



Programming Logic and Design

Seventh Edition

Chapter 5

Looping



Objectives

- In this chapter, you will learn about:
 - The advantages of looping
 - Using a loop control variable
 - Nested loops
 - Avoiding common loop mistakes
 - Using a `for` loop
 - Common loop applications



Understanding the Advantages of Looping

- Looping makes computer programming efficient and worthwhile
- Write one set of instructions to operate on multiple, separate sets of data
- Loop: a structure that repeats actions while some condition continues

Understanding the Advantages of Looping (continued)

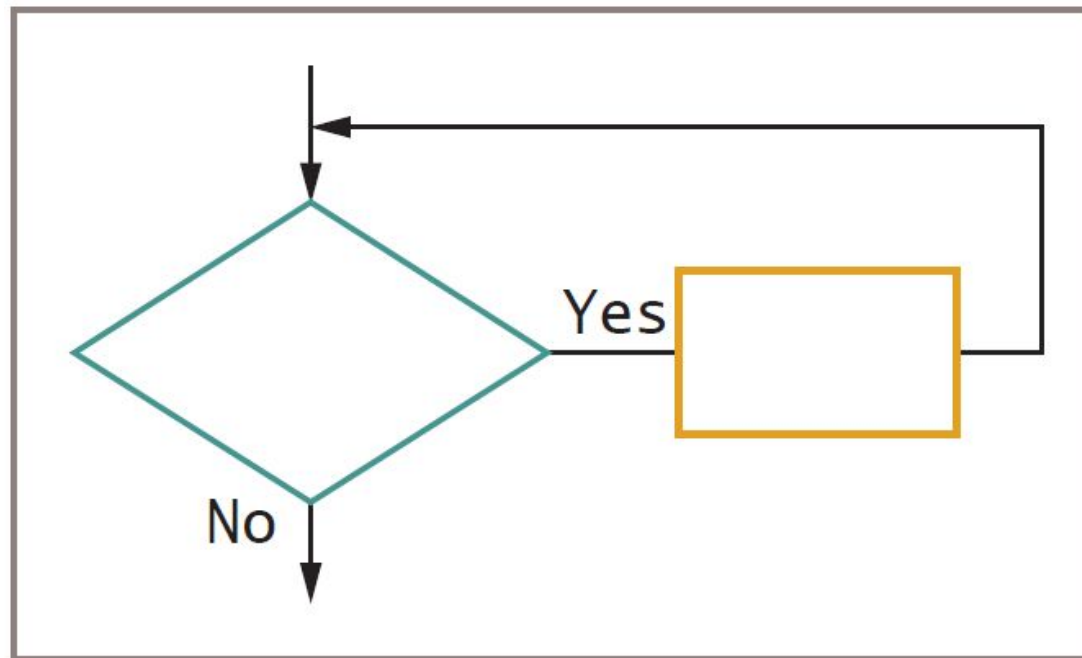


Figure 5-1 The loop structure



Using a Loop Control Variable

- As long as a condition remains true, the statements in a `while` loop's body execute
- Control number of repetitions
 - **Loop control variable** initialized before entering loop
 - Loop control variable tested
 - Body of loop must alter value of loop control variable
- Repetitions controlled by:
 - Counter
 - Sentinel value



Using a Definite Loop with a Counter

- **Definite loop**
 - Executes a predetermined number of times
- **Counter-controlled loop**
 - Program counts loop repetitions
- Loop control variables altered by:
 - **Incrementing**
 - **Decrementing**

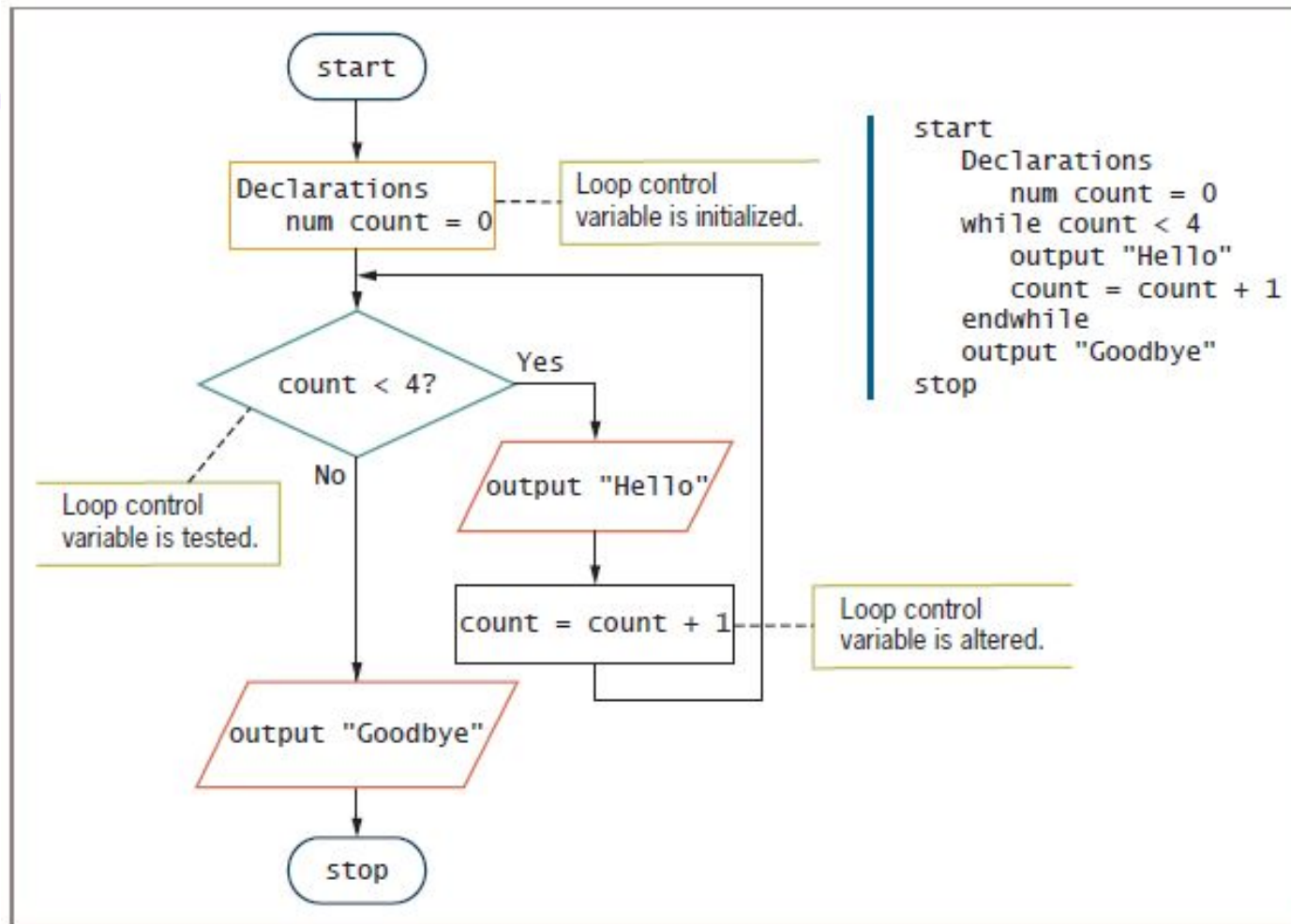


Figure 5-3 A counted `while` loop that outputs *Hello* four times

Using an Indefinite Loop with a Sentinel Value

- **Indefinite loop**
 - Performed a different number of times each time the program executes
 - The user decides how many times the loop executes

