logic expressed using >= and <

Using Relational Comparison Operators (continued -5)

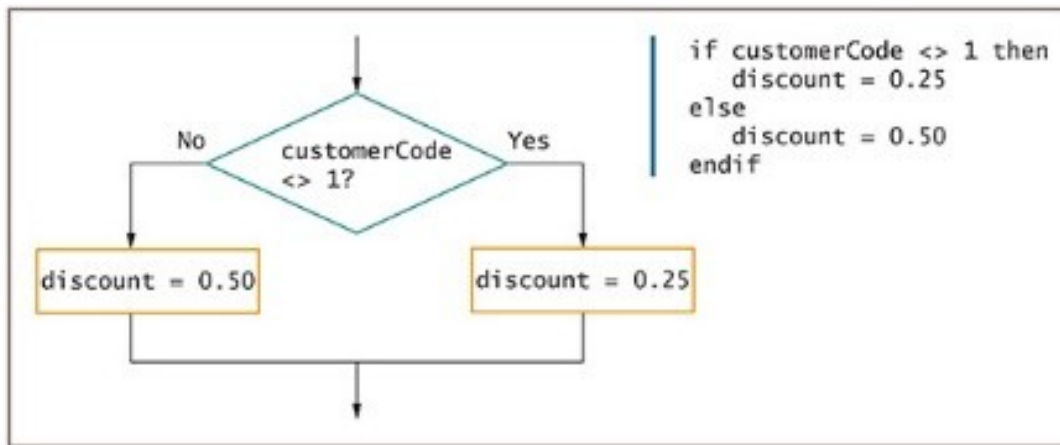


Figure 4-6 Using a negative comparison

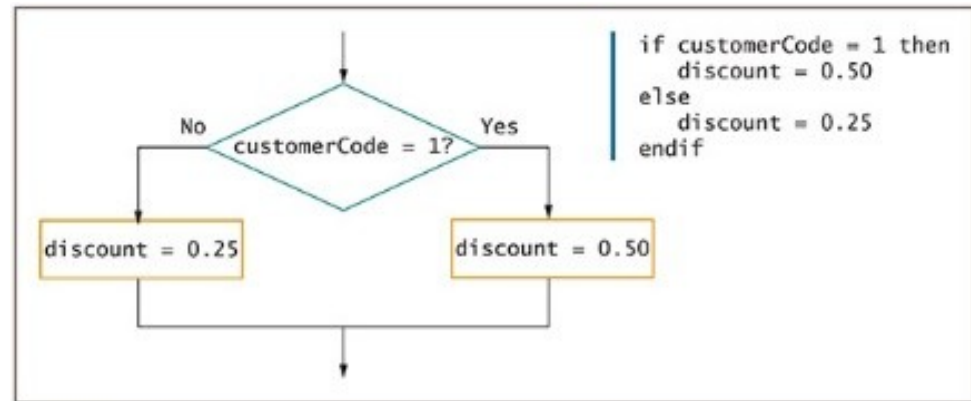


Figure 4-7 Using the positive equivalent of the negative comparison in Figure 4-6

Avoiding a Common Error with Relational Operators

- Common errors
 - Using the wrong operator
 - $BIG > small$
 - $small < BIG$
 - 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**) or a **cascading if statement**

Understanding AND Logic (continued)

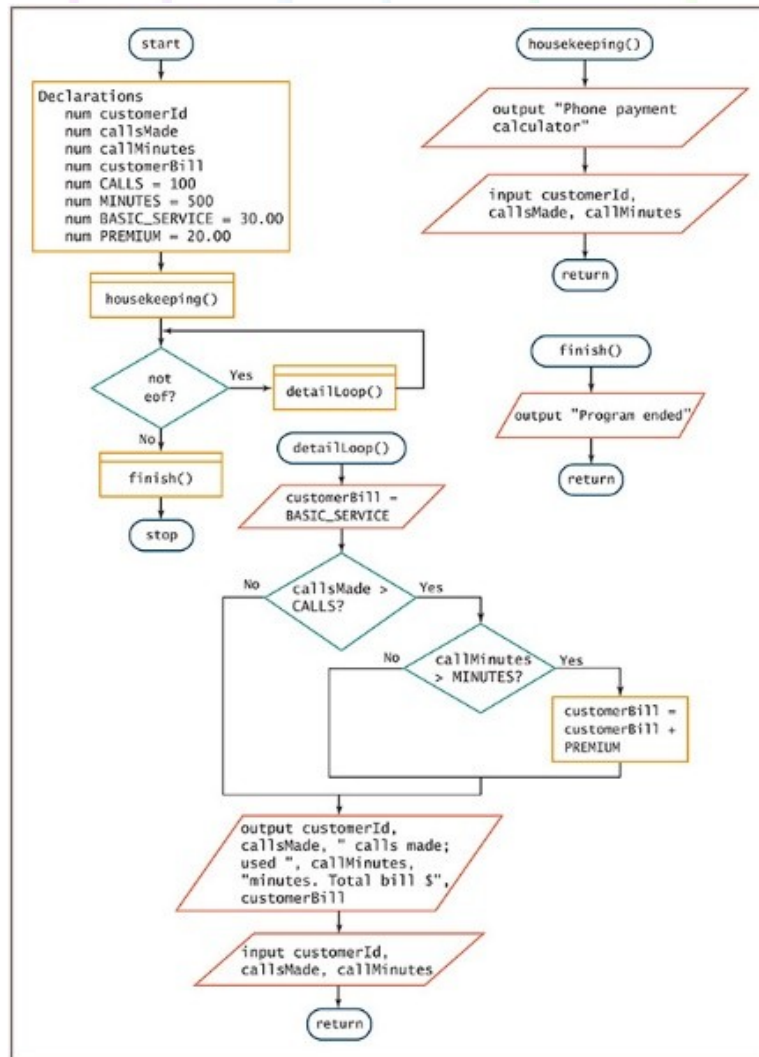


Figure 4-8 Flowchart and pseudocode for cell phone billing program (continues)

-1)
(continued)



Figure 4-8 Flowchart and pseudocode for cell phone billing program

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

Nesting AND Decisions for Efficiency

(continued -1)

