CIS1400 – Programming Logic and Technique

Topic 6 → Understanding Functions

modifications by CREngland

Chapter Topics

6.1 Introduction to Functions: Generating Random Numbers

6.2 Writing Your Own Functions

6.3 More Library Functions

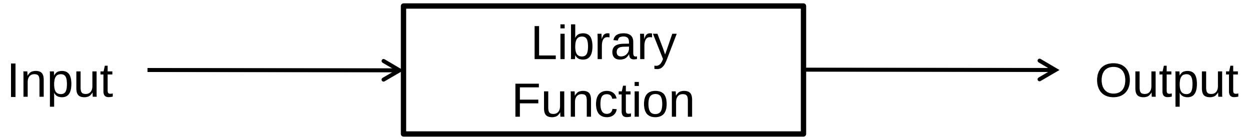
Sample Program: Commission Rate Program

6.1 Introduction to Functions

- A module is a group of statements that exists within a program for the purpose of performing a specific task. (*Chapter 3 -- Modules*)
- A function is *similar* to a module:
  - Group of statements that perform a specific task
  - Call function to execute it
- A function is *different* than a module:
  - Returns a value back to program that called it
    - Through return statement
  - Returned value can be used like other program values
    - Assigned to variable
    - Displayed on screen
    - Used as part of an expression

6.1 Introduction to Functions

- Many languages provide libraries of pre-written functions (aka library functions)
  - Built into programming language
    - Stored in special files when compiler/interpreter is installed
  - Usually common tasks and save time for the programmer because it allows for code reuse
    - Mathematical Functions
    - Data Type Conversion Functions
    - String Functions
    - Formatting Functions
  - Often viewed as a "*black box*"
    - Details of process not as important as input and output requirements



Input

Library Function

Output

6.1 Introduction to Functions

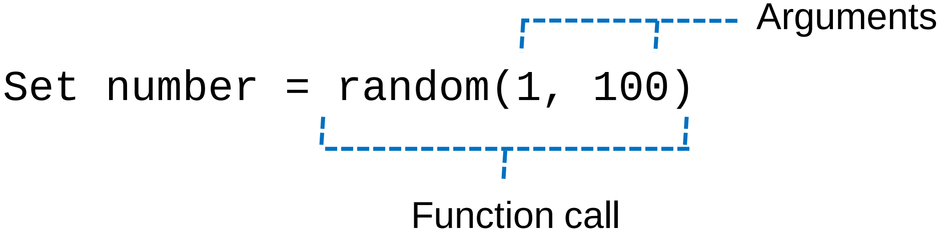
The Random Number Generator function is useful in:

- Game programs
- Simulation programs
- Statistical programs
- Computer security such as encryption

Visual Basic
Generating Random Numbers can be found in VB Language Companion page 75

How random function works:

Figure 6-2 A statement that calls the random function

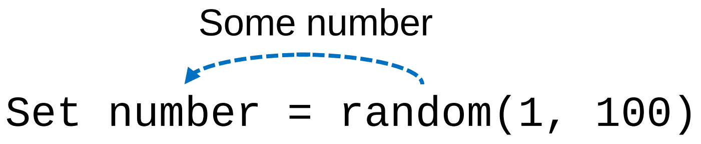


```
Set number = random(1, 100)
```

Arguments

Function call

Figure 6-3 The random function returns a value



Some number

```
Set number = random(1, 100)
```

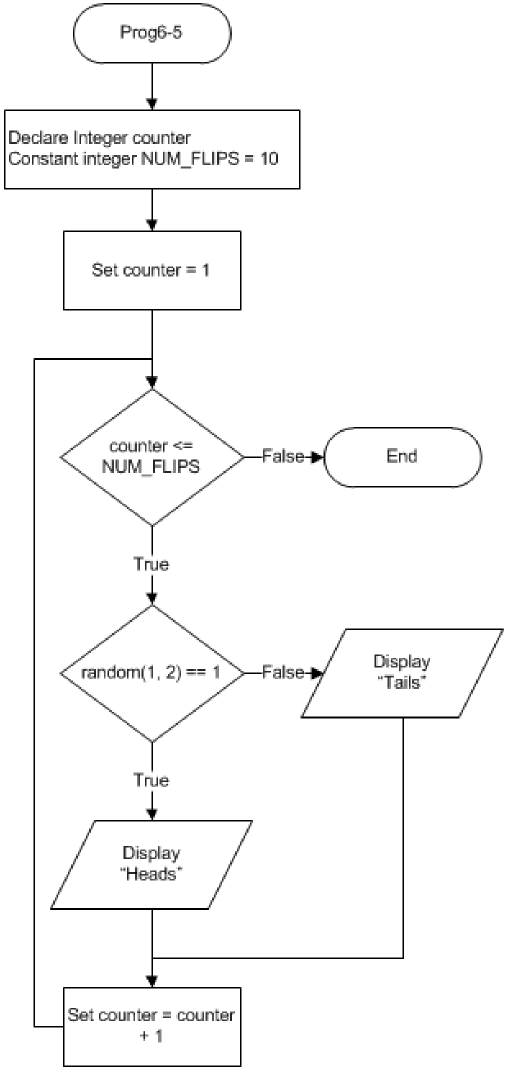
A random number in the range of 1 through 100 will be assigned to the `number` variable.

Random Number Example – Program 6-5

```
// Declare a counter variable
Declare Integer counter

// constant for the number of flips
Constant Integer NUM_FLIPS = 10

For counter = 1 To NUM_FLIPS
  // Simulate the coin flip.
  If random(1, 2) == 1 Then
    Display “Heads”
  Else
    Display “Tails”
  End If
End For
```



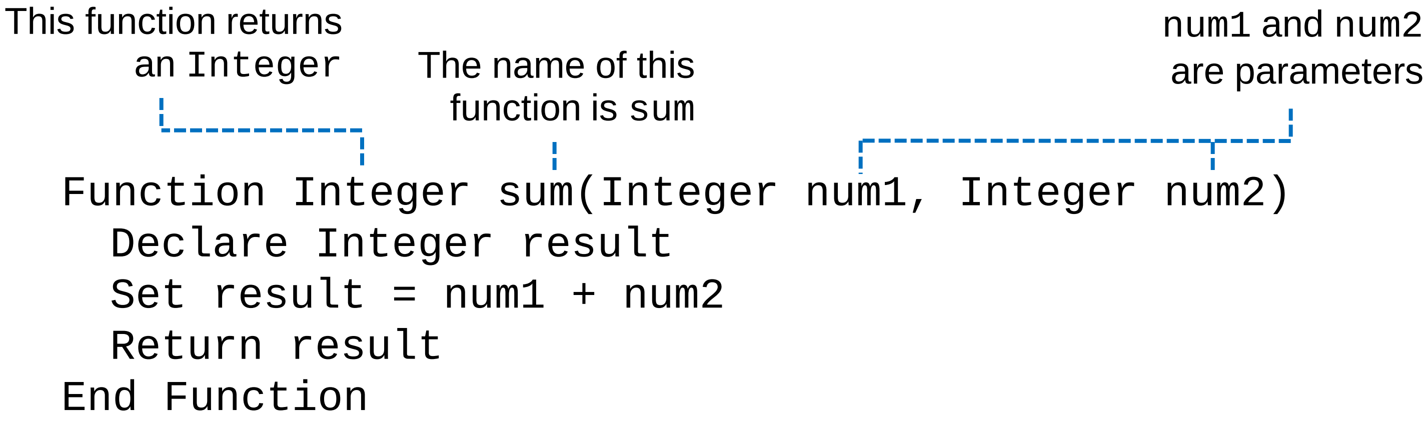
6.2 Writing Your Own Functions

Most languages allow programmers to write functions.