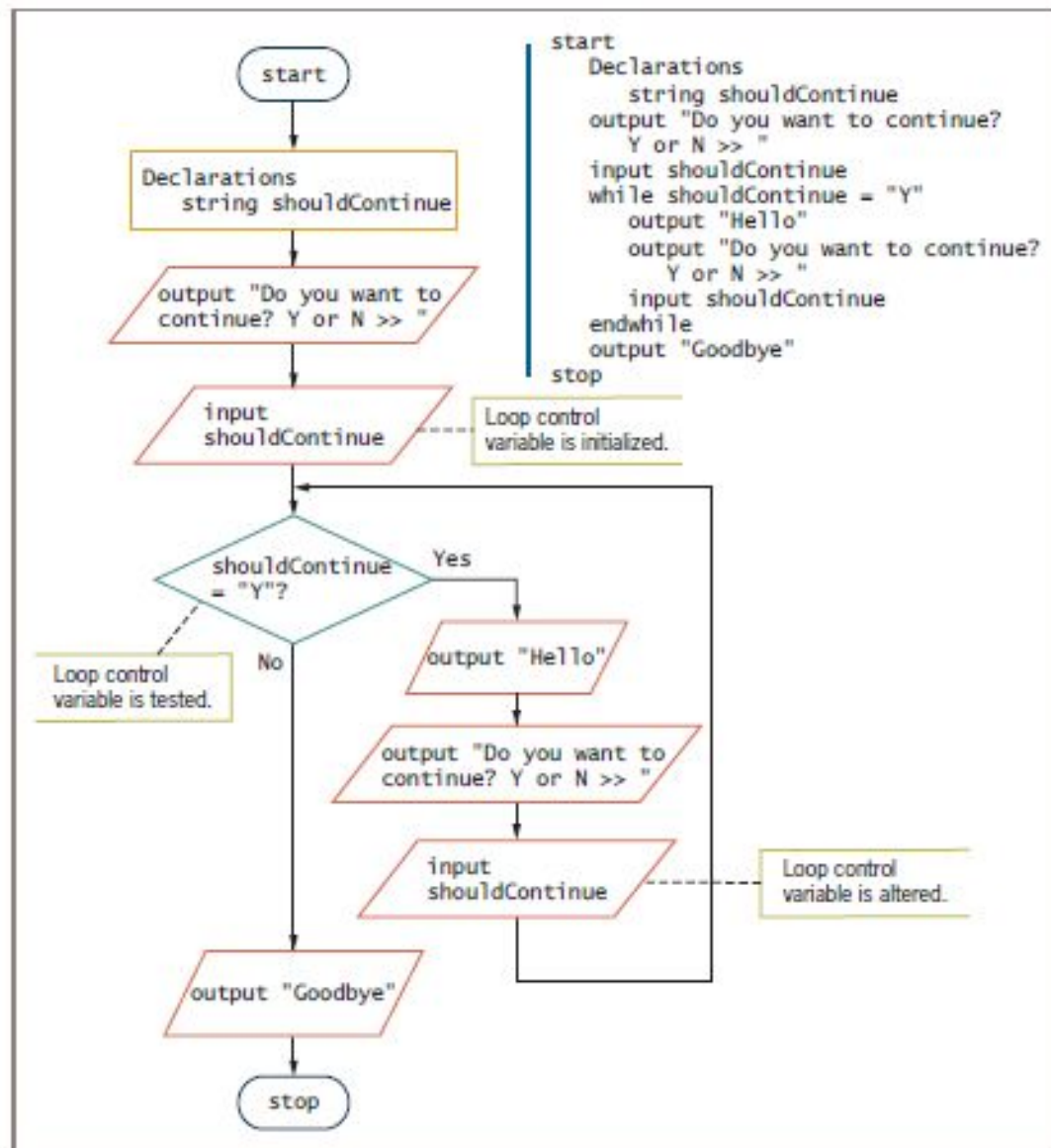


Figure 5-4 An indefinite `while` loop that displays *Hello* as long as the user wants to continue

Understanding the Loop in a Program's Mainline Logic

- Three steps should occur in every properly functioning loop
 - Provide a starting value for the variable that will control the loop
 - Test the loop control variable to determine whether the loop body executes
 - Alter the loop control variable



Nested Loops

- **Nested loops:** loops within loops
- **Outer loop:** the loop that contains the other loop
- **Inner loop:** the loop that is contained
- Needed when values of two (or more) variables repeat to produce every combination of values

Nested Loops (continued)

