Programming Logic and Design

Seventh Edition

Chapter 4

Making Decisions

Objectives

In this chapter, you will learn about:

- Boolean expressions and the selection structure
- The relational comparison operators
- AND logic
- OR logic
- Making selections within ranges
- Precedence when combining `AND` and `OR` operators

Boolean Expressions and 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

Boolean Expressions and the Selection Structure (cont.)

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

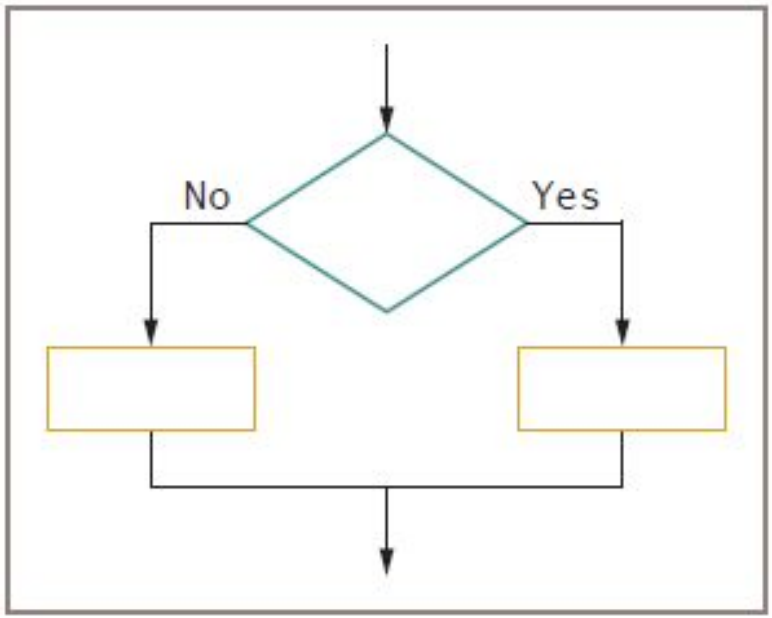


Figure 4-1 The dual-alternative selection structure

Boolean Expressions and the Selection Structure (cont.)

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

General Syntax of "if statement":

```
    if (condition)
{
    //Block of C statements here
    //These statements will only execute if the condition is true
}
```

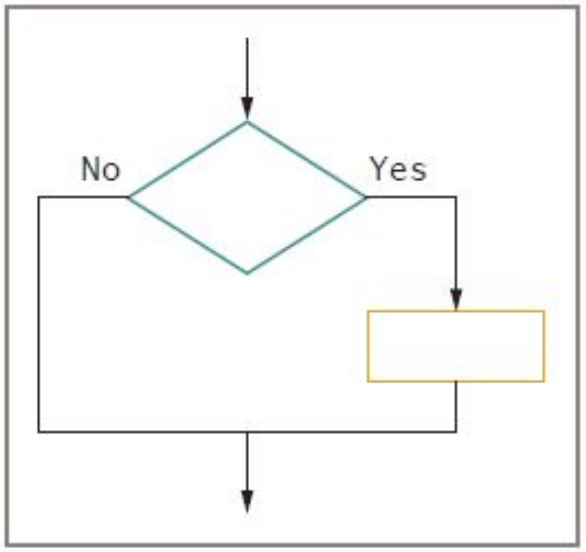


Figure 4-2 The single-alternative selection structure

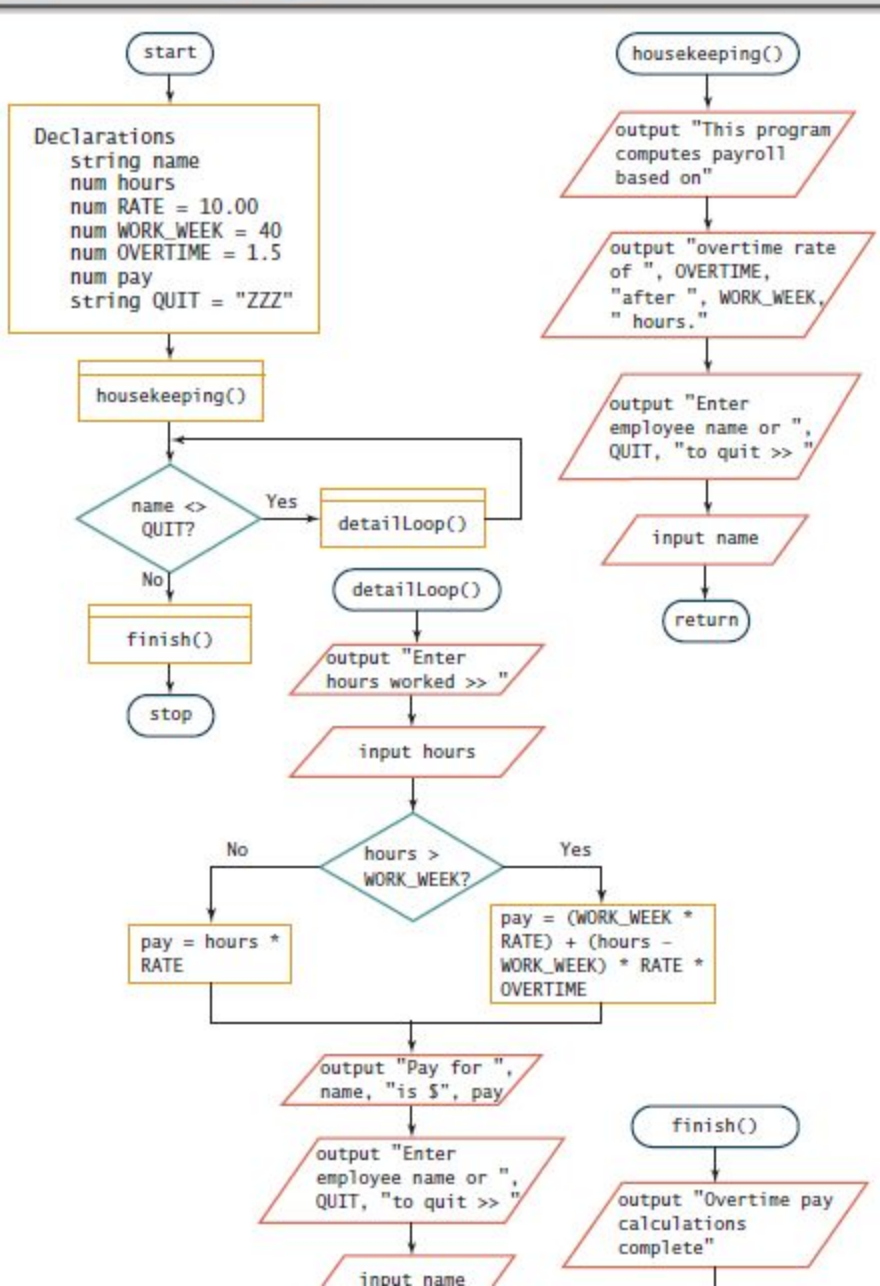

start
Declarations
string name
num hours
num RATE = 10.00
num WORK_WEEK = 40
num OVERTIME = 1.5
num pay
string QUIT = "ZZZ"
housekeeping()
name <> QUIT?
Yes
No
detailLoop()
finish()
stop
housekeeping()
output "This program computes payroll based on"
output "overtime rate of ", OVERTIME, "after ", WORK_WEEK, " hours."
output "Enter employee name or ", QUIT, "to quit >> "
input name
return
detailLoop()
output "Enter hours worked >> "
input hours
hours > WORK_WEEK?
No
Yes
pay = hours * RATE
pay = (WORK_WEEK * RATE) + (hours - WORK_WEEK) * RATE * OVERTIME
output "Pay for ", name, "is $", pay
output "Enter employee name or ", QUIT, "to quit >> "
input name
finish()
output "Overtime pay calculations complete"

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

```
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
```

Boolean Expressions and the Selection Structure (cont.)

