Good Program Design

1. An Overview of Computers and Programming

- Understanding Computer Systems
- Understanding Simple Program Logic
- Understanding the Program Development Cycle
- Using Pseudo-code Statements and Flowchart Symbols
- Using a Sentinel Value to End a Program
- Understanding the Evolution of Programming Models
- Chapter Summary
- Exercises

1.1 Understanding Computer Systems

- A computer system:
  - Hardware
  - Software:
    - Application software
    - System software
- Computer hardware and software accomplish three major operations in most programs:
  - Input: Data items
  - Processing
  - Output: storage devices, cloud

1.2 Understanding Simple Program Logic

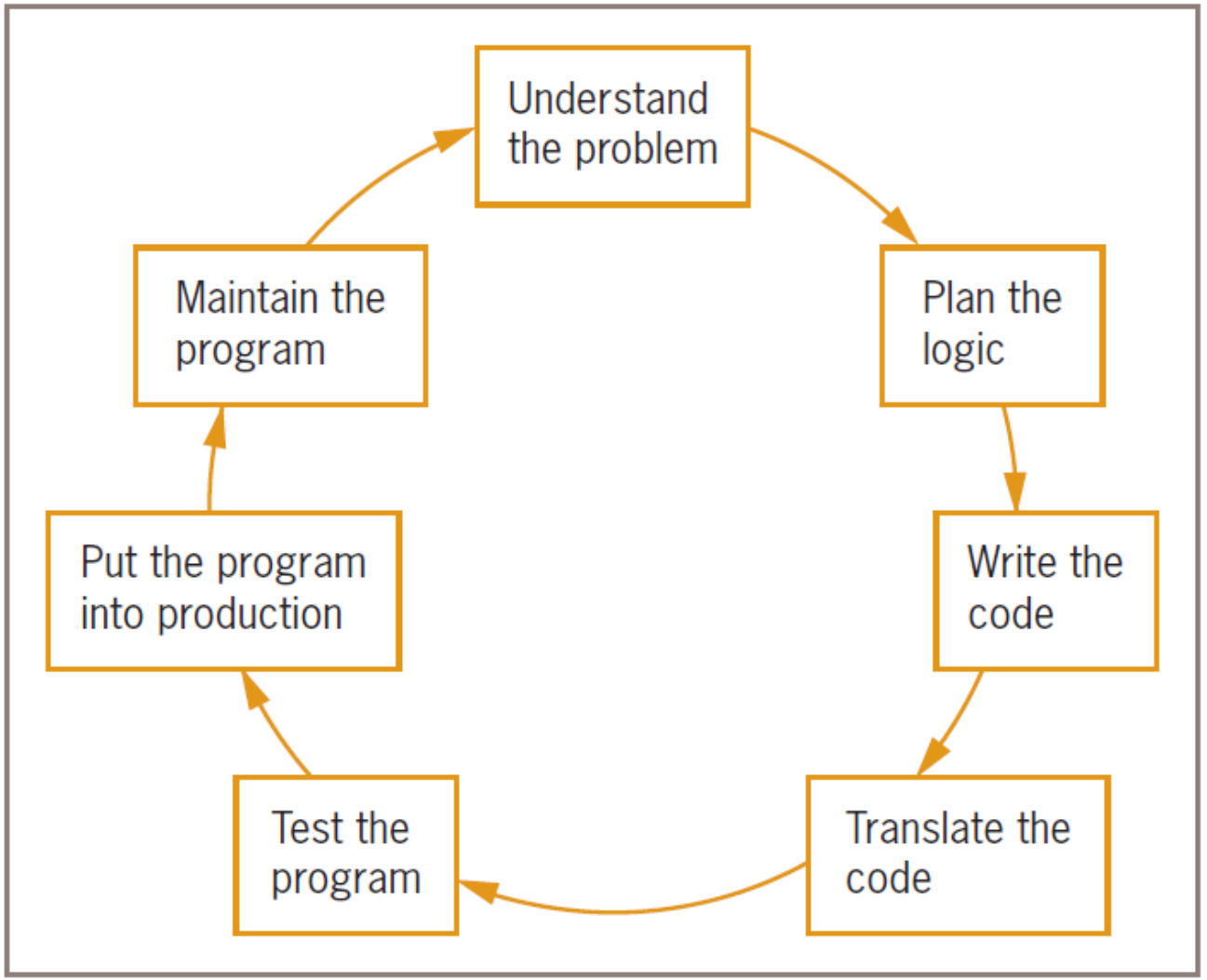
```
input myNumber
set myAnswer = myNumber * 2
output myAnswer
```

- The instruction to **input myNumber** is an example of an input operation. A **variable** is a named memory location whose value can vary.
- The instruction **set myAnswer = myNumber** * 2 is an example of a processing operation.
- In the number-doubling program, the **output myAnswer** instruction is an example of an output operation.

1.3 Understanding the Program Development Cycle

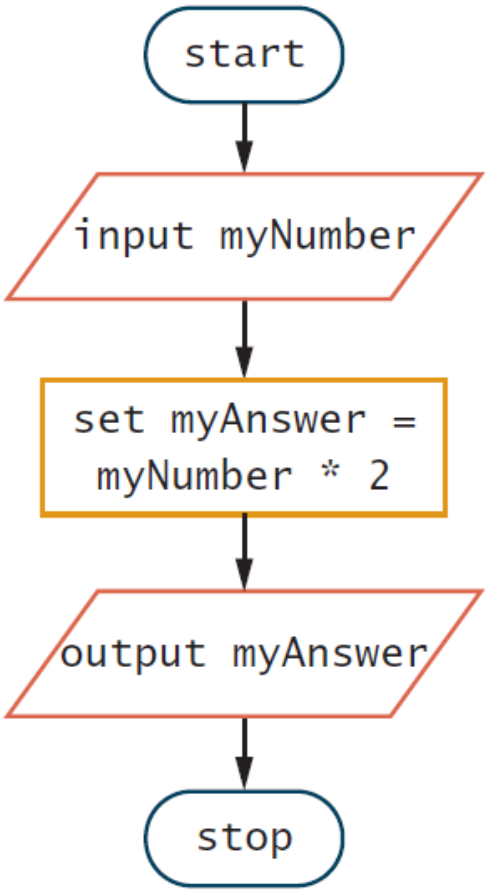
1. Understand the problem: end users, documentation
2. Plan the logic: algorithm, desk-checking
3. Code the program: syntax
4. Use software (a compiler or interpreter) to translate the program into machine language: high-level programming language, low-level machine language
5. Test the program: debugging
6. Put the program into production: conversion
7. Maintain the program: maintenance

The Program Development Cycle



1.4 Using Pseudocode Statements and Flowchart Symbols

Flowchart



Pseudocode

```
start
    input myNumber
    set myAnswer = myNumber * 2
    output myAnswer
stop
```

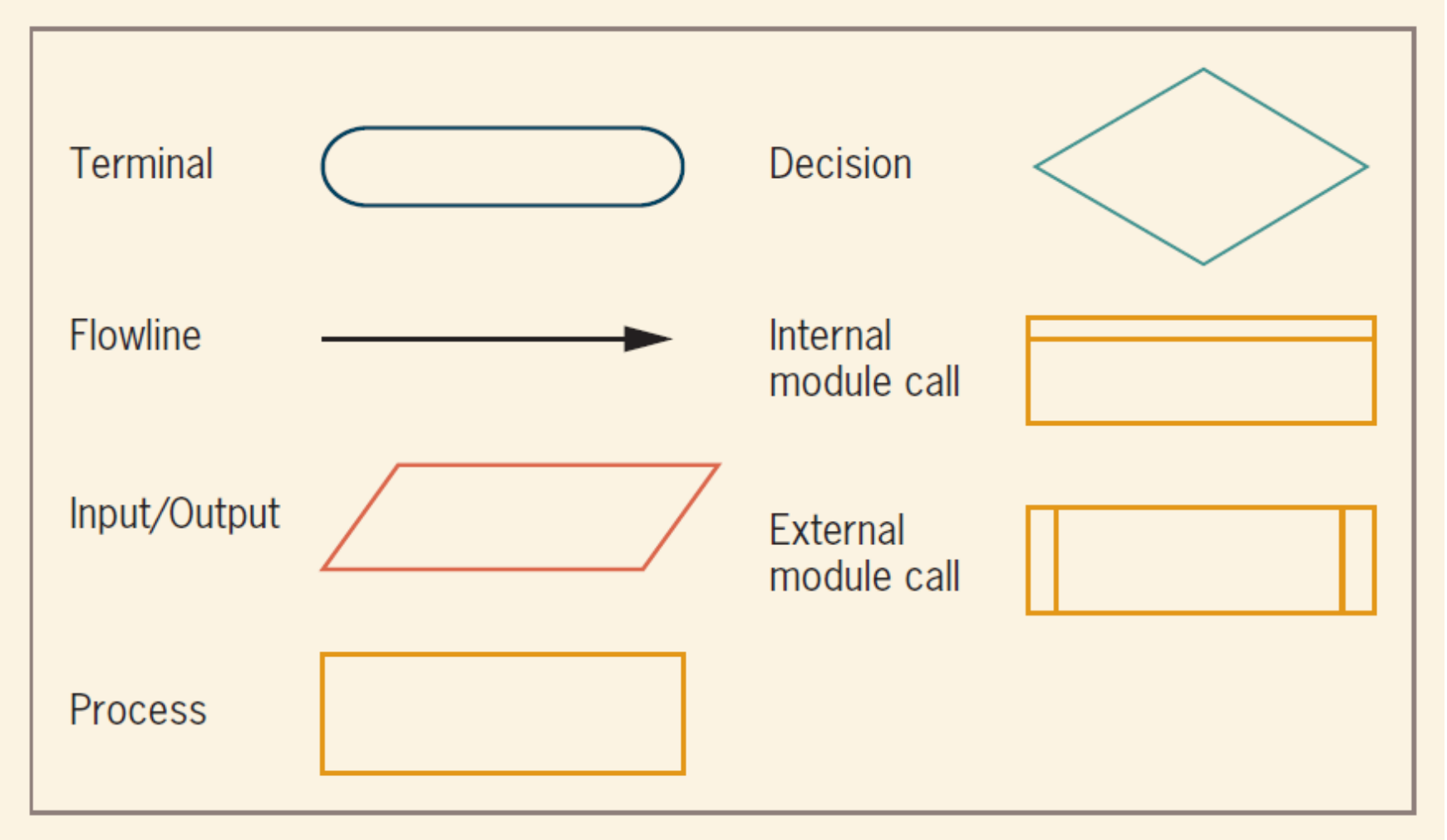
Pseudocode Standards

- Programs begin with **start** and end with **stop**; these two words are always aligned.
- Whenever a module name is used, it is followed by a set of parentheses.
- Modules begin with the module name and end with **return**. The module name and **return** are always aligned.
- Each program statement performs one action—for example, input, processing, or output.
- Program statements are indented a few spaces more than **start** or the module name.
- Each program statement appears on a single line if possible. When this is not possible, continuation lines are indented.
- Program statements begin with lowercase letters.
- No punctuation is used to end statements.

Drawing Flowcharts

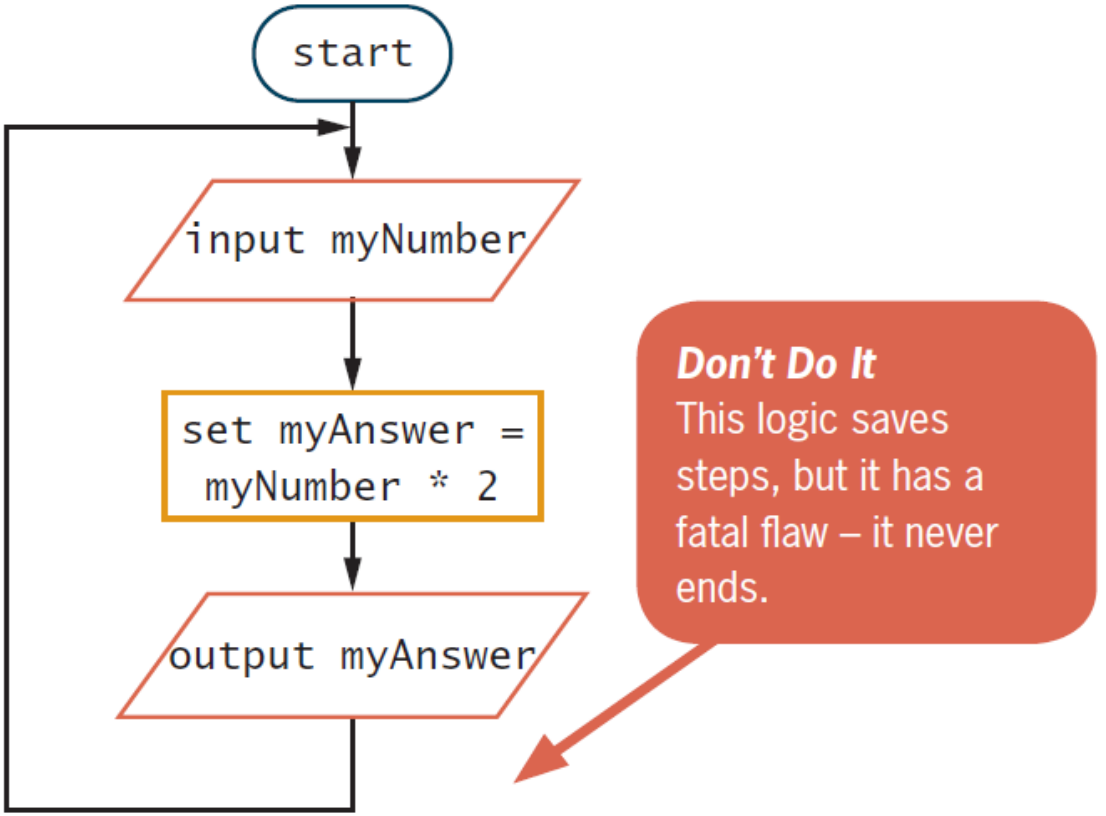
- input symbol
- processing symbol
- output symbol
- flowline
- terminal symbol

