Programming Logic and Design
Ninth 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
  - Using a posttest loop
  - Characteristics shared by all structured loops
  - Common loop applications
  - Similarities and differences between selections and loops

Appreciating the Advantages of Looping

- Looping makes computer programming efficient and worthwhile
- Write one set of instructions to operate on multiple, separate sets of data
  - Less time required for design and coding
  - Fewer errors
  - Shorter compile time
- Loop: a structure that repeats actions while some condition continues

Appreciating the Advantages of Looping (continued -1)

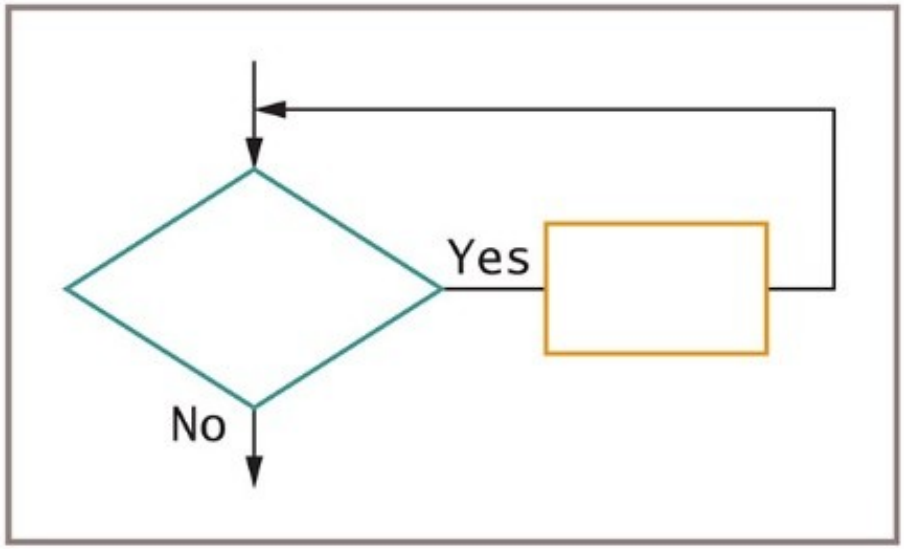


Figure 5-1 The loop structure

Appreciating the Advantages of Looping (continued -2)

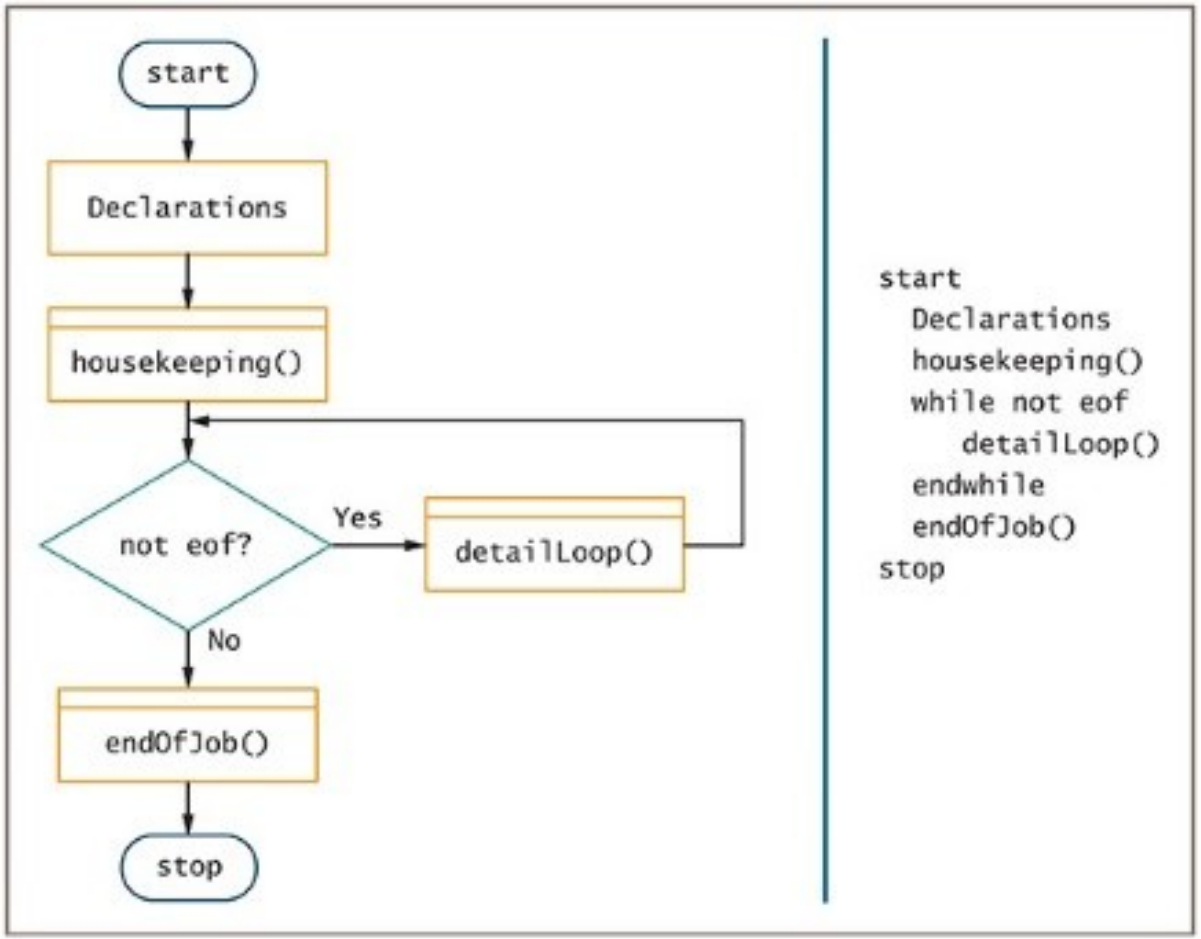


Figure 5-2 The mainline logic common to many business programs

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 – used to create a definite counter-controlled loop
  - Sentinel value – used to create an indefinite

Using a Definite Loop with a Counter

- **Definite loop**
  - Executes a predetermined number of times
- **Counter-controlled loop** or **counted loop**
  - Program counts loop repetitions
- Loop control variables altered by:
  - **Incrementing**
  - **Decrementing**
- **Counter**
  - Any numeric variable that counts the number of times an event has occurred, usually starts with 0

Using a Definite Loop with a Counter (continued -1)

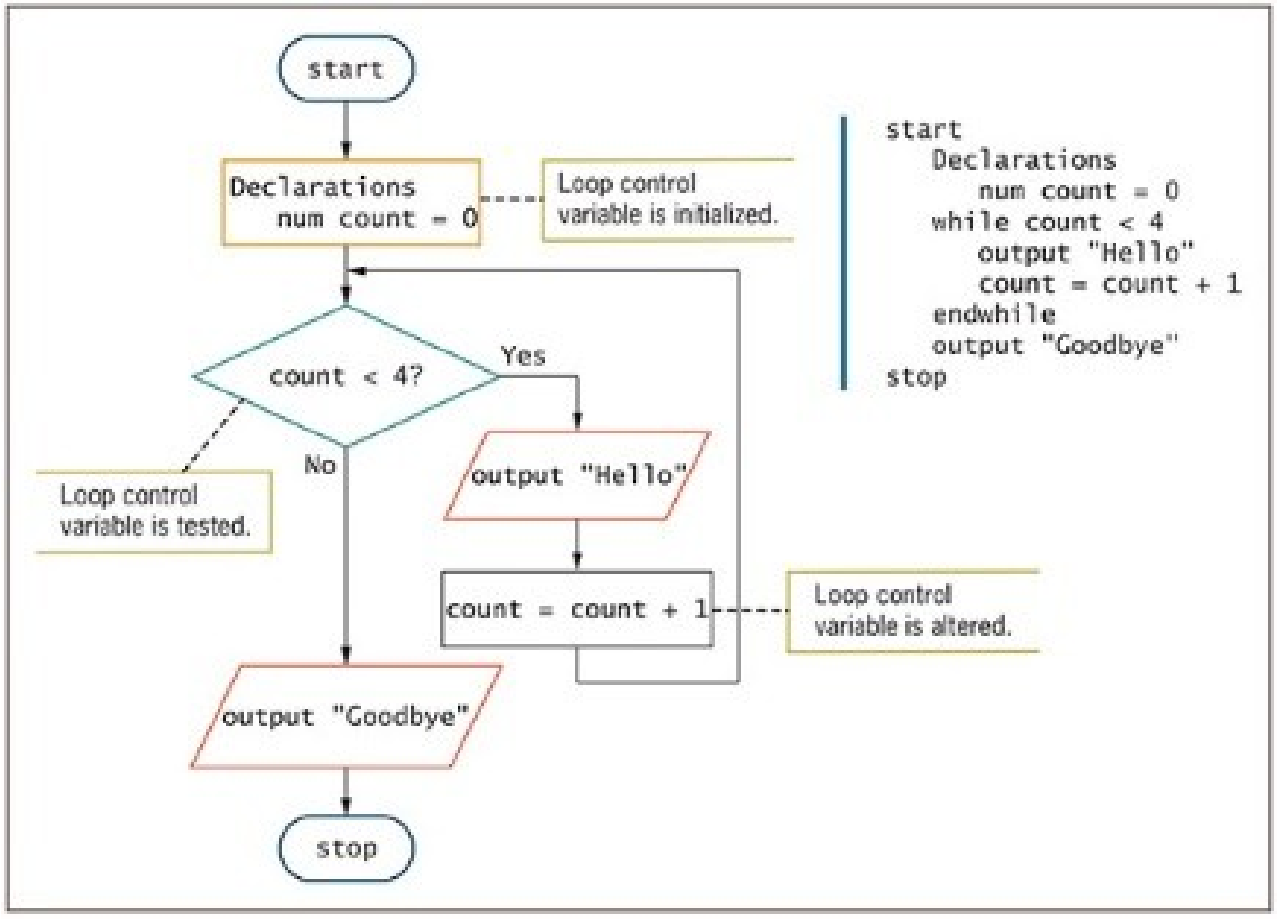


Figure 5-3 A counted while loop that outputs *Hello* four times

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

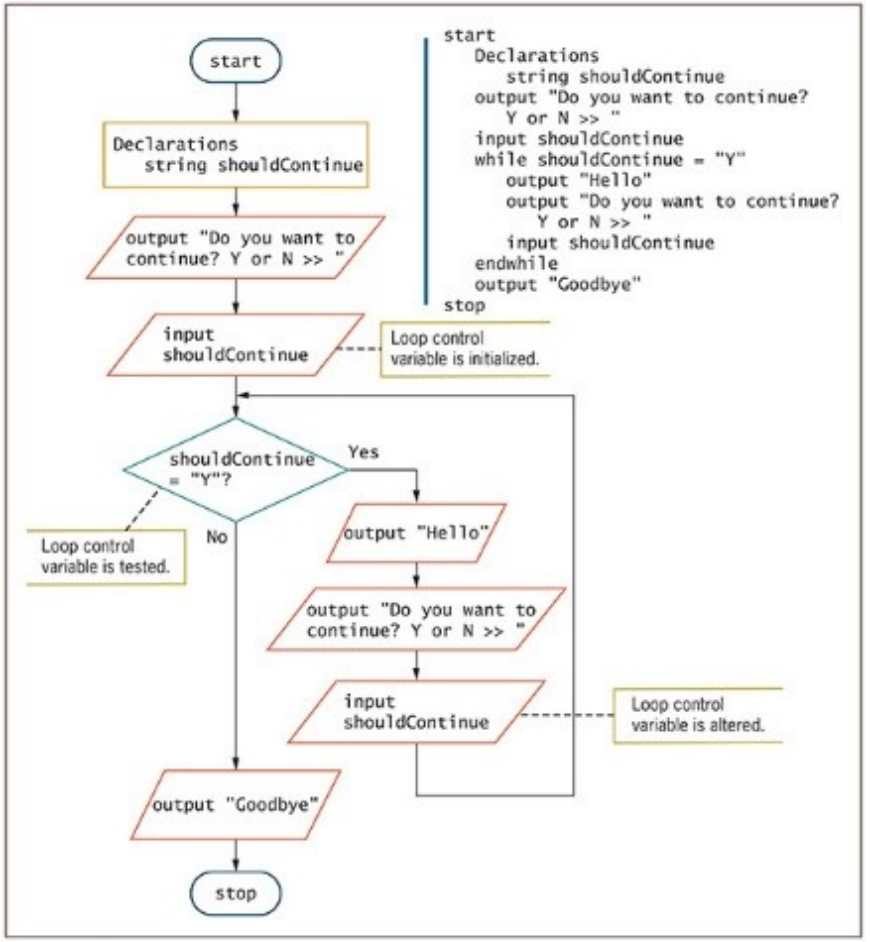


Figure 5-4 An indefinite `while` loop that displays *Hello* as long as the user wants to continue