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

If Statement Example:

```
#include <stdio.h>
int main ()
{
  /* local variable definition */
   int a = 10;
  /* check the Boolean condition using if statement */
   if( a < 20 ) {
     /* if condition is true then print the following */
      printf("a is less than 20\n" );
   }
   printf("value of a is : %d\n", a);
   return 0;
}
```

a is less than 20

value of a is : 10

Using Relational Comparison Operators

- **Relational comparison operators**
  - Six types supported by all modern programming languages
  - Two values compared can be either variables or constants
- **Trivial expressions**
  - Will always evaluate to the same result
  - Examples:
    - true for `20 = 20?`
    - false for `30 = 40?`

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 (!=).

Table 4-1 Relational comparison operators

Using Relational Comparison Operators (cont.)

- Any decision can be made with only three types of comparisons: =, >, and <
  - The >= and <= operators are not necessary but make code more readable
- "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 (cont.)

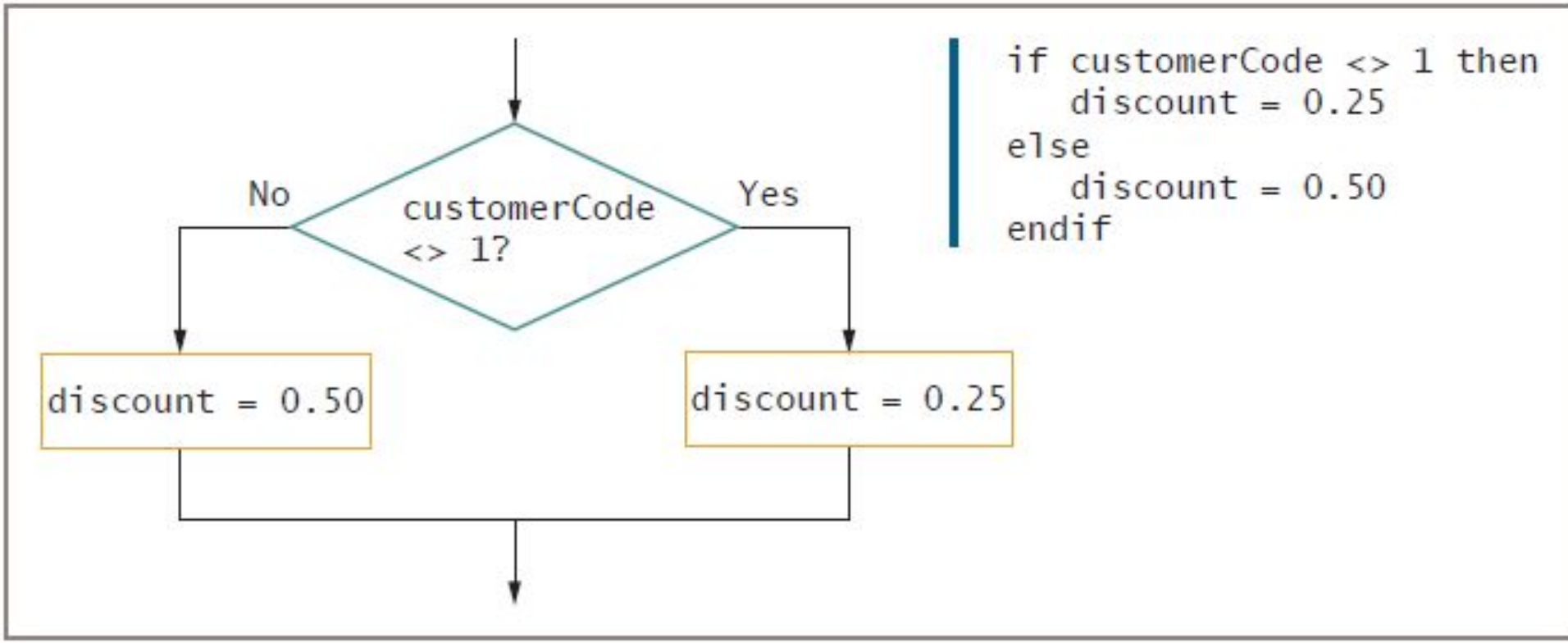


Figure 4-5 Using a negative comparison

Using Relational Comparison Operators (continued)

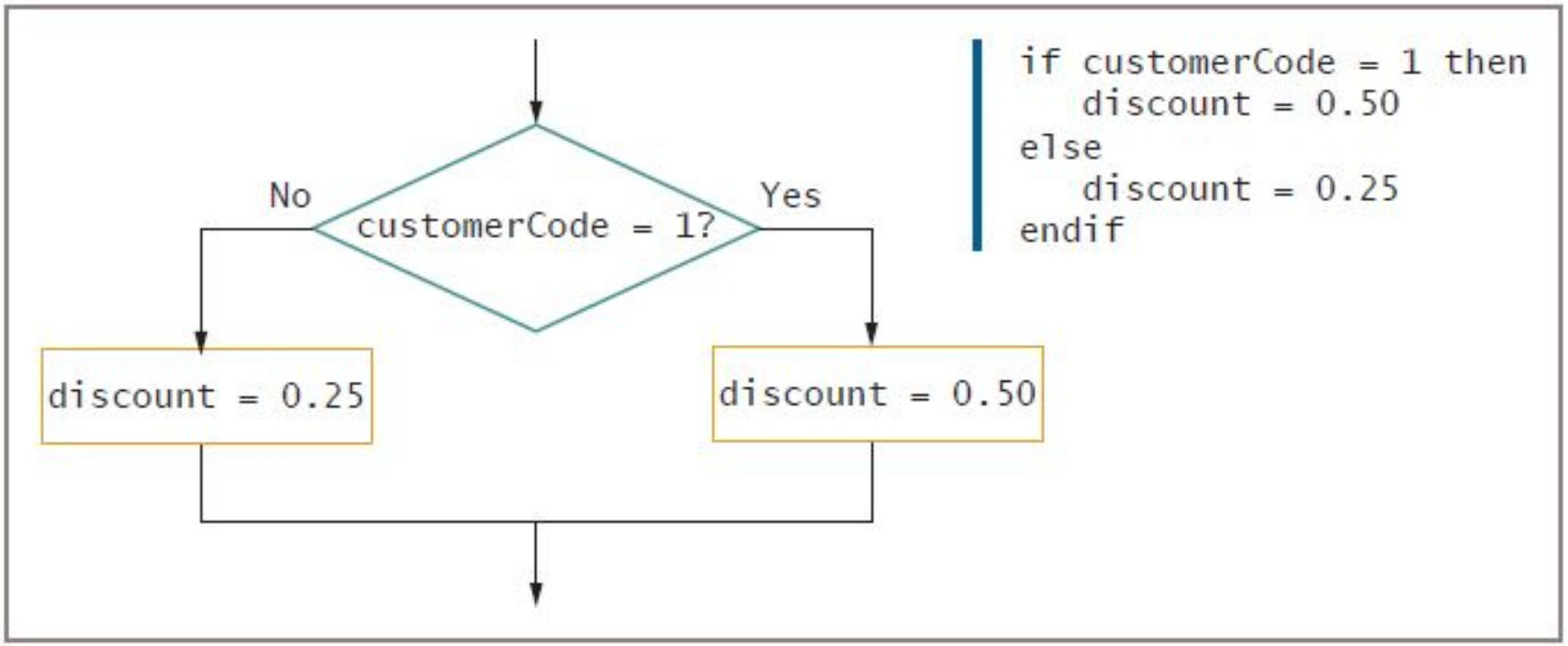


Figure 4-6 Using the positive equivalent of the negative comparison in Figure 4-5

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

For example: you work for a cell phone company that charges customers as follow:

- The basic monthly service bill is $30.
- An additional $20 is billed to customers who make more than 100 calls that last for a total more than 500 minutes.
- The logic needed for this billing program includes an **AND** decision, decision that test a condition with two parts both of two tests evaluate to true.