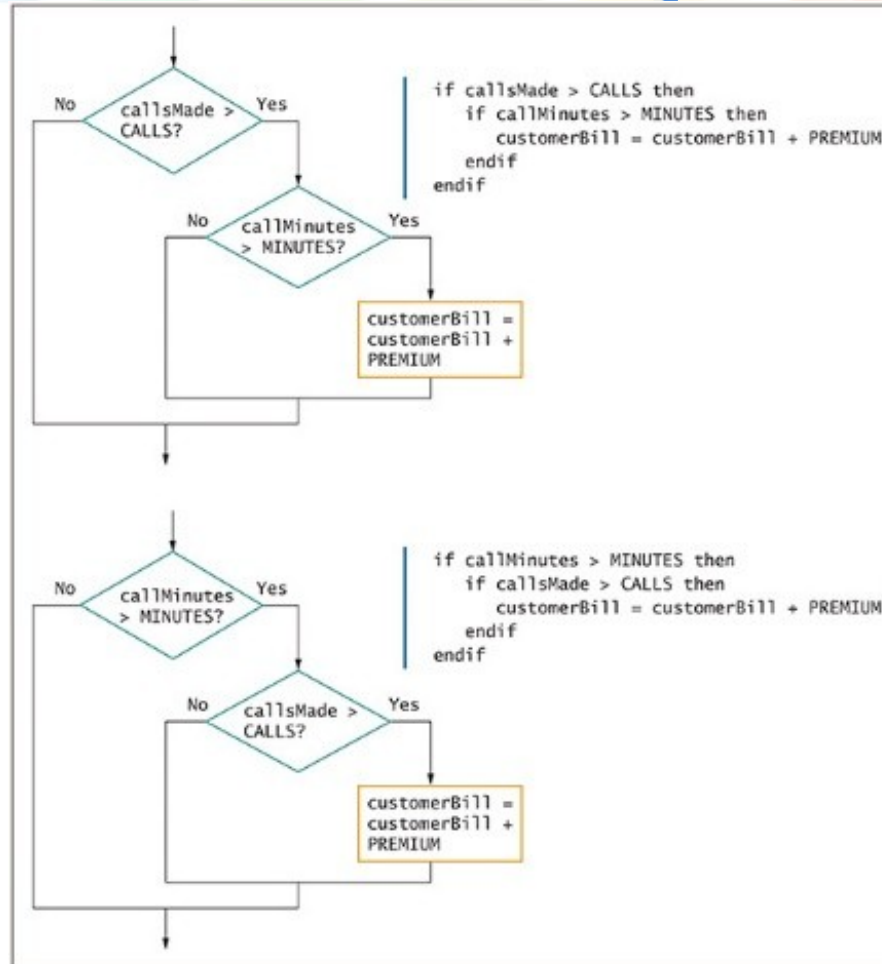


Figure 4-9 Two ways to produce cell phone bills using identical criteria



Using the AND Operator

- **Conditional AND operator** (simply an **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 -1)

- Combine several Boolean expressions that have true/false meaning into a single expression using the AND operator
- In an AND operation both expressions must be true in order for the entire expression itself to be true
- Example:
 - If I want to go to the movie AND I have enough money then I will go to the movie.

Using the AND Operator (continued -2)

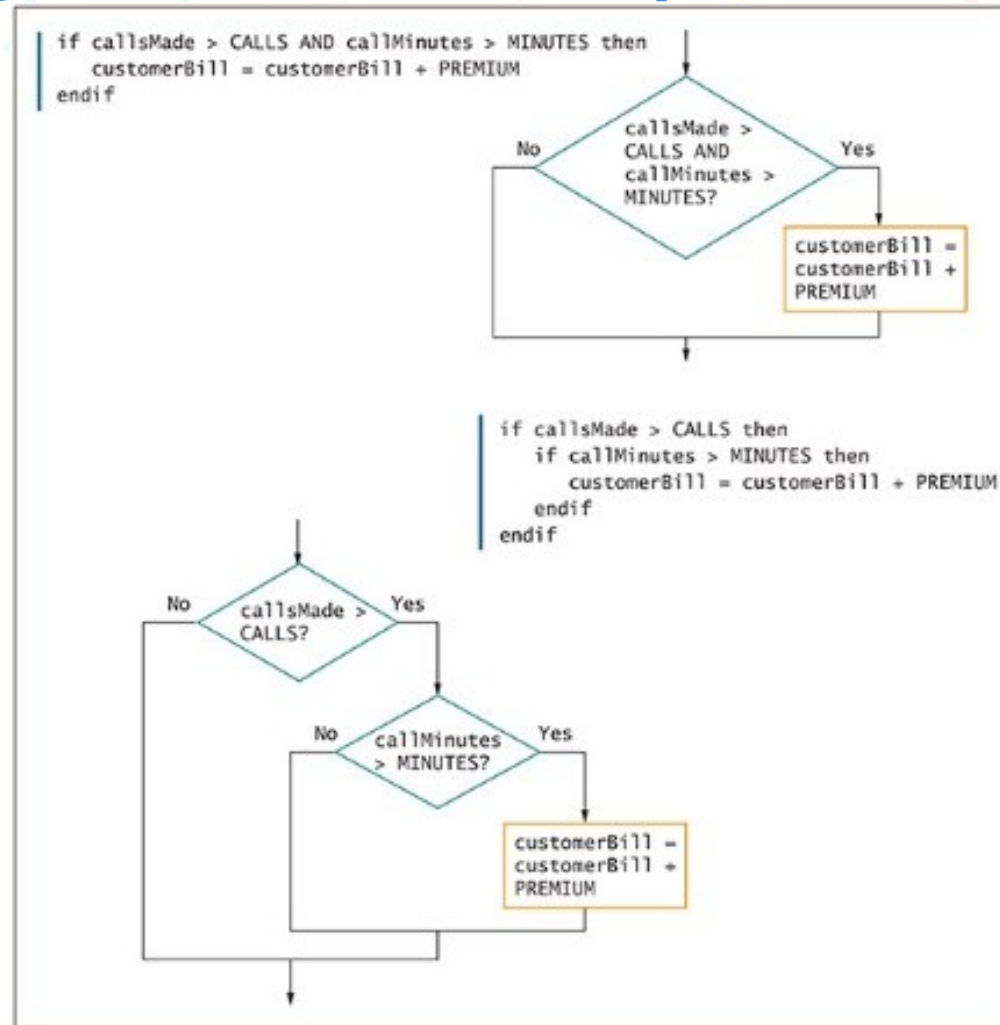


Figure 4-10 Using an AND operator and the logic behind it

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 a complete Boolean expression on both sides
- Expressions should not be trivial