The code for a function is known as a function definition.

- Definition contains the following parts:
  - A header
    - Starting point of the function. Specifies 'Function' keyword, data type of value returned, name of function, and any parameter variables

Figure 6-7 Parts of the function header



This function returns an Integer

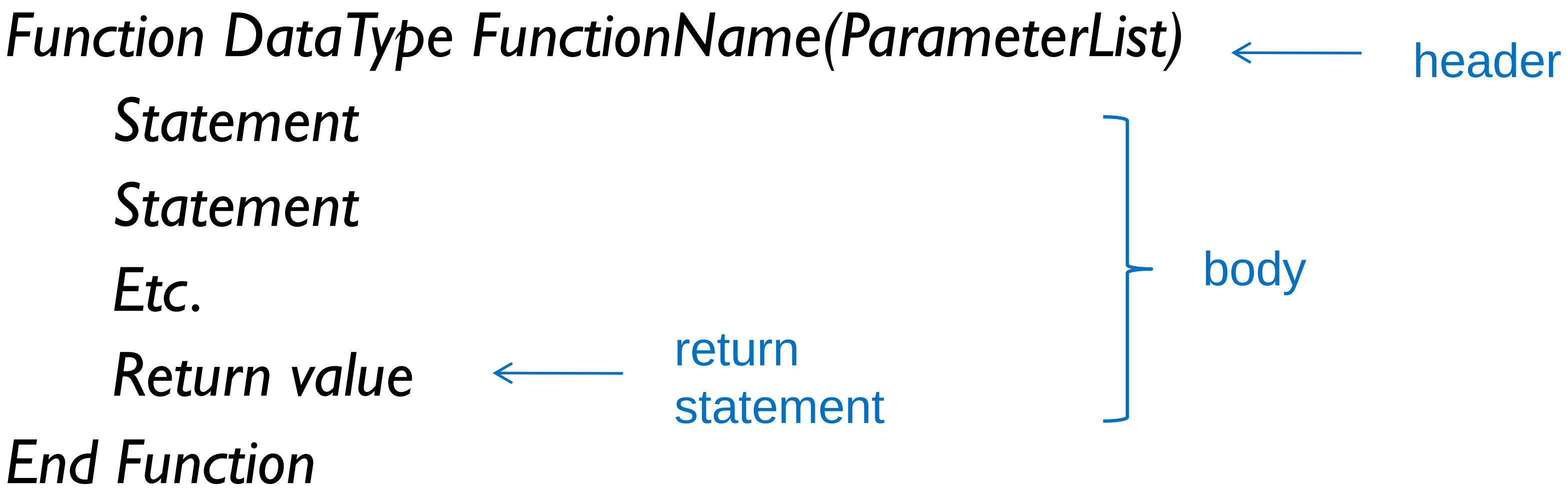
The name of this function is sum

num1 and num2 are parameters

```
Function Integer sum(Integer num1, Integer num2)
  Declare Integer result
  Set result = num1 + num2
  Return result
End Function
```

6.2 Writing Your Own Functions

- Definition contains the following parts:
  - A body
    - Statements that execute when the function is called
  - A return statement
    - Value returned when the function ends
- Pseudocode Format:



```
Function DataType FunctionName(ParameterList)
    Statement
    Statement
    Etc.
    Return value
End Function
```