Using an Indefinite Loop with a Sentinel Value (continued -1)

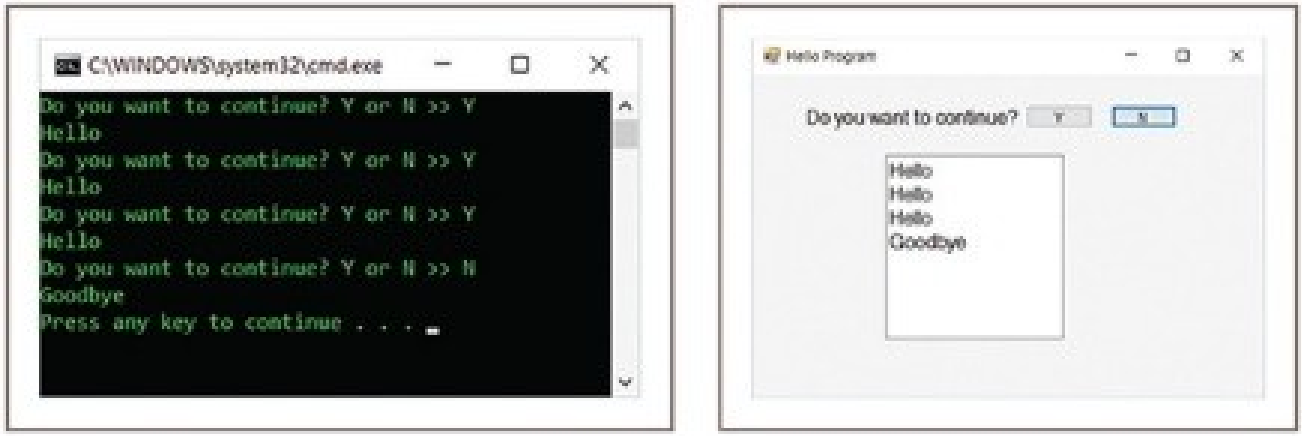


Figure 5-5 Typical executions of the program in Figure 5-4 in two environments

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 within the loop

Understanding the Loop in a Program's Mainline Logic (continued -1)

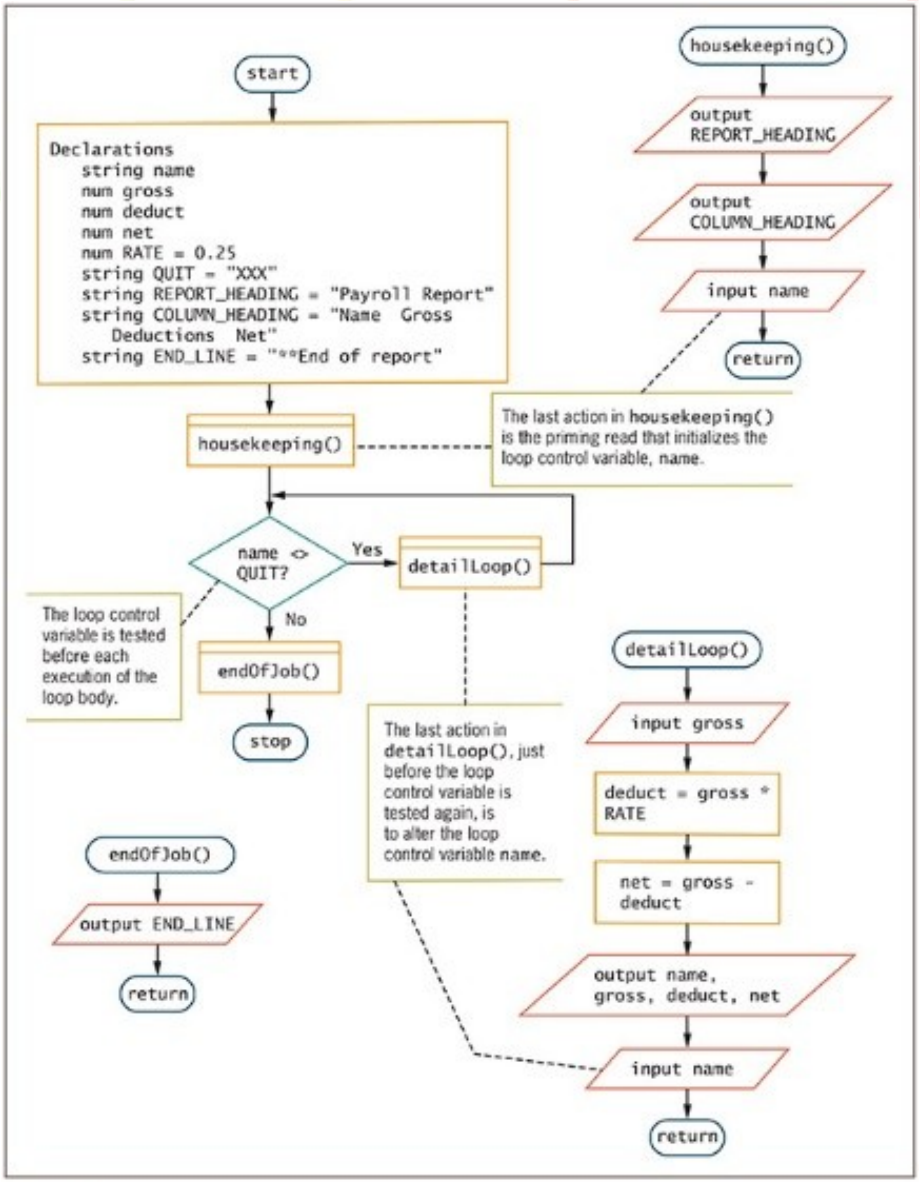


Figure 5-6 A payroll program showing how the loop control variable is used

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 -1)

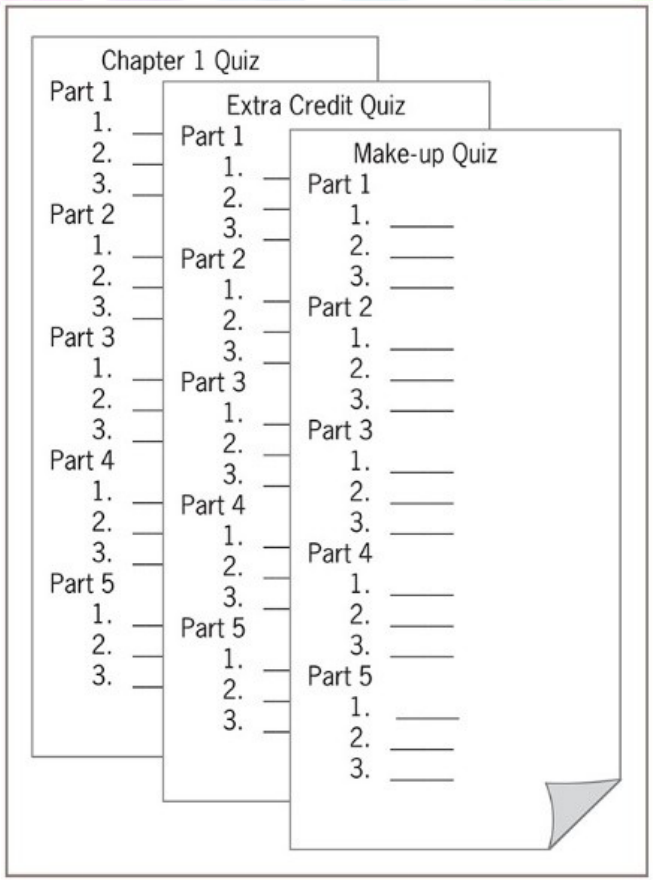


Figure 5-7 Quiz answer sheets

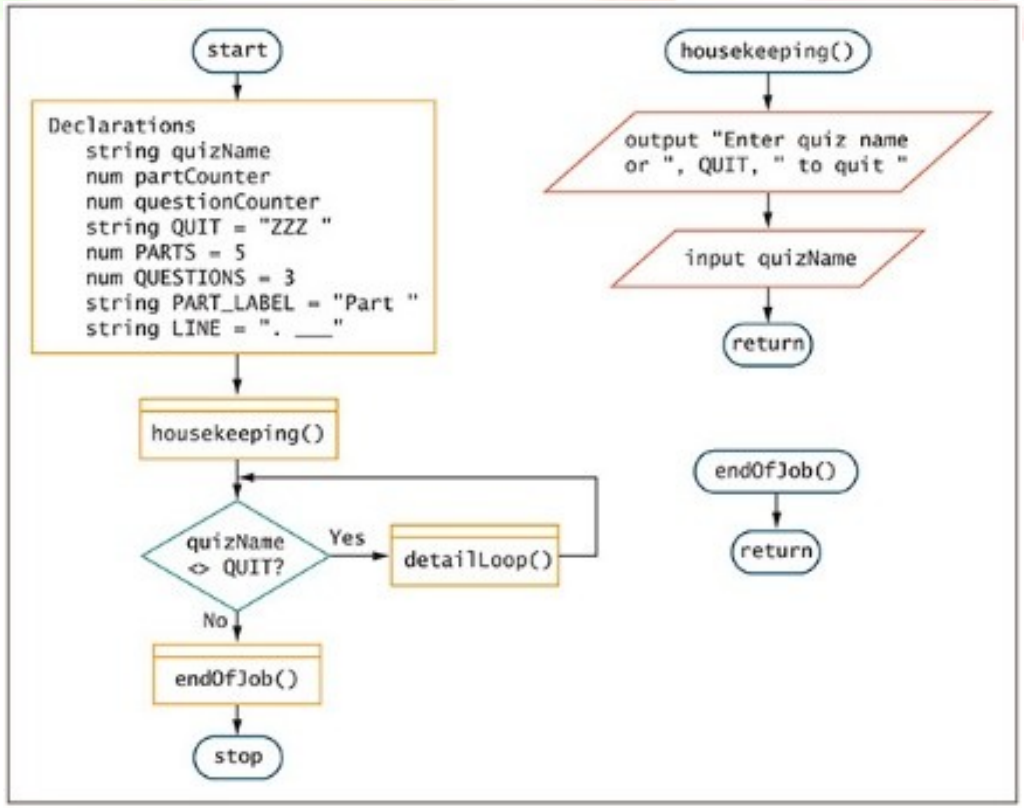


Figure 5-8 Flowchart and pseudocode for AnswerSheet program *(continues)*

Nested Loops (continued-2)