Flowchart Symbols



Repeating Instructions

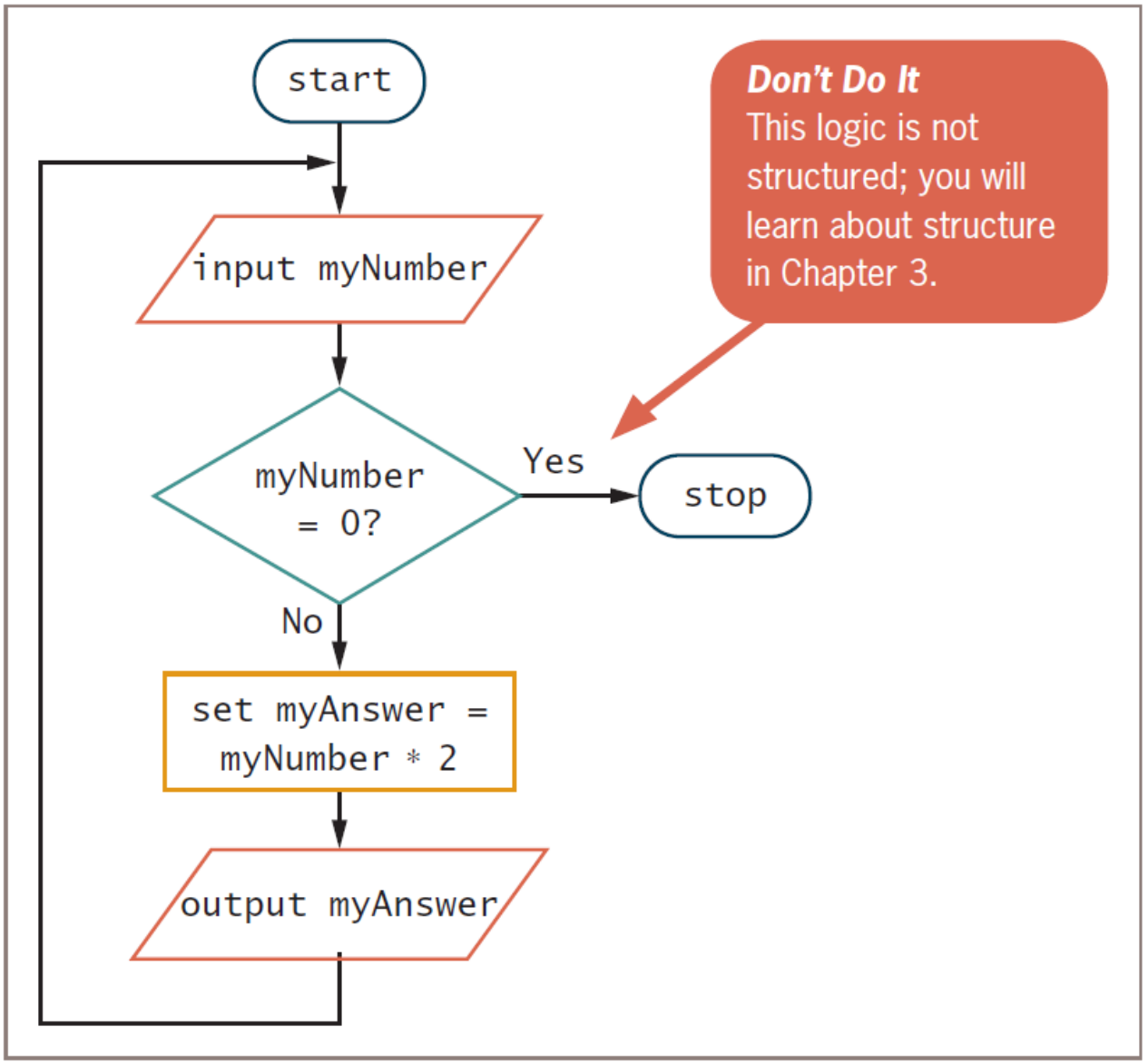
```
start
  input myNumber
  set myAnswer = myNumber * 2
  output myAnswer
  input myNumber
  set myAnswer = myNumber * 2
  output myAnswer
```



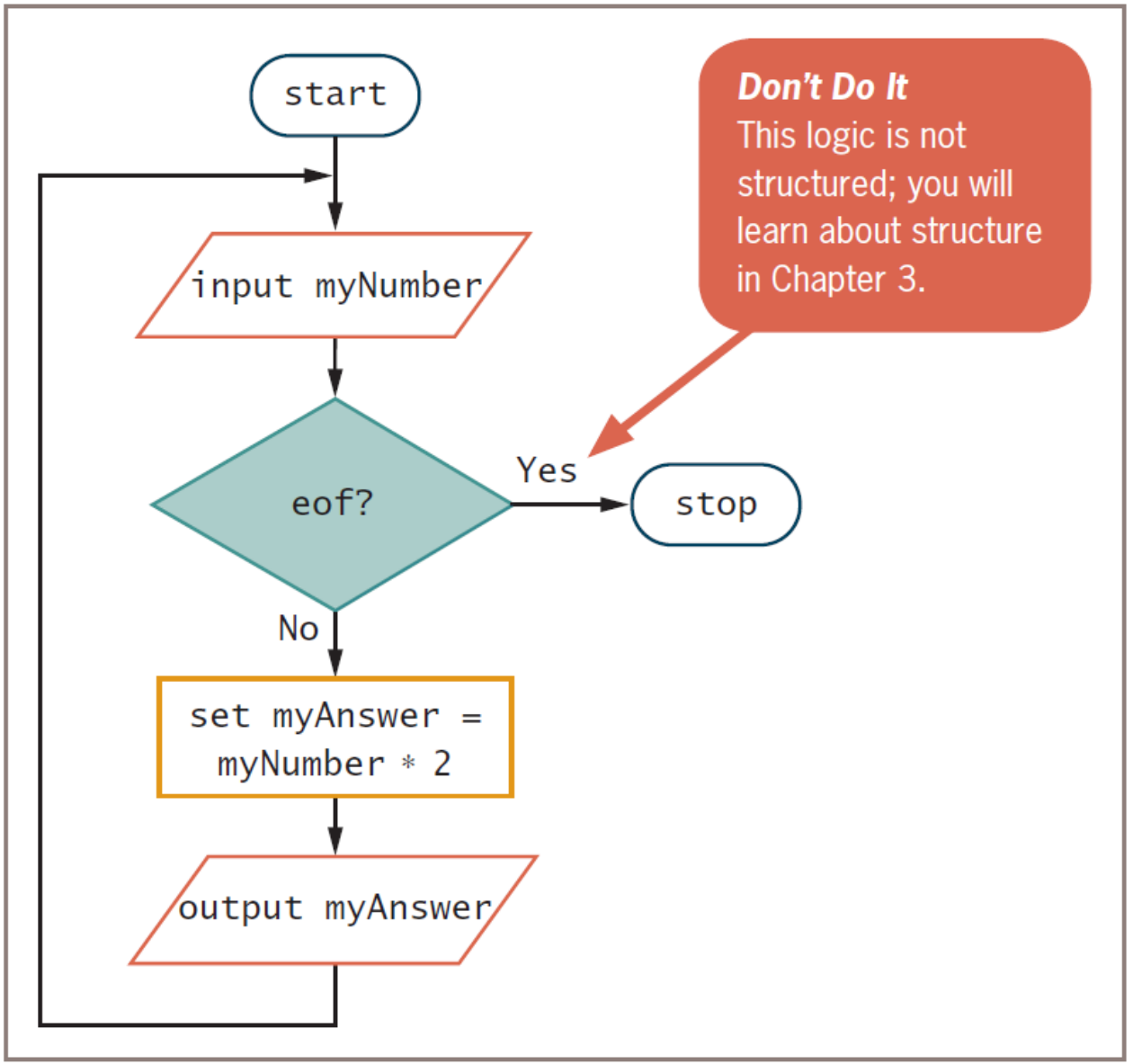
1.5 Using a Sentinel Value to End a Program

- making a decision
- decision symbol
- dummy value
- sentinel value
- eof

A Program with Sentinel Value of 0



A Program using eof



1.6 Understanding the Evolution of Programming Models

- **Procedural programming** focuses on the procedures that programmers create.
- **Object-oriented programming** focuses on objects, or “things,” and describes their features (also called attributes) and behaviors.

1.7 Summary

- input, processing, and output.
- logic, syntax errors, logical errors.
- understanding the problem, planning the logic, coding the program, translating the program, testing the program, putting the program into production, and maintaining it.
- flowcharts, pseudo-code
- sentinel value

1.8 Exercies - Identifying shapes

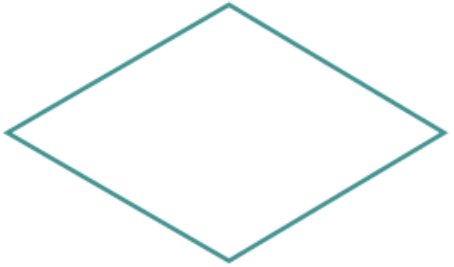
1. Input
2. Processing
3. Output
4. Decision
5. Terminal

A.

B.

C.

D.



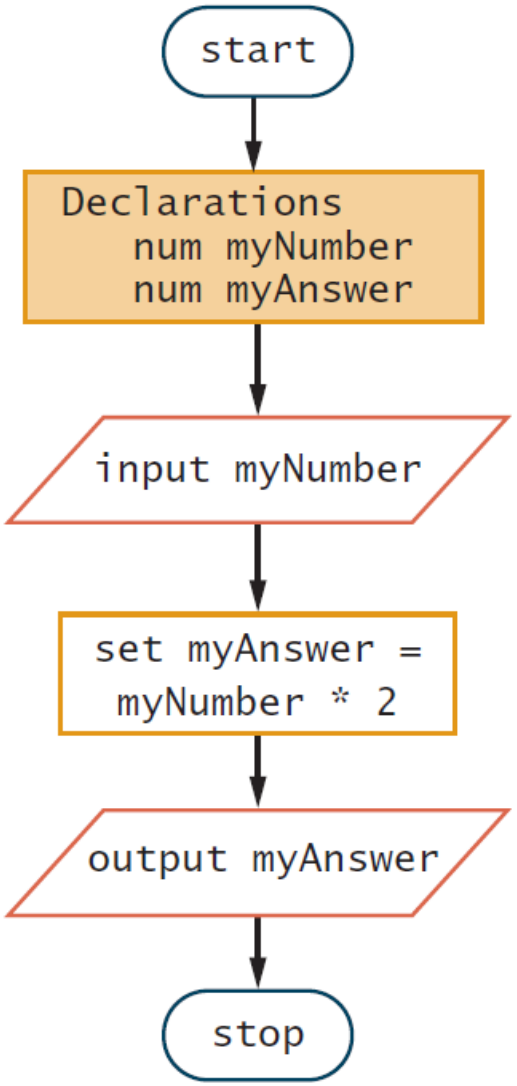
2. Elements of High-Quality Programs

- Declaring and Using Variables and Constants
- Understanding the Advantages of Modularization
- Modularizing a Program
- Creating Hierarchy Charts
- Features of Good Program Design
- Summary
- Exercises

2.1 Declaring and Using Variables and Constants

- Numeric constant, string constant
- Variables:
  - Declaration
    - identifier
    - data type
  - Garbage
- Naming Variables:
  - Camel casing: lastName
  - Pascal casing: LastName
  - Hungarian notation: stringLastName
  - Snake casing: last_name
  - Mixed case with underscores: Last_Name

Variable Declarations




```
start
  Declarations
     num myNumber
     num myAnswer
  input myNumber
  set myAnswer = myNumber * 2
  output myAnswer
stop
```

Rules for Naming Variables

- Variable names must be one word:
  - **interestRate**
- Variable names must start with a letter
- Variable names should have some appropriate meaning:
  - **set interestEarned = initialInvestment * interestRate**

Assigning Values to Variables

- assignment statement:
  - **set myAnswer = myNumber** * 2
  - **set someNumber** = 2
  - **set someNumber** = 3 + 7
  - **set someOtherNumber = someNumber**
  - **set someOtherNumber = someNumber** * 5
- some programming languages:
  - **myAnswer = myNumber** * 2
  - **someNumber** = 2

Data Types of Variables

- numeric variable
  - **myAnswer** = **myNumber** * 2
  - **taxRate** = 2.5
- string variable
  - **lastName** = "**Lincoln**"