header

body

return statement

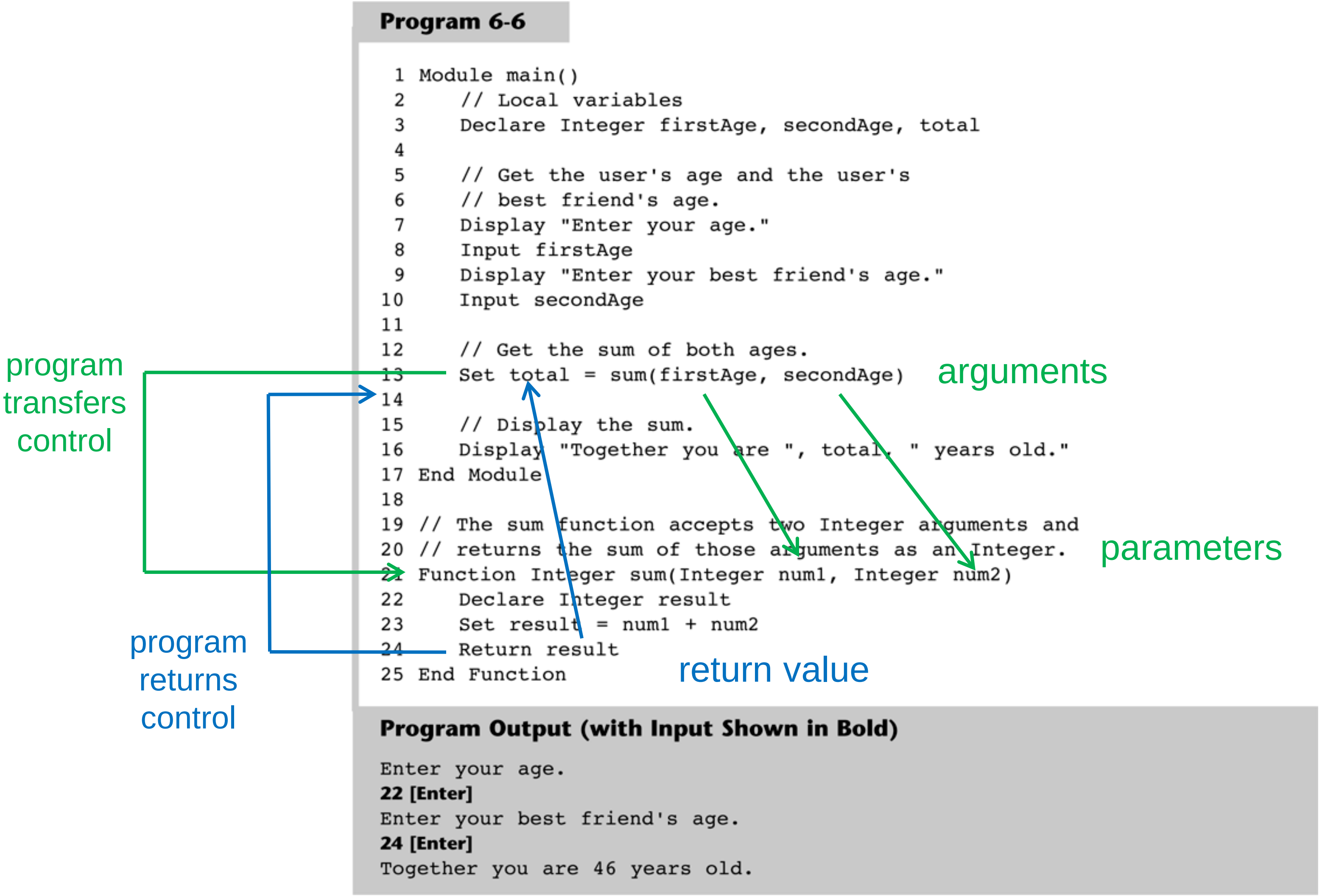
6.2 Writing Your Own Functions

- A call must be made to the function in order for the statements in the body to execute.
  - Used in assignment, expressions, calculations, or display statements
  - Pseudocode Format

```
Module main( )
    Statement
     Set retVariable = NameOfFunction(ArgumentList)
    Statement
    Etc.
End Module
```

Assignment call statement

Functions Example – Program 6-6



Program 6-6

```
Module main()
    // Local variables
    Declare Integer firstAge, secondAge, total

    // Get the user's age and the user's
    // best friend's age.
    Display "Enter your age."
    Input firstAge
    Display "Enter your best friend's age."
    Input secondAge

    // Get the sum of both ages.
    Set total = sum(firstAge, secondAge)

    // Display the sum.
    Display "Together you are ", total, " years old."
End Module

// The sum function accepts two Integer arguments and
// returns the sum of those arguments as an Integer.
Function Integer sum(Integer num1, Integer num2)
    Declare Integer result
    Set result = num1 + num2
    Return result
End Function
```

Program Output (with Input Shown in Bold)

```
Enter your age.
22 [Enter]
Enter your best friend's age.
24 [Enter]
Together you are 46 years old.
```

program transfers control

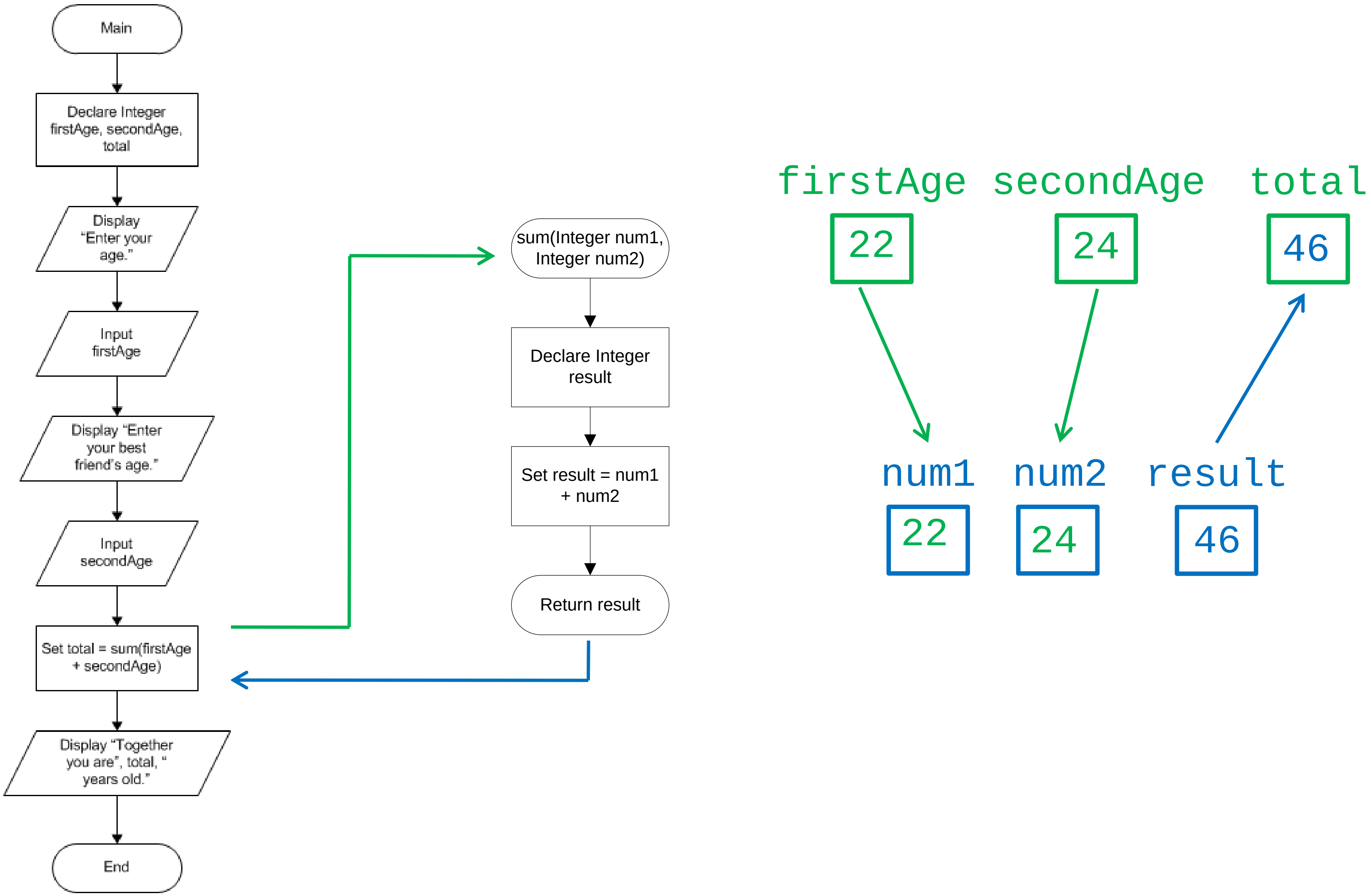
arguments

parameters

program returns control

return value

Functions Example – Program 6-6



Main

Declare Integer firstAge, secondAge, total

Display "Enter your age."

Input firstAge

Display "Enter your best friend's age."

Input secondAge

Set total = sum(firstAge + secondAge)

Display "Together you are", total, " years old."