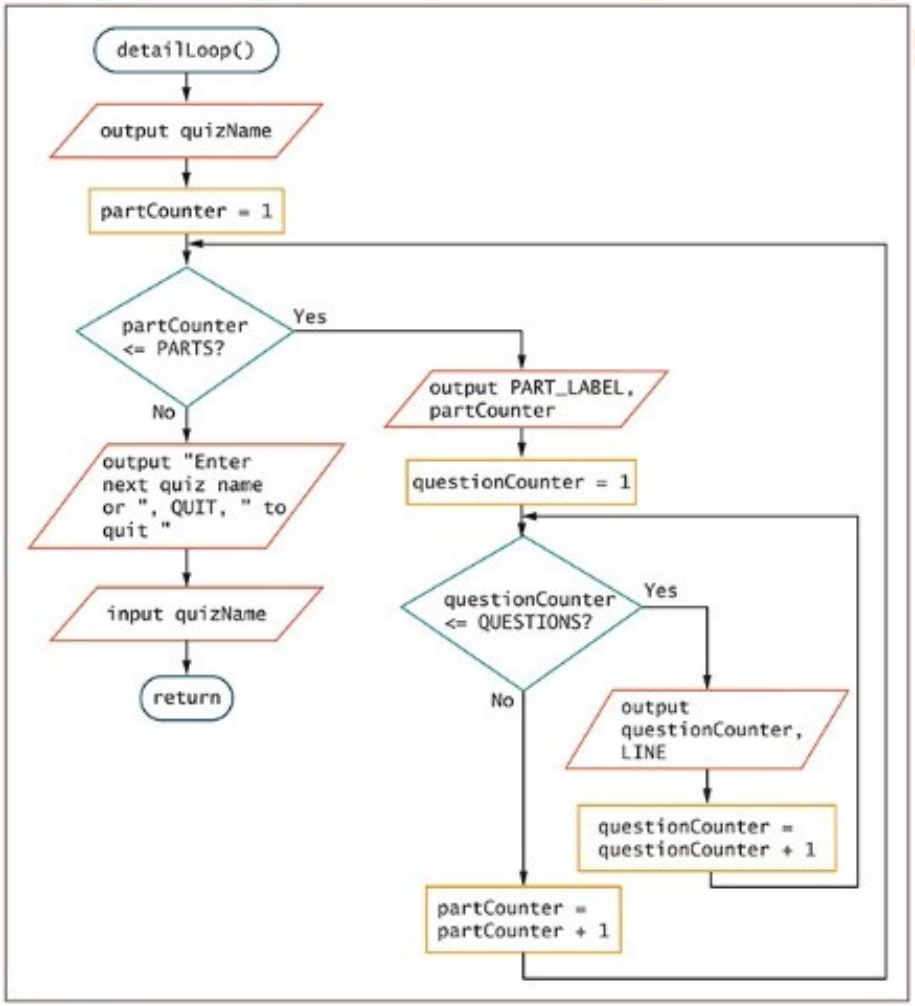


Figure 5-8 Flowchart and pseudocode for AnswerSheet program *(continues)*

Nested Loops (continued -3)

(continued)

```
start
   Declarations
      string quizName
      num partCounter
      num questionCounter
      string QUIT = "ZZZ"
      num PARTS = 5
      num QUESTIONS = 3
      string PART_LABEL = "Part "
      string LINE = ". _____"
   housekeeping()
   while quizName <> QUIT
      detailLoop()
   endwhile
   endOfJob()
stop

housekeeping()
   output "Enter quiz name or ", QUIT, " to quit "
   input quizName
return

detailLoop()
   output quizName
   partCounter = 1
   while partCounter <= PARTS
      output PART_LABEL, partCounter
      questionCounter = 1
      while questionCounter <= QUESTIONS
         output questionCounter, LINE
         questionCounter = questionCounter + 1
      endwhile
      partCounter = partCounter + 1
   endwhile
   output "Enter next quiz name or ", QUIT, " to quit "
   input quizName
return

endOfJob()
return
```

Figure 5-8 Flowchart and pseudocode for AnswerSheet program

Nested Loops (continued -4)

- **Nested Loop facts:**
  - Nested loops never overlap. An inner loop is always completely contained within an outer loop
  - An inner loop goes through all of its iterations each time its outer loop goes through just one iteration
  - The total number of iterations executed by a nested loop is the number of inner loop iterations times the number of outer loop iterations

Avoiding Common Loop Mistakes

- Mistake: failing to initialize the loop control variable
  - Example: get name statement remove
    - Value of name unknown or garbag
    - Program may end before any labels printed
    - 100 labels printed with an invalid nan

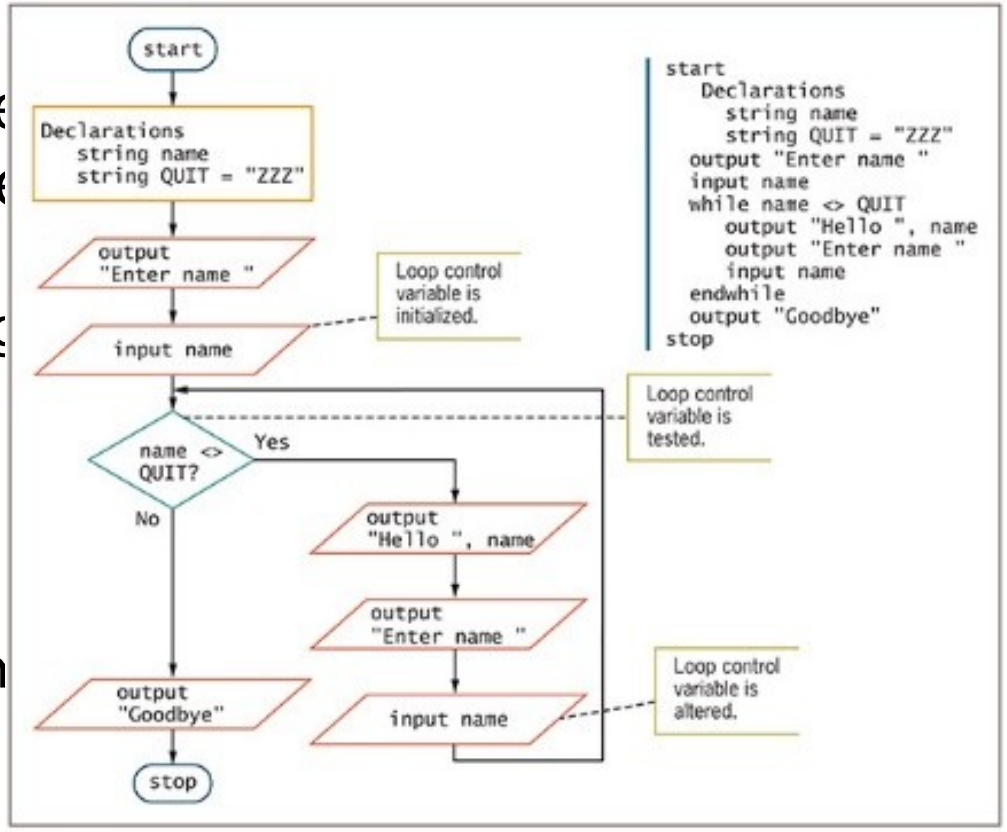


Figure 5-9 Correct logic for greeting program

Avoiding Common Loop Mistakes (continued -1)

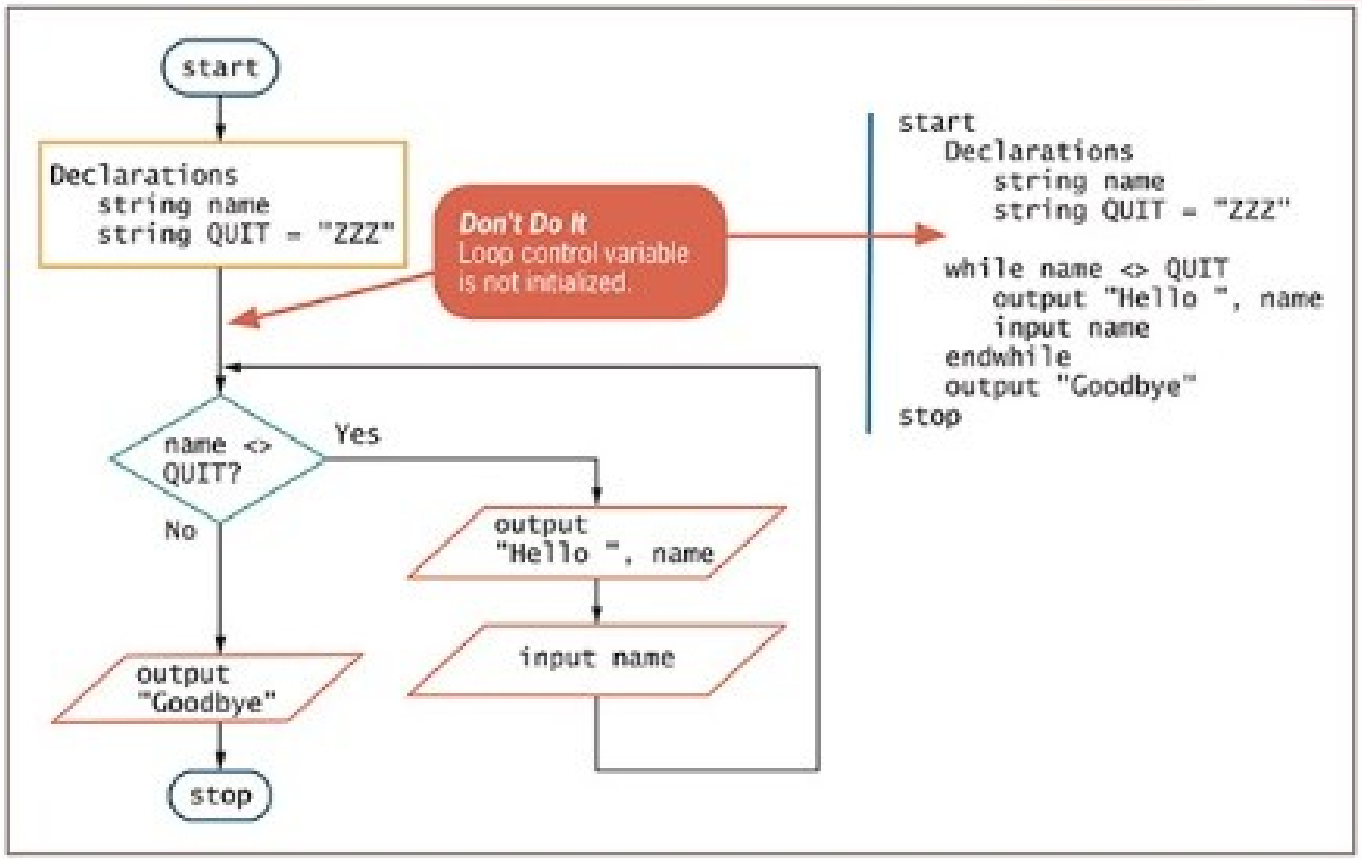


Figure 5-10 Incorrect logic for greeting program because the loop control variable initialization is missing