Declaring Named Constants

- named constant:
  - **num SALES_TAX_RATE** = 0.06

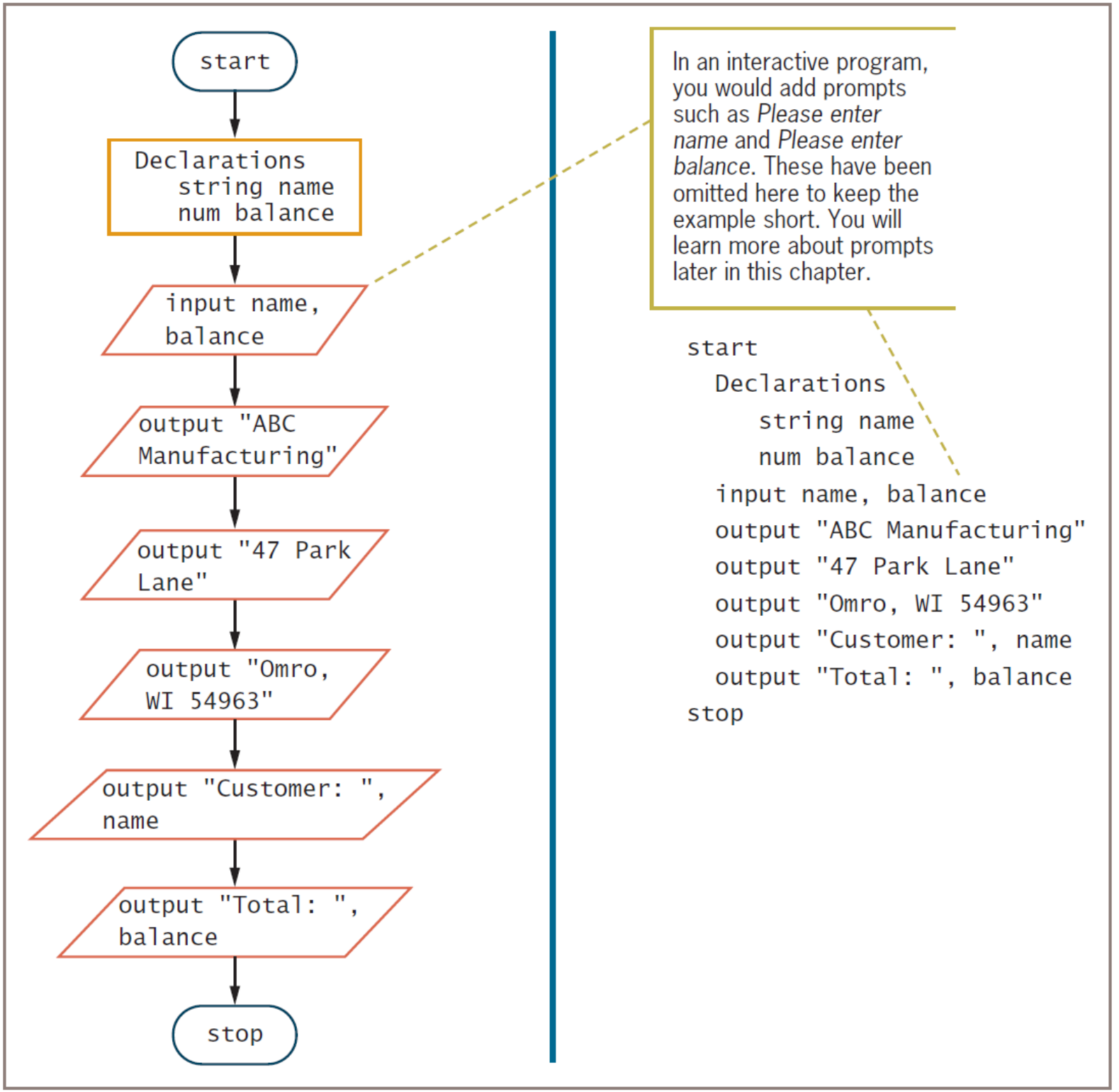
- identical meaning
  - **taxAmount** = **price** * 0.06
  - **taxAmount** = **price** * **SALES_TAX_RATE**

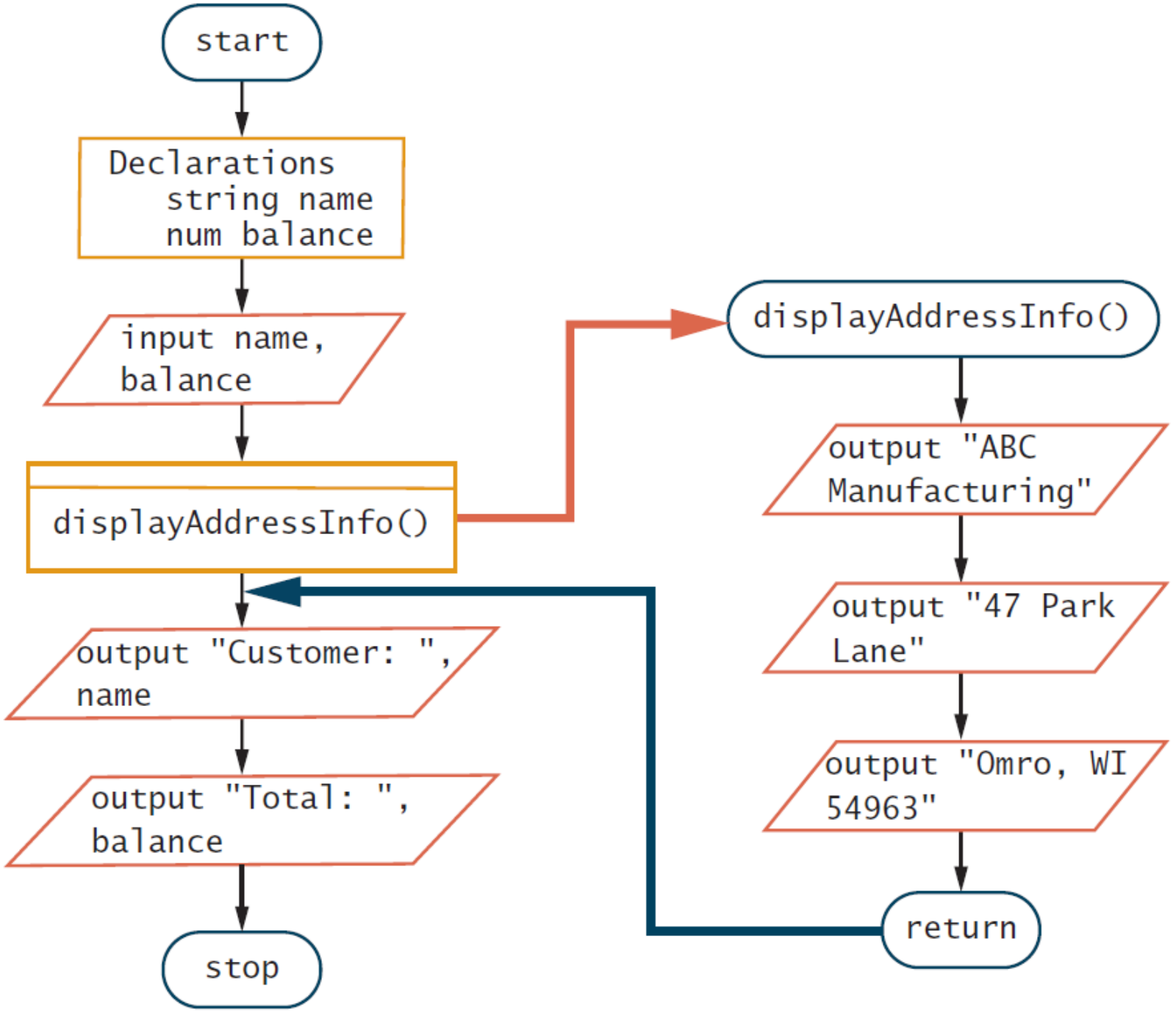
2.2 Understanding the Advantages of Modularization

- Modules, subroutines, procedures, functions, methods.
- Modularization, functional decomposition.
- Reasons:
  - Modularization provides abstraction
  - Modularization helps multiple programmers to work on a problem
  - Modularization allows you to reuse work more easily:
    - Reusability
    - Reliability

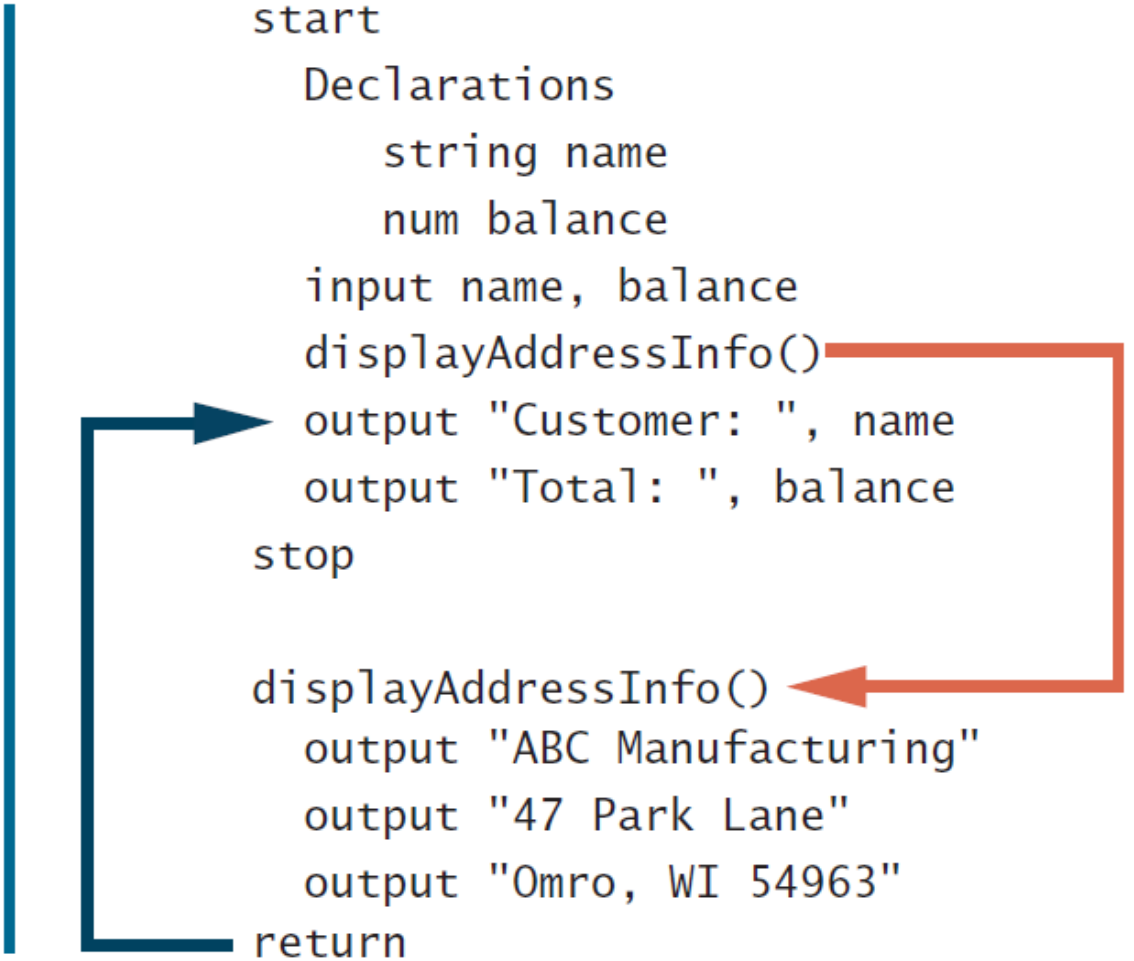
2.3 Modularizing a Program

- main program contains the basic steps, or the mainline logic
- create a module:
  - A header—The module header
  - A body—The module body
  - A **return** statement


start
Declarations
string name
num balance
input name, balance
output "ABC Manufacturing"
output "47 Park Lane"
output "Omro, WI 54963"
output "Customer: ", name
output "Total: ", balance
stop
In an interactive program, you would add prompts such as *Please enter name* and *Please enter balance*. These have been omitted here to keep the example short. You will learn more about prompts later in this chapter.
start
Declarations
string name
num balance
input name, balance
output "ABC Manufacturing"
output "47 Park Lane"
output "Omro, WI 54963"
output "Customer: ", name
output "Total: ", balance
stop


start
Declarations
string name
num balance
input name,
balance
displayAddressInfo()
output "Customer: ",
name
output "Total: ",
balance
stop
displayAddressInfo()
output "ABC
Manufacturing"
output "47 Park
Lane"
output "Omro, WI
54963"
return

Main Program that Calls displayAddressInfo() Module



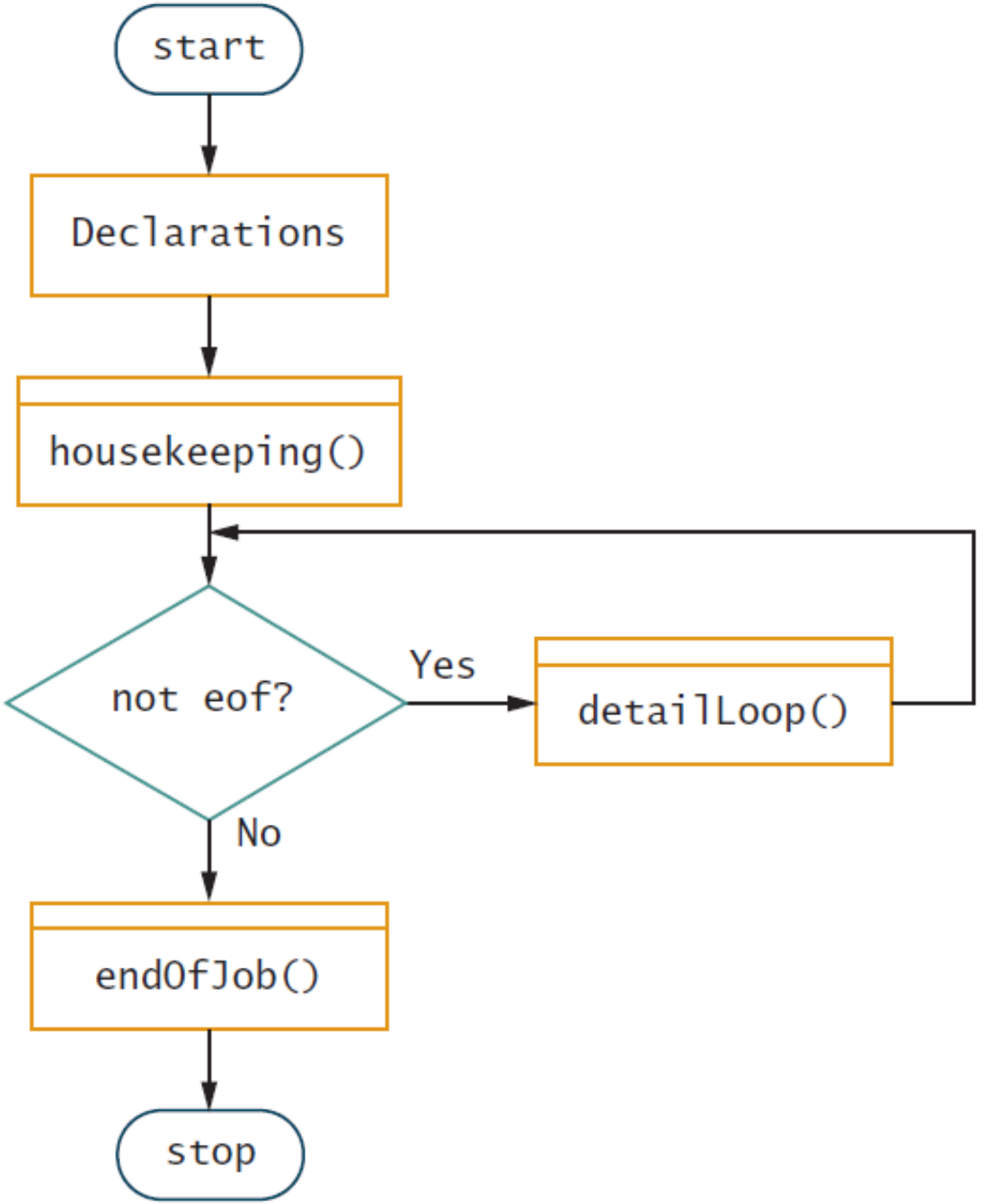
Declaring Variables and Constants within Modules

```
start
  Declarations
     string name
     num balance
  input name, balance
  displayAddressInfo()
  output "Customer: ", name
  output "Total: ", balance
stop

displayAddressInfo()
  Declarations
     string LINE1 = "ABC Manufacturing"
     string LINE2 = "47 Park Lane"
     string LINE3 = "Omro, WI 54963"
  output LINE1
  output LINE2
  output LINE3
return
```