Avoiding Common Errors in an AND Selection (continued -1)

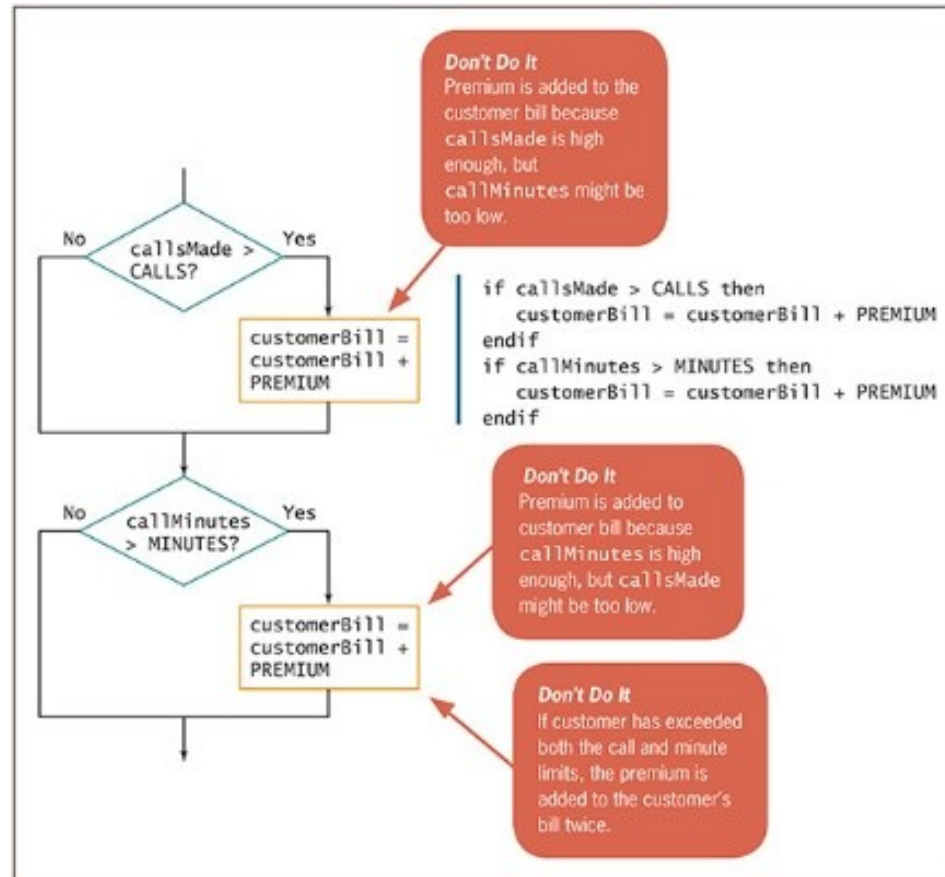


Figure 4-11 Incorrect logic to add a \$20 premium to the bills of cell phone customers who meet two criteria



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?”

Understanding OR Logic (continued -1)

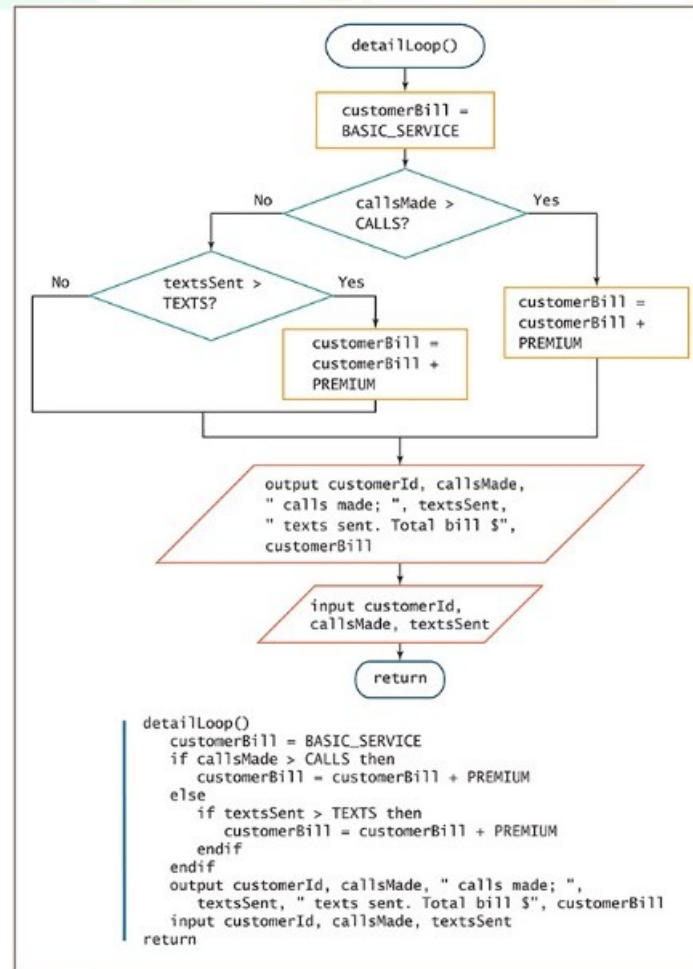


Figure 4-12 Flowchart and pseudocode for cell phone billing program in which a customer must meet one or both of two criteria to be billed a premium

Writing OR Selections 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

Writing OR Selections for Efficiency

(continued -1)