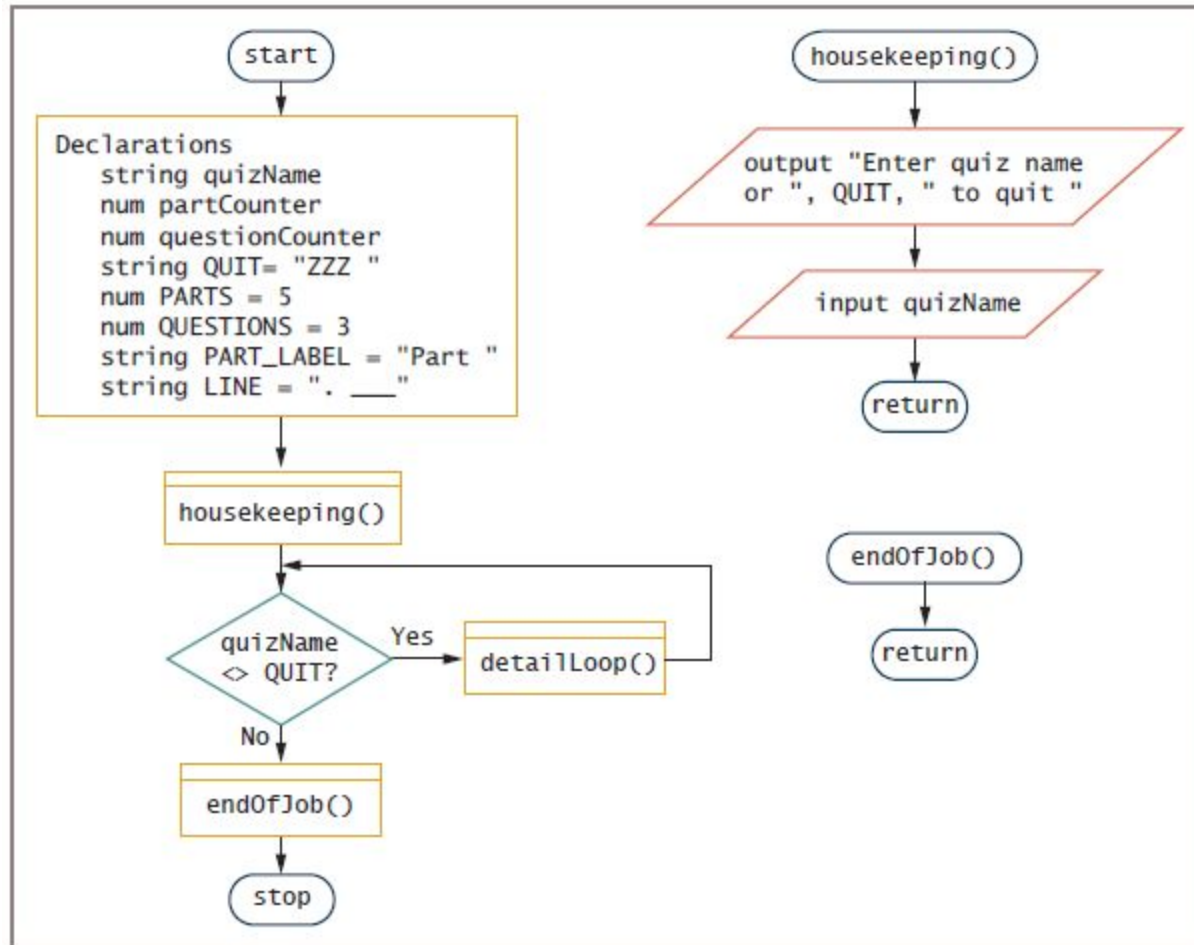


Figure 5-8 Flowchart and pseudocode for AnswerSheet program



Avoiding Common Loop Mistakes

- Mistake: neglecting to initialize the loop control variable
 - Example: `get name` statement removed
 - Value of `name` unknown or garbage
 - Program may end before any labels printed
 - 100 labels printed with an invalid name

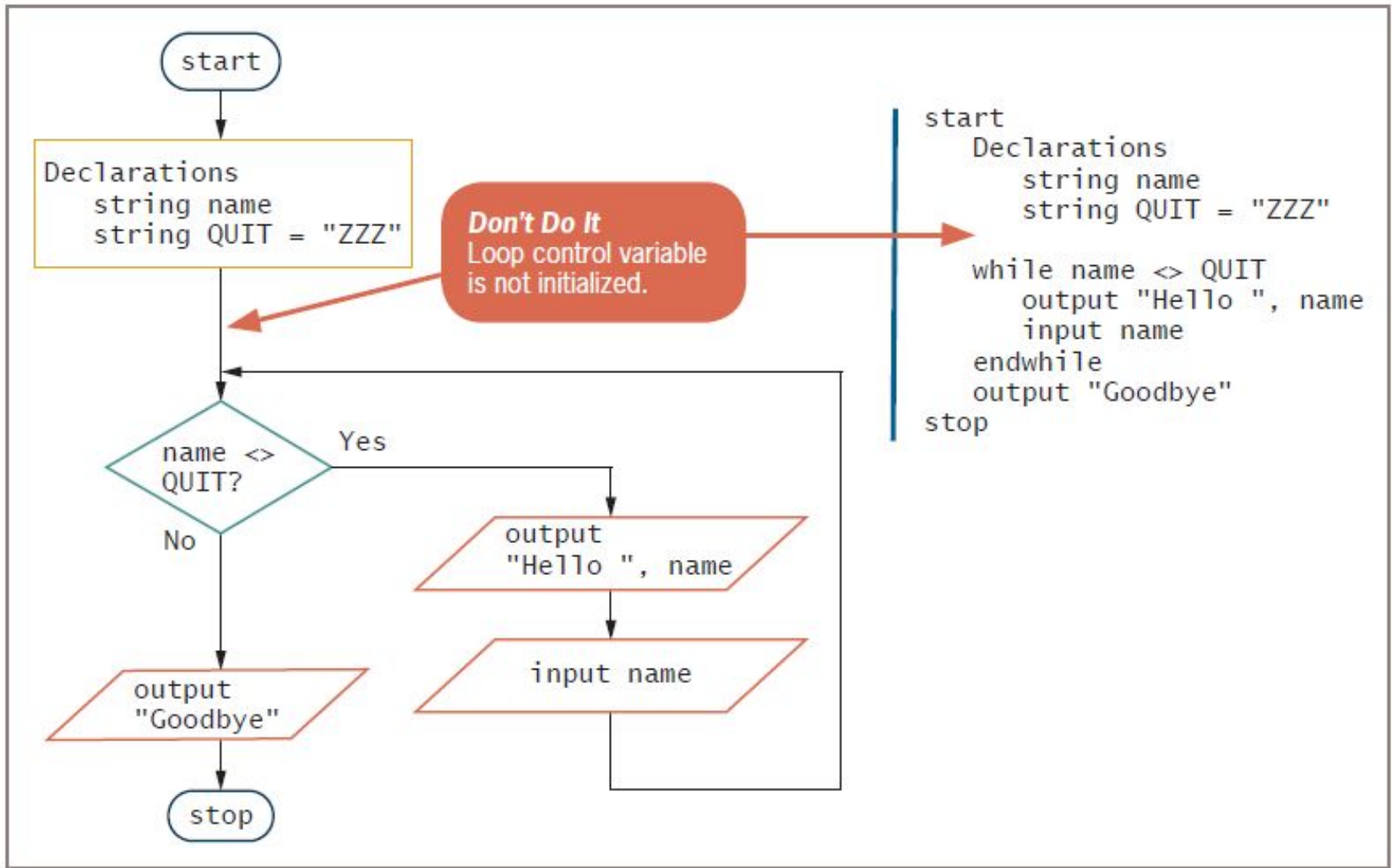


Figure 5-10 Incorrect logic for greeting program because the loop control variable initialization is missing

Avoiding Common Loop Mistakes (continued)

- Mistake: neglecting to alter the loop control variable
 - Remove `get name` instruction from outer loop
 - User never enters a name after the first one
 - Inner loop executes infinitely
- Always incorrect to create a loop that cannot terminate