Understanding the Most Common Configuration for Mainline Logic

- **Housekeeping tasks**: perform at the beginning of a program
  - variable and constant declarations
  - displaying instructions
  - displaying report headings
  - inputting the first piece of data
- **Detail loop tasks** do the core work: execute repeatedly for each set of input data until there are no more.
- **End-of-job tasks**: displaying totals or other final messages and closing any open files.




```
start
  Declarations
  housekeeping()
  while not eof
     detailLoop()
  endwhile
  endOfJob()
stop
```

Sample Payroll Report

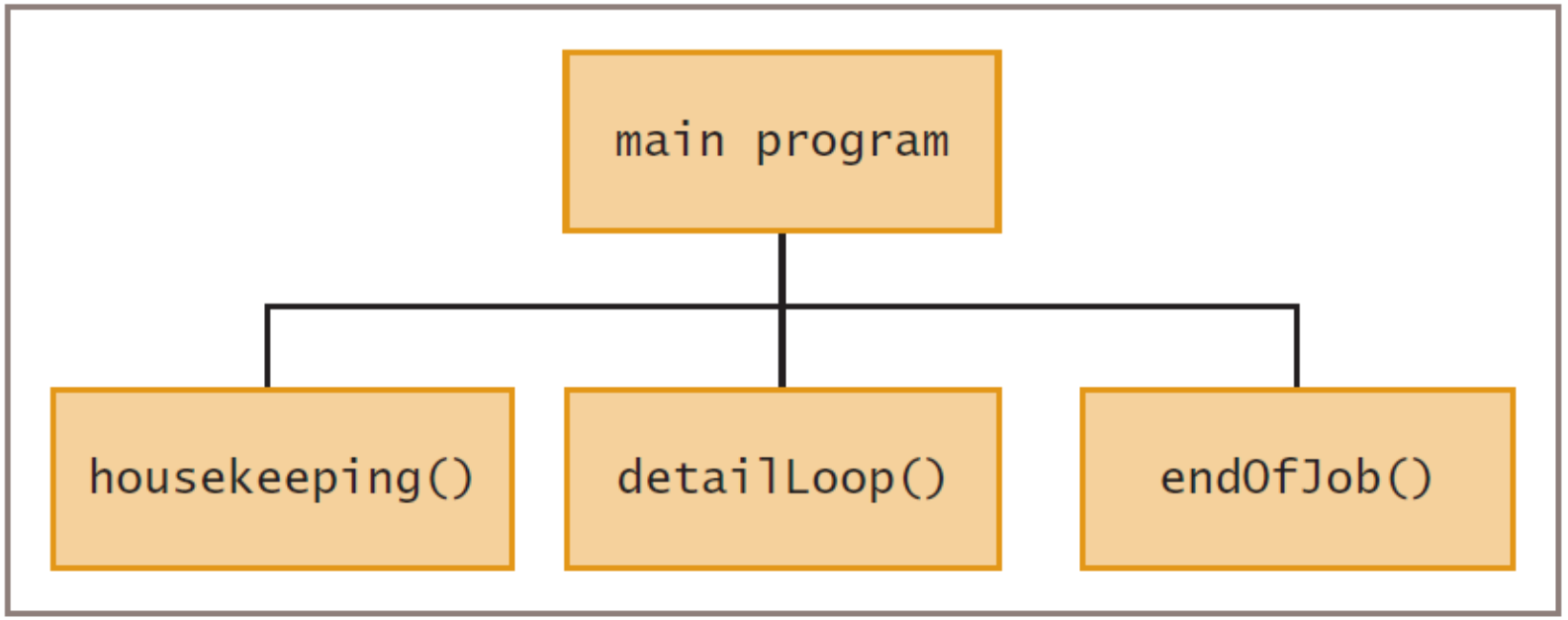
Payroll Report

Name	Gross	Deductions	Net
Andrews	1000.00	250.00	750.00
Brown	1400.00	350.00	1050.00
Carter	1275.00	318.75	956.25
Young	1100.00	275.00	825.00

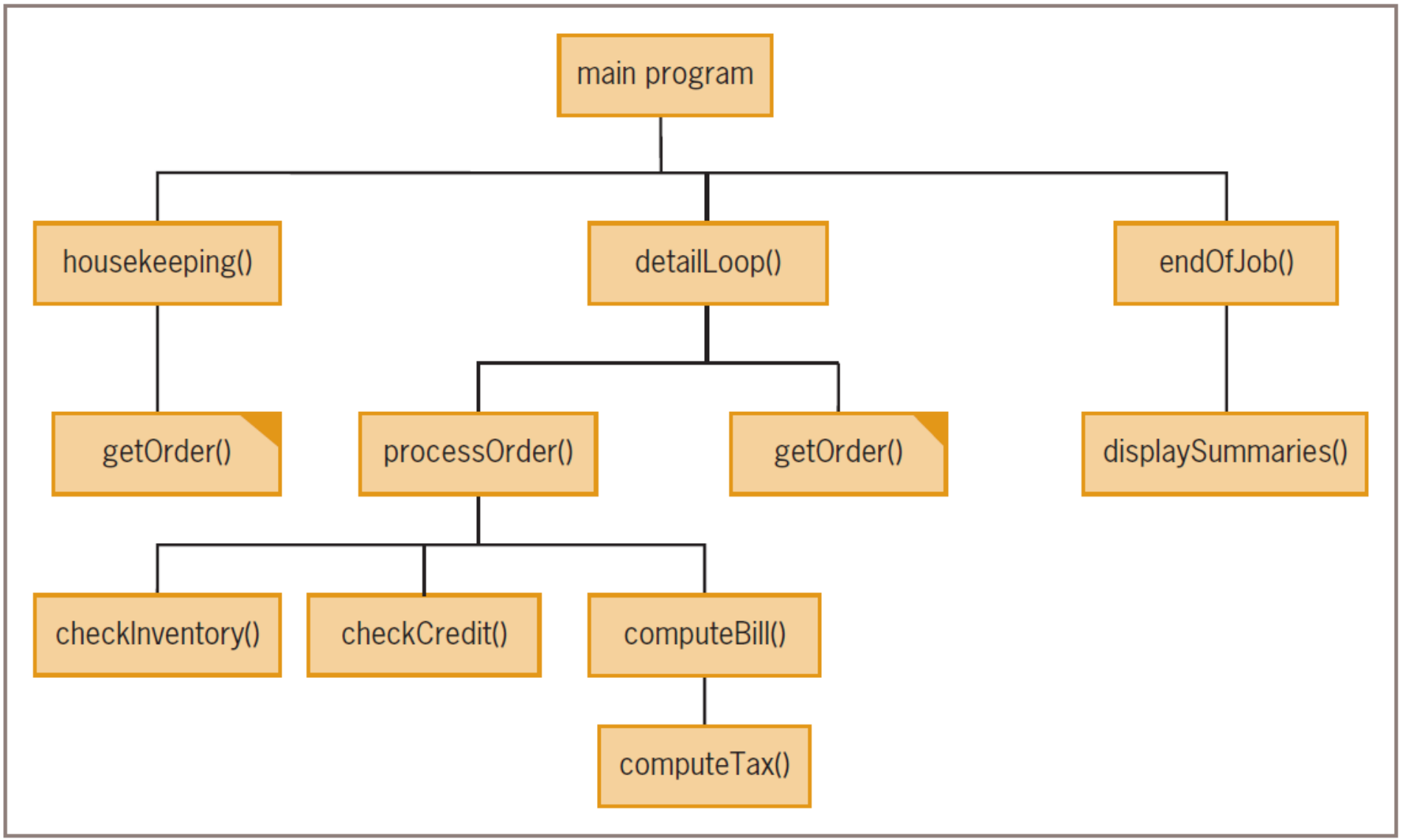
***End of report

2.4 Creating Hierarchy Charts

- A hierarchy chart tells you only which modules exist within a program and which modules call others.



Billing program hierarchy chart



2.5 Features of Good Program Design

- Provide program comments where appropriate
- Choose identifiers thoughtfully
- Strive to design clear statements within your programs and modules
- Write clear prompts and echo input
- Continue to maintain good programming habits as you develop your programming skills

2.6 Summary

- Programs contain data in three different forms: literals (or unnamed constants), variables, and named constants.
- Programmers break down programming problems into smaller, cohesive units called modules, subroutines, procedures, functions, or methods.
- When you create a module, you include a header, a body, and a return statement.
- A hierarchy chart illustrates modules and their relationships.
- As programs become larger and more complicated, the need for good planning and design increases.

2.7 Exercises

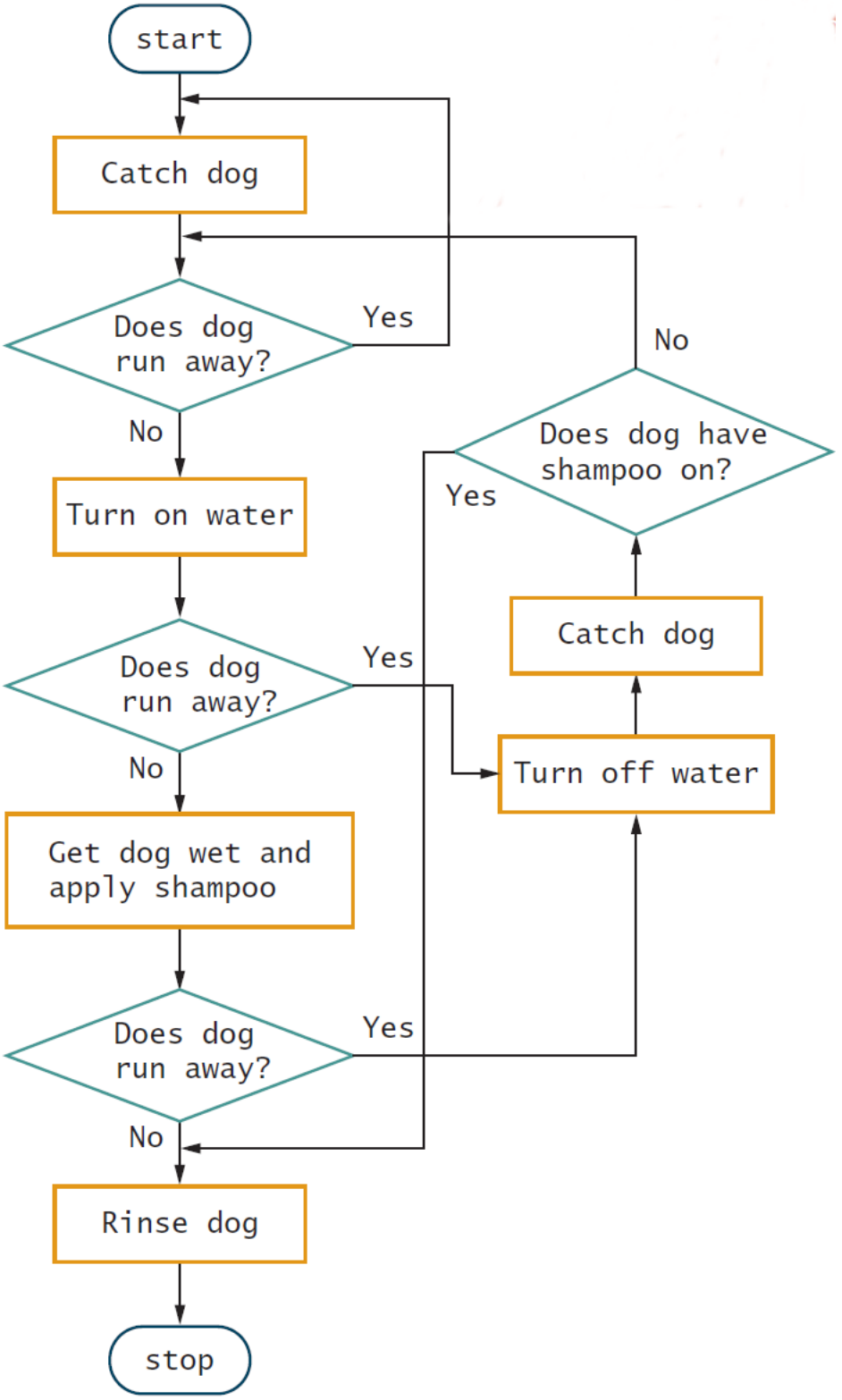
- Draw the hierarchy chart and then plan the logic for a program that calculates a person's body mass index (BMI). BMI is a statistical measure that compares a person's weight and height. The program uses three modules. The first prompts a user for and accepts the user's height in inches. The second module accepts the user's weight in pounds and converts the user's height to meters and weight to kilograms. Then, it calculates BMI as weight in kilograms divided by height in meters squared, and displays the results. There are 2.54 centimeters in an inch, 100 centimeters in a meter, 453.59 grams in a pound, and 1,000 grams in a kilogram. Use named constants whenever you think they are appropriate. The last module displays the message *End of job*.
- Revise the BMI-determining program to execute continuously until the user enters 0 for the height in inches.