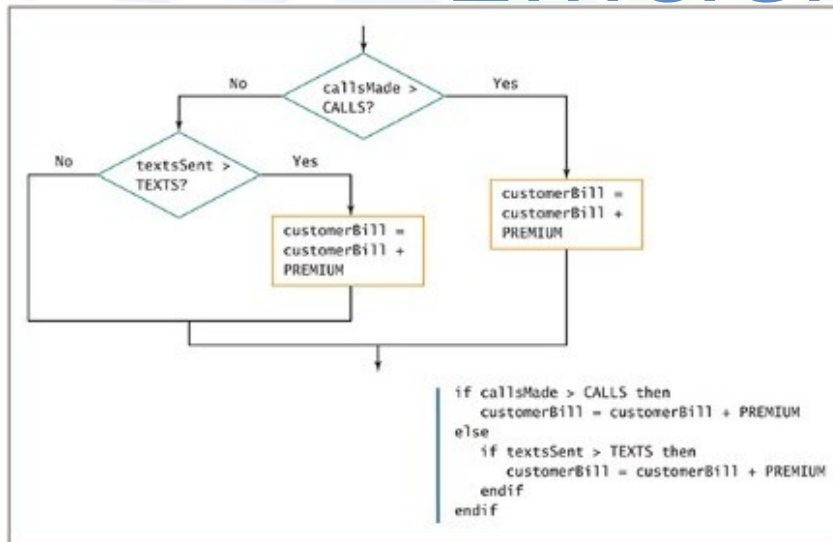


Figure 4-13 Two ways to assign a premium to bills of customers who meet one of two criteria (continues)

(continued)

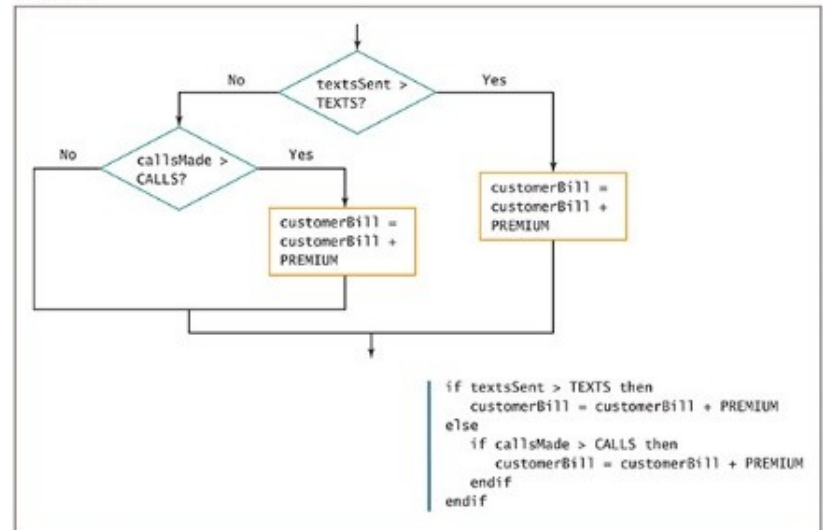


Figure 4-13 Two ways to assign a premium to bills of customers who meet one of two criteria



Using the OR Operator

- **Conditional OR operator** (simply an **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

Avoiding Common Errors in an OR Selection

- Second question must be a self-contained structure with one entry and exit point
- Request for *A and B* in English logically means a request for *A or B*
 - Examples
 - “Add \$20 to the bill of anyone who makes more than 100 calls and to anyone who has used more than 500 minutes”
 - “Add \$20 to the bill of anyone who has made more than 100 calls or has used more than 500 minutes”

Avoiding Common Errors in an OR Selection (continued -1)

- Make sure Boolean expression are complete
- Make sure that selections are structured
- Make sure to use OR selections when they are required
- Make sure that expression are not trivial

Avoiding Common Errors in an OR Selection

(continued -2)

