Programming Logic and Design
Sixth 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: structure that repeats actions while some condition continues

Understanding the Advantages of Looping (continued)

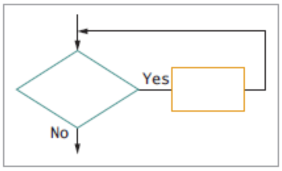


Figure 5-1 The loop structure

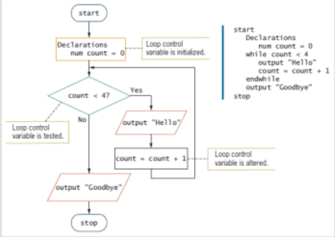
Using a Loop Control Variable

- As long as a Boolean expression remains true, `while` loop's body executes
- 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** predetermined number of times
- **Counter-controlled** loop
  - Program counts loop repetitions
- Loop control variables altered by:
  - **Incrementing**
  - **Decrementing**

Using a Definite Loop with a Counter (continued)

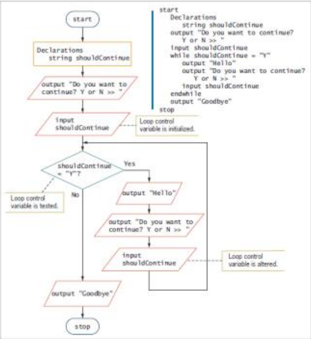

start
Declarations
num count = 0
Loop control variable is initialized.
count < 4?
Yes
No
Loop control variable is tested.
output "Hello"
count = count + 1
Loop control variable is altered.
output "Goodbye"
stop

```
start
   Declarations
      num count = 0
   while count < 4
      output "Hello"
      count = count + 1
   endwhile
   output "Goodbye"
stop
```

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
- Three crucial steps
  - Starting value to control the loop must be provided
  - Comparison must be made using the value that controls the loop
  - Within the loop, value that controls the loop must be altered


start
Declarations
string shouldContinue
output "Do you want to continue? Y or N >> "
input shouldContinue
Loop control variable is initialized.
shouldContinue = "Y"?
Yes
No
Loop control variable is tested.
output "Hello"
output "Do you want to continue? Y or N >> "
input shouldContinue
Loop control variable is altered.
output "Goodbye"
stop

```
start
   Declarations
      string shouldContinue
   output "Do you want to continue?
      Y or N >> "
   input shouldContinue
   while shouldContinue = "Y"
      output "Hello"
      output "Do you want to continue?
         Y or N >> "
      input shouldContinue
   endwhile
   output "Goodbye"
stop
```

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

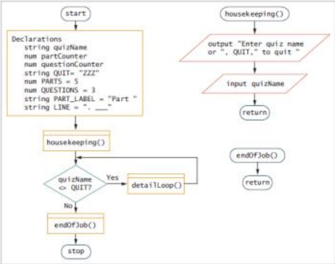


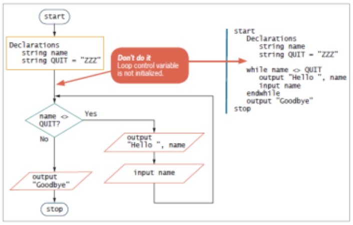
Figure 5-8 Flowchart and pseudocode for `AnswerSheet` program

Avoiding Common Loop Mistakes

- 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

Avoiding Common Loop Mistakes (continued)

- 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



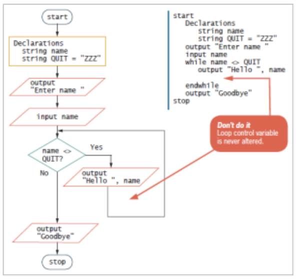
Don't do it
Loop control variable is not initialized.

```
start
   Declarations
      string name
      string QUIT = "ZZZ"

   while name <> QUIT
      output "Hello ", name
      input name
   endwhile
   output "Goodbye"
stop
```

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



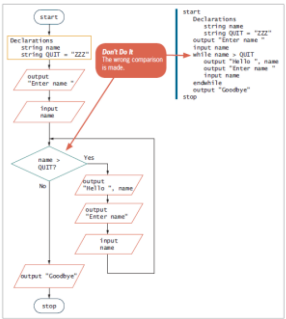
```
start
   Declarations
      string name
      string QUIT = "ZZZ"
   output "Enter name "
   input name
   while name <> QUIT
      output "Hello ", name

   endwhile
   output "Goodbye"
stop
```

Don't do it
Loop control variable is never altered.