3. Understanding Structure

- The Disadvantages of Unstructured Spaghetti Code
- Understanding the Three Basic Structures
- Using a Priming Input to Structure a Program
- Understanding the Reasons for Structure
- Recognizing Structure
- Structuring and Modularizing Unstructured Logic
- Chapter Summary
- Exercises

3.1 The Disadvantages of Unstructured Spaghetti Code

- spaghetti code, unstructured programs
- might produce correct results
- difficult to read and maintain
- its logic would be hard to follow
- prone to error
- difficult to reuse

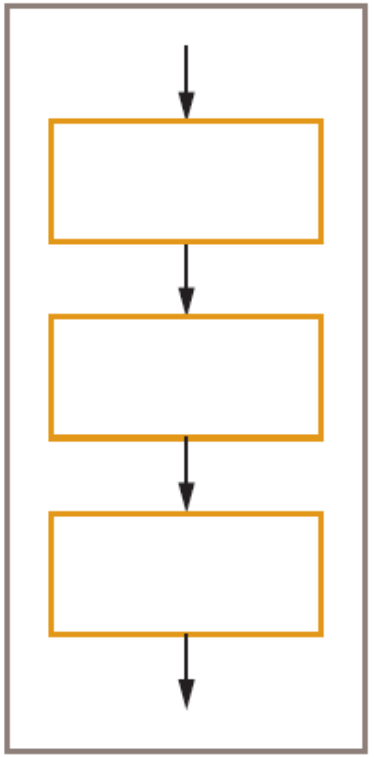

start
Catch dog
Does dog run away?
Yes
No
Turn on water
Does dog run away?
Yes
No
Get dog wet and apply shampoo
Does dog run away?
Yes
No
Rinse dog
stop
Does dog have shampoo on?
No
Yes
Catch dog
Turn off water

3.2 Understanding the Three Basic Structures

- Any program can be constructed using one or more of only three structures:
  - sequence
  - selection
  - loop
- A structure is a basic unit of programming logic

The Sequence Structure

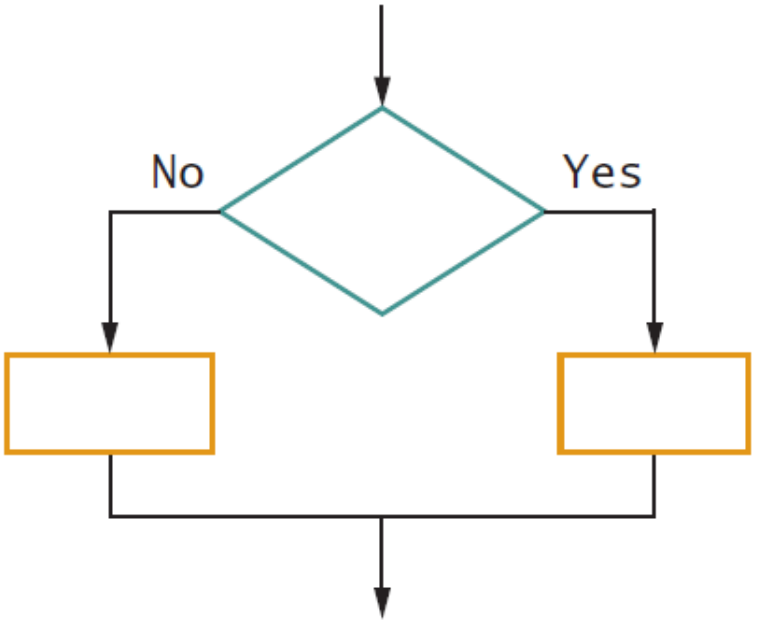
- A sequence can contain any number of tasks, but there is no option to branch off and skip any of the tasks.



The Selection Structure

- One of two courses of action is taken based on the answer to a question.
- A flowchart that describes a selection structure begins with a decision symbol, and the branches of the decision must join at the bottom of the structure.

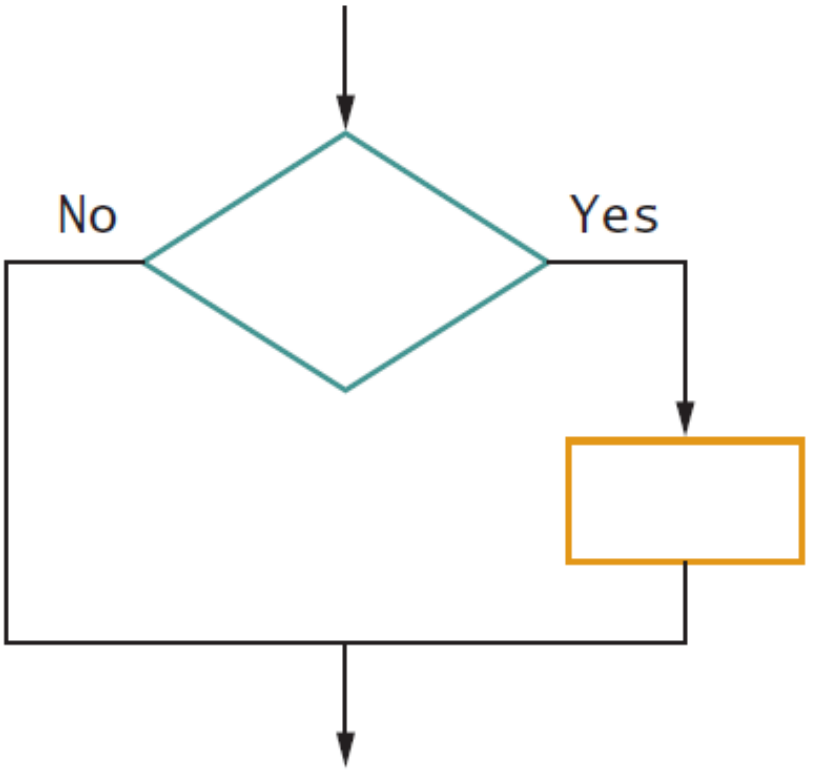
```
if someCondition is true then
   do oneProcess
else
   do theOtherProcess
endif
```



Single-alternative selection structure

- Note that it is perfectly correct for one branch of the selection to be a “do nothing” branch.

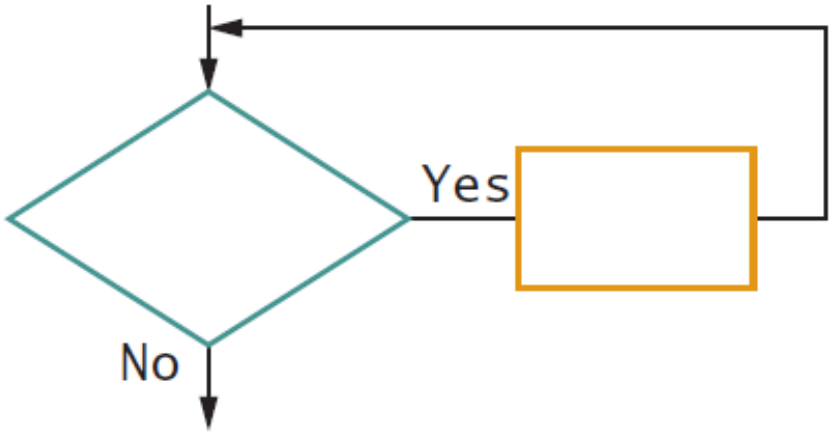
```
if it is raining then
   take an umbrella
endif
```



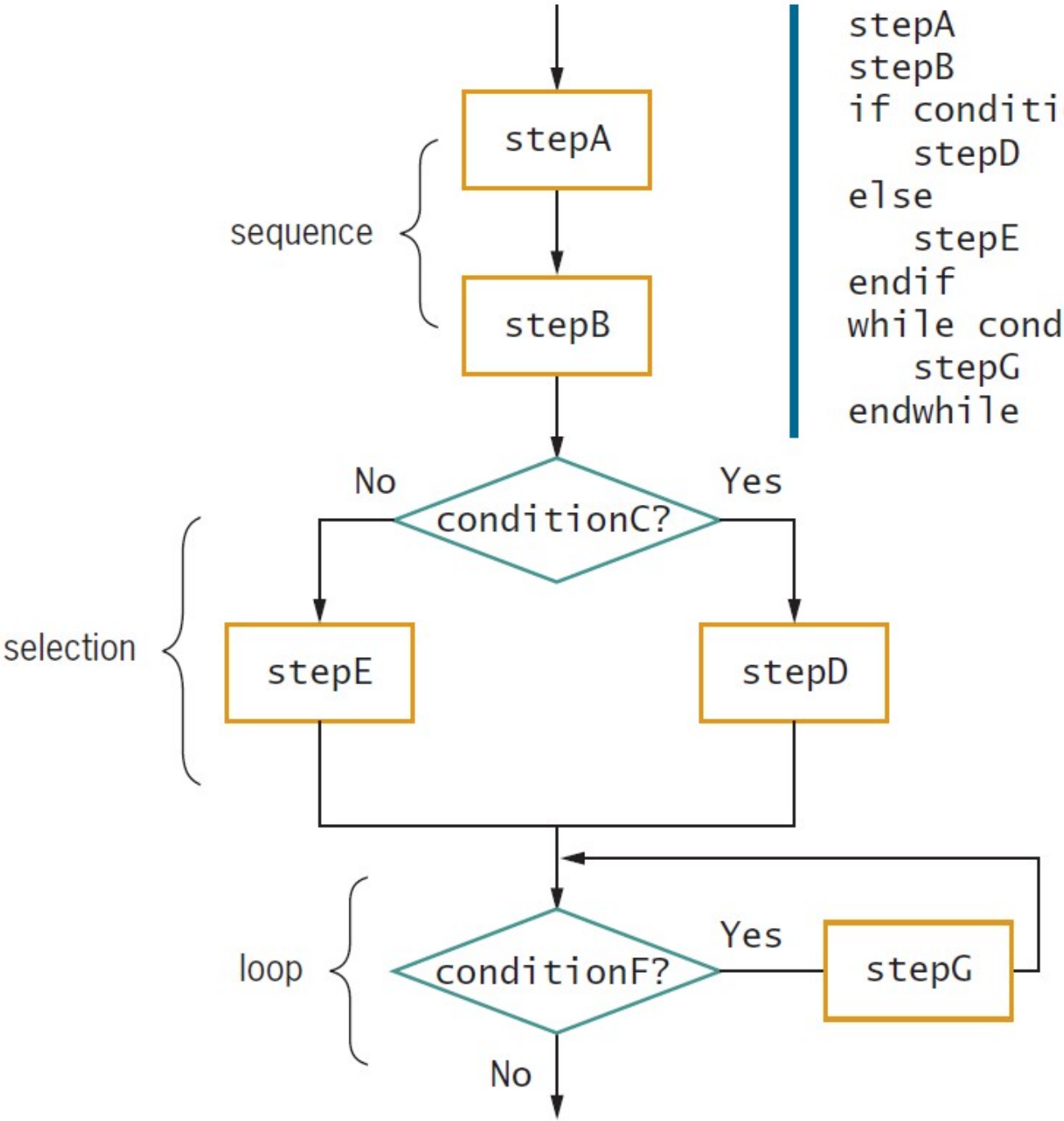
The Loop Structure

- A loop continues to repeat actions while a condition remains true.

```
while testCondition continues to be true do
    someProcess
endwhile
```