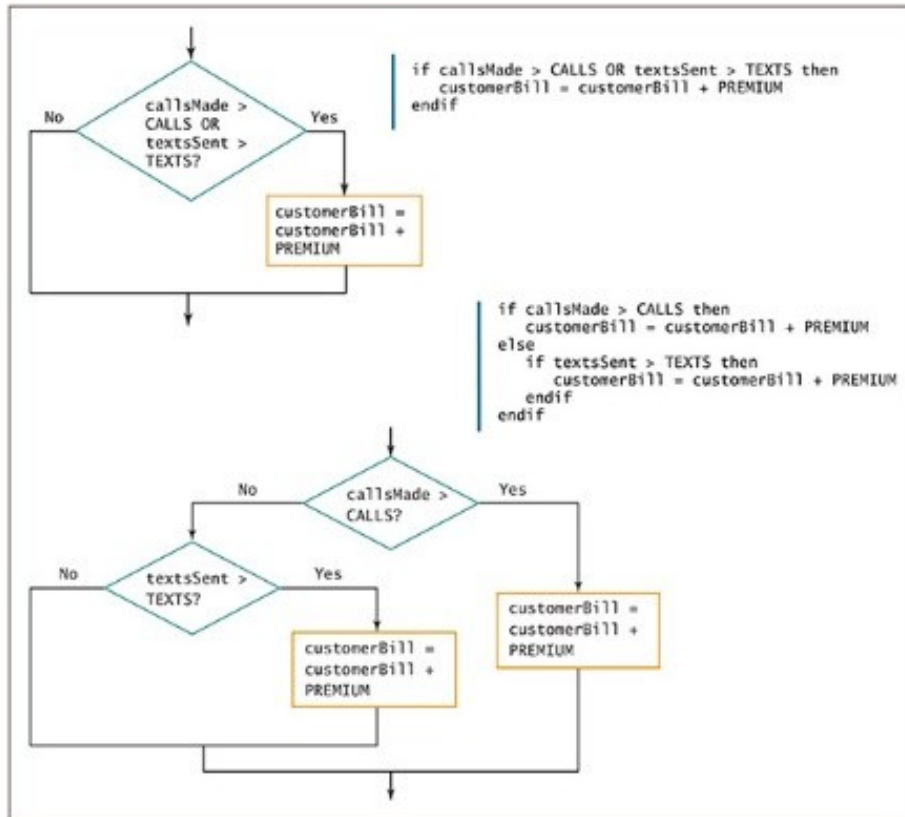


Figure 4-14 Using an OR operator and the logic behind it

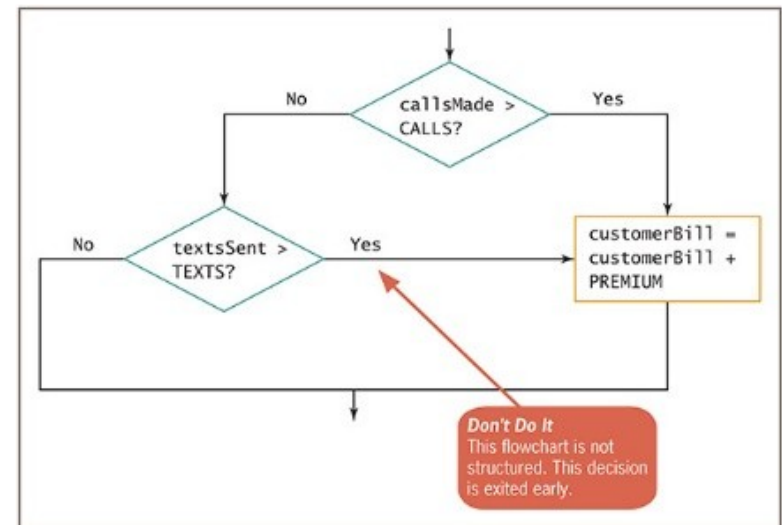


Figure 4-15 Unstructured flowchart for determining customer cell phone bill

Avoiding Common Errors in an OR Selection

(continued -3)

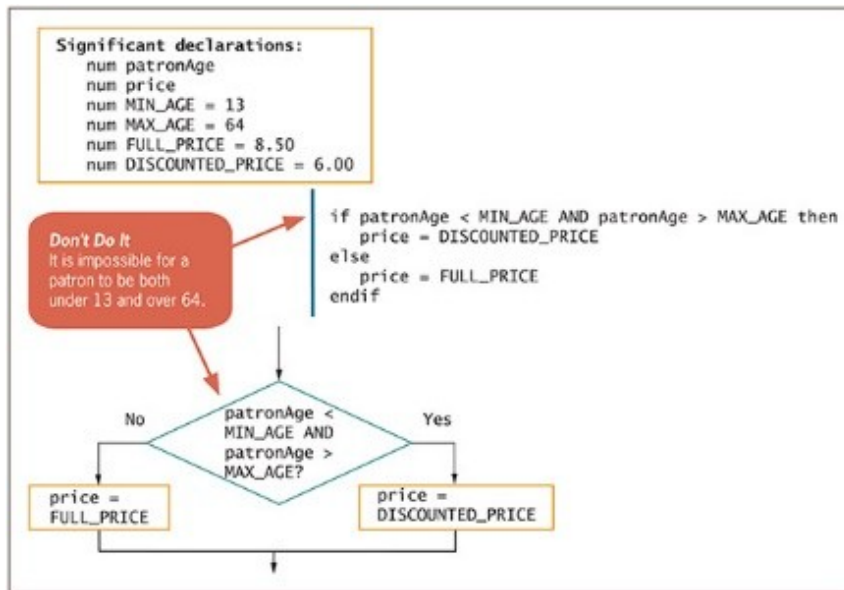


Figure 4-16 Incorrect logic that attempts to provide a discount for young and old movie patrons

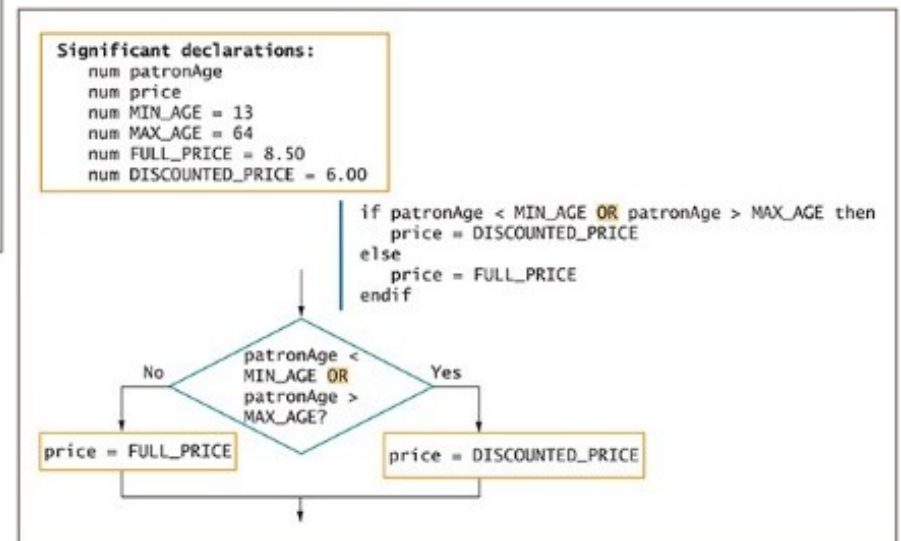


Figure 4-17 Correct logic that provides a discount for young and old movie patrons

Avoiding Common Errors in an OR Selection (continued -4)

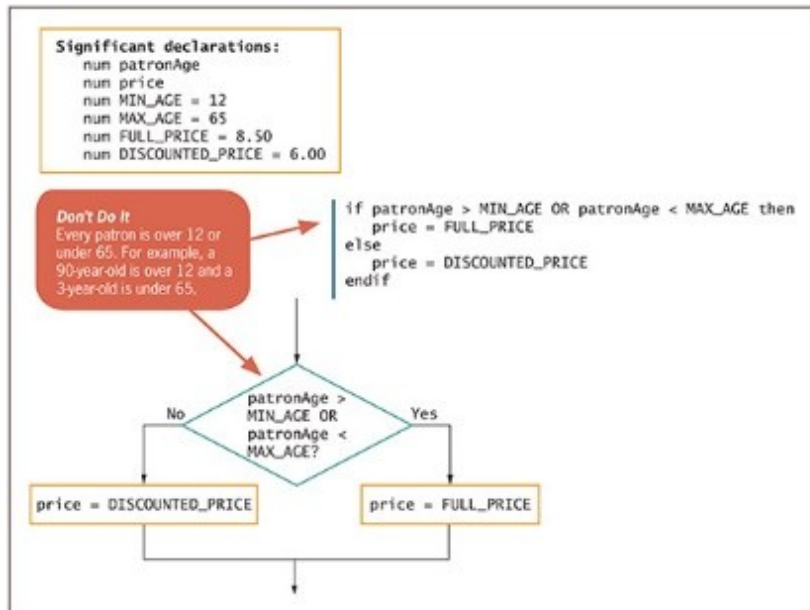


Figure 4-18 Incorrect logic that attempts to charge full price for patrons whose age is over 12 and under 65