End

sum(Integer num1, Integer num2)

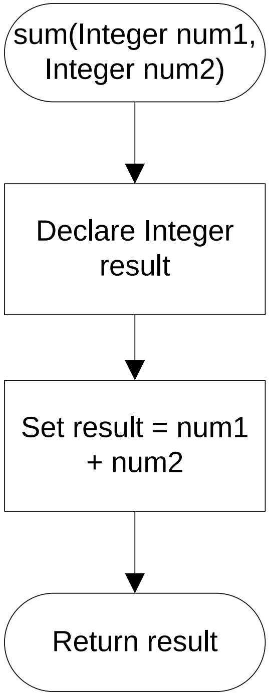
Declare Integer result

Set result = num1 + num2

Return result

6.2 Writing Your Own Functions

- While you can pass many arguments *into* a function, you can only return *one* value
- Like modules, functions
  - Simplify code, increase the speed of development, and ease the facilitation of teamwork
  - Should be flowcharted separately
    - Starting terminal shows function name with parameters
    - Ending terminal shows return with value or expression



6.2 Writing Your Own Functions

- IPO (input, processing, and output) is tool used when designing functions
  - Input column shows description of data *passed to function* as arguments
  - Processing column shows brief *description of process* function performs
  - Output column describes *data returned from* function

IPO Chart for the sum Function		
Input	Processing	Output
Two ages	Adds ages together	Sum of ages as Integer

6.3 More Library Functions

Mathematical Functions*

- Functions typically accept one or more values as arguments, perform a mathematical operation using the arguments, and return the results

Set result = sqrt(16)

- Returns the square root of 16

Set area = power(4, 2)

- Raises the value of 4 to the power of 2

Visual Basic
Mathematical functions can be found in VB Language Companion Table 6-2

** some programming languages may not implement these functions*

6.3 More Library Functions

Other Common Mathematical Functions*

- *abs* calculates the absolute value of a number
- *cos* returns the cosign of an argument
- *round* rounds to the nearest integer
- *sin* returns the sine of an argument
- *tan* returns the tangent of an argument

See Table 6-2 on page 250.

** some programming languages may not implement these functions*

6.3 More Library Functions

Data Type Conversion Functions*

- Library functions that convert values from one data type to another
  - *toInteger* converts a real to an integer
  - *toReal* converts an integer to a real
- Real numbers can store integers
- Integers cannot store real numbers
  - Loss of precision
- Type mismatch errors will occur without converting values

See Table 6-3 on page 251.

* *some programming languages may not implement these functions*

6.3 More Library Functions

Formatting Functions*

- Allow to format a number in a certain way
- *currencyFormat* will be used to format a number to a currency

```
Declare Real amount = 6450.879
Display currencyFormat(amount)
```

- Display would be *$6,450.88*

Visual Basic
Formatting functions can be found in VB Language Companion Table 6-3

VB example

```
Dim amount As Double = 6450.879
Dim resultStr As String
resultStr = amount.ToString(“c”)
Console.Write(resultStr)
```

* *some programming languages may not implement these functions*

6.3 More Library Functions

Visual Basic
String methods can be found in VB Language Companion page 67

String Functions*

- Allow for working with strings
- *length* function returns the length of a function
- *append* function joins multiple strings together
- *toUpper* and *toLower* converts a string to upper or lower case
- *substring* can extract a character or a portion of a string out of a string
- *contains* identifies similar strings within two strings
- *stringToInteger* and *stringToReal* converts string that stores a number, to a number data type
- *isInteger* and *isReal* test numbers to see if it can be converted to a string

** some programming languages may not implement these functions*

Sample Program: Commission Rate Program

"*In The Spotlight*" page 240

- Calculate sales commission based upon sales
- Employee advanced pay is subtracted from commission
- ***What is required for each phase of the program?***

1. What must be read as input?
   - □ Get monthly sales
   - □ Get advanced pay
2. How will the input be processed?
   - □ Use monthly sales to determine commission rate
   - □ Calculate pay based upon commission and advanced pay
3. What will be done with the output?
   - □ Display amount of pay; negative pay should be reimbursed

Sample Program: Commission Rate Program

IPO Chart for the `getSales` Function		
Input	Processing	Output
None	Prompts the user to enter amount of monthly sales	Monthly sales as `Real`

IPO Chart for the `getAdvancedPay` Function		
Input	Processing	Output
None	Prompts the user to enter amount of advanced pay	Amount of advanced pay as `Real`

Sample Program: Commission Rate Program

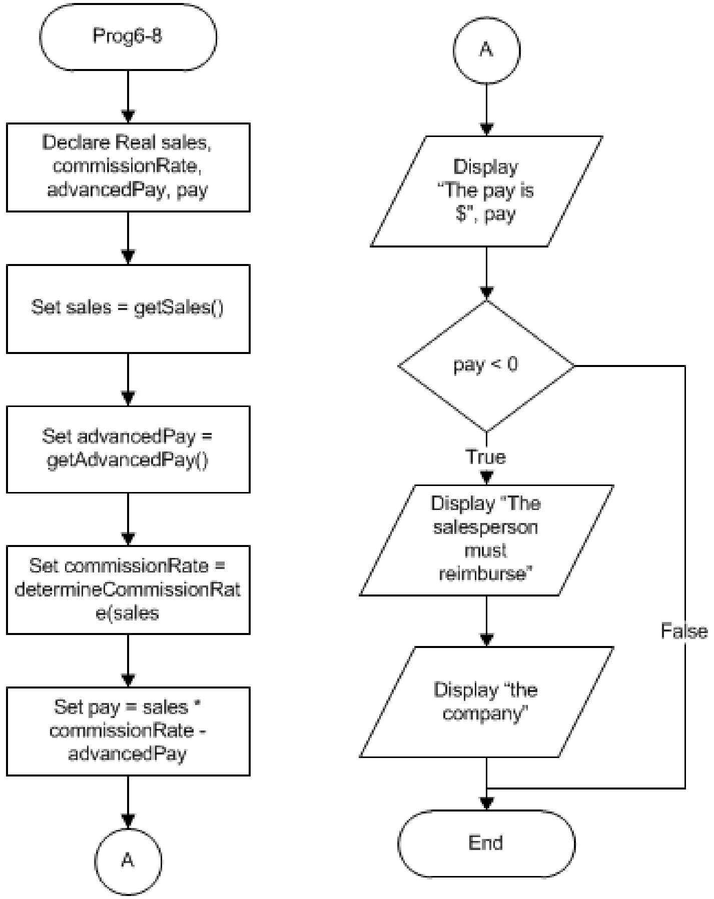
IPO Chart for the `determineCommissionRate` Function		
Input	Processing	Output
Amount of monthly sales	Determine the commission rate based upon monthly sales: Less than $10,000.00 → 10% $10,000.00 – 14,999.99 → 12% $15,000.00 – 17,999.99 → 14% $18,000.00 – 21,999.99 → 16% $22,000 or more → 18%	Commission rate as `Real`

Sample Program: Commission Rate Program

Main module pseudocode

```
Module main()
  // Local variables
  Declare Real sales, commissionRate, advancedPay, pay
  // Get the amount of sales
  Set sales = getSales()
  // Get the amount of advanced pay
  Set advancedPay = getAdvancedPay()
  // Determine the commission rate
  Set commissionRate = determineCommissionRate(sales)
  // Calculate the pay
  Set pay = sales * commissionRate – advancedPay
  // Display the amount of pay
  Display “The pay is $”, pay
  // Determine whether the pay is negative
  If pay < 0 Then
    Display “The salesperson must reimburse”
    Display “the company.”
  End If
End Module
```