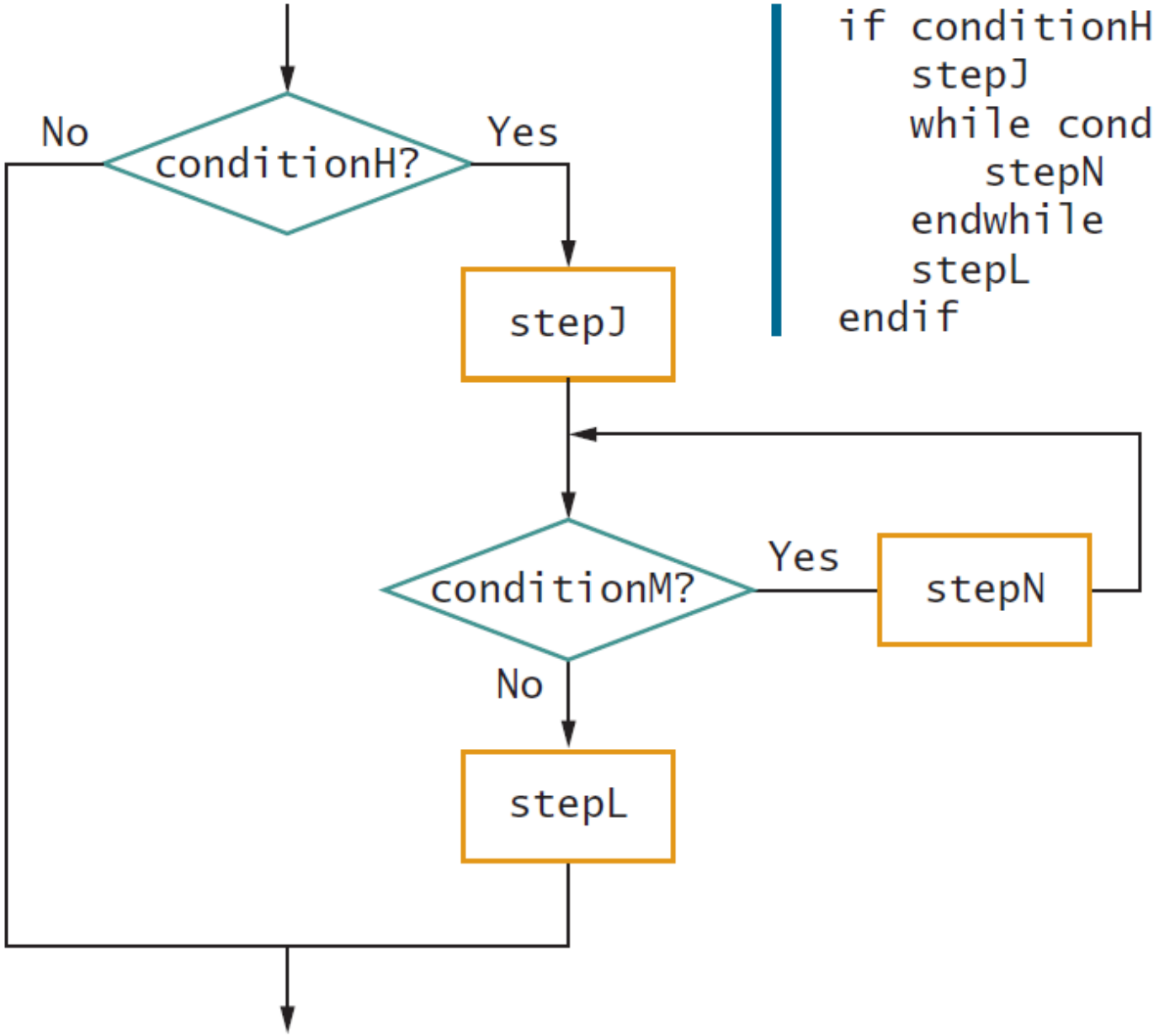


Combining Structures - stacking structures

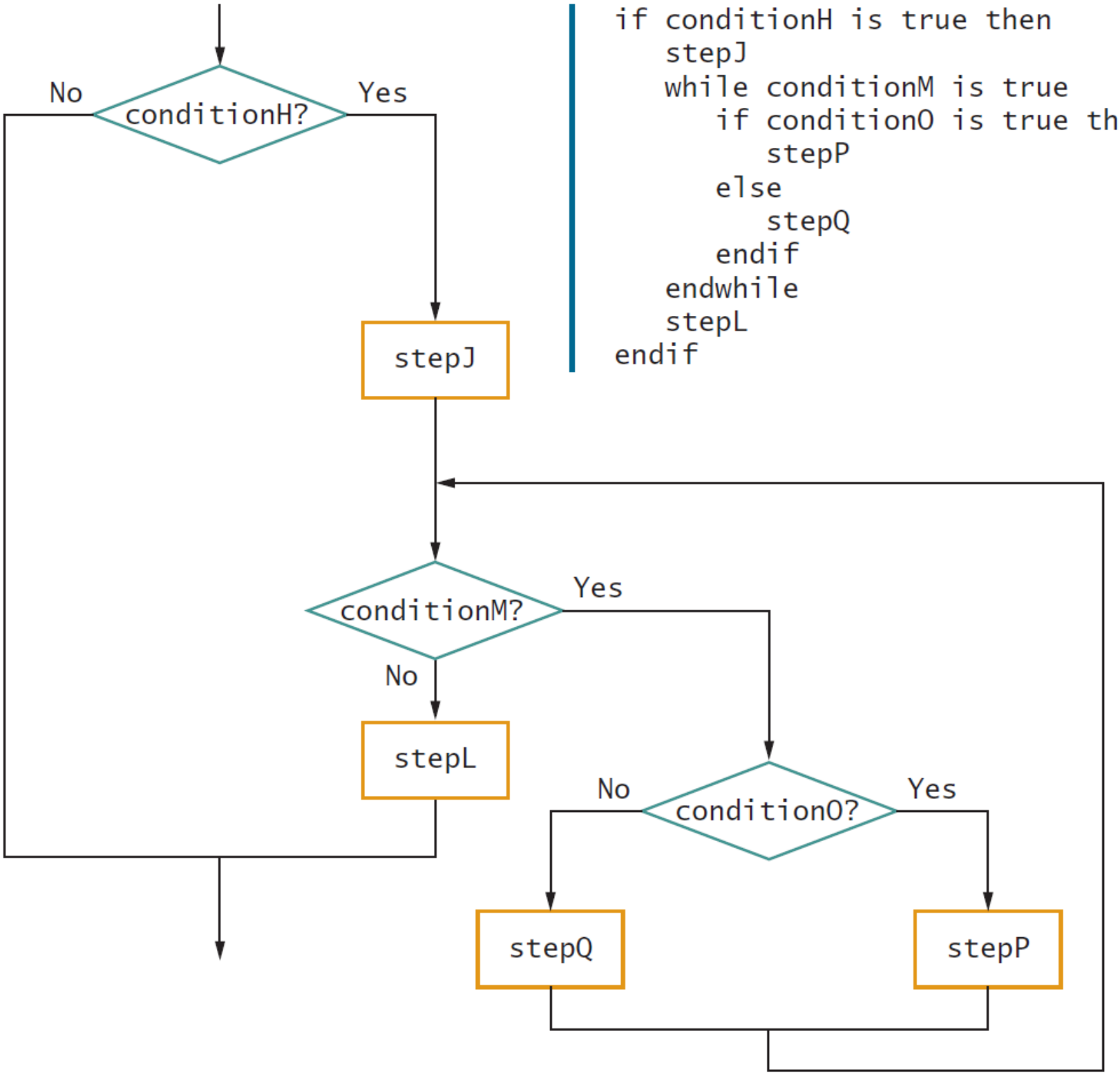



```
stepA
stepB
if conditionC is true then
   stepD
else
   stepE
endif
while conditionF is true
   stepG
endwhile
```

Combining Structures - nesting structures

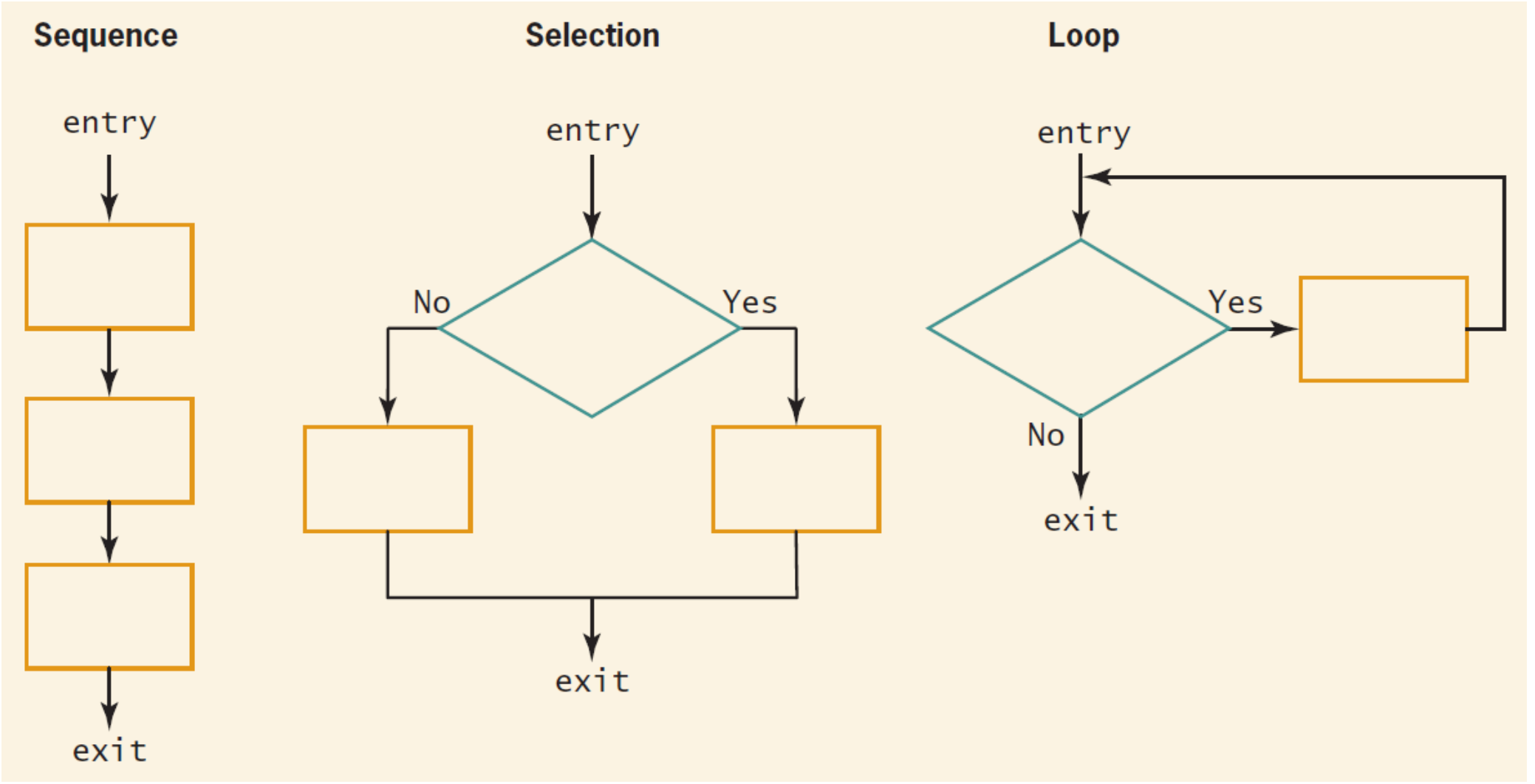



```
if conditionH is true then
   stepJ
   while conditionM is true
      stepN
   endwhile
   stepL
endif
```

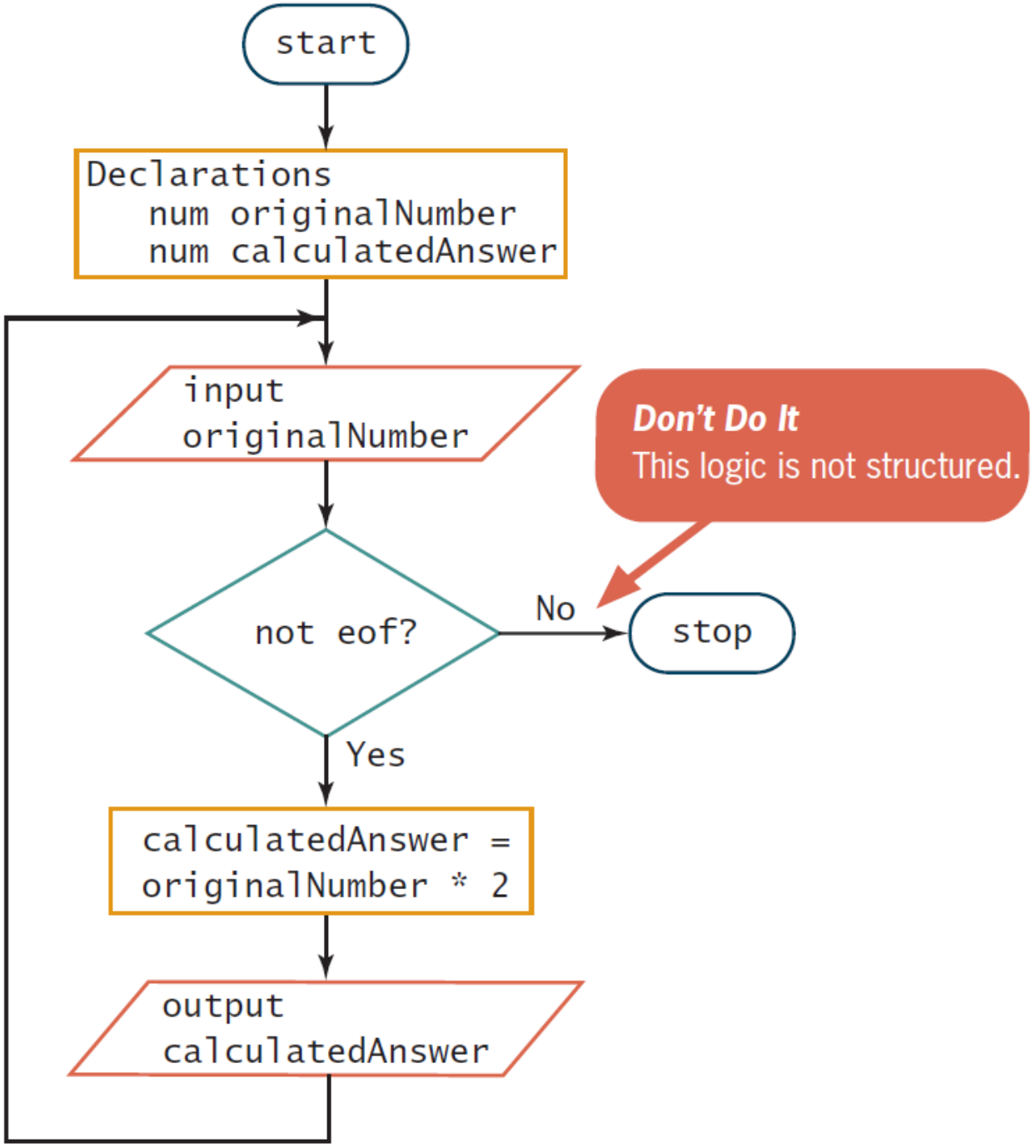



```
if conditionH is true then
   stepJ
   while conditionM is true
      if conditionO is true then
         stepP
      else
         stepQ
      endif
   endwhile
   stepL
endif
```