- **AND decision**
  - Requires that both of two tests evaluate to true
  - Requires a **nested decision** (**nested `if`**) or a **cascading `if` statement**

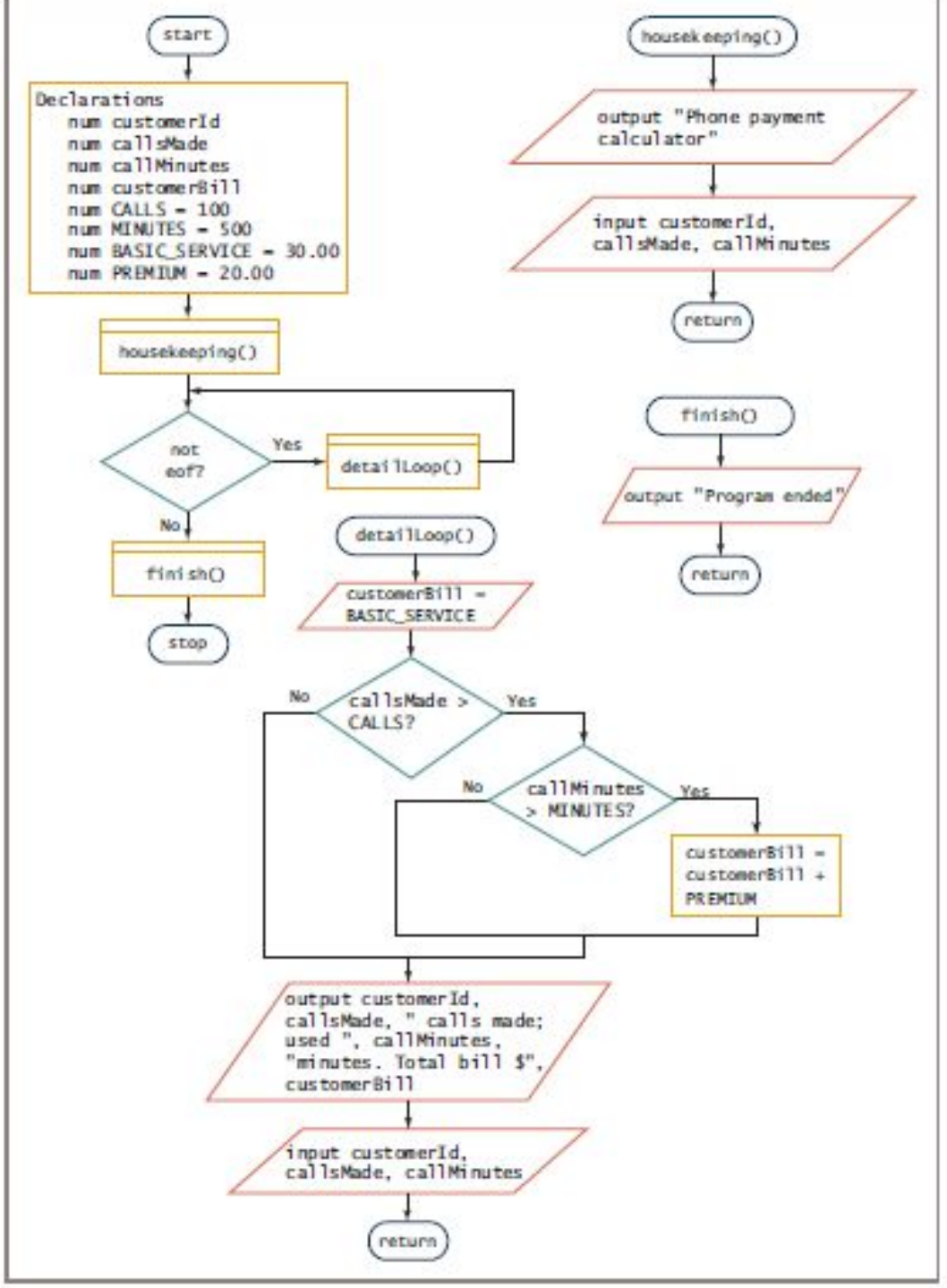


Figure 4-7 Flowchart and pseudocode for cell phone billing program

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

```
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
```