Avoiding Common Loop Mistakes (continued -2)

- 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 t create a loop that cannot terminate

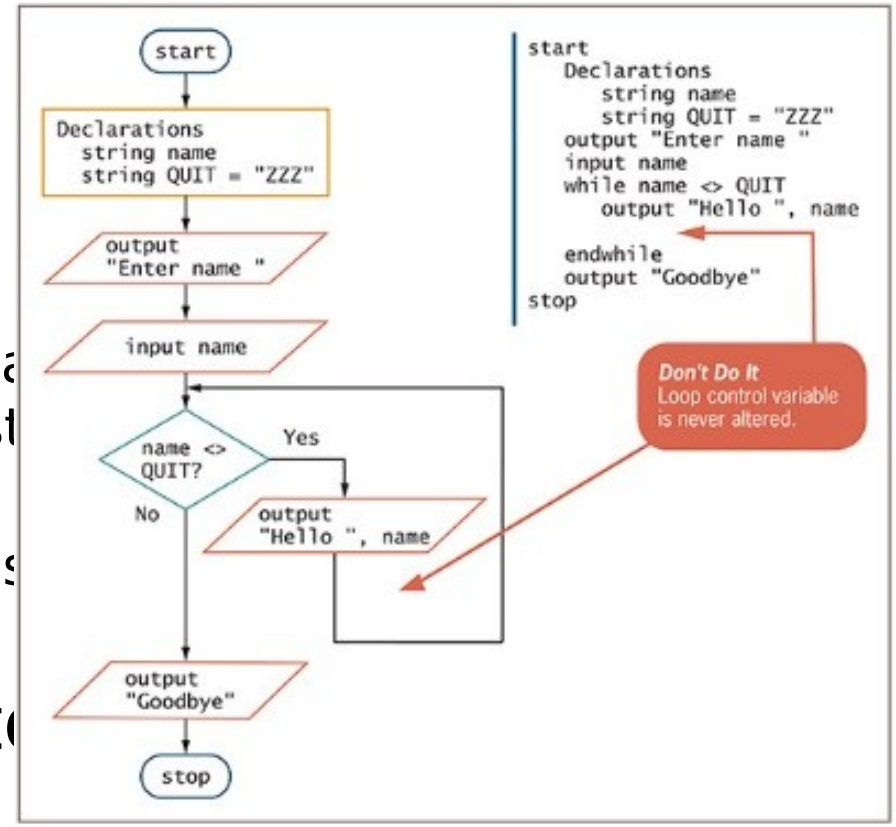


Figure 5-11 Incorrect logic for greeting program because the loop control variable is not altered

Avoiding Common Loop Mistakes (continued -3)

- Mistake: using the wrong comparison when testing 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

Avoiding Common Loop Mistakes (continued 4)

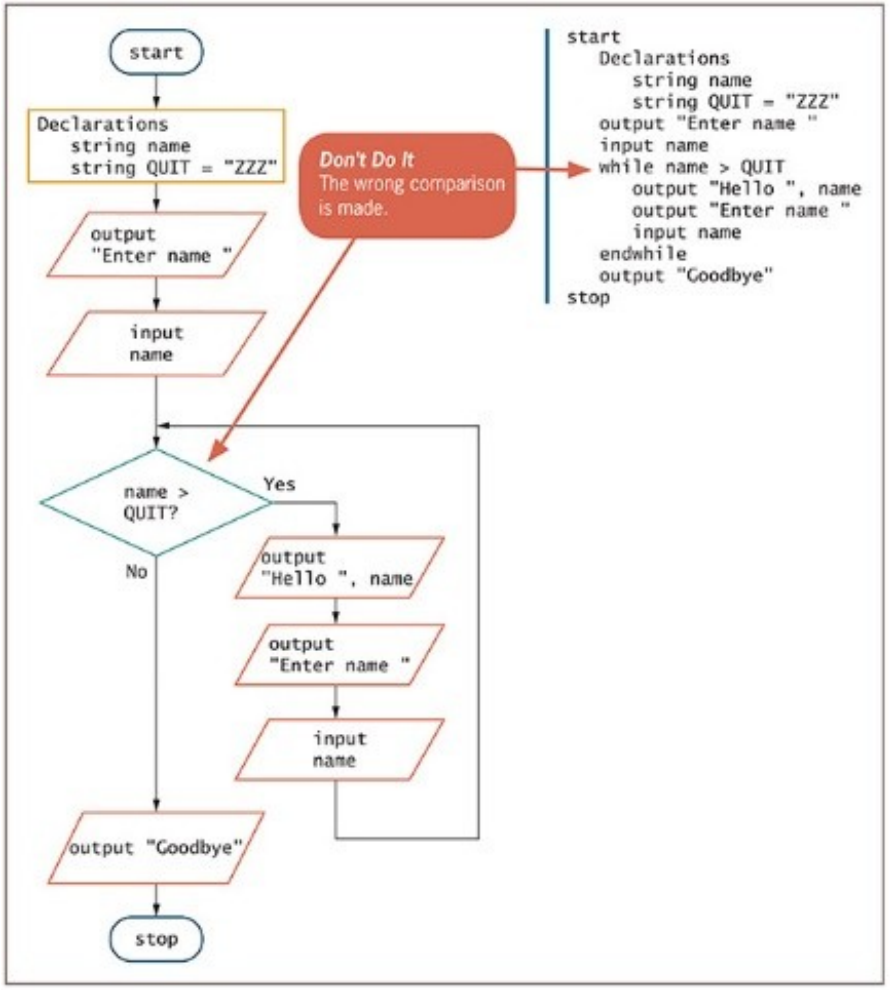


Figure 5-12 Incorrect logic for greeting program because the wrong test is made with the loop control variable

Avoiding Common Loop Mistakes (continued -5)

- 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

Avoiding Common Loop Mistakes (continued -6)

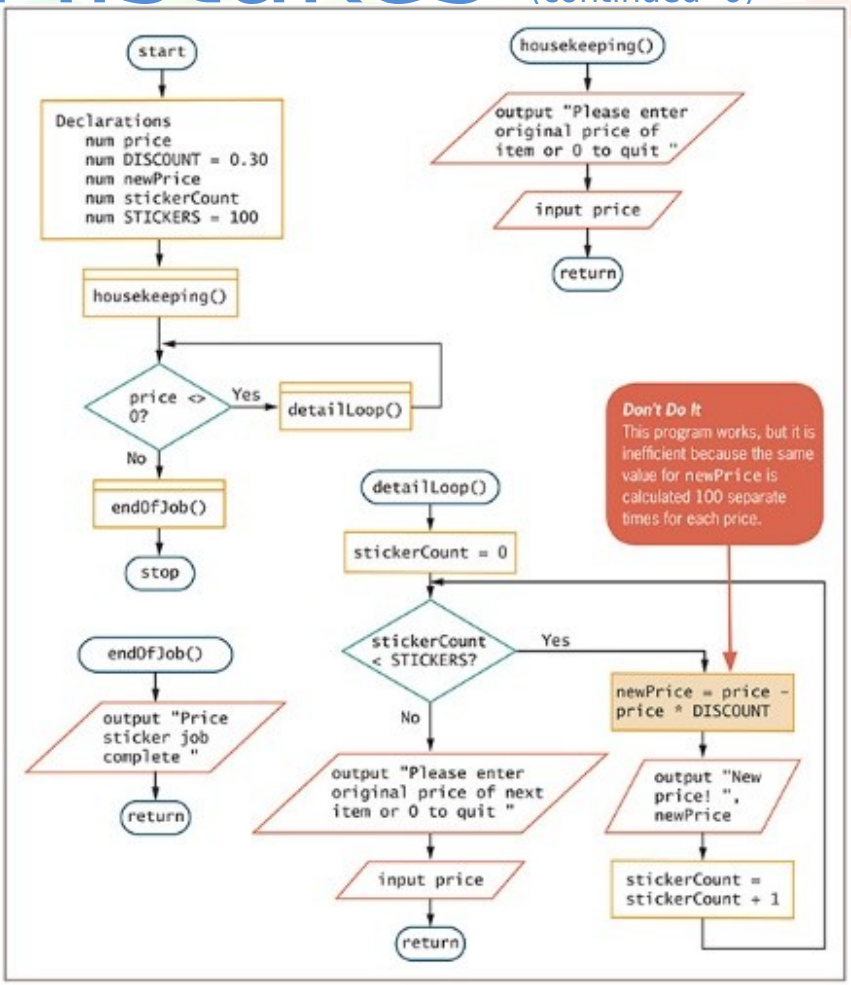


Figure 5-13 Inefficient way to produce 100 discount price stickers for differently priced items
(continues)

Avoiding Common Loop Mistakes

(continued)

```
start
   Declarations
      num price
      num DISCOUNT = 0.30
      num newPrice
      num stickerCount
      num STICKERS = 100
   housekeeping()
   while price <> 0
      detailLoop()
   endwhile
   endOfJob()
stop

housekeeping()
   output "Please enter original price of item or 0 to quit "
   input price
return

detailLoop()
   stickerCount = 0
   while stickerCount < STICKERS
      newPrice = price - price * DISCOUNT
      output "New price! ", newPrice
      stickerCount = stickerCount + 1
   endwhile
   output "Please enter original price of
      next item or 0 to quit "
   input price
return

endOfJob()
   output "Price sticker job complete"
return
```

Don't Do It
This program works, but it is inefficient because the same value for `newPrice` is calculated 100 separate times for each price.