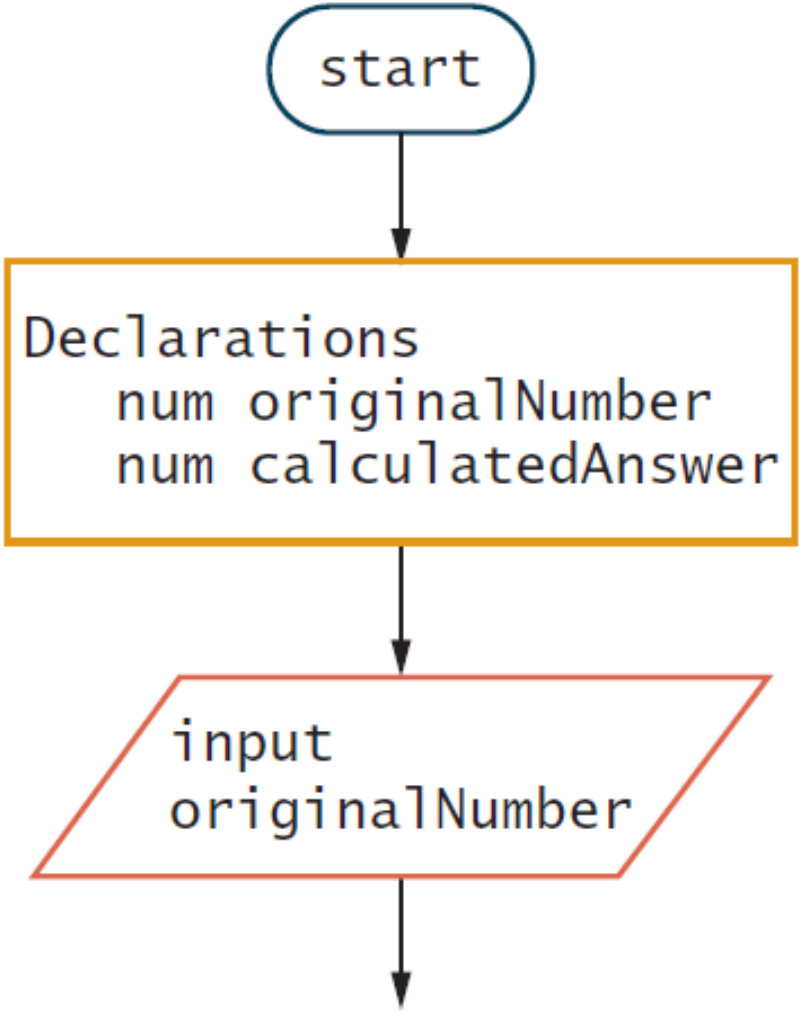
The Three Structures



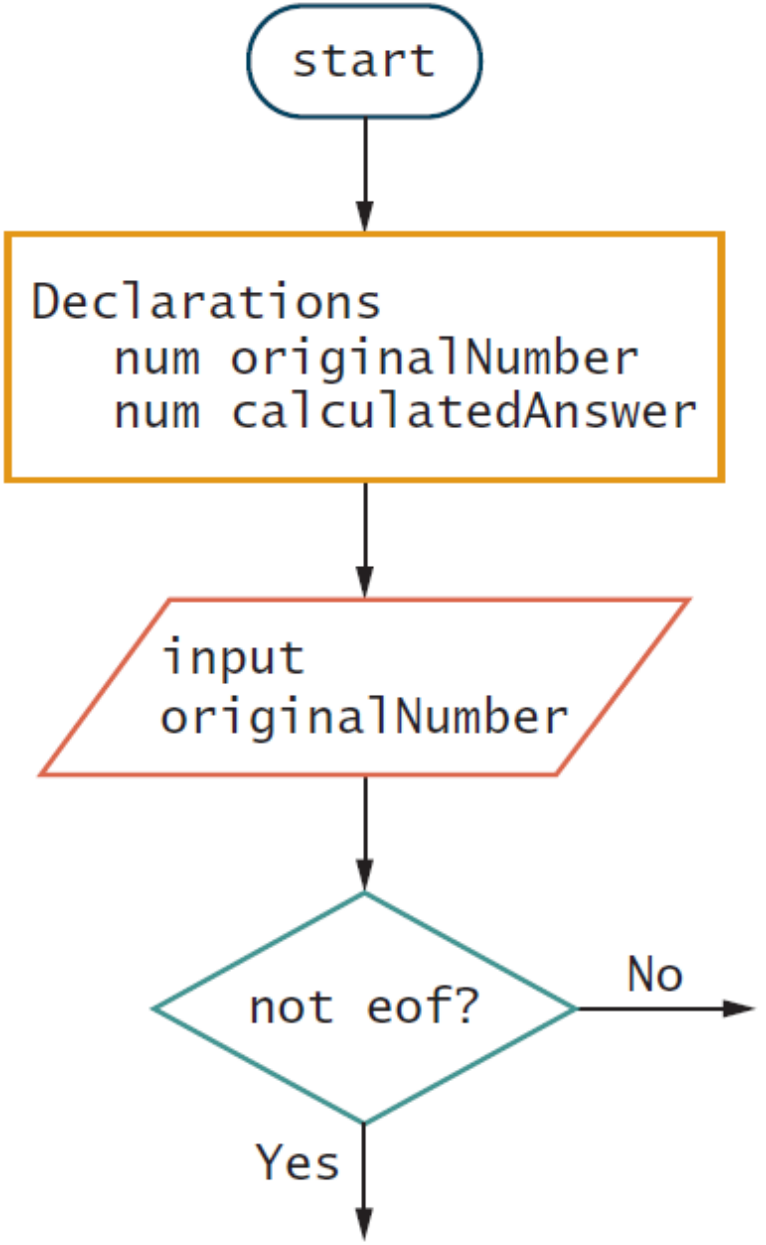
3.3 Using a Priming Input to Structure a Program



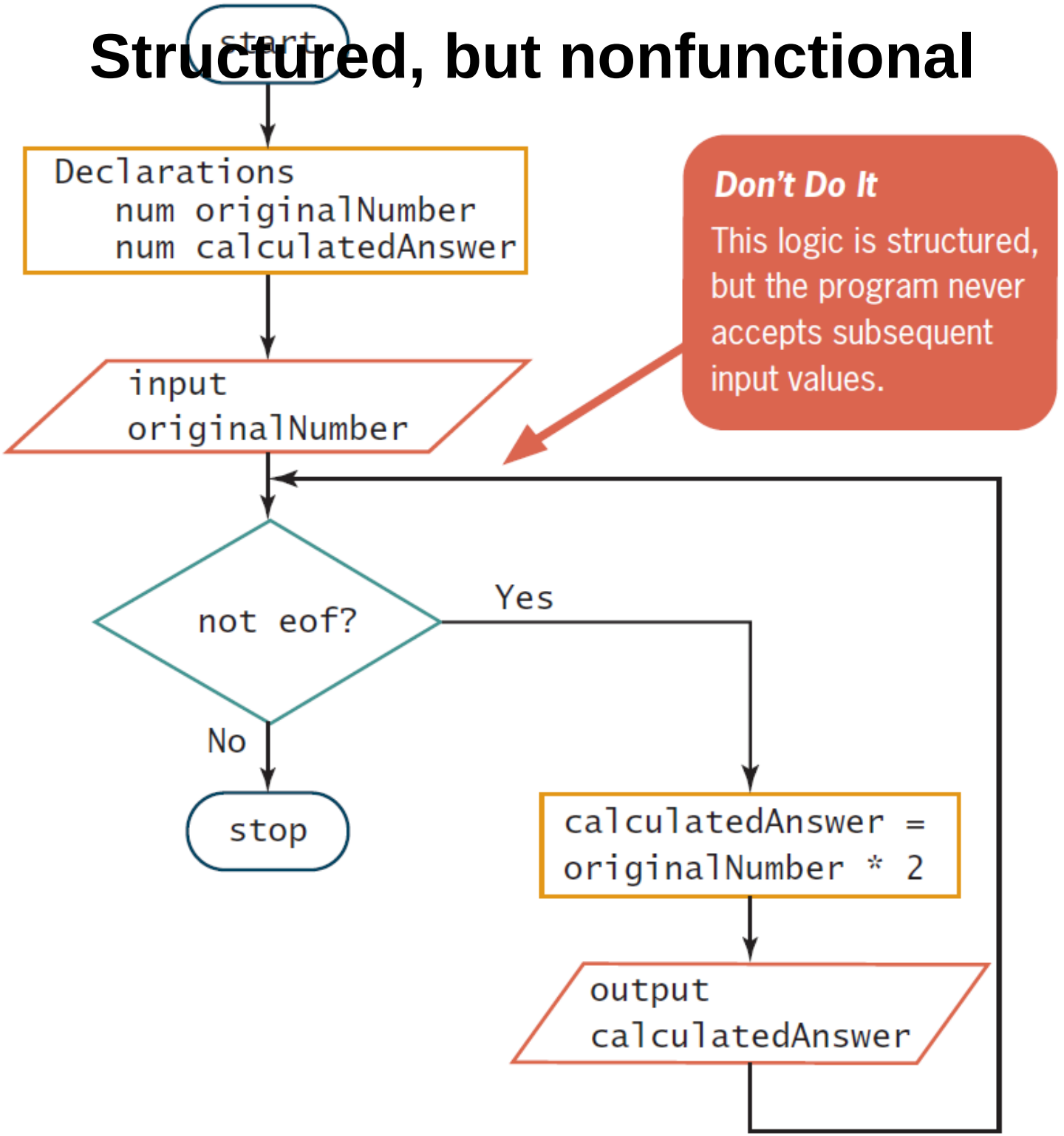
- Is this portion of the flowchart structured? Yes, it is a sequence of two tasks—making declarations and inputting a value



- In a structured while loop, the rules are:
  - ask a question
  - If the answer indicates you should execute the loop body, then you do so.
  - After you execute the loop body, then you must go right back to ask the question again.

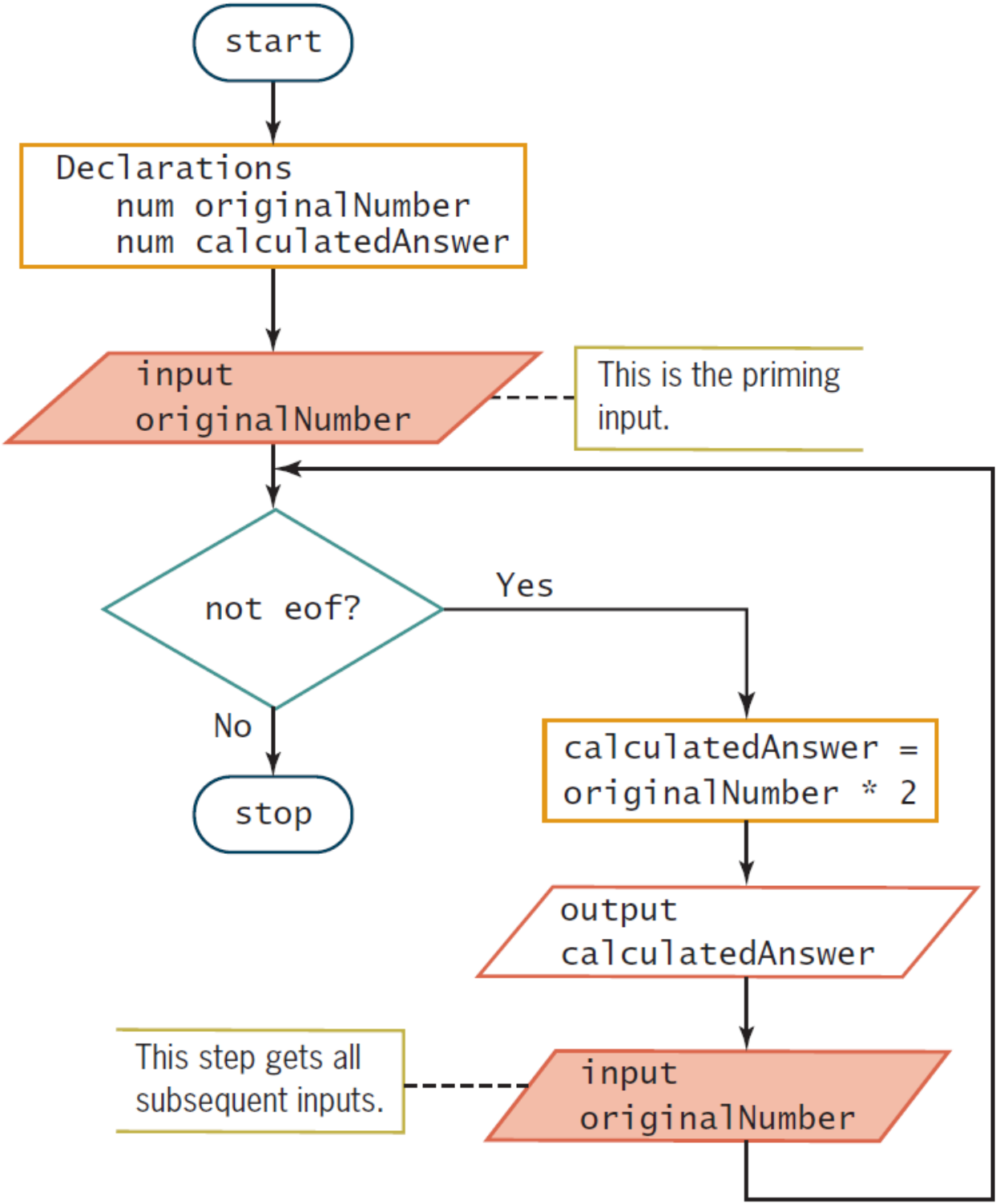


Structured, but nonfunctional

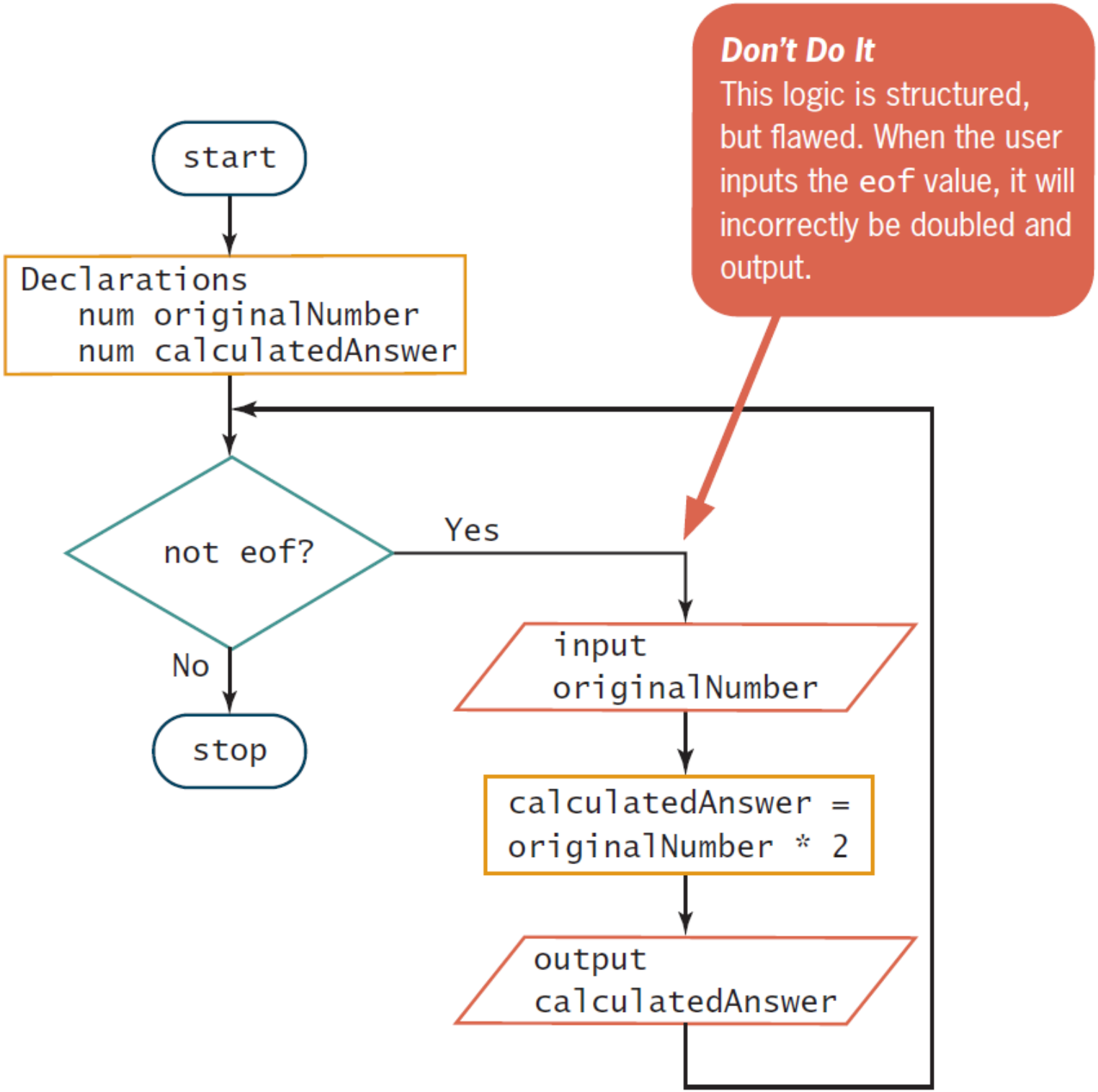


structured and work as intended?