Figure 5-10 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



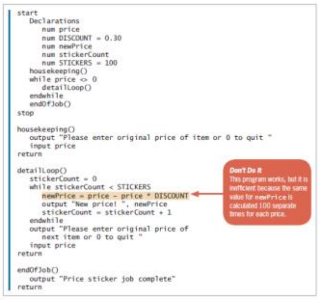
Don't Do It
The wrong comparison is made.

```
start
   Declarations
      string name
      string QUIT = "ZZZ"
   output "Enter name "
   input name
   while name > QUIT
      output "Hello ", name
      output "Enter name "
      input name
   endwhile
   output "Goodbye"
stop
```

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

Avoiding Common Loop Mistakes (continued)

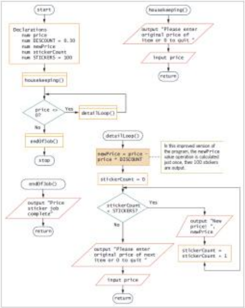
- Mistake: including statements inside the loop that belong outside the loop
  - Example: discount every item by 30 percent
  - Inefficient because the same value is calculated 100 separate times for each price that is entered
  - Move outside loop for efficiency



```
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



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
  - Evaluates
  - Increments
- 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` to 0
- Checks `count` against the limit value 3
- If evaluation is true, `for` statement body prints the label
- Increases `count` by 1

Using a `for` Loop (continued)

- `while` statement could be used in place of `for` statement
- **Step value**: number used to increase a loop control variable on each pass through a loop
  - Programming languages can:
    - Require a statement that indicates the step value
    - Have a step value default of 1
- Specify step value when each pass through the loop changes the loop control variable by value other than 1

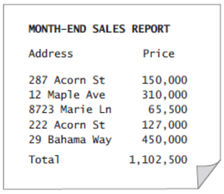
Common Loop Applications

- 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 one
    - Accumulator increments by some value

Common Loop Applications (continued)

- Accumulate total real estate prices
  - Declare numeric variable at beginning
  - Initialize the accumulator to 0
  - Read each transaction's data record
  - Add its value to accumulator variable
  - Read the next record until `eof`
- Variables exist only for the life of the application
  - Run the application a second time; variables occupy different memory location

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

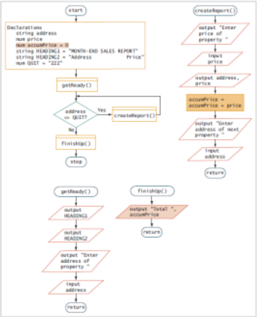
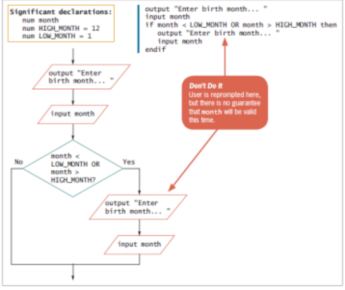


Figure 5-17 Flowchart and pseudocode for real estate sales report program

Common Loop Applications (continued)

- Using a loop to validate data
  - When prompting a user for data, no guarantee that data is valid
- **Validate data**: make sure data falls in acceptable ranges
- Example: user enters birth month
  - If number is less than 1 or greater than 12
    - Display error message and stop the program
    - Assign default value for the month
    - Reprompt the user for valid input

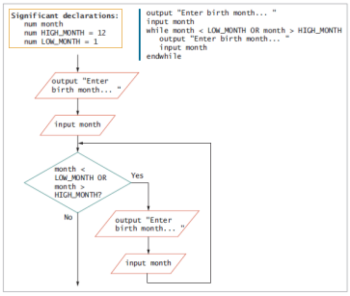


```
Significant declarations:
   num month
   num HIGH_MONTH = 12
   num LOW_MONTH = 1
```