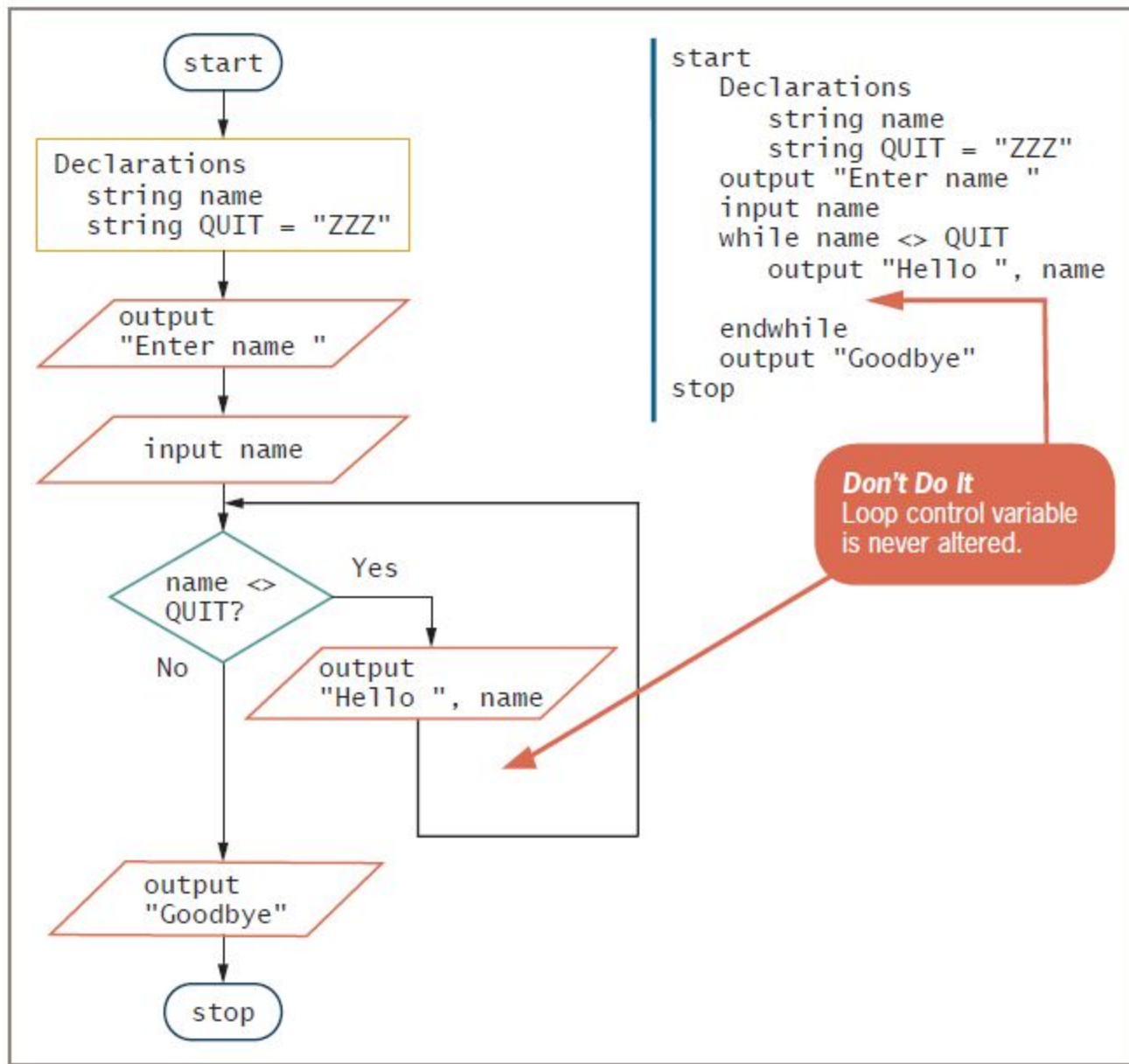


Figure 5-11 Incorrect logic for greeting program because the loop control variable is not altered

Avoiding Common Loop Mistakes (continued)

- Mistake: using the wrong comparison with the loop control variable
 - Programmers must use correct comparison
 - Seriousness depends on actions performed within a loop
 - Overcharge insurance customer by one month
 - Overbook a flight on airline application
 - Dispense extra medication to patients in pharmacy

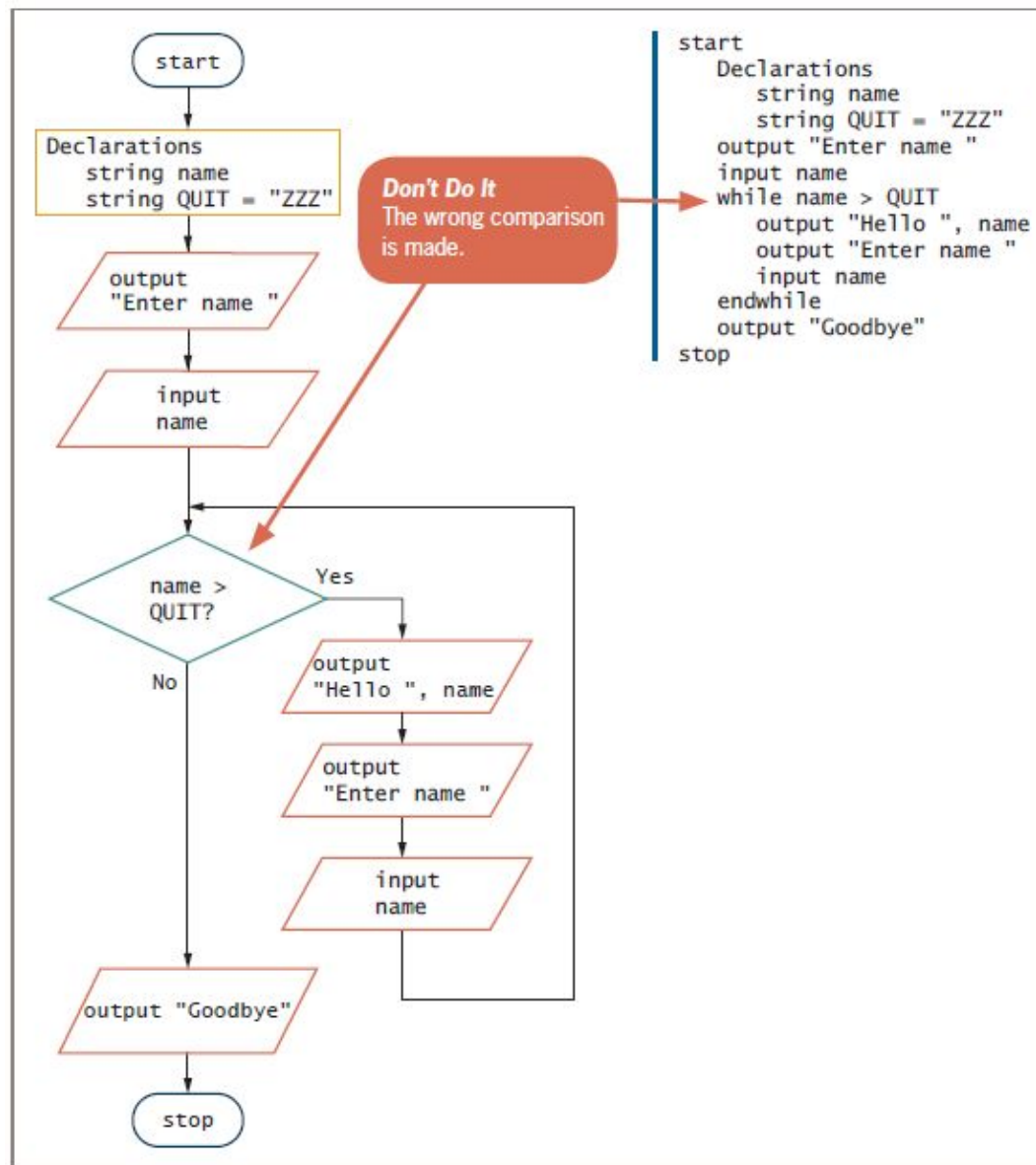


Figure 5-12 Incorrect logic for greeting program because the wrong test is made with the loop control variable