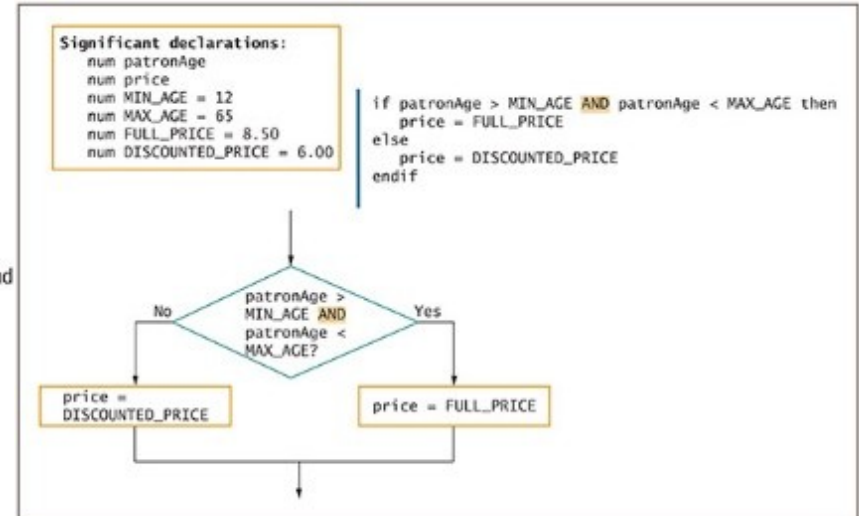


Figure 4-19 Correct logic that charges full price for patrons whose age is over 12 and under 65

Understanding NOT Logic

- The **NOT** operator
 - Reverses the meaning of a Boolean expression
 - if NOT (age < 18) then
 - output "Can register to vote"
 - Endif
 - If NOT true, it is false
 - If NOT false, it is true
 - Is a unary operator
 - Takes only one operator

Avoiding a Common Error in a NOT Expression

- Be careful not to create trivial expressions
- Incorrect code:
if NOT (employeeDept = 1) OR NOT (employeeDept = 2) then
 output "Employee is not in Department 1 or 2"
Endif
- Correct code:
if NOT (employeeDept = 1 OR employeeDept = 2) then
 output "Employee is not in Department 1 or 2"
endif

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 \geq or \leq
 - No: use $<$ or $>$

Making Selections Within Ranges

(continued -1)

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

Figure 4-20 Discount rates based on items ordered

Making Selections Within Ranges (continued -2)

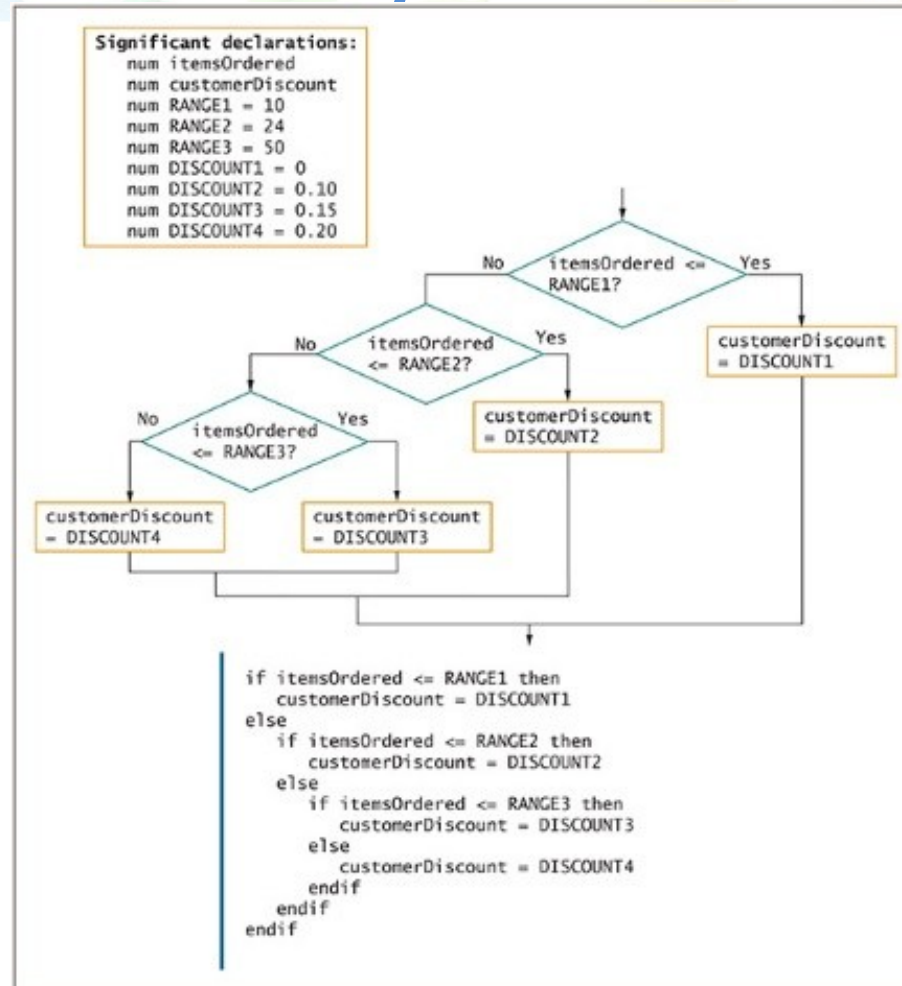
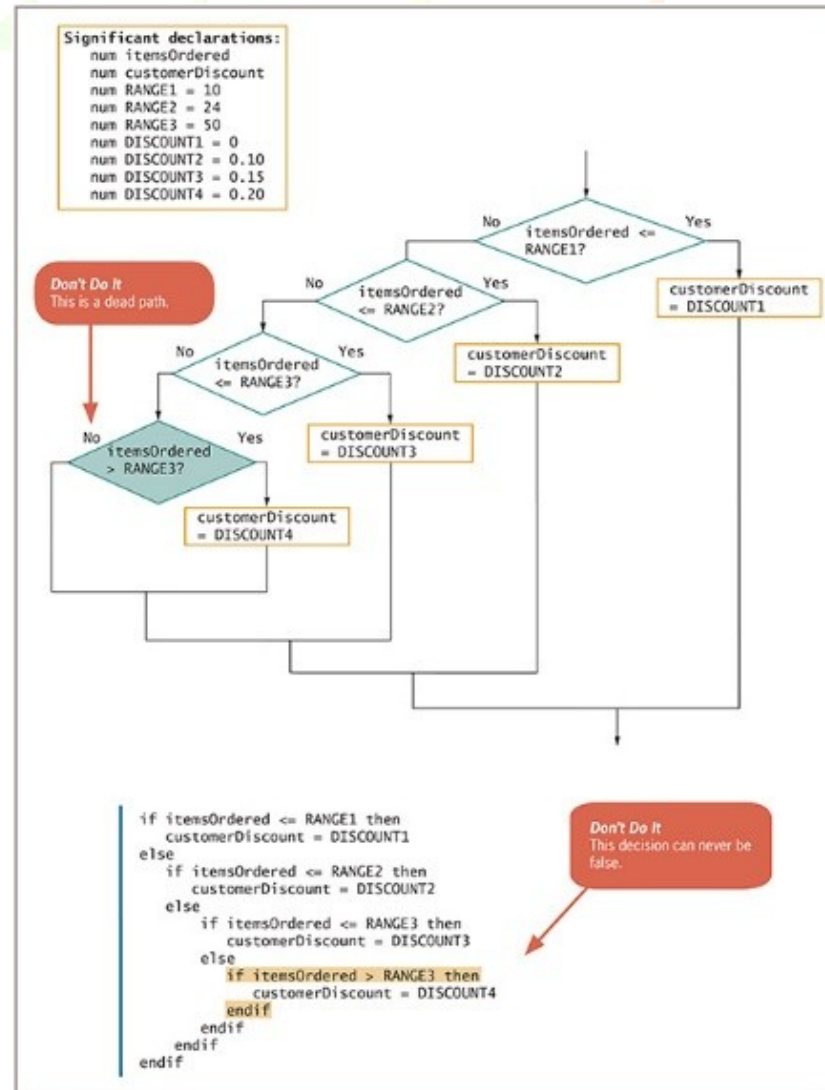