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

Avoiding Common Loop Mistakes

(continued -8)

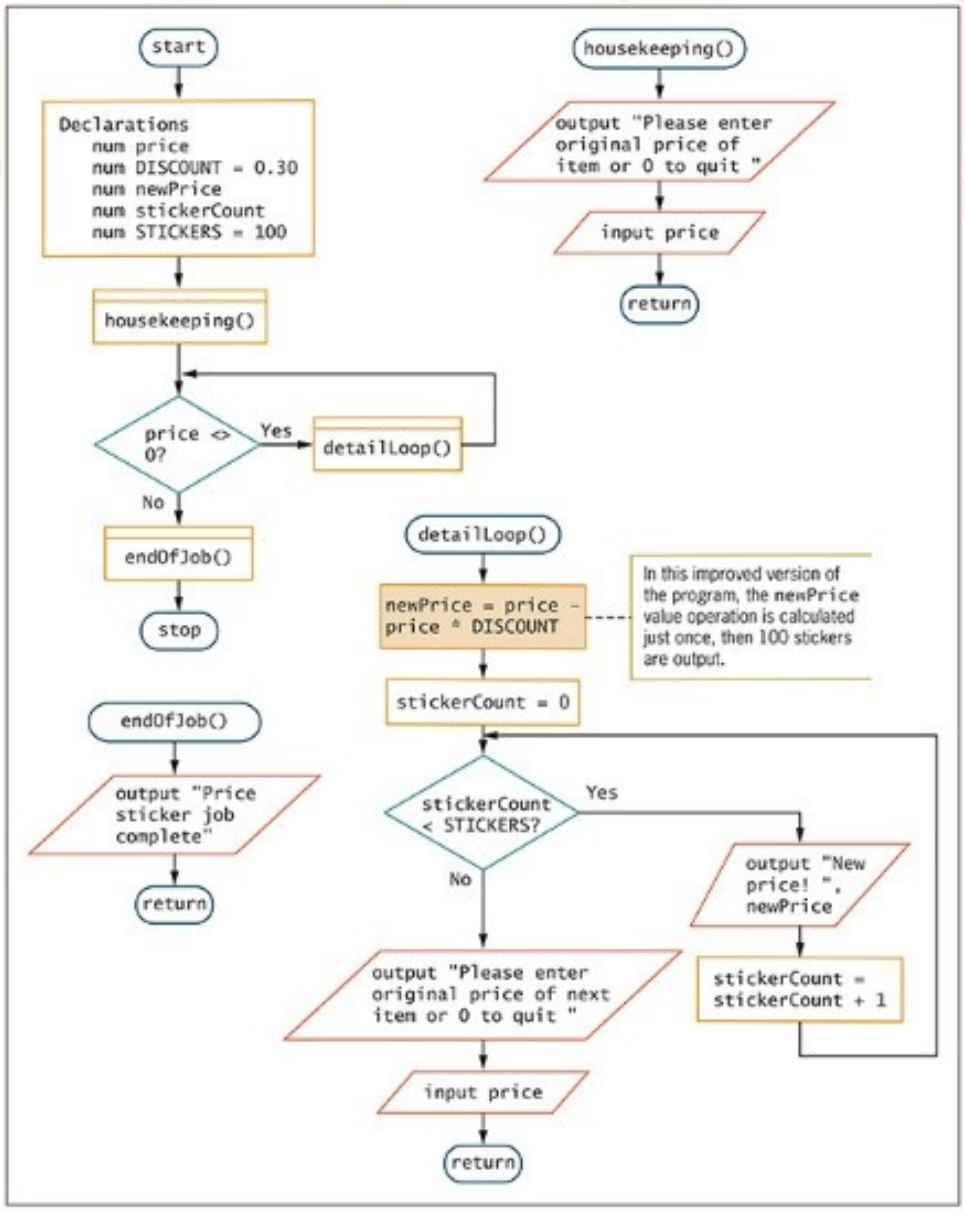


Figure 5-14 Improved discount sticker-making program *(continues)*

Avoiding Common Loop Mistakes

(continued -9)

(continued)

```
start
   Declarations
      num price
      num DISCOUNT = 0.30
      num newPrice
      num stickerCount
      num STICKERS = 100
   housekeeping()
   while price <> 0
      detailLoop()
   endwhile
   endOfJob()
stop

housekeeping()
   output "Please enter original price of item or 0 to quit "
   input price
return

detailLoop()
   newPrice = price - price * DISCOUNT
   stickerCount = 0
   while stickerCount < STICKERS
      output "New price! ", newPrice
      stickerCount = stickerCount + 1
   endwhile
   output "Please enter original price of next item or 0 to quit "
   input price
return

endOfJob()
   output "Price sticker job complete"
return
```

In this improved version of the program, the newPrice value operation is calculated just once, then 100 stickers are output.

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
  - Tests
  - Alters

```
count = 0
while count <= 3
   output "Hello"
   count = count + 1
endwhile
```

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

Figure 5-15 Comparable while and for statements that each output *Hello* four times

Using a for Loop (continued -1)

- 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 **step value**

Using a for Loop (continued -2)

- **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 -3)

- `while` statement could be used in place of `for` statement
- **Pretest loop**: the loop control variable is tested before each iteration
  - `for` loops and `while` loops are pretest loops

Using a Posttest Loop

- **Posttest loop**: the loop control variable is tested after each iteration
- In a **posttest loop**, the loop body executes at least one time because the loop control variable is not tested until after the first iteration.
  - **do-while loop** is a posttest loop
- Example

```
do
    pay a bill
while more bills remain to be paid
```

Using a Posttest Loop (continued -1)

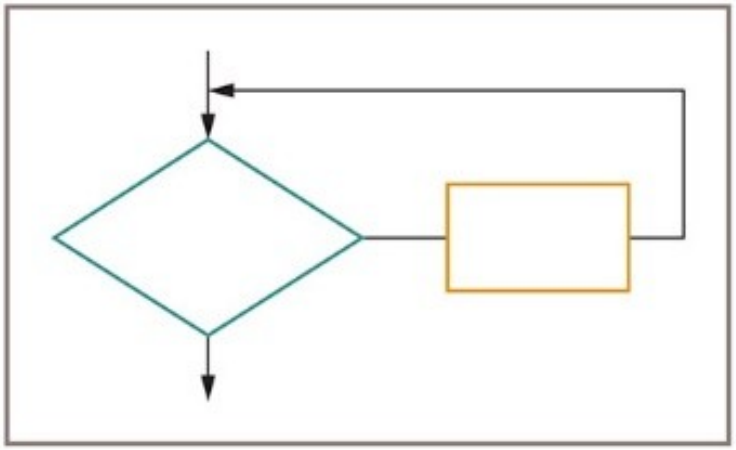


Figure 5-16 The `while` loop, which is a pretest

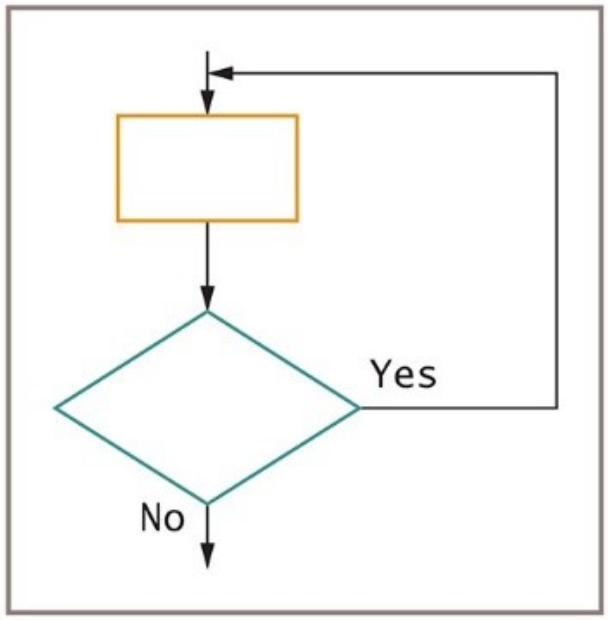


Figure 5-17 Structure of a `do-while` loop, which is a posttest loop

Using a Posttest Loop (continued -2)

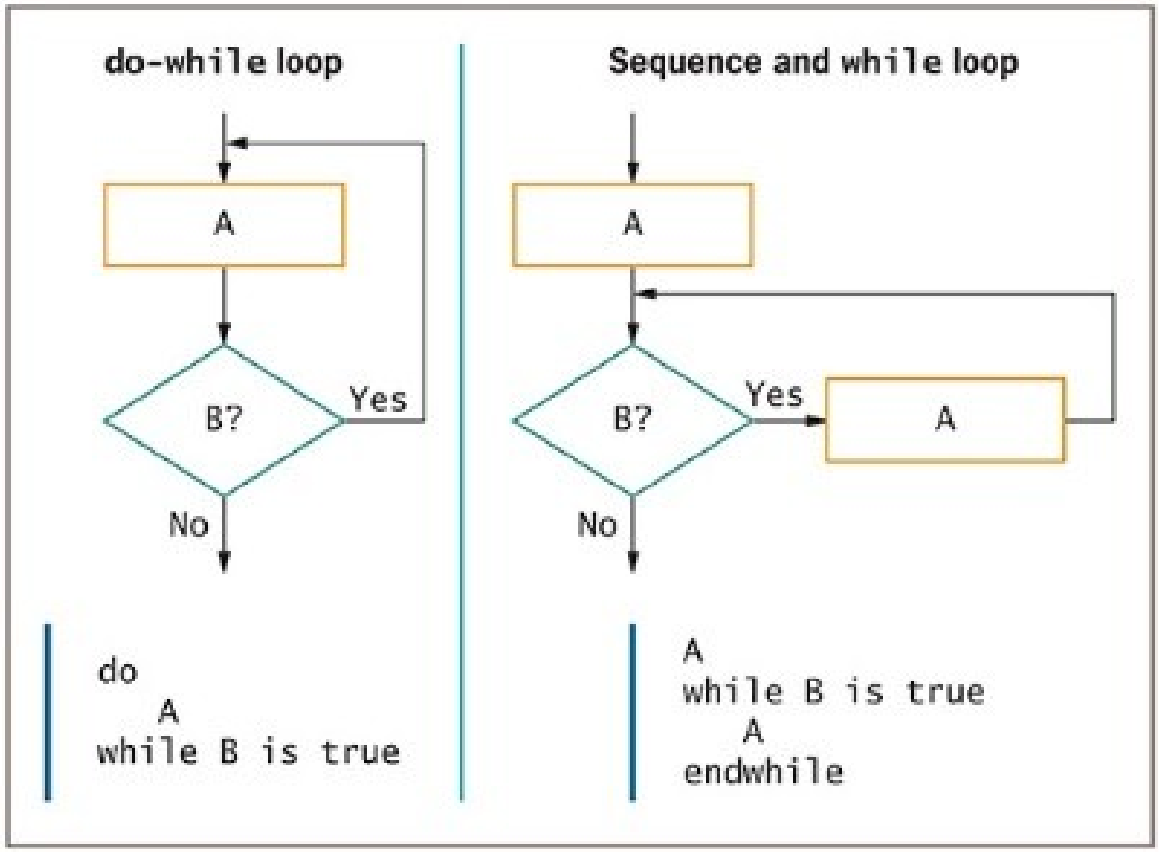


Figure 5-18 Flowchart and pseudocode for do-while loop and while loop that do the same thing

Recognizing the Characteristics Shared by Structured Loop

- All structured loops have these characteristics:
  - The loop-controlling evaluation must provide either the entry to or exit from the structure
  - The loop-controlling evaluation provides the only entry to or exit from the structure
- Some languages support a **do-until loop**, which is a posttest loop that iterates until the loop controlling evaluation is false

Recognizing the Characteristics Shared by Structured Loop (continued -1)