Sample Program: Commission Rate Program

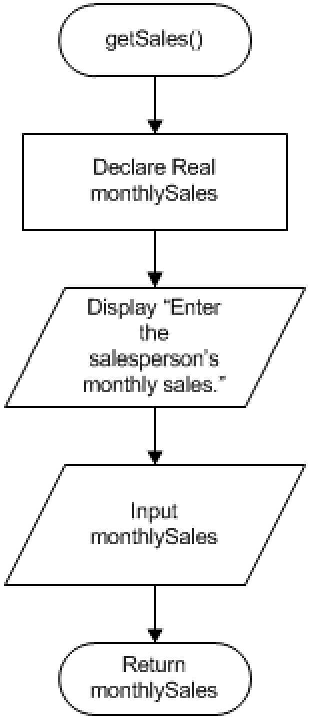

Prog6-8
Declare Real sales, commissionRate, advancedPay, pay
Set sales = getSales()
Set advancedPay = getAdvancedPay()
Set commissionRate = determineCommissionRate(sales
Set pay = sales * commissionRate - advancedPay
A
A
Display "The pay is $", pay
pay < 0
True
False
Display "The salesperson must reimburse"
Display "the company"
End

Main module flowchart

Sample Program: Commission Rate Program

getSales() function

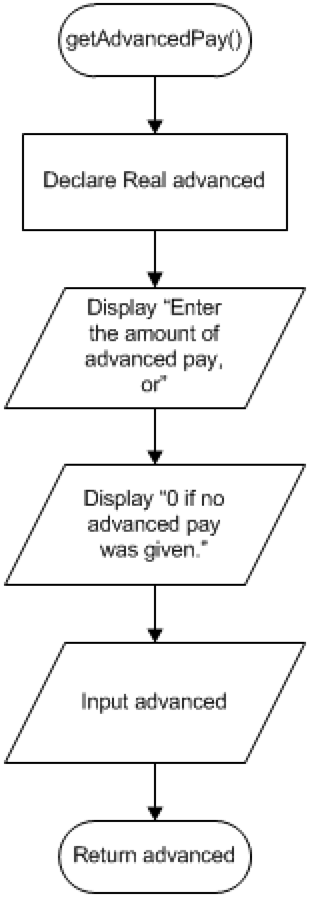
```
// The getSales function gets a salesperson’s
// monthly sales from the user and returns
// that value as a Real.
Function Real getSales()
  // Local variable to hold the monthly sales
  Declare Real monthlySales
  // Get the amount of monthly sales
  Display “Enter the salesperson’s monthly sales.”
  Input monthlySales
  // Return the amount of monthly sales
  Return monthlySales
End Function
```


getSales()
Declare Real monthlySales
Display “Enter the salesperson’s monthly sales.”
Input monthlySales
Return monthlySales

Sample Program: Commission Rate Program

getAdvancedPay() function

```
// The getAdvancedPay function gets the amount of
// advanced pay given to the salesperson and
// returns that amount as a Real.
Function Real getAdvancedPay()
  // Local variable to hold the advanced pay
  Declare Real advanced
  // Get the amount of advanced pay
  Display “Enter the amount of advanced pay, or”
  Display “0 if no advanced pay was given”
  Input advanced
  // Return the advanced pay
  Return advanced
End Function
```

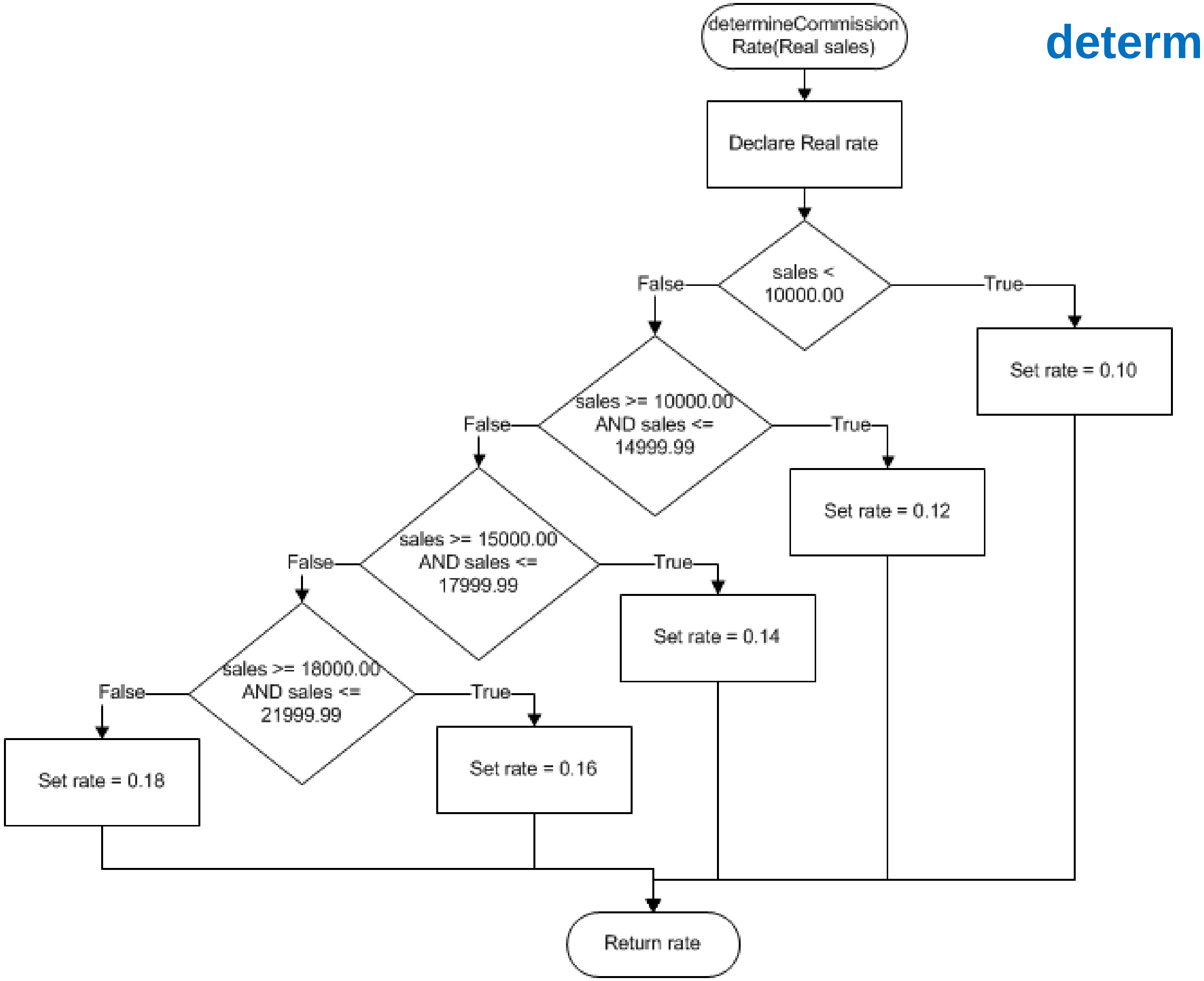

getAdvancedPay()
Declare Real advanced
Display "Enter the amount of advanced pay, or"
Display "0 if no advanced pay was given."
Input advanced
Return advanced