Avoiding Common Loop Mistakes (continued)

- Mistake: including statements inside the loop that belong outside the loop
 - Example: discount every item by 30 percent
 - Inefficient because the same value is calculated 100 separate times for each price that is entered
 - Move outside the loop for efficiency

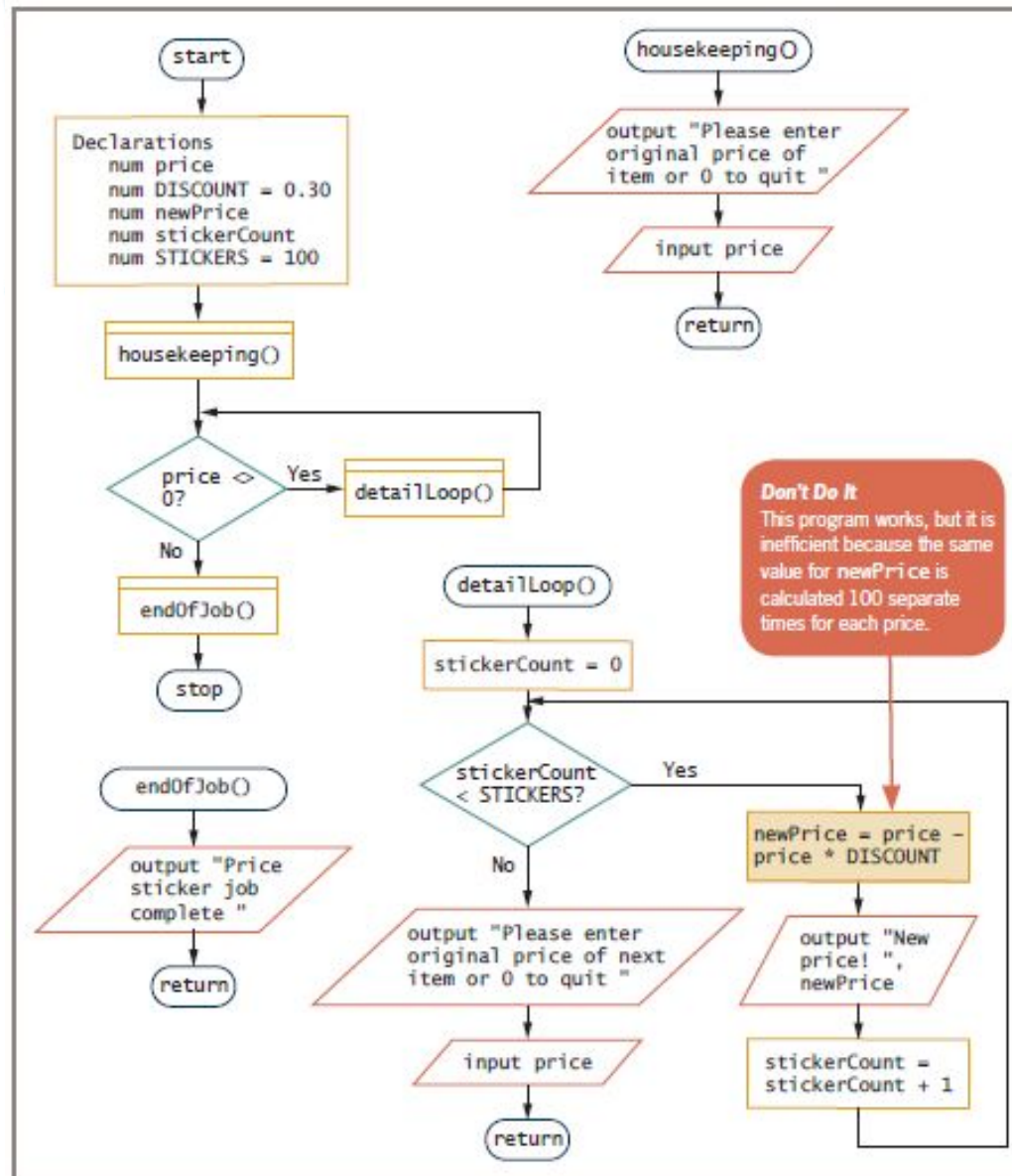


Figure 5-13 Inefficient way to produce 100 discount price stickers for differently priced items