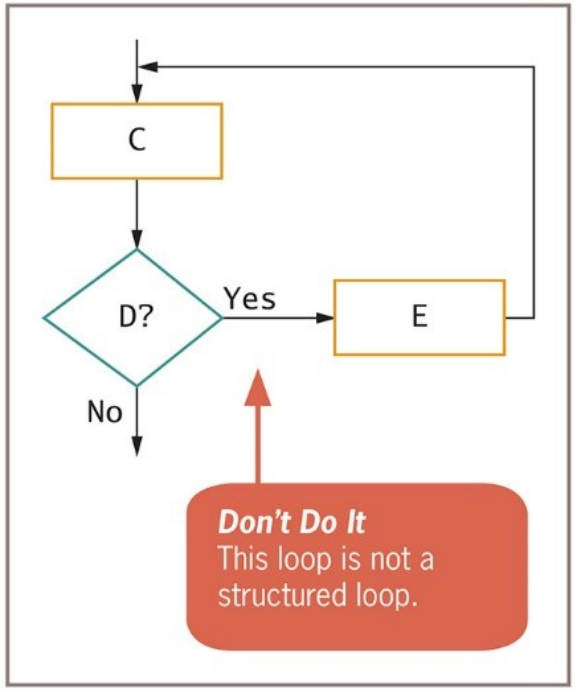


Figure 5-19 Unstructured loop

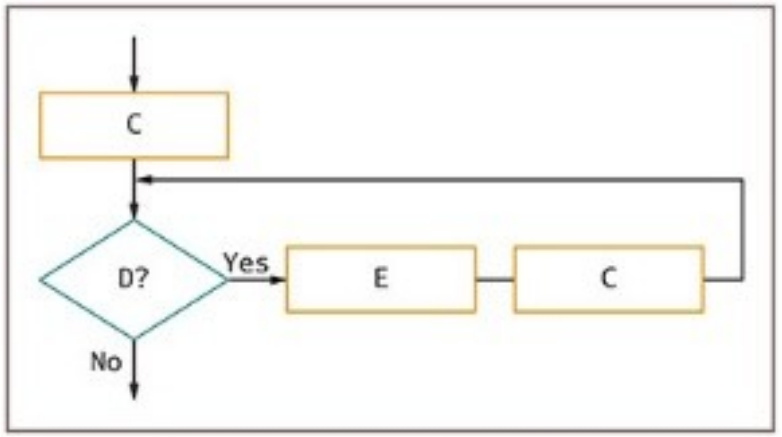


Figure 5-20 Sequence and structured loop that accomplish the same tasks as Figure 5-19

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

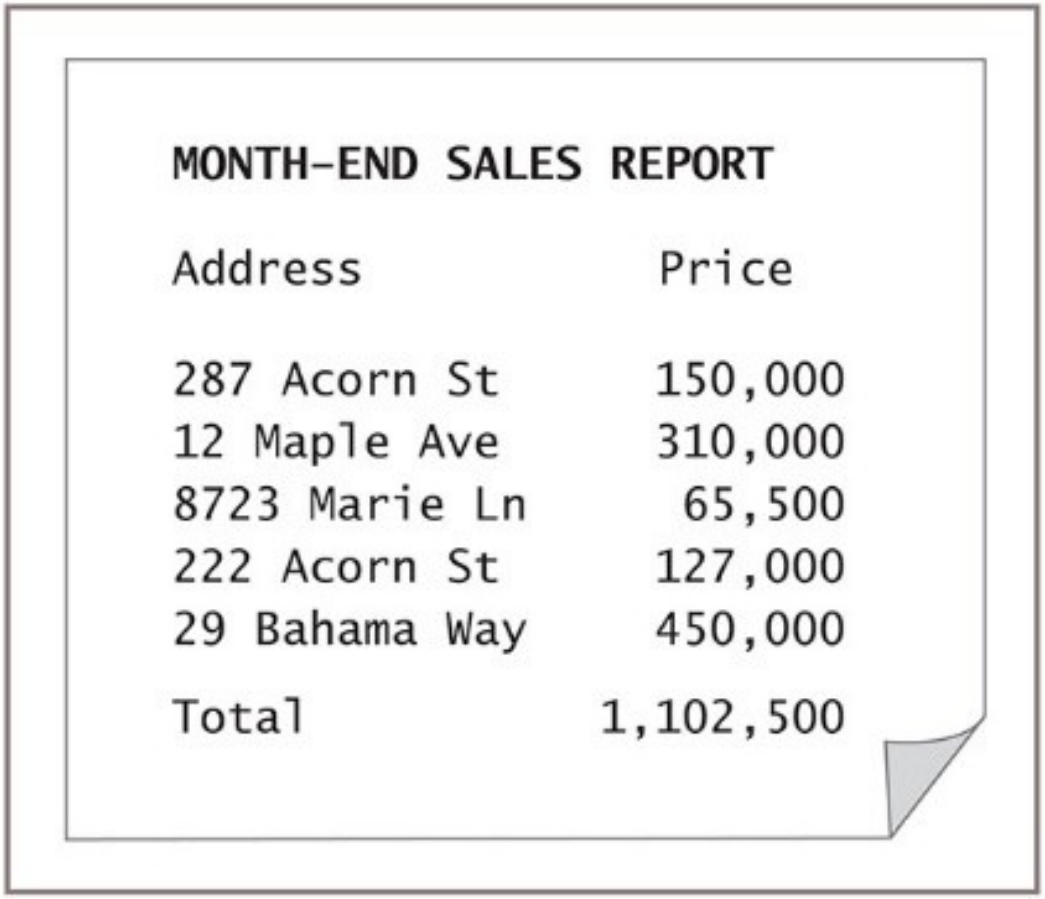
Common Loop Applications

(continued -1)

- 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 -2)



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 5-21 Month-end real estate sales report

Common Loop Applications

(continued -3)

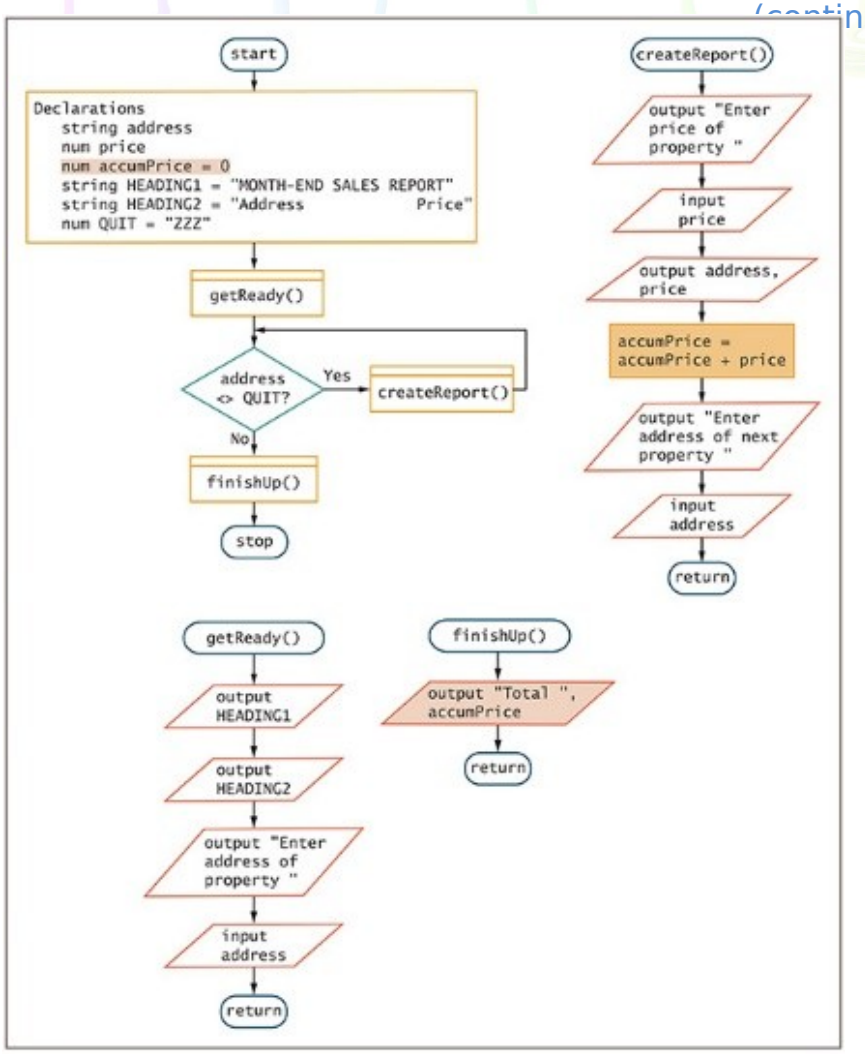


Figure 5-22 Flowchart and pseudocode for real estate sales report program *(continues)*

(continued)

```
start
   Declarations
      string address
      num price
      num accumPrice = 0
      string HEADING1 = "MONTH-END SALES REPORT"
      string HEADING2 = "Address            Price"
      num QUIT = "ZZZ"
   getReady()
   while address <> QUIT
      createReport()
   endwhile
   finishUp()
stop

getReady()
   output HEADING1
   output HEADING2
   output "Enter address of property "
   input address
return

createReport()
   output "Enter price of property "
   input price
   output address, price
   accumPrice = accumPrice + price
   output "Enter address of next property "
   input address
return

finishUp()
   output "Total ", accumPrice
return
```

Figure 5-22 Flowchart and pseudocode for real estate sales report program

Common Loop Applications

(continued -4)

- 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

Common Loop Applications

(continued -5)

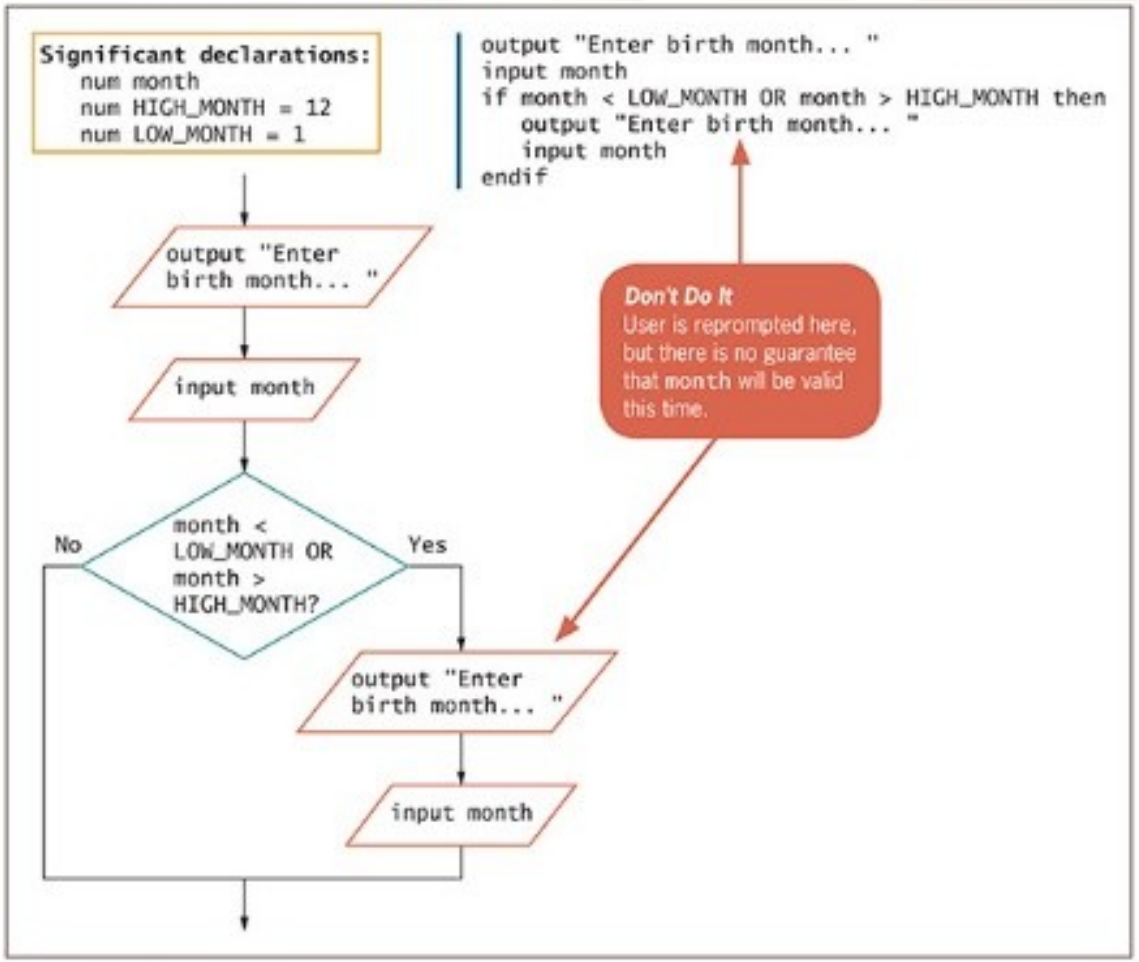


Figure 5-23 Reprompting a user once after an invalid month is entered

Common Loop Applications

(continued -6)