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

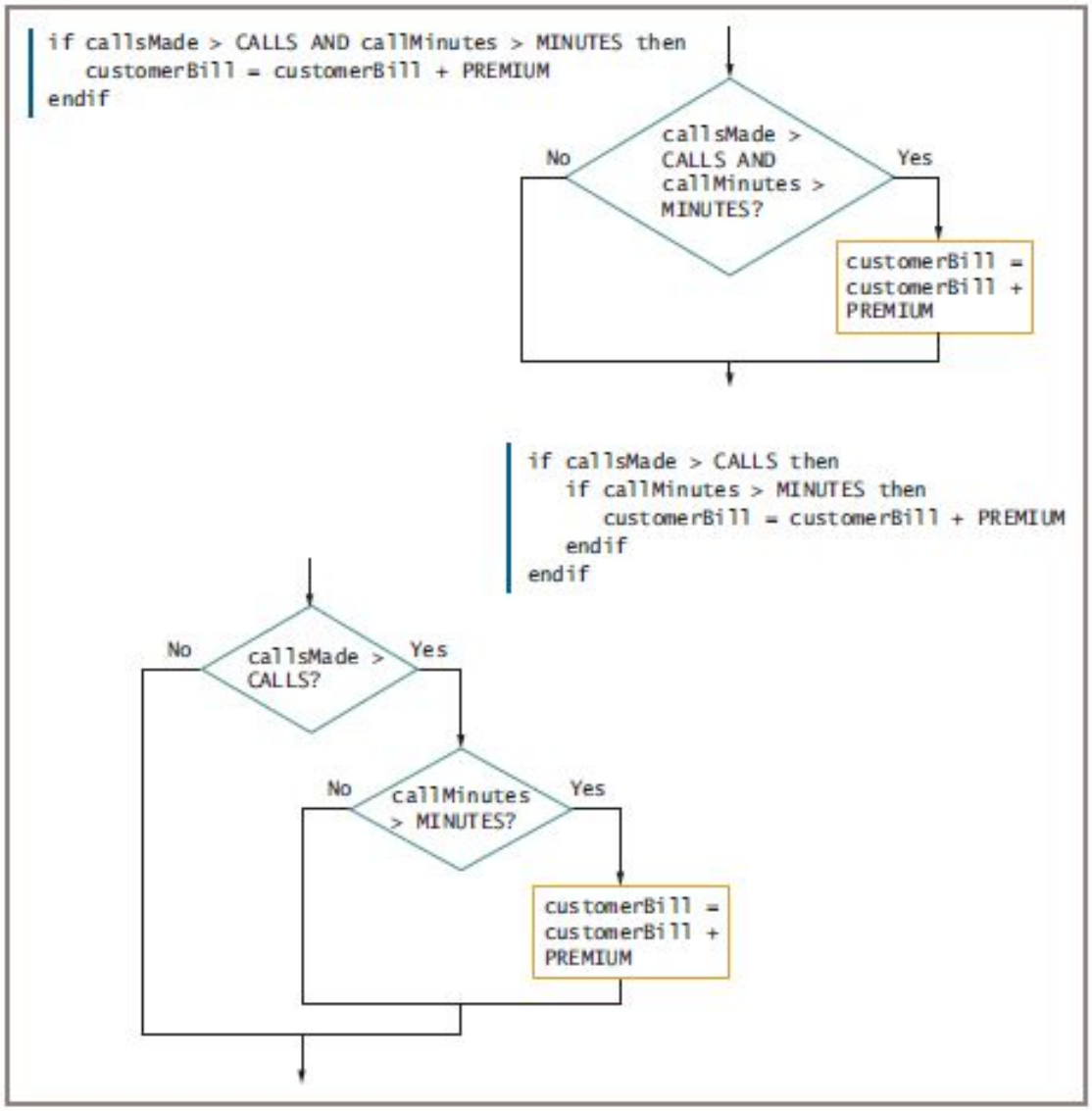


Figure 4-9 Using an AND operator and the logic behind it

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 a 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
  - Eliminate as many extra decisions as possible

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

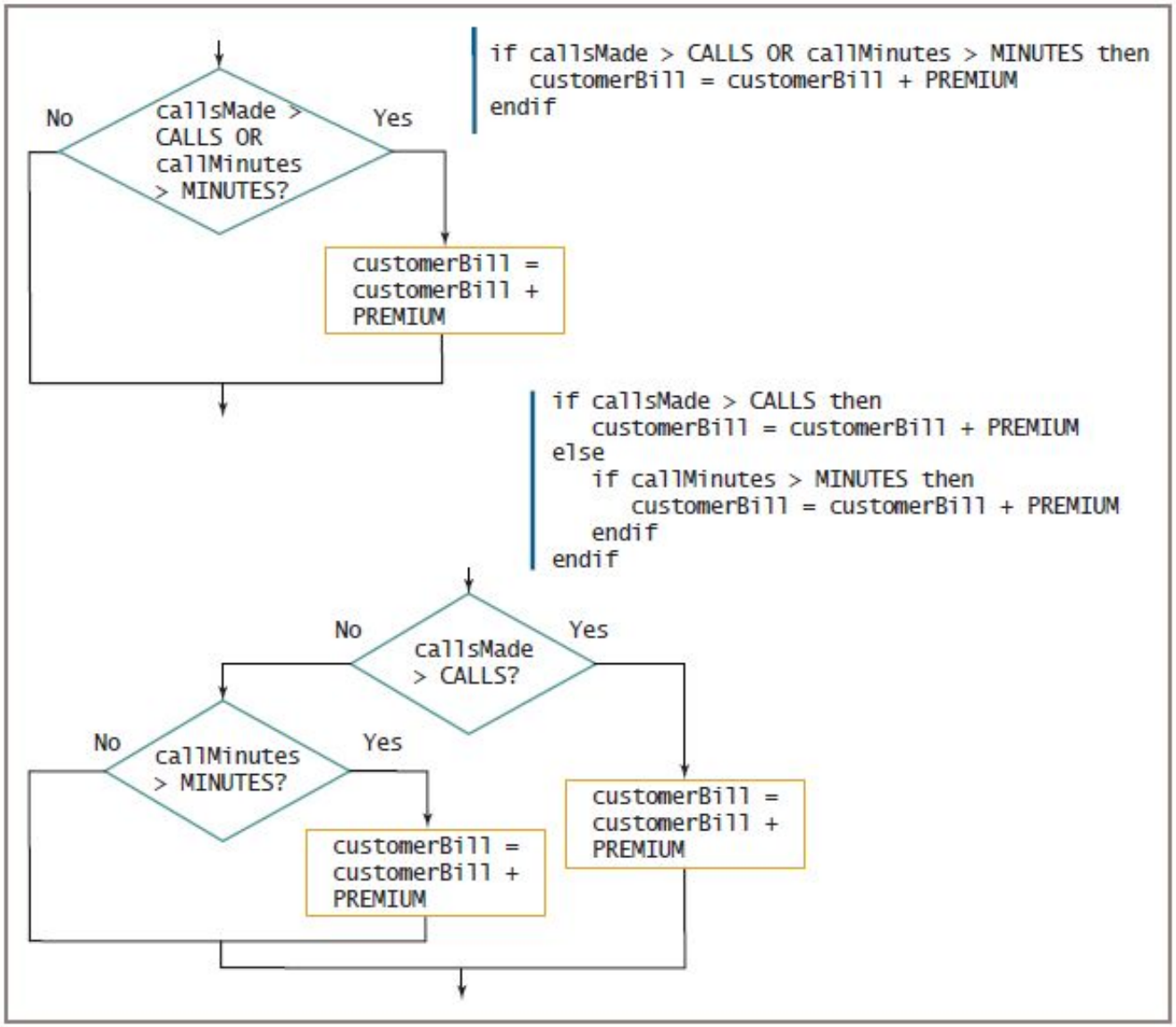


Figure 4-13 Using an OR operator and the logic behind it

Avoiding Common Errors in an OR Selection

- Second question must be 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
  - Example
    - "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 (cont.)