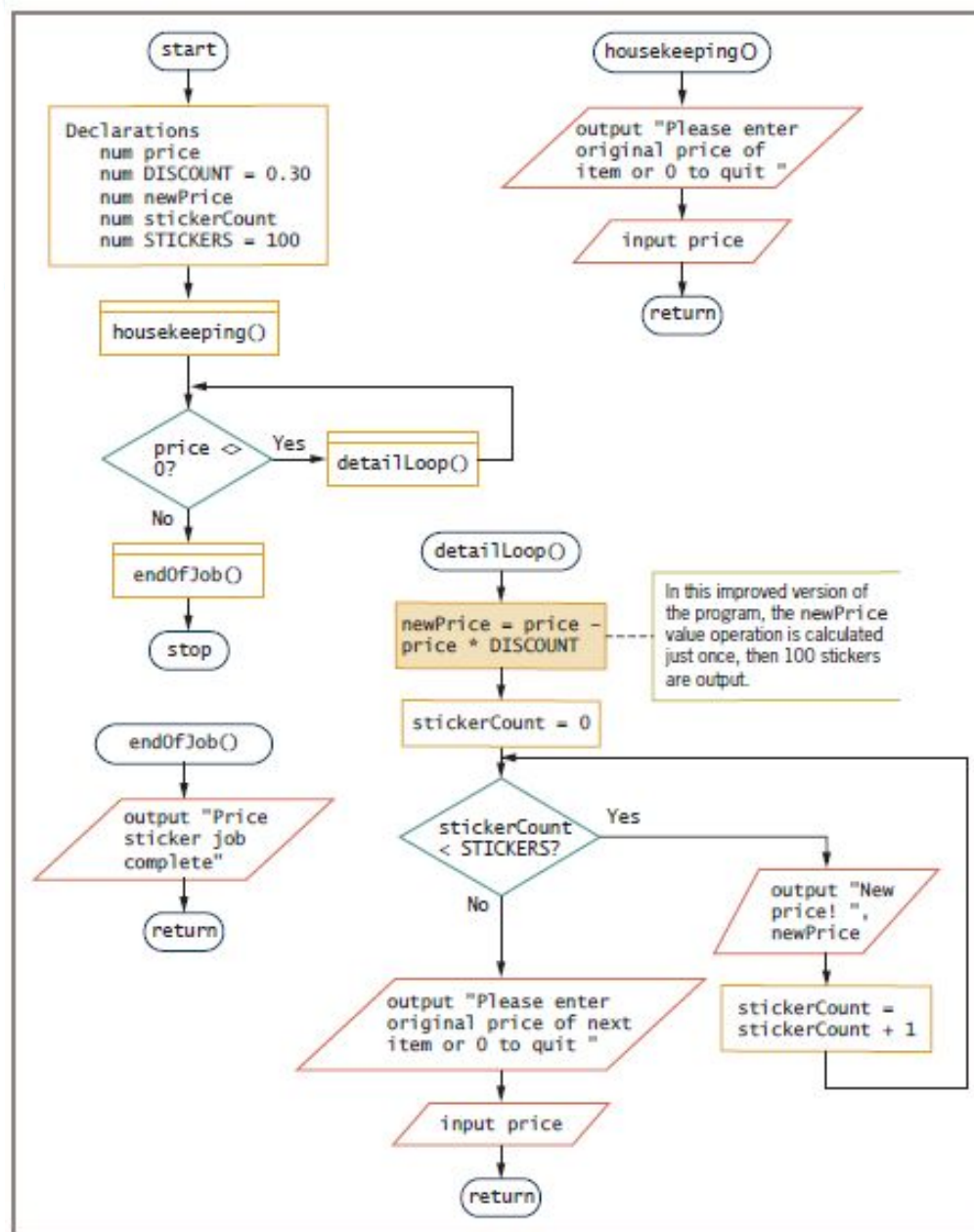


Figure 5-14 Improved discount sticker-making program



Using a for Loop

- **for statement** or **for loop** is a definite loop
- Provides three actions in one structure
 - Initializes
 - Evaluates
 - Alters
- Takes the form:

```
for loopControlVariable = initialValue to  
finalValue step stepValue  
do something  
endfor
```

Using a `for` Loop (continued)

- Example

```
for count = 0 to 3 step 1
    output "Hello"
endfor
```

- Initializes `count` variable to 0
- Checks `count` variable against the limit value 3
- If evaluation is true, `for` statement body prints the word “Hello”
- Increases `count` by 1



Using a `for` Loop (continued)

- `while` statement could be used in place of `for` statement
- **Step value:** the amount by which a loop control variable changes
 - Can be positive or negative (incrementing or decrementing the loop control variable)
 - Default step value is 1
 - Programmer specifies a step value when each pass through the loop changes the loop control variable by a value other than 1



Using a `for` Loop (continued)

- **Pretest loop:** the loop control variable is tested before each iteration
 - `for` loops and `while` loops are pretest loops
- **Posttest loop:** the loop control variable is tested after each iteration
 - `do...while` is a posttest loop



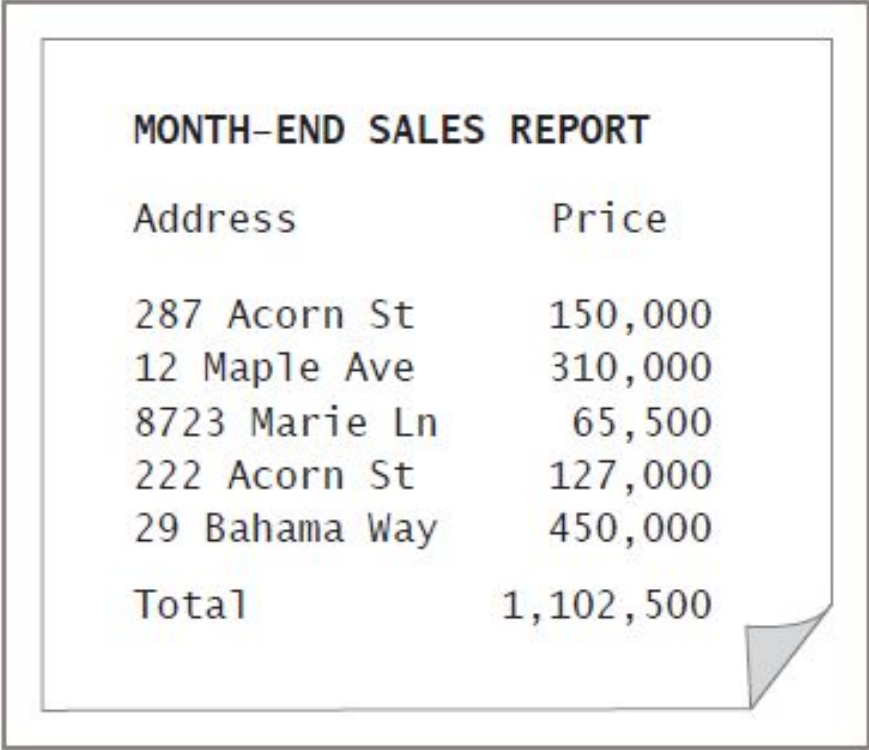
Common Loop Applications

- Using a loop to accumulate totals
 - Examples
 - Business reports often include totals
 - List of real estate sold and total value
- **Accumulator:** variable that gathers values
 - Similar to a counter
 - Counter increments by 1
 - Accumulator increments by some value

Common Loop Applications (continued)

- Accumulators require three actions
 - Initialize the accumulator to 0
 - Accumulators are altered: once for every data set processed
 - At the end of processing, accumulators are output
- **Summary reports**
 - Contain only totals with no detail data
 - Loops are processed but detail information is not printed

Common Loop Applications (continued)



MONTH-END SALES REPORT	
Address	Price
287 Acorn St	150,000
12 Maple Ave	310,000
8723 Marie Ln	65,500
222 Acorn St	127,000
29 Bahama Way	450,000
Total	1,102,500