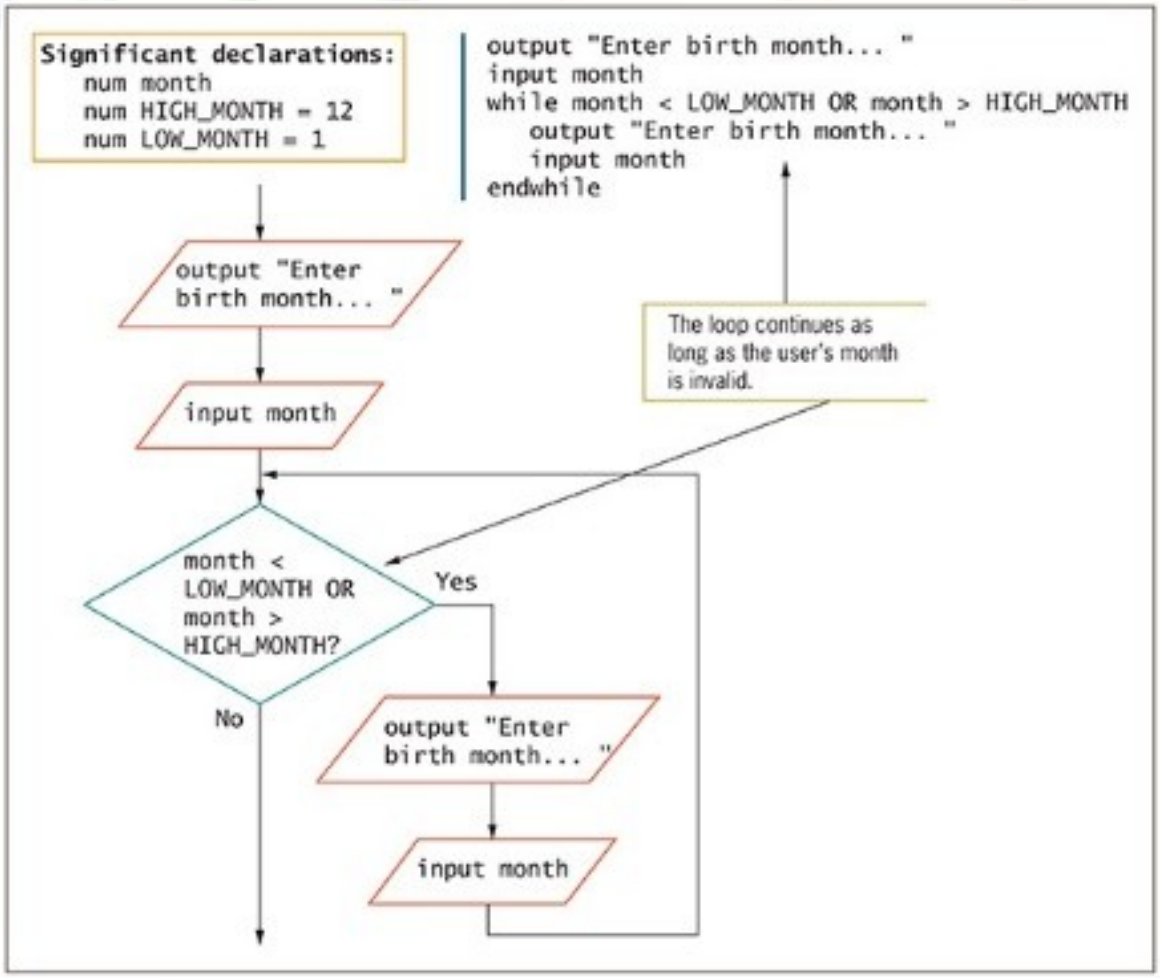


Figure 5-24 Reprompting a user continuously after an invalid month is entered

Common Loop Applications

(continued -7

- 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 -8)

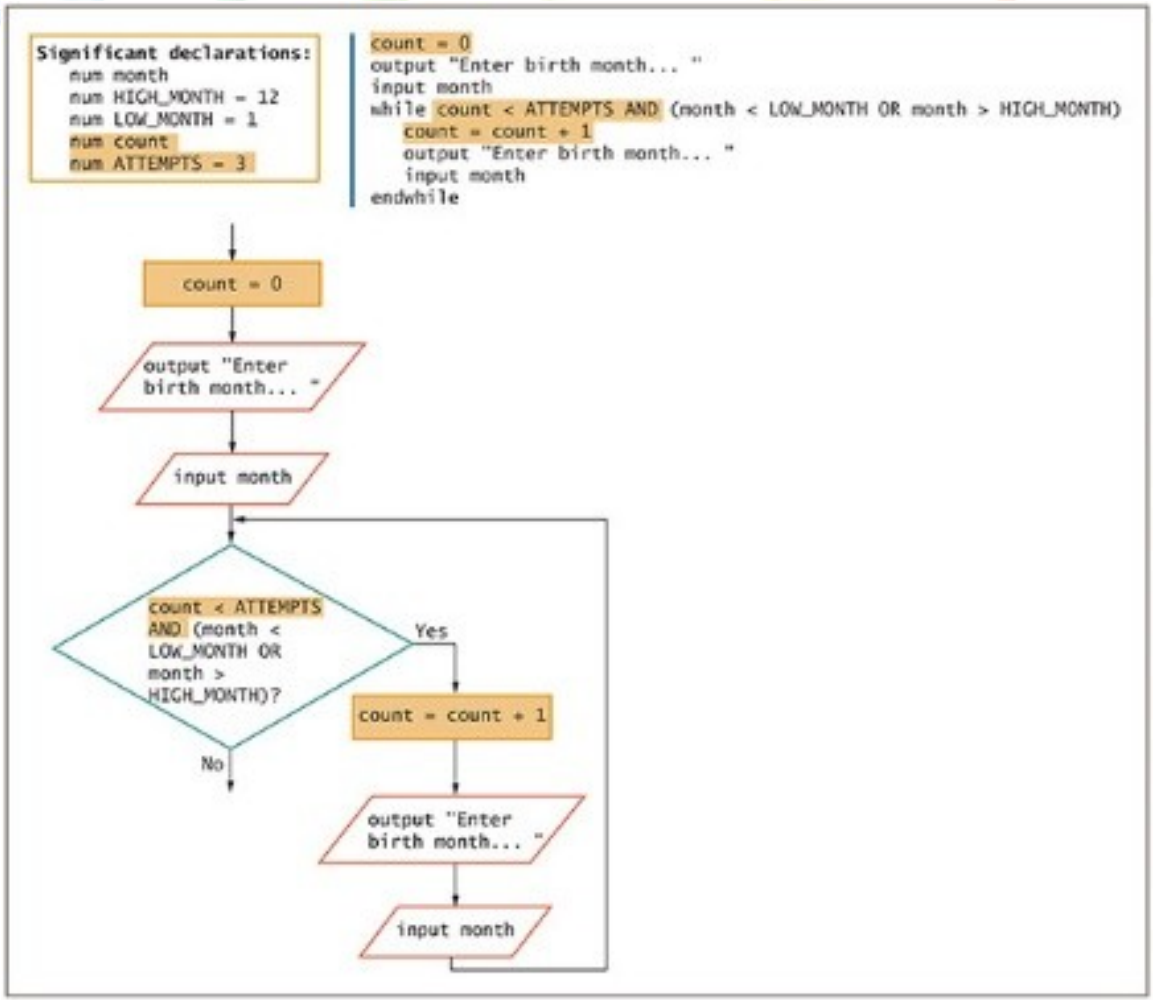


Figure 5-25 Limiting user reprompts

Common Loop Applications

(continued -9)

- 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()` or `isUpper()`
  - Accept user data as strings
  - Use built-in methods to convert to correct data types

Common Loop Applications

(continued -10)

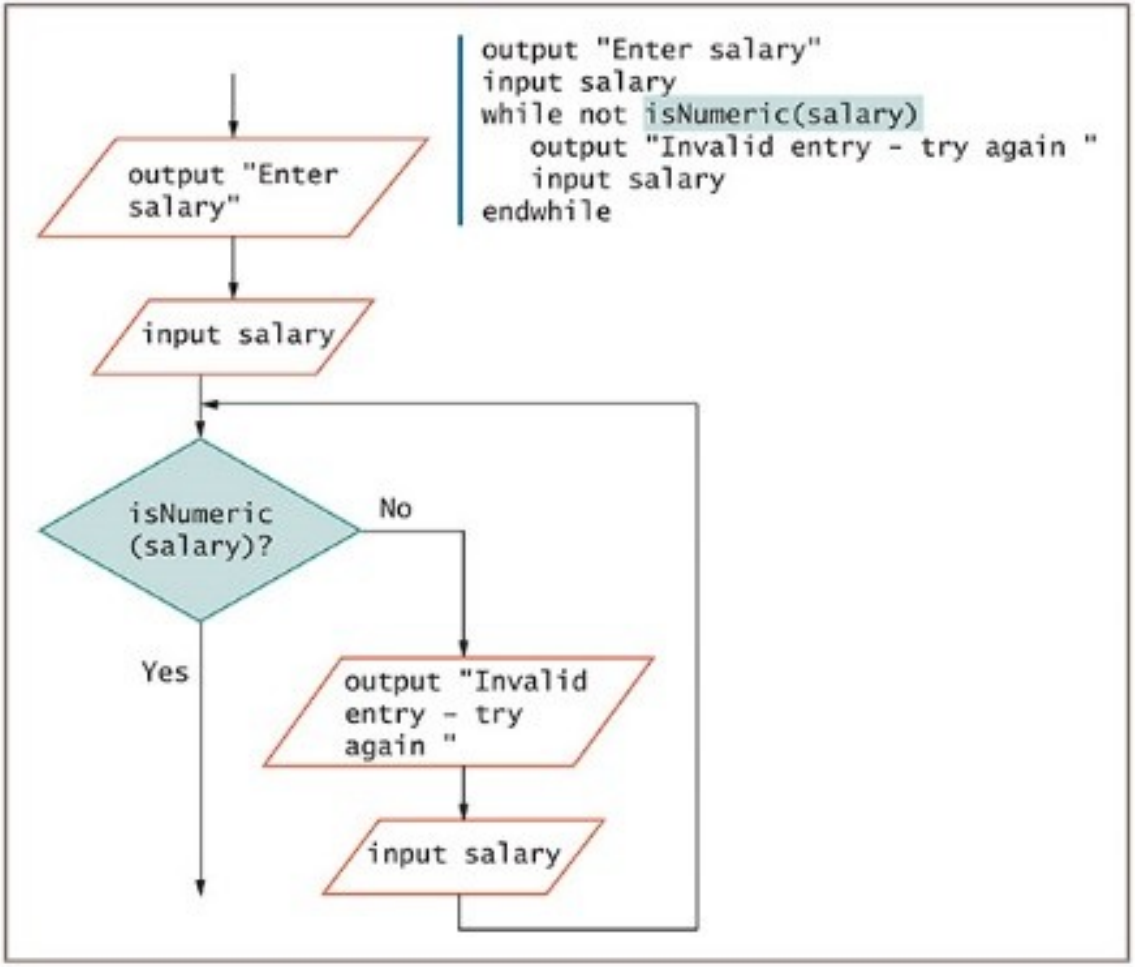


Figure 5-26 Checking data for correct type

Common Loop Applications

(continued -11)