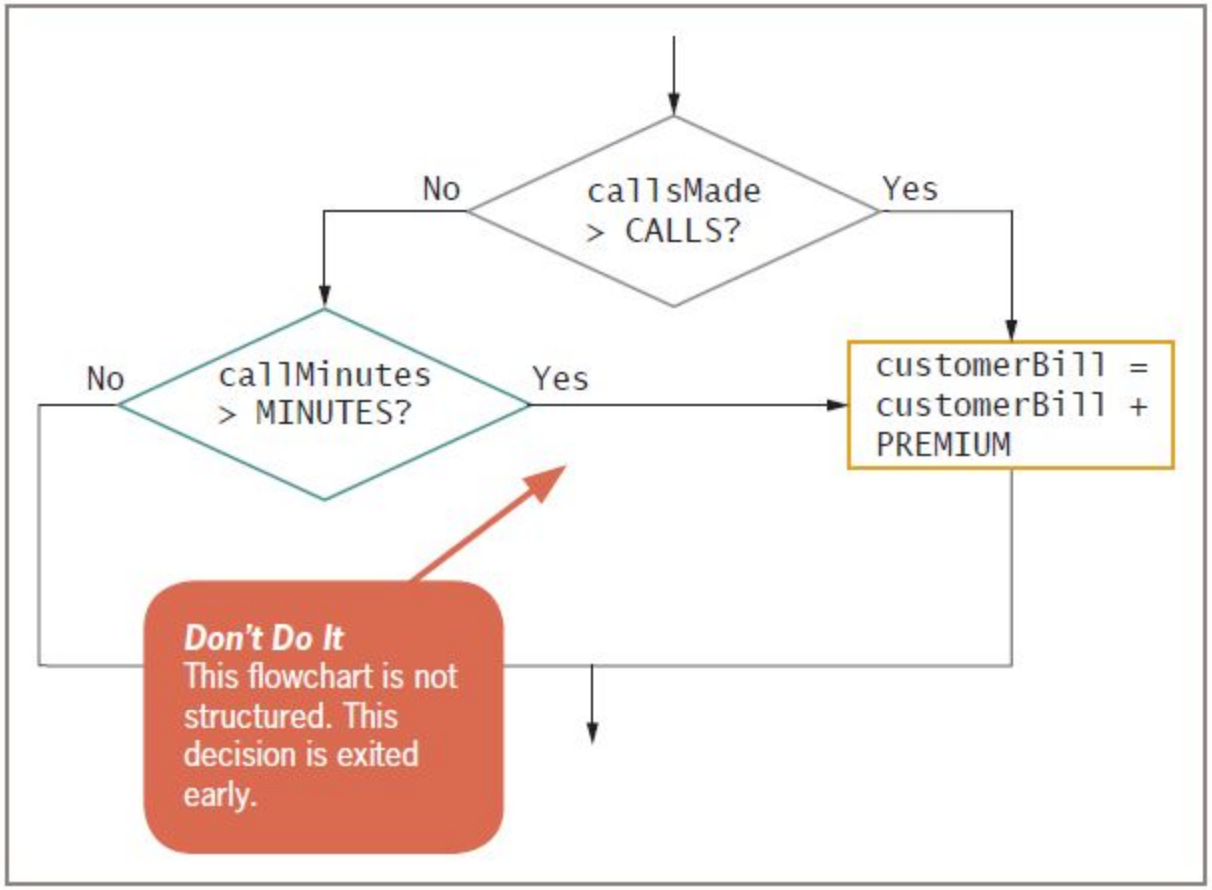


Figure 4-14 Unstructured flowchart for determining customer cell phone bill

Avoiding Common Errors in an OR Selection (continued)

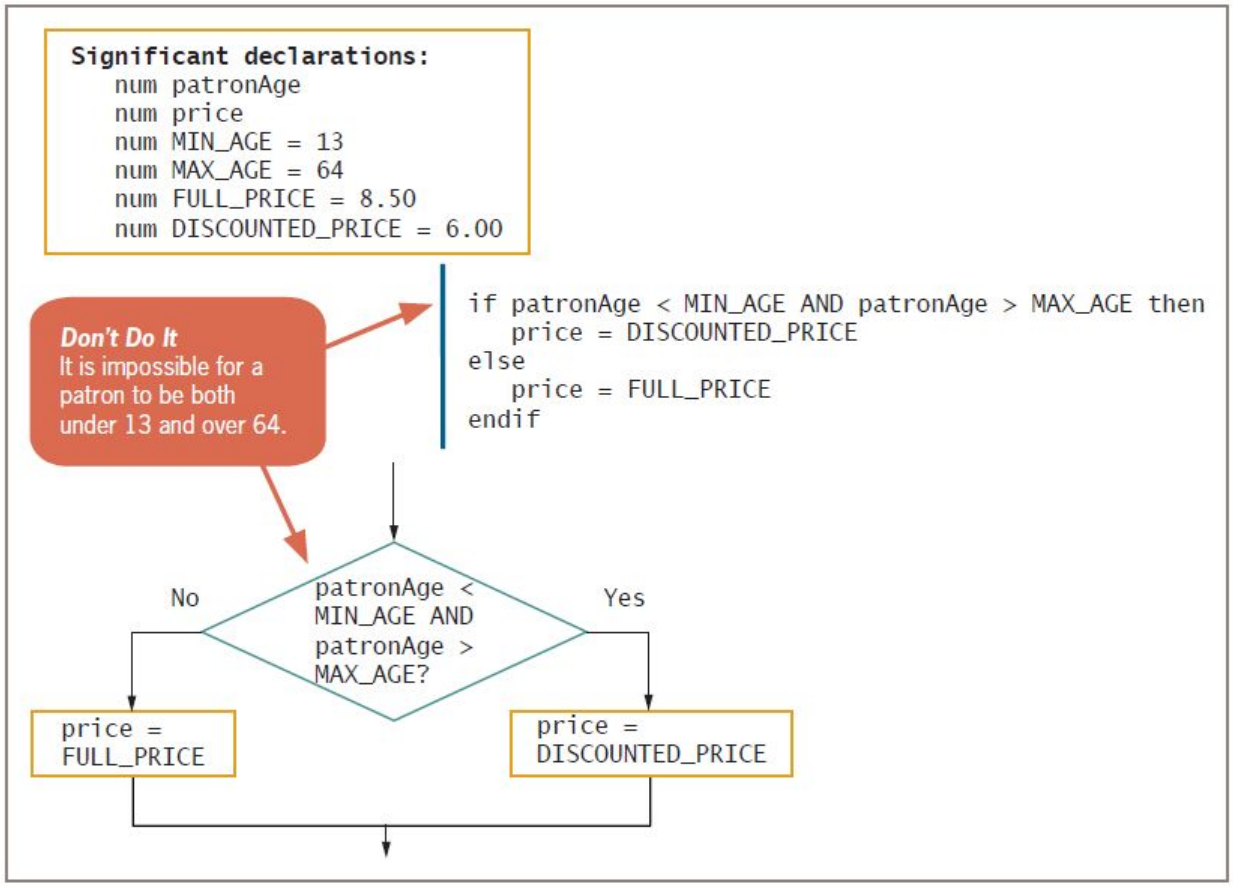


Figure 4-15 Incorrect logic that attempts to provide a discount for young and old movie patrons

Avoiding Common Errors in an OR Selection (continued)