Figure 5-16 Month-end real estate sales report

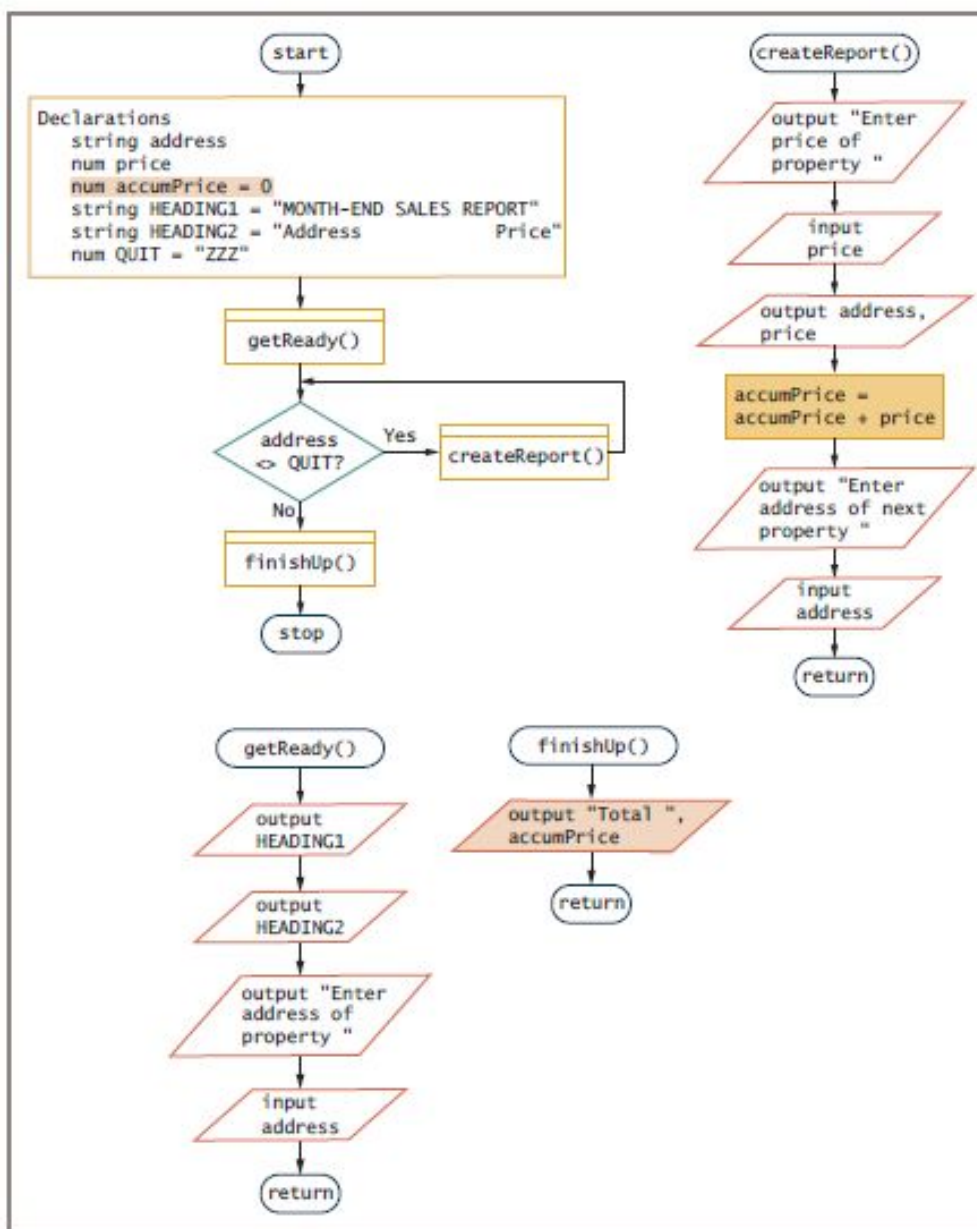


Figure 5-17 Flowchart and pseudocode for real estate sales report program

Common Loop Applications (continued)

- Using a loop to validate data
 - **Defensive programming:** preparing for all possible errors before they occur
 - When prompting a user for data, no guarantee that data is valid
 - **Validate data:** make sure data falls in acceptable ranges (month values between 1 and 12)
 - **GIGO:** Garbage in, garbage out
 - Unvalidated input will result in erroneous output

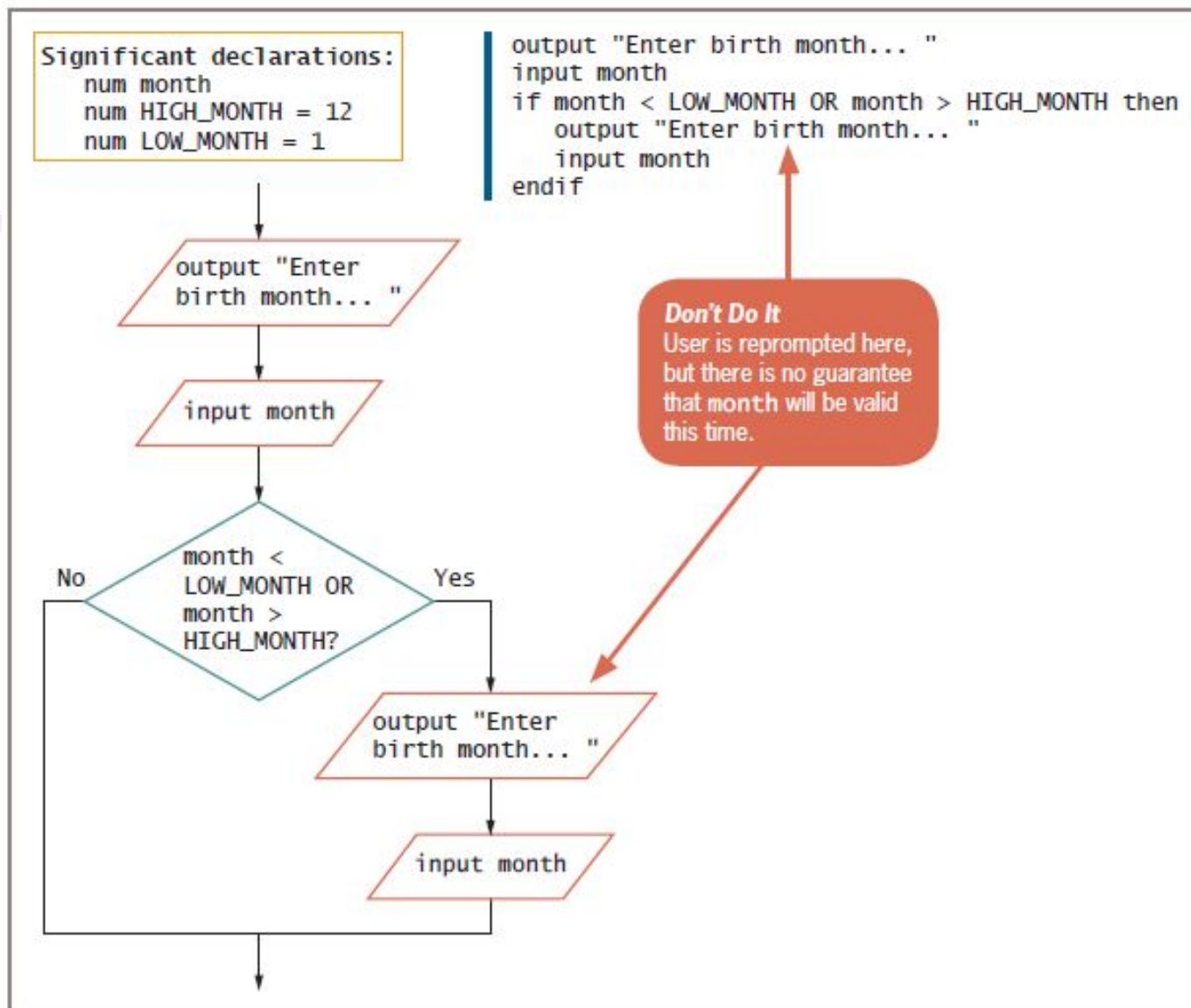


Figure 5-18 Reprompting a user once after an invalid month is entered

Significant declarations:

```
num month  
num HIGH_MONTH = 12  
num LOW_MONTH = 1
```

```
output "Enter birth month... "  
input month  
while month < LOW_MONTH OR month > HIGH_MONTH  
    output "Enter birth month... "  
    input month  
endwhile
```