Figure 4-21 Flowchart and pseudocode of logic that selects correct discount based on items ordered

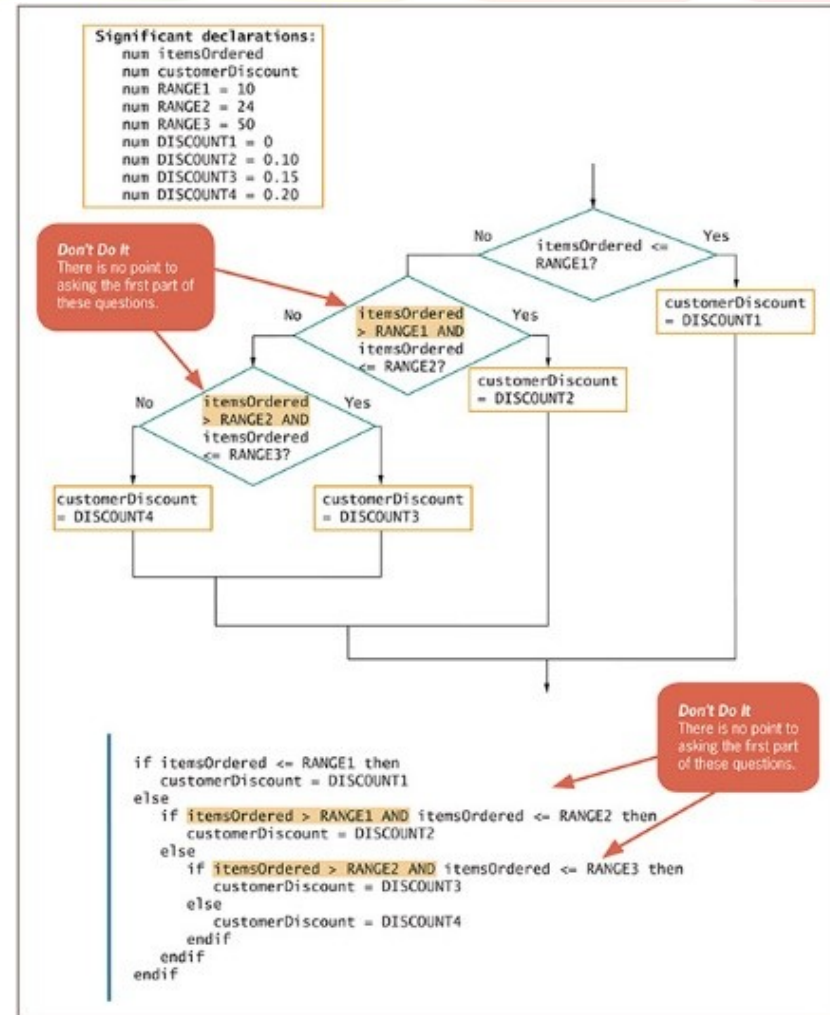
Avoiding Common Errors When Using Range Checks

- Avoid a **dead** or **unreachable path**
 - 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 testing the same range limit multiple times

Eliminating Dead Paths



Avoid Testing the Same Range Limit Multiple Times



Understanding Precedence When Combining AND and OR Operators

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

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

Understanding Precedence When Combining AND and OR Operators

(continued -1)

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

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

Understanding Precedence When Combining AND and OR Operators

(continued -2)

- 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 Operators

(continued -3)

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

Understanding Precedence When Combining AND and OR Operators

(continued -4)