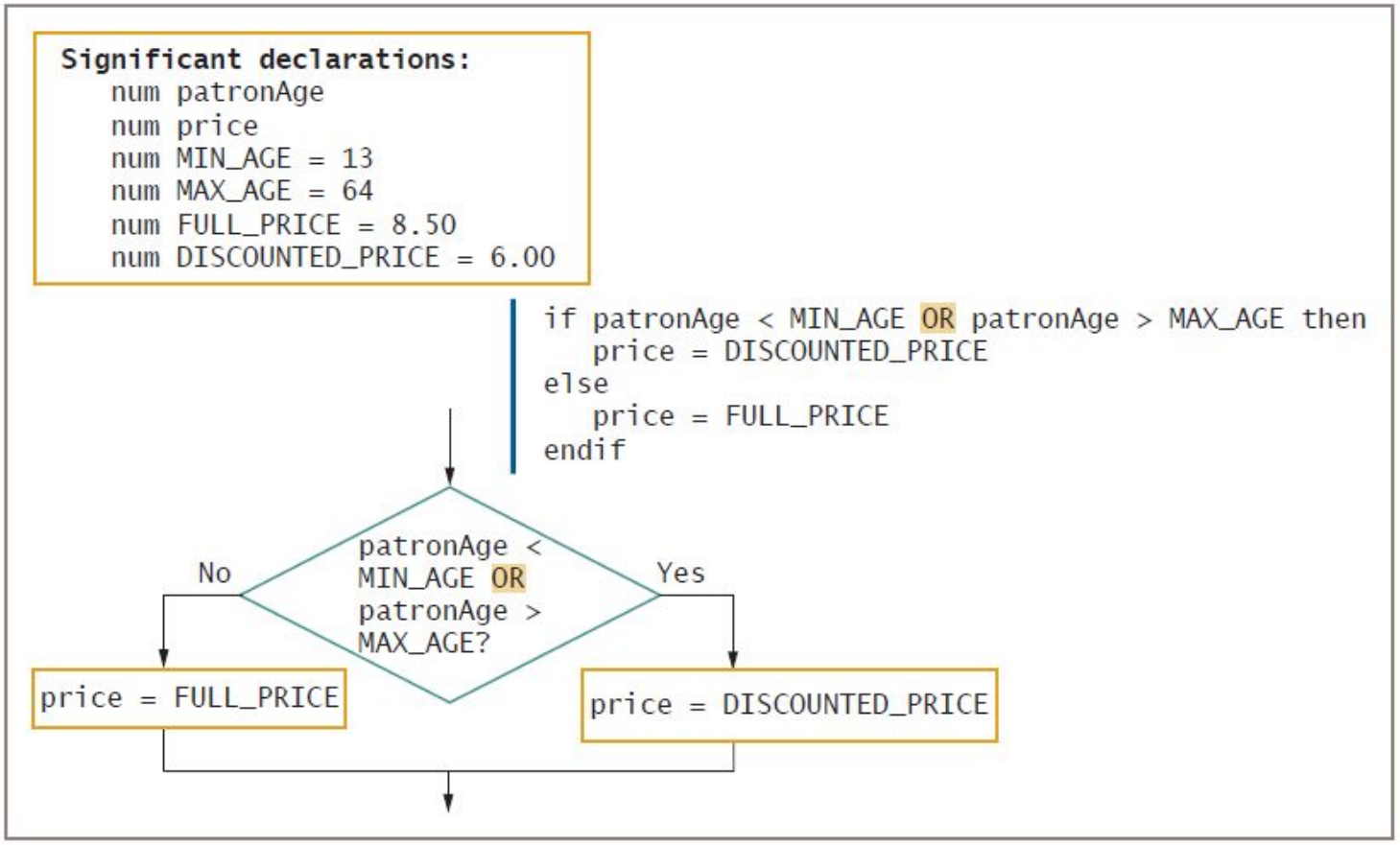


Figure 4-16 Correct logic that provides a discount for young and old movie patrons

Avoiding Common Errors in an OR Selection (continued)

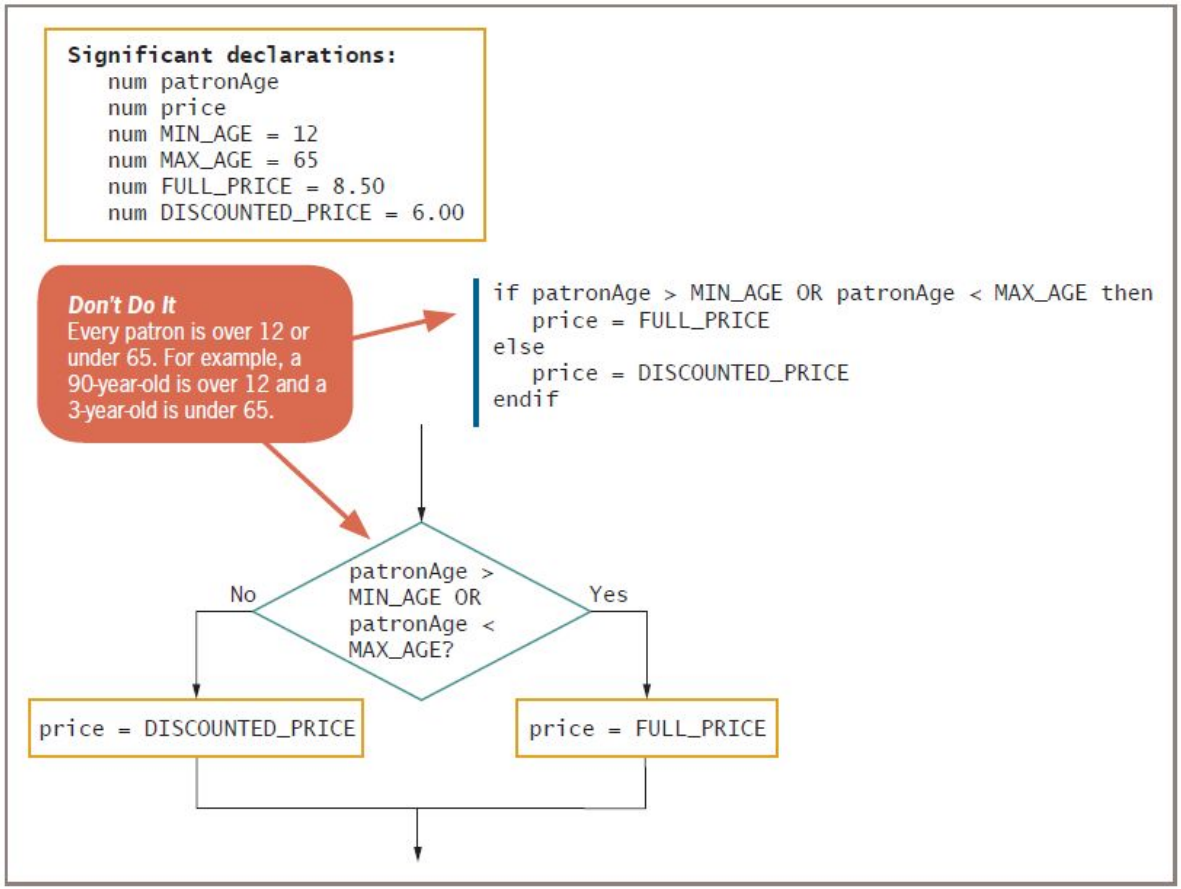


Figure 4-17 Incorrect logic that attempts to charge full price for patrons whose age is over 12 and under 65

Avoiding Common Errors in an OR Selection (continued)