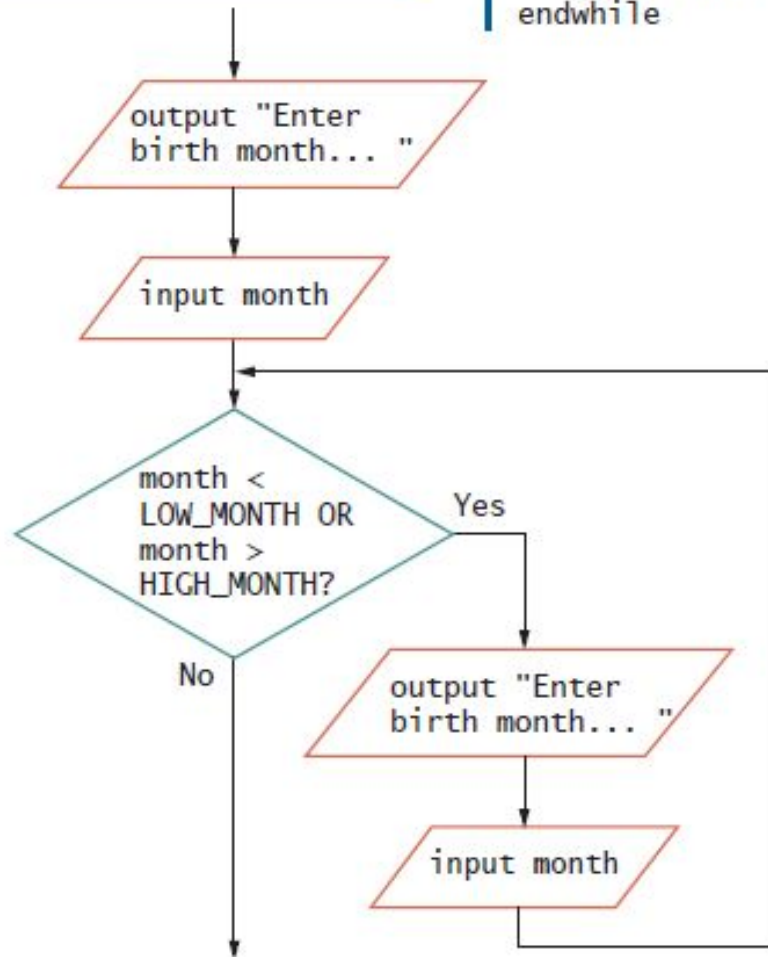


Figure 5-19 Reprompting a user continuously after an invalid month is entered

Common Loop Applications (continued)

- Limiting a reprompting loop
 - Reprompting can be frustrating to a user if it continues indefinitely
 - Maintain a count of the number of reprompts
 - **Forcing** a data item means:
 - Override incorrect data by setting the variable to a specific value

Common Loop Applications (continued)

- Validating a data type
 - Validating data requires a variety of methods
 - `isNumeric()` or similar method
 - Provided with the language translator you use to write your programs
 - Black box
 - `isChar()` or `isWhitespace()`
 - Accept user data as strings
 - Use built-in methods to convert to correct data types

Common Loop Applications (continued)

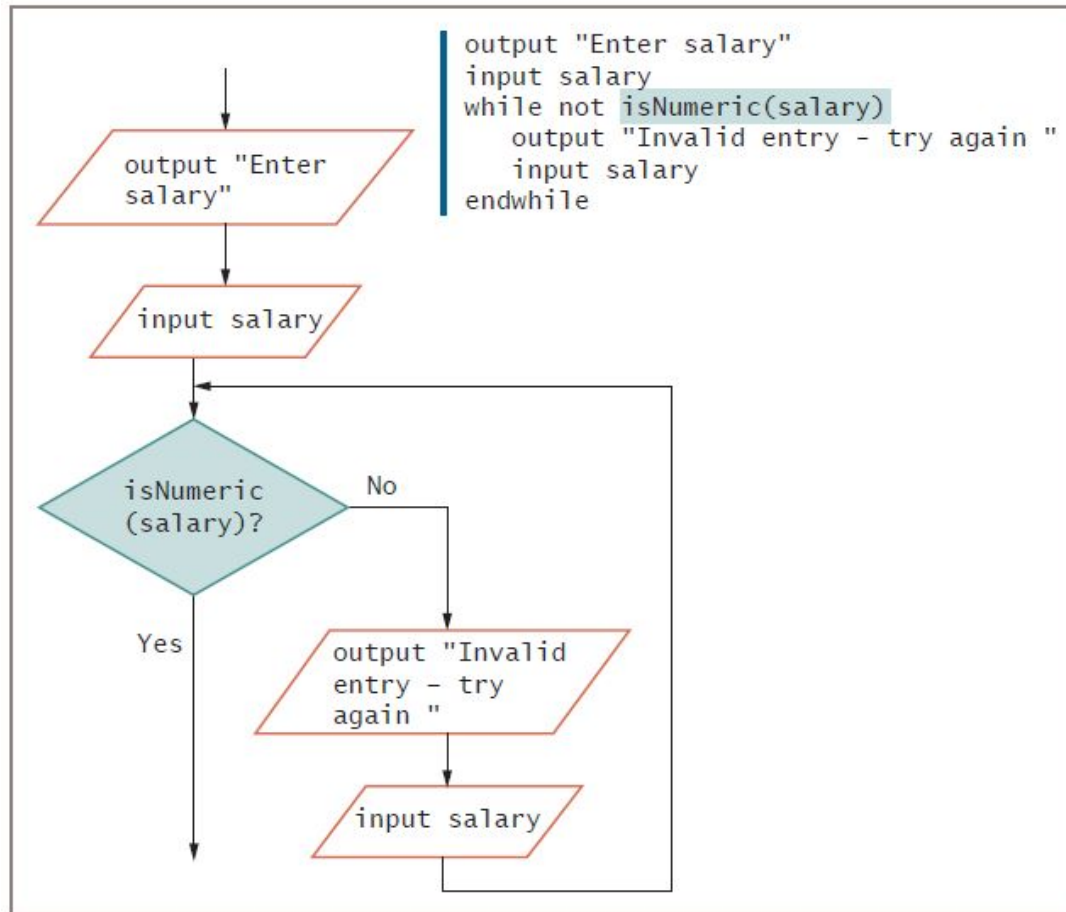


Figure 5-21 Checking data for correct type



Common Loop Applications (continued)

- Validating reasonableness and consistency of data
 - Many data items can be checked for reasonableness
 - Good defensive programs try to foresee all possible inconsistencies and errors



Summary

- Loops write one set of instructions that operate on multiple, separate sets of data
- Three steps must occur in every loop
 - Initialize the loop control variable
 - Compare the variable to some value
 - Alter the variable that controls the loop
- Nested loops: loops within loops
- Nested loops maintain two individual loop control variables
 - Alter each at the appropriate time



Summary (continued)

- Common mistakes made by programmers
 - Neglecting to initialize the loop control variable
 - Neglecting to alter the loop control variable
 - Using the wrong comparison with the loop control variable
 - Including statements inside the loop that belong outside the loop
- Most computer languages support a `for` statement
 - `for` loop used when the number of iterations is known
- Loops are used to accumulate totals in business reports and to reprompt users for valid data