- 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

Common Loop Applications

(continued -12)

- **Comparing Selecti and Loops**
- Selection Structure
  - The two logical pa (True and False) join together
- Loop Structure
  - One of the logical branches returns t the same decision

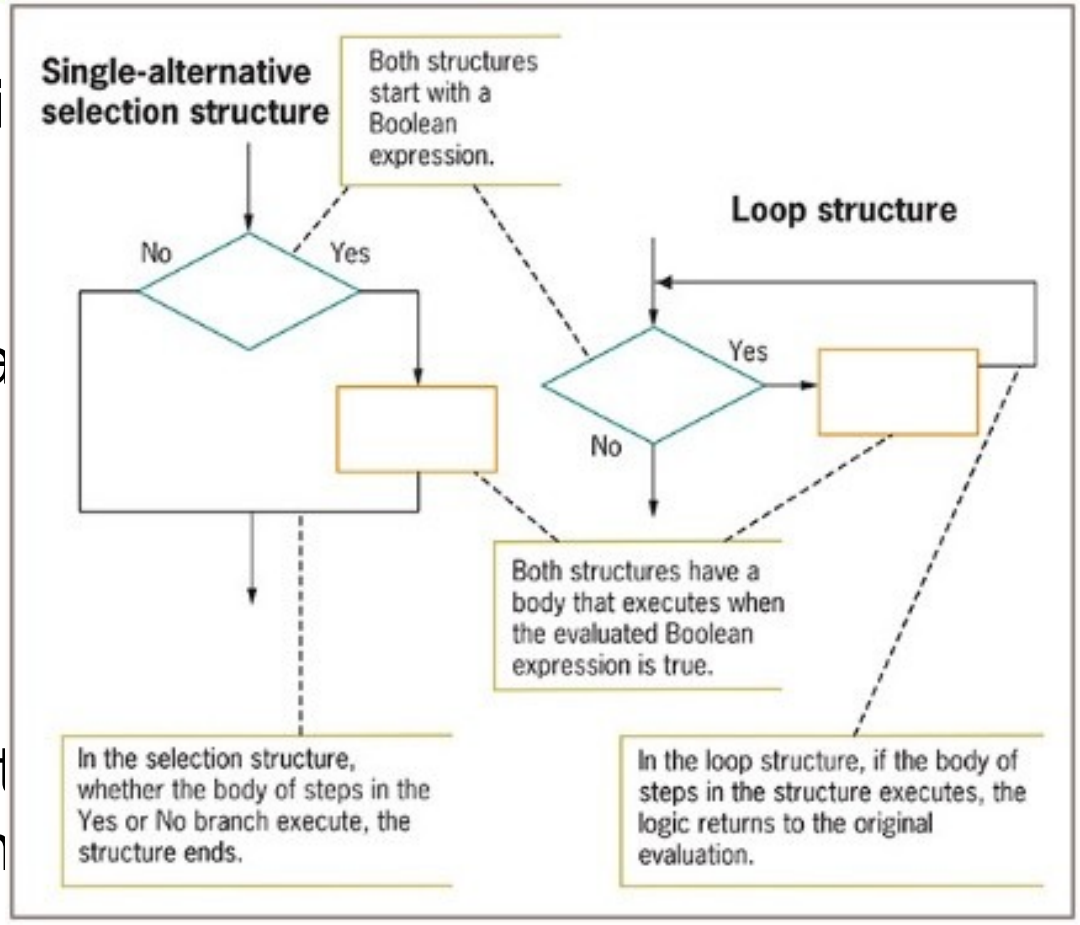


Figure 5-27 Comparing a selection and a loop

Common Loop Applications (continued -13)

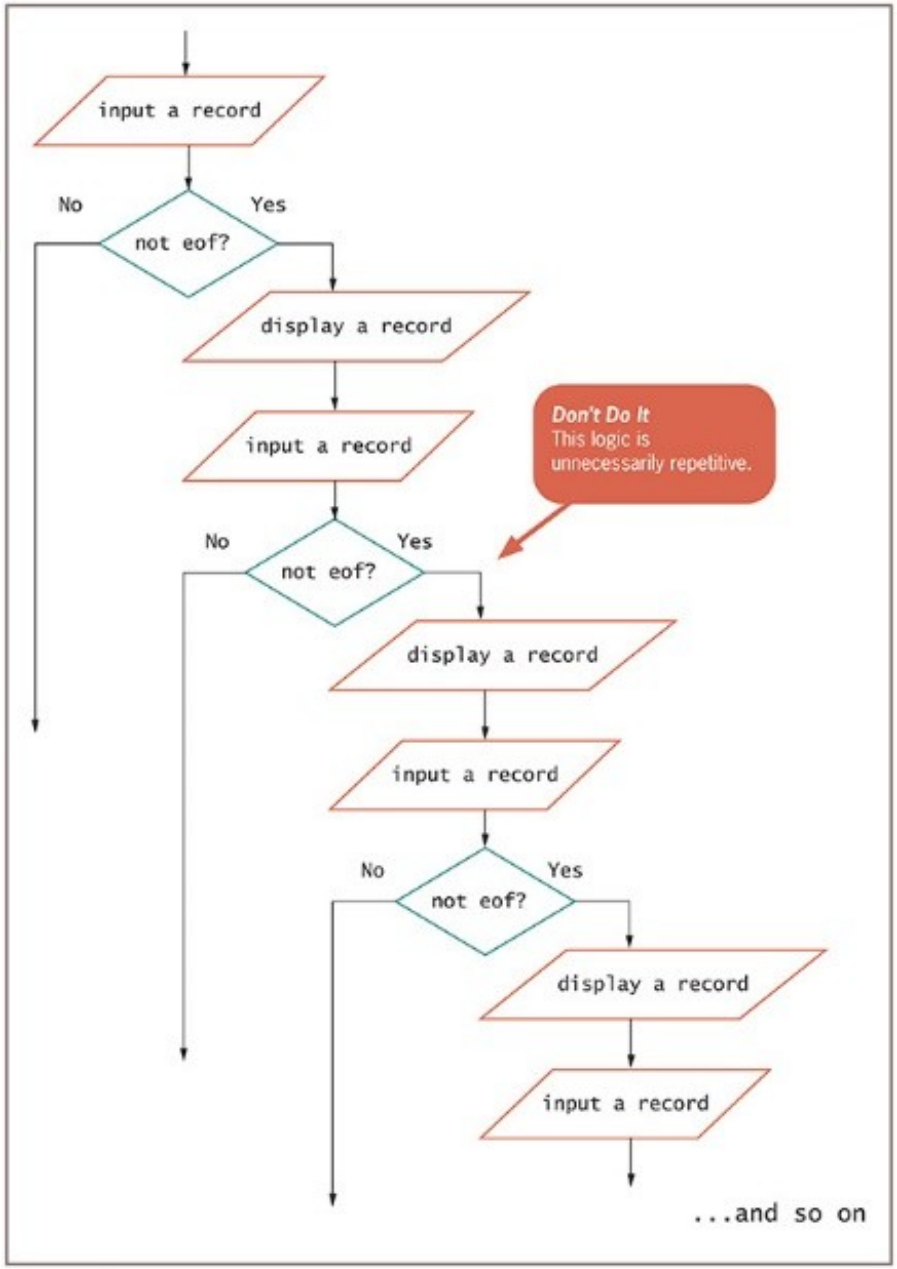


Figure 5-28 Inefficient logic for reading and displaying employee records

Common Loop Applications

(continued -14)

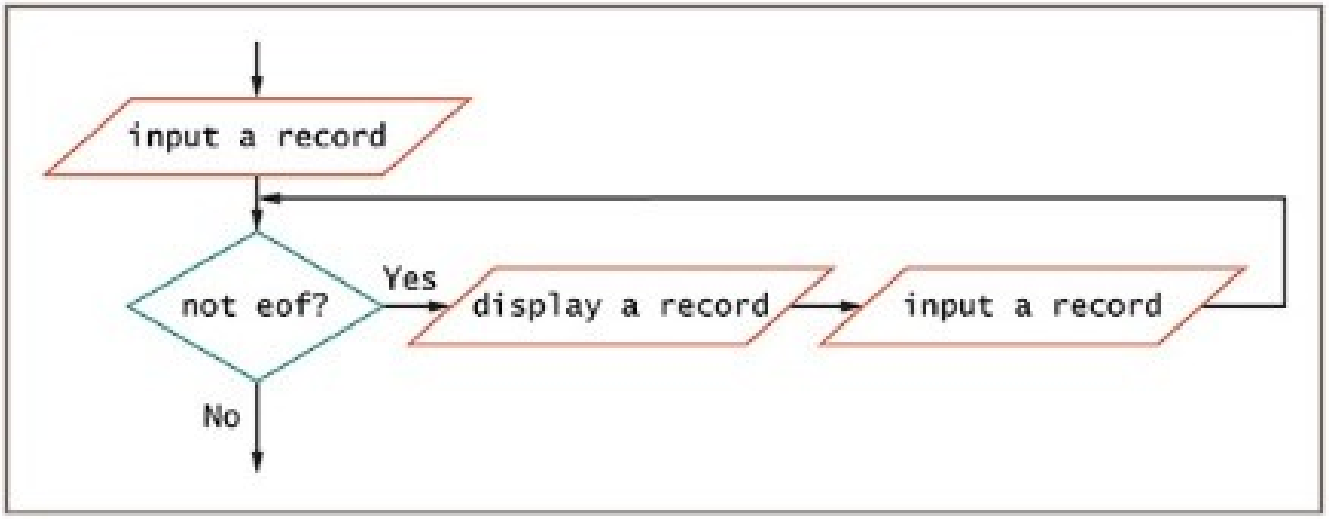


Figure 5-29 Efficient and structured logic for getting and displaying employee records

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 -1)

- 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` loop
  - `for` loop used when the number of iterations is known
- In a posttest loop, the loop body executes at least one time because the loop control

Summary (continued -2)

- Loops are used to accumulate totals in business reports and to reprompt users for valid data
- In the selection structure the two logical paths that emerge from a test join together following their actions
- In the loop structure, the paths that emerge from the test do not join together, instead, one of the paths eventually returns to the same test