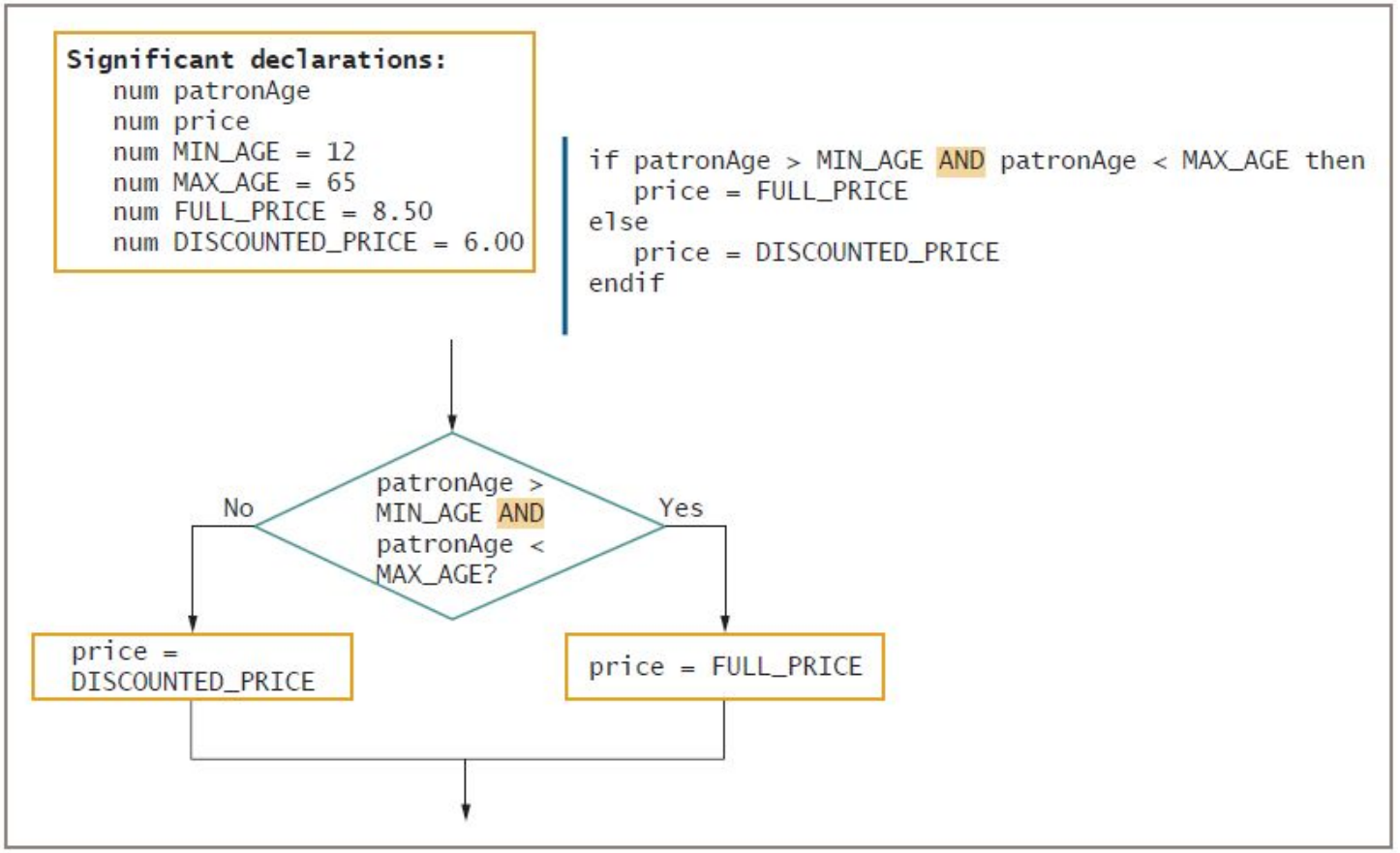


Figure 4-18 Correct logic that charges full price for patrons whose age is over 12 and under 65

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

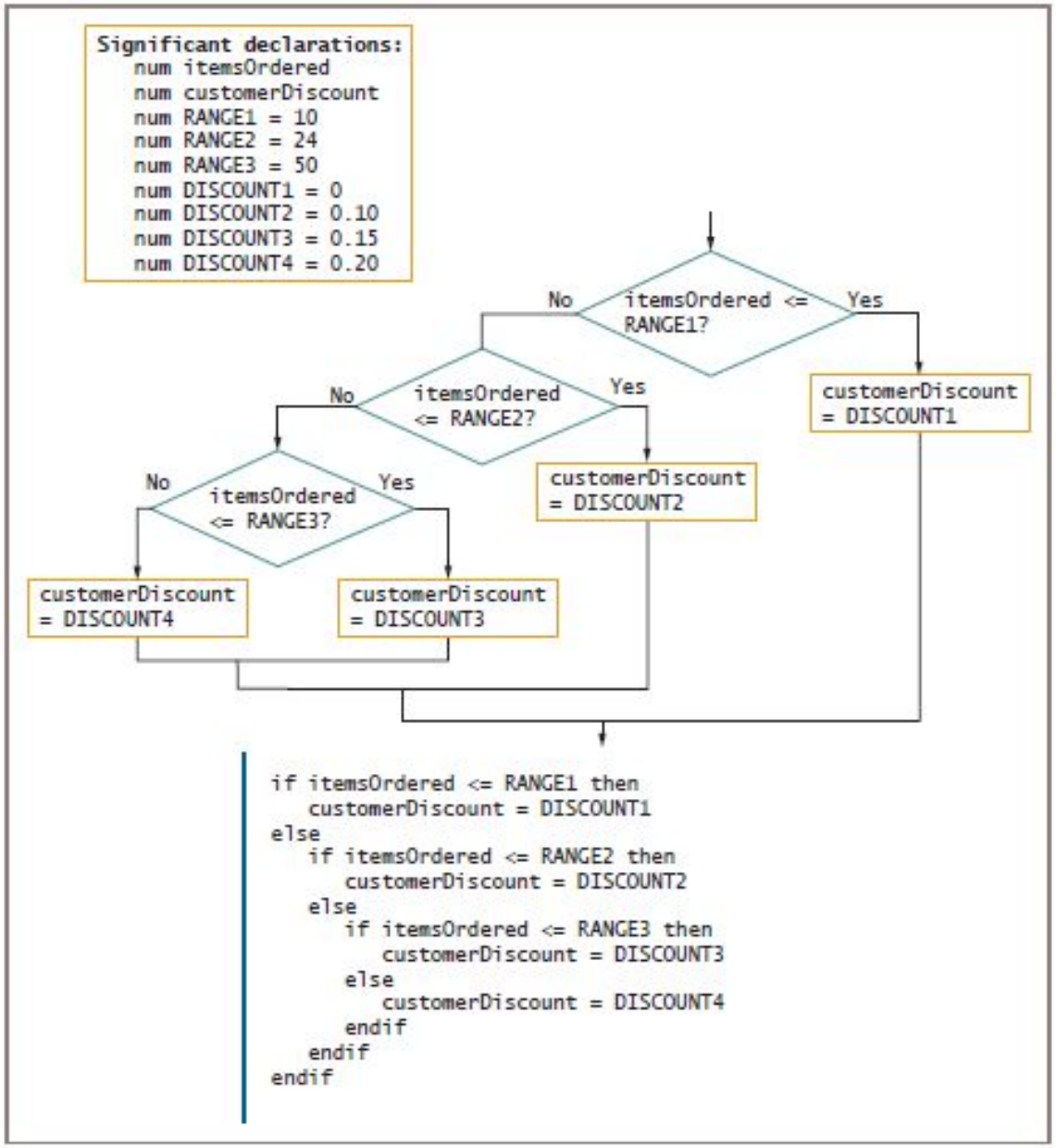


Figure 4-20 Flowchart and pseudocode of logic that selects correct discount based on items

Avoiding Common Errors When Using Range Checks

- Avoid 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 asking a question when the logic has already determined the outcome

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 (cont'd)

- 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 (cont'd)

- 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 (cont'd)