```
output "Enter birth month... "
input month
if month < LOW_MONTH OR month > HIGH_MONTH then
   output "Enter birth month... "
   input month
endif
```

output "Enter birth month... "
input month
month < LOW_MONTH OR month > HIGH_MONTH?
No
Yes
output "Enter birth month... "
input month
Don't Do It
User is reprompted here, but there is no guarantee that month will be valid this time.

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
```

output "Enter birth month... "
input month
month < LOW_MONTH OR month > HIGH_MONTH?
Yes
No
output "Enter birth month... "
input month

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

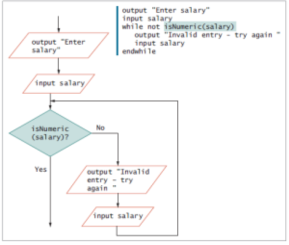
Common Loop Applications (continued)

- Limiting a reprompting loop
  - Reprompting can be frustrating to a user if it continues indefinitely
  - Maintain 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)

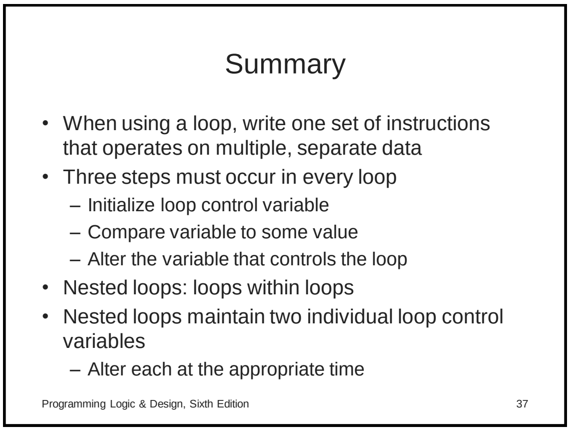


```
output "Enter salary"
input salary
while not isNumeric(salary)
   output "Invalid entry - try again "
   input salary
endwhile
```

Figure 5-21 Checking data for correct type

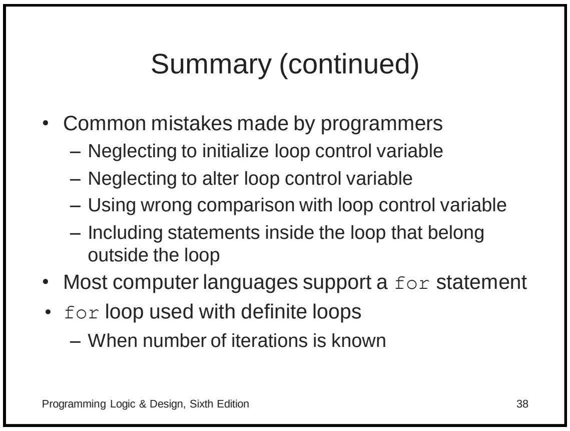
Common Loop Applications (continued)

- 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



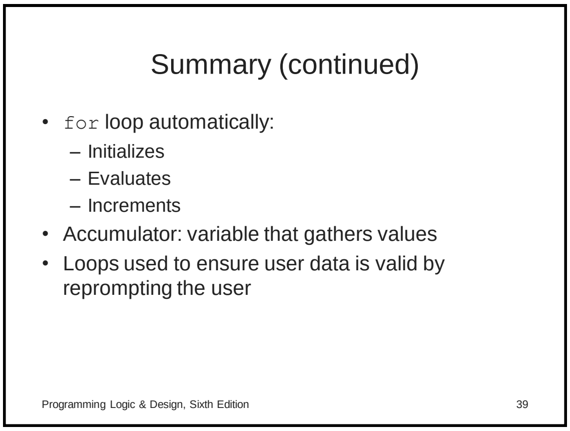
Summary

- When using a loop, write one set of instructions that operates on multiple, separate data
- Three steps must occur in every loop
  - Initialize loop control variable
  - Compare 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 loop control variable
  - Neglecting to alter loop control variable
  - Using wrong comparison with loop control variable
  - Including statements inside the loop that belong outside the loop
- Most computer languages support a `for` statement
- `for` loop used with definite loops
  - When number of iterations is known



Summary (continued)

- `for` loop automatically:
  - Initializes
  - Evaluates
  - Increments
- Accumulator: variable that gathers values
- Loops used to ensure user data is valid by reprompting the user