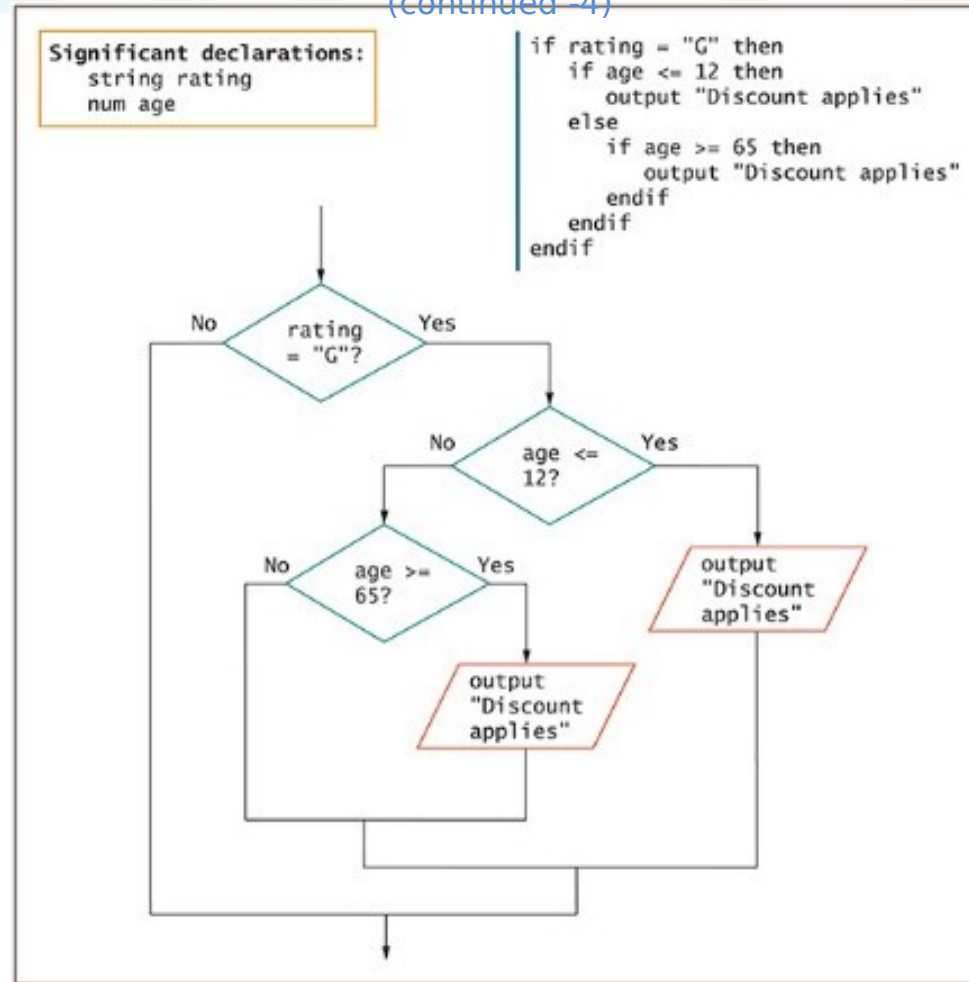


Figure 4-24 Nested decisions that determine movie patron discount



Understanding the case Structure

- Case structure – specialized selection structure
- Use when there are several distinct possible values for a single variable
- And each value requires a different subsequent action

Understanding the case Structure

(continued -1)

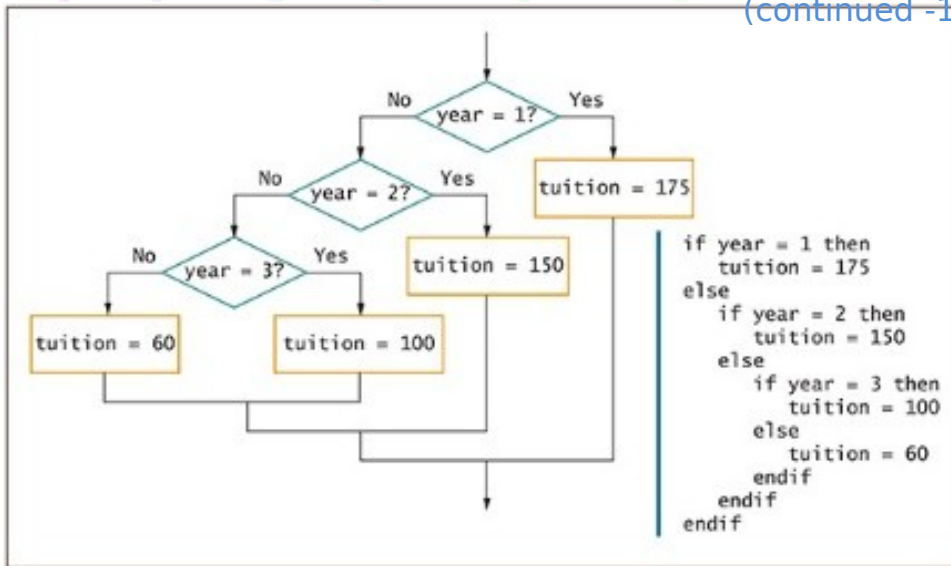


Figure 4-25 Flowchart and pseudocode of tuition decisions

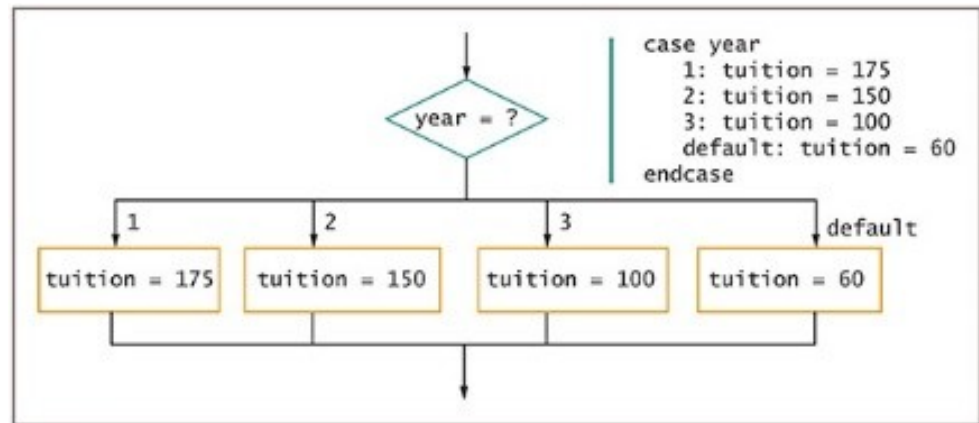


Figure 4-26 Flowchart and pseudocode of case structure that determines tuition



Summary

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



Summary

(continued -1)

- For a range check:
 - Make comparisons with the highest or lowest values in each range
 - Eliminate unnecessary or previously answered questions
- The AND operator takes precedence over the OR operator
- Case structure is a specialized selection structure that can be used when there are several distinct possible values for a single variable, and each value requires a different subsequent action