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

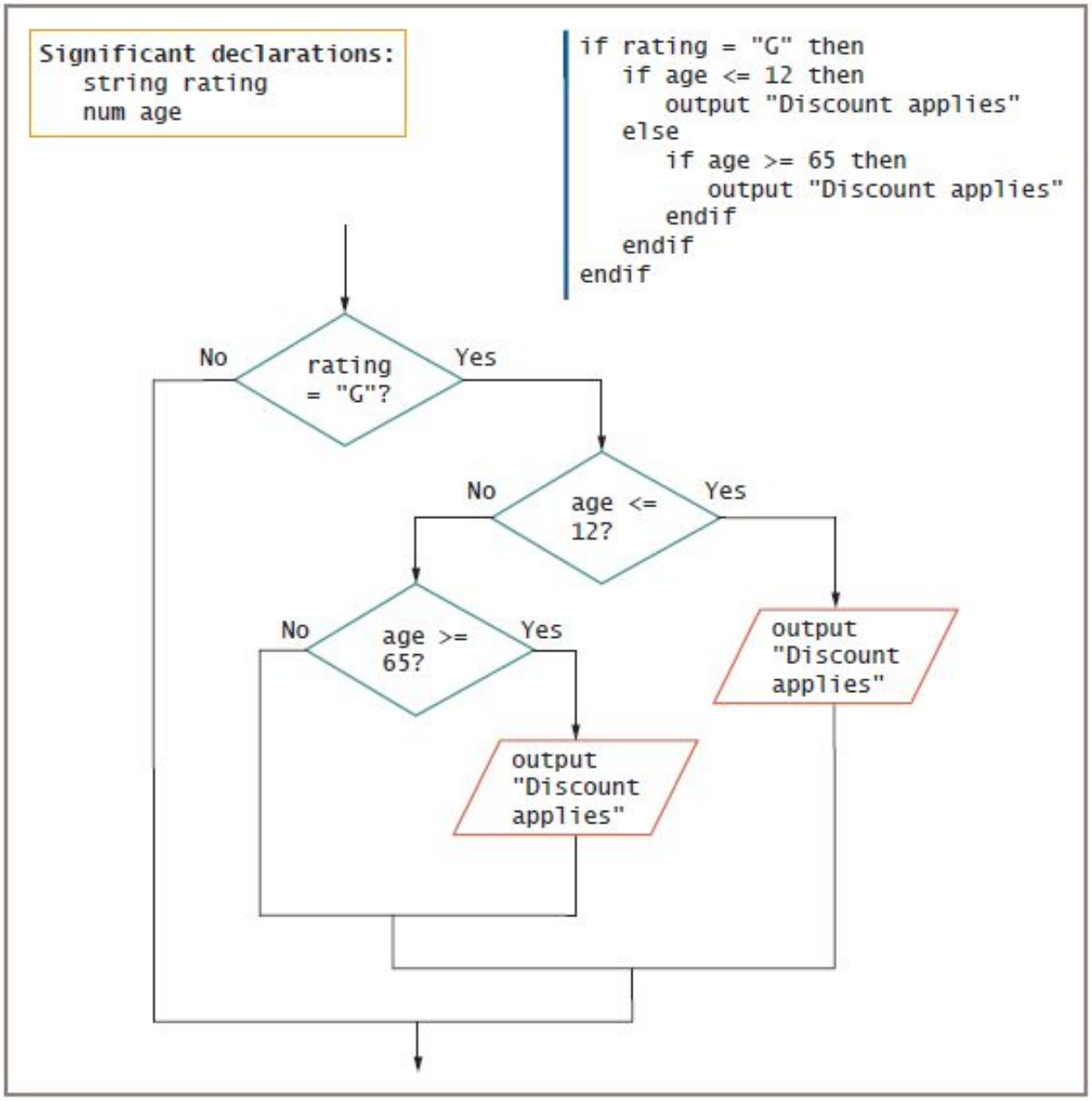


Figure 4-23 Nested decisions that determine movie patron discount

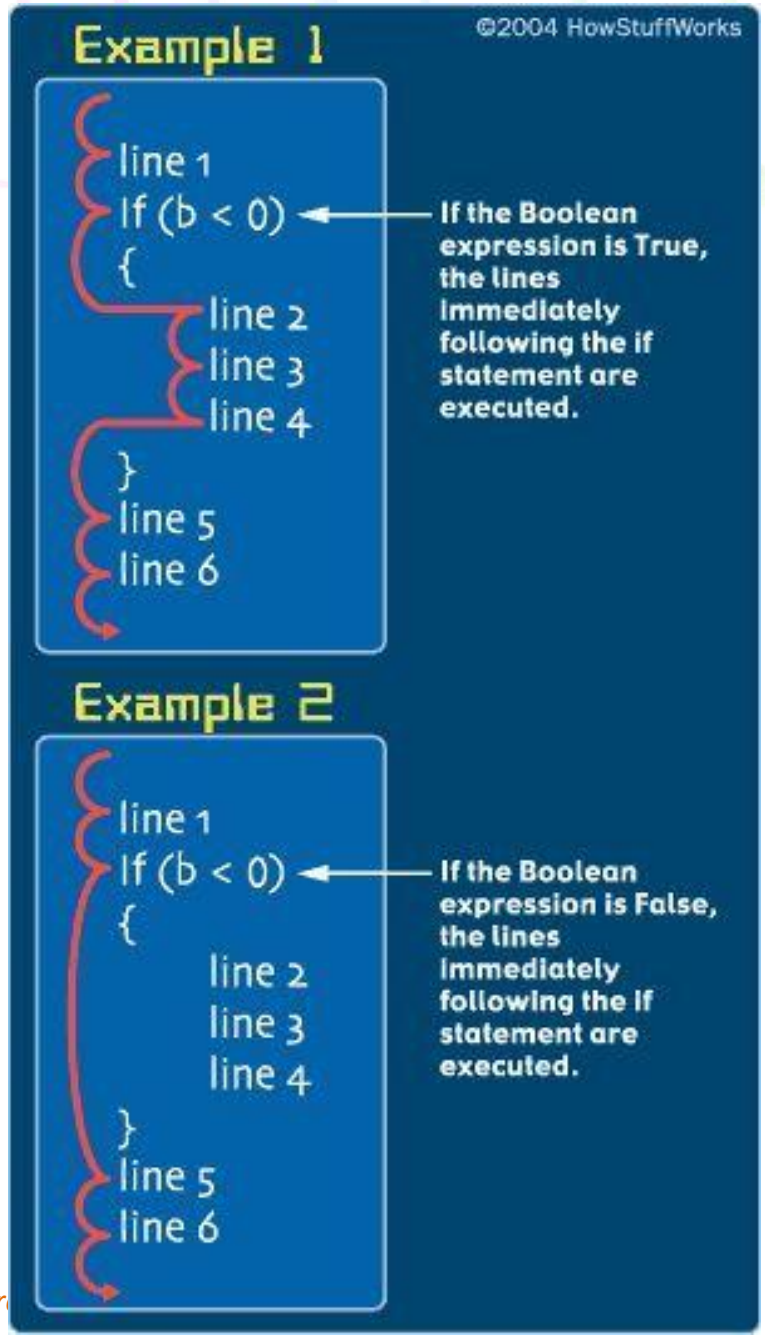
Summary

- Decisions involve evaluating Boolean expressions
- Use relational operators to compare values
- 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

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
- The `AND` operator takes precedence over the `OR` operator

In this Program, you'll learn how to Reverse a Sentence Using Recursion.



To properly understand this Program to Reverse a Sentence Using Recursion you should know the following:

- C Functions; C User-defined functions; C Recursion
- The below Program takes the input from the user and displays it in reverse order.

Program to Reverse a Sentence Using Recursion

```
#include <stdio.h>
void reverse();
void main()
{
        printf("Please enter a sentence: ");
        reverse();
        }
        void reverse()
        {
        char c;
        scanf("%c", &c);
        if (c != '\n')
        {
                reverse();
                printf("%c", c);
        }
}
```

C Program to Print an Integer in Just 3 Lines of Code

The variable is displayed on the screen using printf( ) function.

Data types in C language:

for **integers**, it is **int**, for **characters** it is **char**, for **floating point** data it's **float** and so on. For **large integers**, you can use **long or long long data** type. To store **integers** which are **larger than (2^18-1)** which is the range of **long long** data type you may use **strings**. In the below program we store an integer in a string and then display it.

```
#include <stdio.h>
int main()
{
int number;   // printf() dislpays the formatted output
printf("Enter an integer: ");        // scanf() reads the formatted input and stores them
scanf("%d", &number);              // printf() displays the formatted output
printf("You entered: %d", number);
return 0;
}
```

Output: Enter a integer: 35

You entered: 35

C Programming Language:

C Programming Language:

The course materials can be found on the:

https://devdocs.io/c/