Sample Program: Commission Rate Program

```
// The determineCommissionRate function accepts the
// amount of sales as an argument and returns the
// commission rate as a Real.
Function Real determineCommissionRate(Real sales)
  // Local variable to hold commission rate
  Declare Real rate
  // Determine the commission rate
  If sales < 10000.00 Then
    Set rate = 0.10
  Else If sales >= 10000.00 AND sales <= 14999.99 Then
    Set rate = 0.12
  Else If sales >= 15000.00 AND sales <= 17999.99 Then
    Set rate = 0.14
  Else If sales >= 18000.00 AND sales <= 21999.99 Then
    Set rate = 0.16
  Else
    Set rate = 0.18
  End If
  // Return the commission rate
  Return rate
End Function
```

determineCommissionRate() function pseudocode

Sample Program: Commission Rate Program



determineCommissionRate() function flowchart

Chapter Topics

7.1 Garbage In, Garbage Out

7.2 The Input Validation Loop

7.3 Defensive Programming

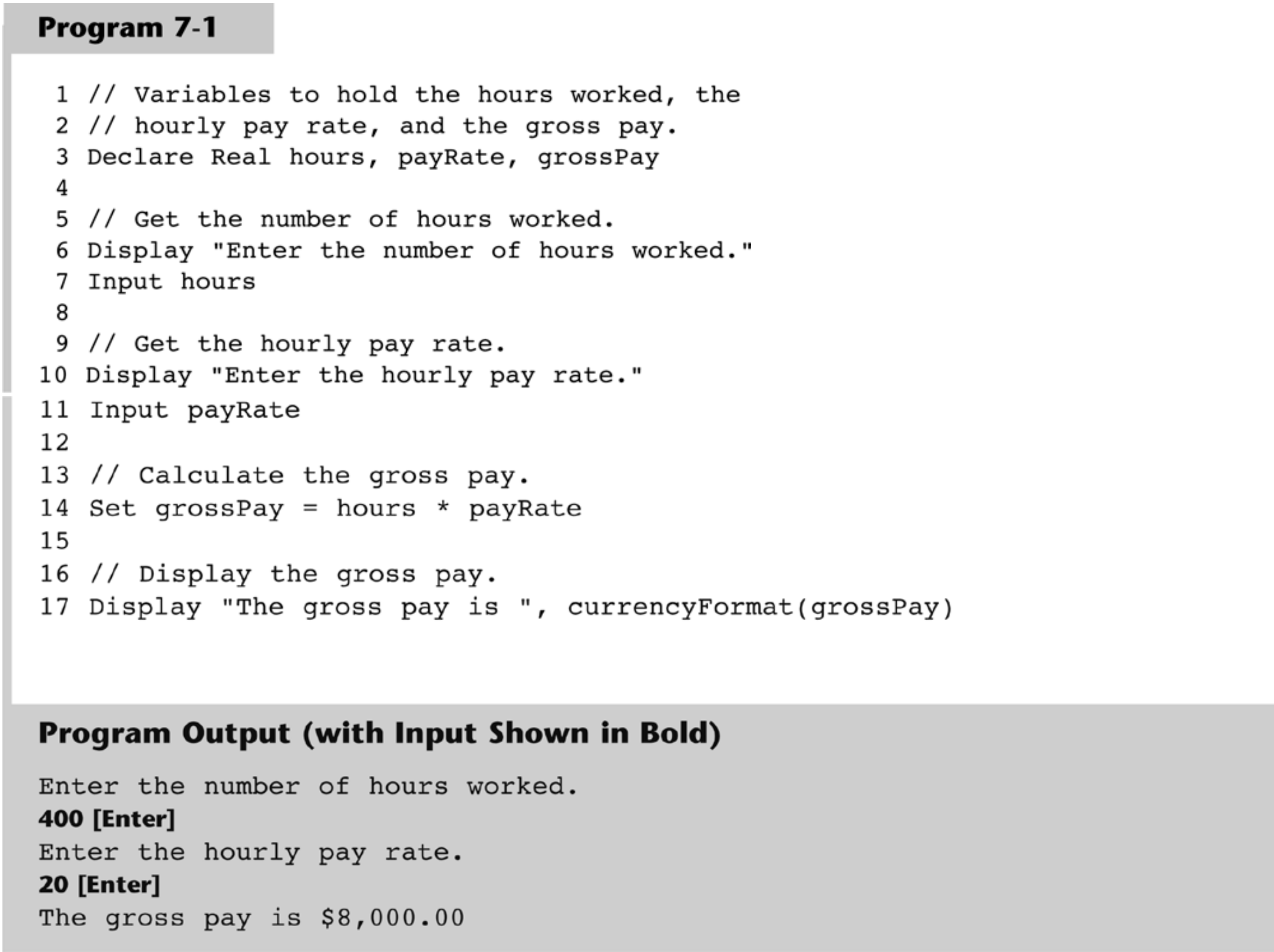
Sample Program: Commission Rate Program (with validation)

7.1 Garbage In, Garbage Out

If a program reads bad data as input, it will produce bad data as output!

- Programs should be designed to accept only good data
- Input Validation
  - All input should be inspected before processing
  - If input is invalid, it should be rejected and the user should be prompted to enter the correct data

7.1 Garbage In, Garbage Out



Program 7-1

```
// Variables to hold the hours worked, the
// hourly pay rate, and the gross pay.
Declare Real hours, payRate, grossPay

// Get the number of hours worked.
Display "Enter the number of hours worked."
Input hours

// Get the hourly pay rate.
Display "Enter the hourly pay rate."
Input payRate

// Calculate the gross pay.
Set grossPay = hours * payRate

// Display the gross pay.
Display "The gross pay is ", currencyFormat(grossPay)
```

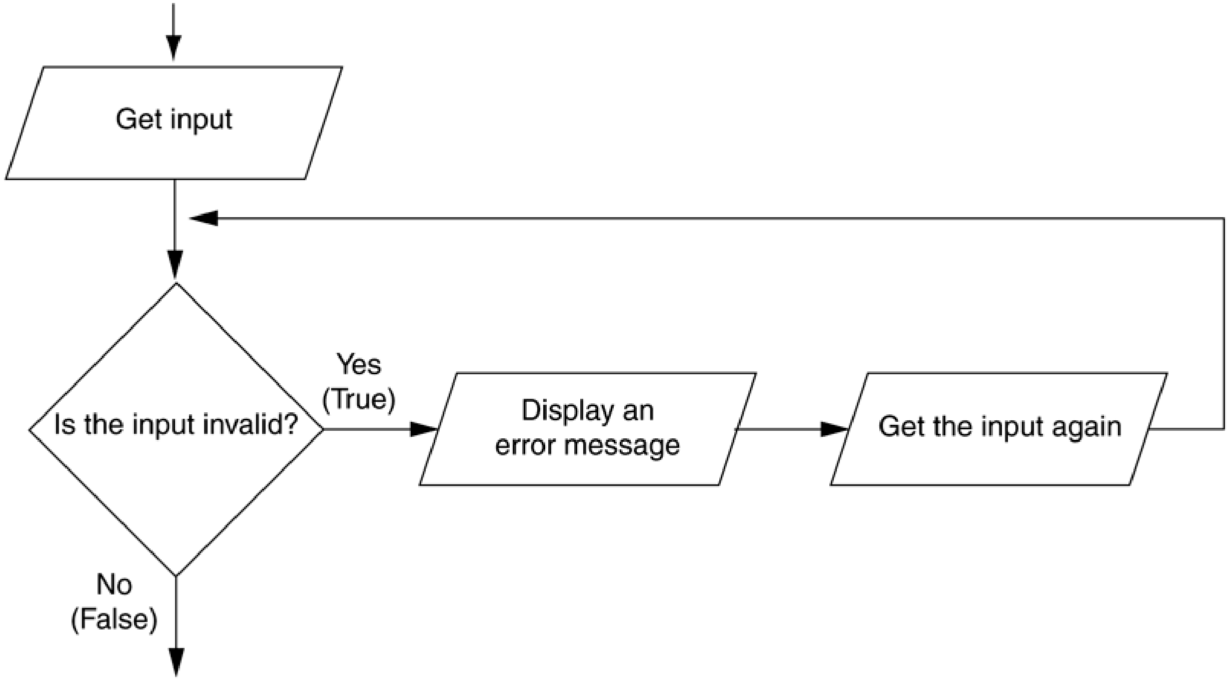
Program Output (with Input Shown in Bold)

```
Enter the number of hours worked.
400 [Enter]
Enter the hourly pay rate.
20 [Enter]
The gross pay is $8,000.00
```

7.2 The Input Validation Loop

Input validation is commonly done with a loop that iterates as long as input is bad

Figure 7-1 Logic containing an input validation loop



7.2 The Input Validation Loop

Priming read is the first input to be tested when using a Pretest Loop

```
// Get a test score
Display “Enter a test score.”
Input score
// Validate the test score.
While score < 0 OR score > 100
  Display “ERROR:  The score cannot be less than 0”
  Display “or greater than 100.”
  Display “Enter the correct score.”
  Input score
End While
```

priming read

7.2 The Input Validation Loop

Posttest Loop can also be used to validate input, eliminating the need for a priming read

```
Do
   Display “Enter a test score.”
   Input score
While score < 0 OR score > 100
```

However, displaying an error message can be more complex

```
Do
   Display “Enter a test score.”
   Input score
   If score < 0 OR score > 100 Then
      Display “ERROR:  The score cannot be less than 0 ”
      Display “or greater than 100.”
   End If
While score < 0 OR score > 100
```

error message display

7.2 The Input Validation Loop

Writing Validation Functions

- For complex validation, it is recommended to write a function.

```
Function Boolean isInvalid(Integer score)
  Declare Boolean status
  If score < 0 OR score > 100 Then
    Set status = True
  Else
    Set status = False
  Return status
End Function
```

7.2 The Input Validation Loop

Writing Validation Functions

- This process can make the code look cleaner

```
// Get a test score
Display “Enter a test score.”
Input score
// Validate the test score.
While isInvalid(score)  // score < 0 OR score > 100
  Display “ERROR:  The score cannot be less than 0 ”
  Display “or greater than 100.”
  Display “The correct score.”
  Input score
End While
```

7.3 Defensive Programming

Input validation is defensive programming

- The practice of anticipating both obvious and unobvious errors that can happen

Types of errors to consider

- Empty input, where a user accidentally hits enter before entering data
  - length() != 0
- The user enters the wrong type of data
  - isInteger() before stringToInteger()
  - isReal() before stringToReal()

7.3 Defensive Programming

Validating String Input

- Some strings must be validated for specific string input

```
// Get the answer to the question
Display “Is your supervisor an effective leader?”
Input answer
// Validate the input
While toLower(answer) != “yes” AND toLower(answer) != “no”
   Display “Please answer yes or no.  Is your supervisor an”
   Display “effective leader?”
   Input answer
End While
```

Accepts any combination of “yes” or “no” in upper and lower case.

7.3 Defensive Programming

Validating String Input

- Programs may require specific string length requirements

```
// Get the new password
Display “Enter your new password: ”
Input password
// Validate the length of the password
While length(password) < 6
  Display “The password must be at least six”
  Display “characters long.  Enter your new password: ”
  Input password
End While
```

Ensures password at least six characters in length.

7.3 Defensive Programming

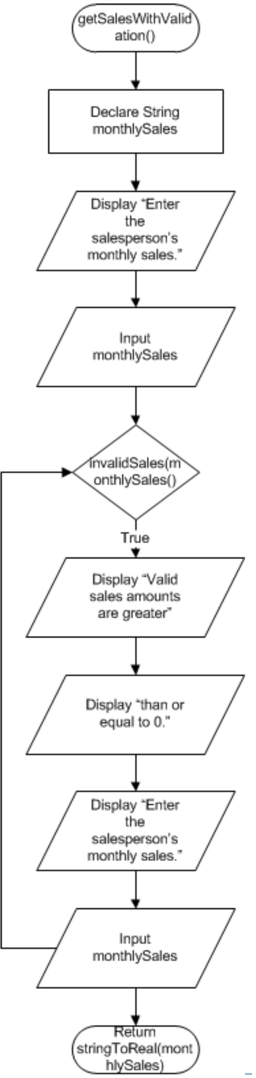
Common errors to be aware of

- State abbreviations should be 2-character strings
- Zip codes should be in the proper format of 5 or 9 digits
- Hourly wages and salary amounts should be numeric values and within acceptable ranges
- Dates should be checked
  - February 29 only valid in leap year
  - February 30 is never valid
- Time measurements should be checked
  - 7 * 24 = 168 hours in a week
- Check for reasonable numbers
  - Birth date in future or too far in past

Sample Program: Commission Rate Program

getSalesWithValidation() function – *with pretest validation*

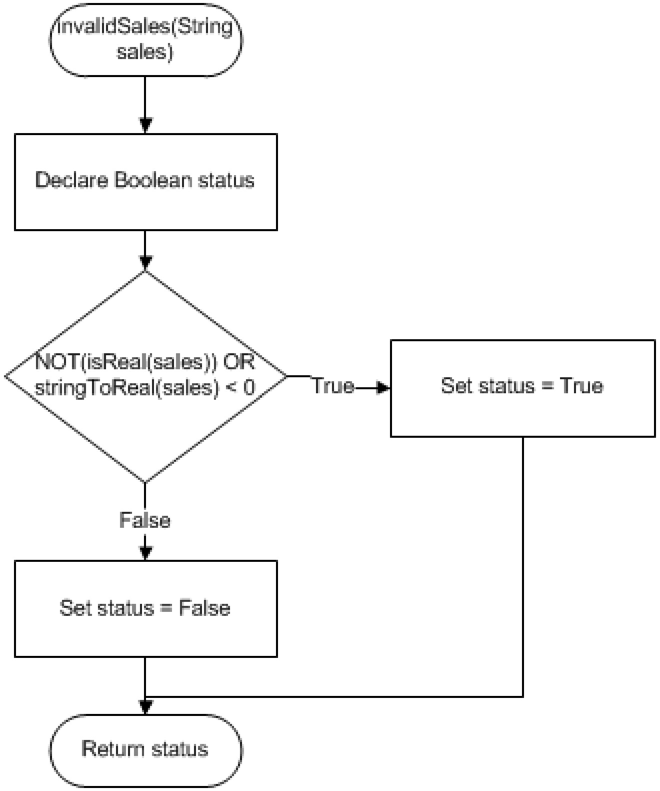
```
// The getSalesWithValidation function gets a
// salesperson’s monthly sales from the user and
// returns that value as a Real.
Function Real getSalesWithValidation()
  // Local variable to hold the monthly sales
  Declare String monthlySales
  // Get the amount of monthly sales
  Display “Enter the salesperson’s monthly sales.”
  Input monthlySales
  // Validate sales with pretest loop
  While invalidSales(monthlySales)
    Display “Valid sales amounts are greater”
    Display “than or equal to zero.”
    Display “Enter the salesperson’s monthly sales.”
    Input monthlySales
  End While
  // Return the amount of monthly sales
  Return stringToReal(monthlySales)
End Function
```


getSalesWithValidation()
Declare String monthlySales
Display "Enter the salesperson's monthly sales."
Input monthlySales
invalidSales(monthlySales()
True
Display "Valid sales amounts are greater"
Display "than or equal to 0."
Display "Enter the salesperson's monthly sales."
Input monthlySales
Return stringToReal(monthlySales)

Sample Program: Commission Rate Program

invalidSales() function – *for pretest validation of getSalesWithValidation() function*

```
// The invalidSales function determines
// if the input sales amount is valid.
Function Boolean invalidSales(String sales)
   // Local variable to hold True or False
   Declare Boolean status
   // If the sales is invalid,
   // set status to True
   If NOT(isReal(sales)) OR
         stringToReal(sales) < 0 Then*
     Set status = True
   Else
     Set status = False
   End If
   // Return the test status
   Return status
End Function
```

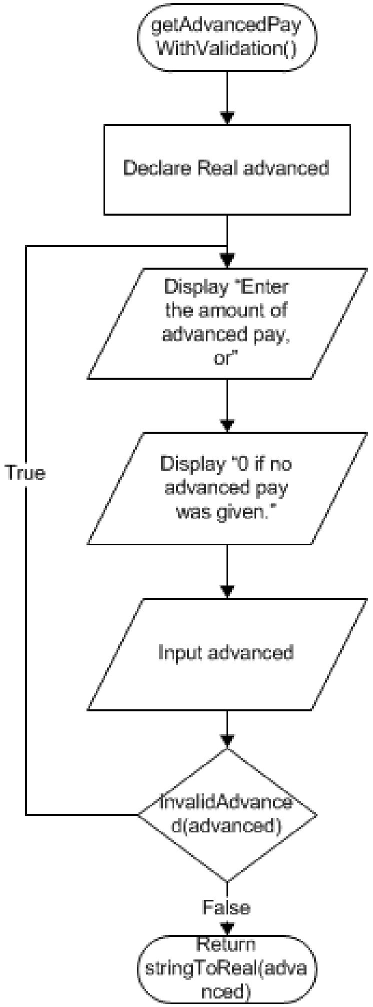


* Only true for short-circuit evaluation.

Sample Program: Commission Rate Program

getAdvancedPayWithValidation() function –
with posttest validation

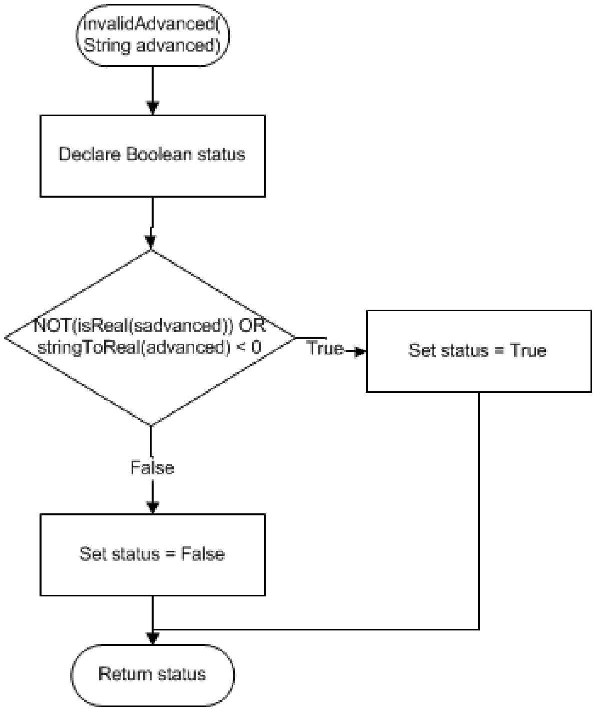
```
// The getAdvancedPayWithValidation function gets
// the amount of advanced pay given to the
// salesperson and returns that amount as a Real.
Function Real getAdvancedPayWithValidation()
  // Local variable to hold the advanced pay
  Declare String advanced
  // Validate advanced pay with posttest loop
  Do
    Display “Enter the amount of advanced pay, or”
    Display “0 if no advanced pay was given”
    Input advanced
  While invalidAdvanced(advanced)
  // Return the advanced pay
  Return stringToReal(advanced)
End Function
```


getAdvancedPayWithValidation()
Declare Real advanced
Display "Enter the amount of advanced pay, or"
Display "0 if no advanced pay was given."
Input advanced
invalidAdvanced(advanced)
True
False
Return stringToReal(advanced)

Sample Program: Commission Rate Program

invalidAdvanced() function – *for posttest validation of getAdvancedPayWithValidation() function*

```
// The invalidAdvanced function determines
// if the input advanced amount is valid.
Function Boolean invalidAdvanced(String advanced)
  // Local variable to hold True or False
  Declare Boolean status
  // If the advanced amount is invalid,
  // set status to True
  If NOT (isReal(advanced)) OR
      stringToReal(advanced < 0) Then*
    Set status = True
  Else
    Set status = False
  End If
  // Return the test status
  Return status
End Function
```


invalidAdvanced(String advanced)
Declare Boolean status
NOT(isReal(sadvanced)) OR stringToReal(advanced) < 0
True
Set status = True
False
Set status = False
Return status

* Only true for short-circuit evaluation.