- A **priming input** or **priming read** is an added statement that gets the first input value in a program.



Structured but incorrect solution

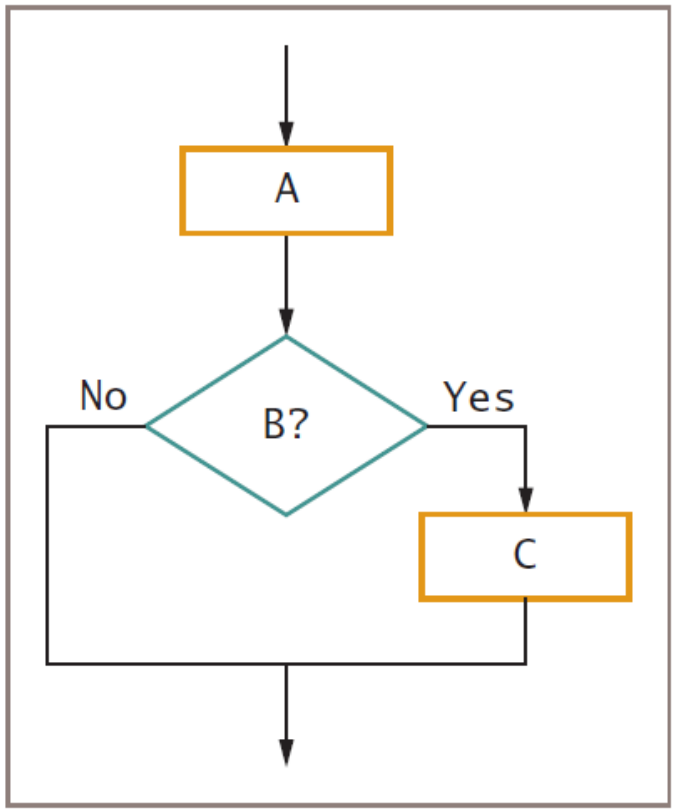


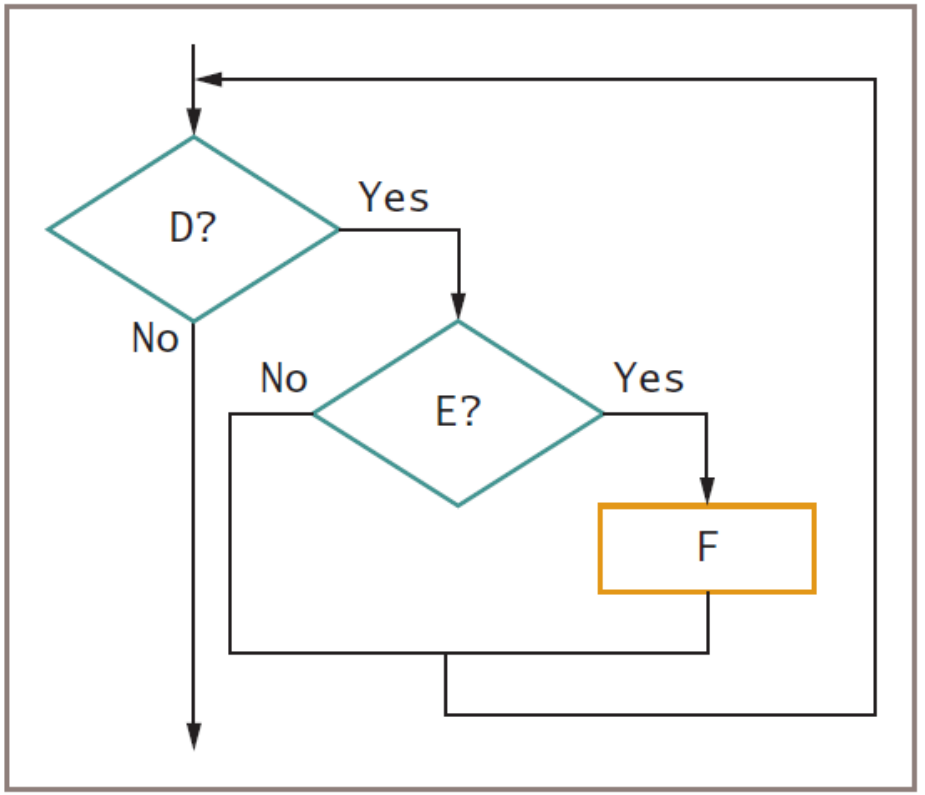
3.4 Understanding the Reasons for Structure

- Clarity—The number-doubling program is small. As programs get bigger, they get more confusing if they are not structured.
- Professionalism—All other programmers (and programming teachers youmight encounter) expect your programs to be structured. It is the way things are done professionally.
- Efficiency—Most newer computer languages support structure and use syntax that lets you deal efficiently with sequence, selection, and looping.
- Maintenance—You and other programmers will find it easier to modify and maintain structured programs as changes are required in the future.
- Modularity—Structured programs can be easily broken down into modules that can be assigned to any number of programmers.

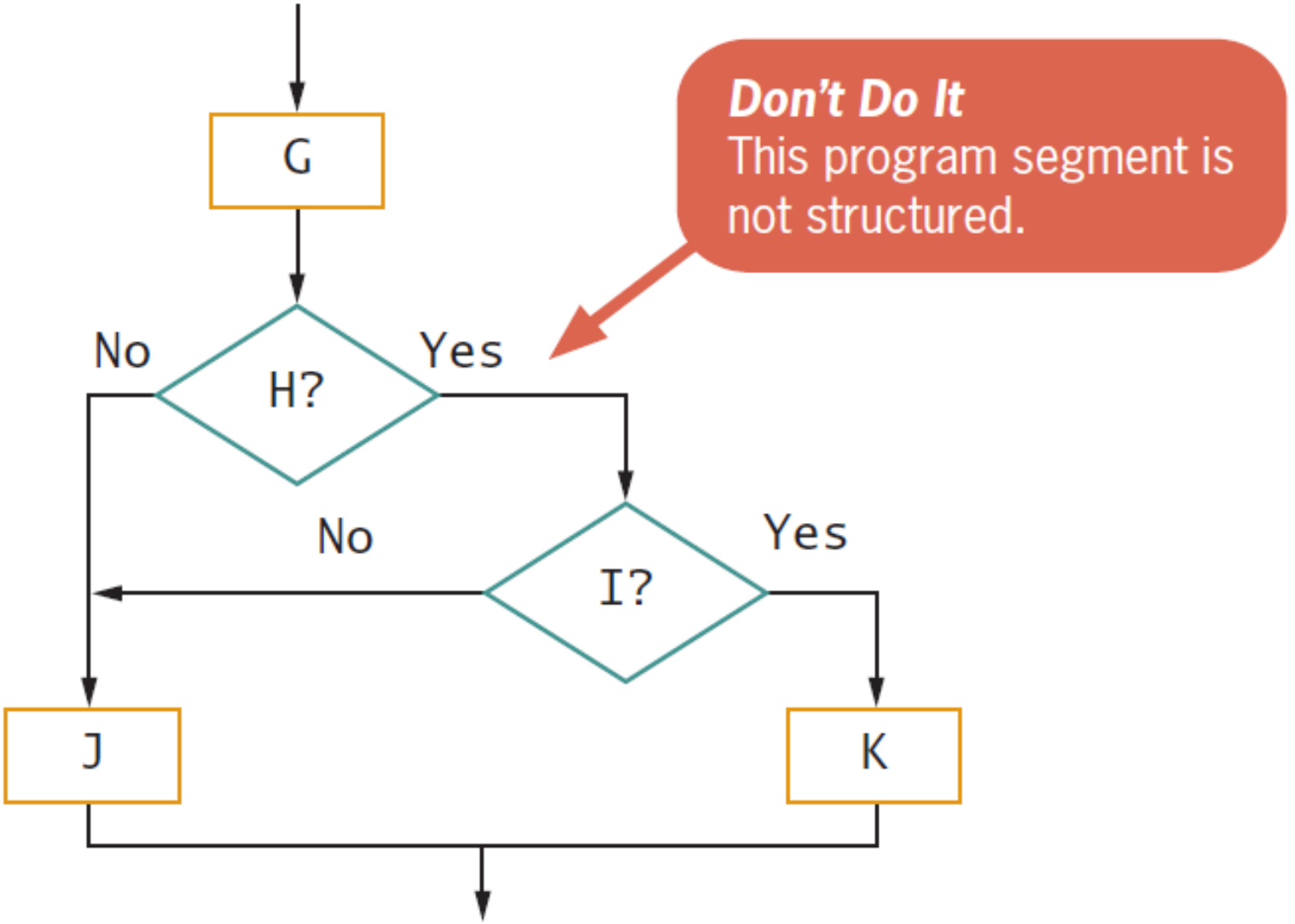
3.5 Recognizing Structure

- It is difficult to detect whether a flowchart of a program's logic is structured.

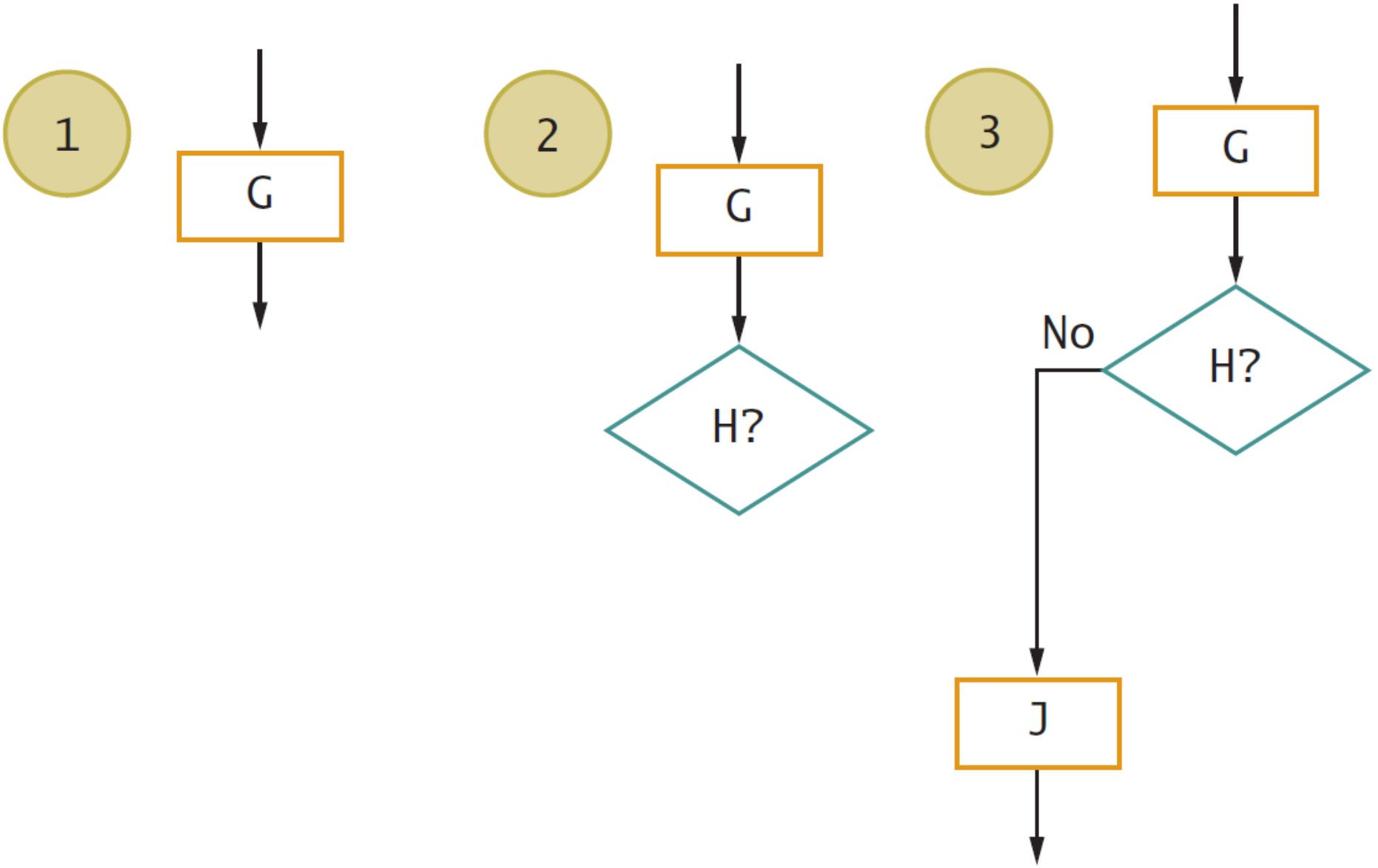




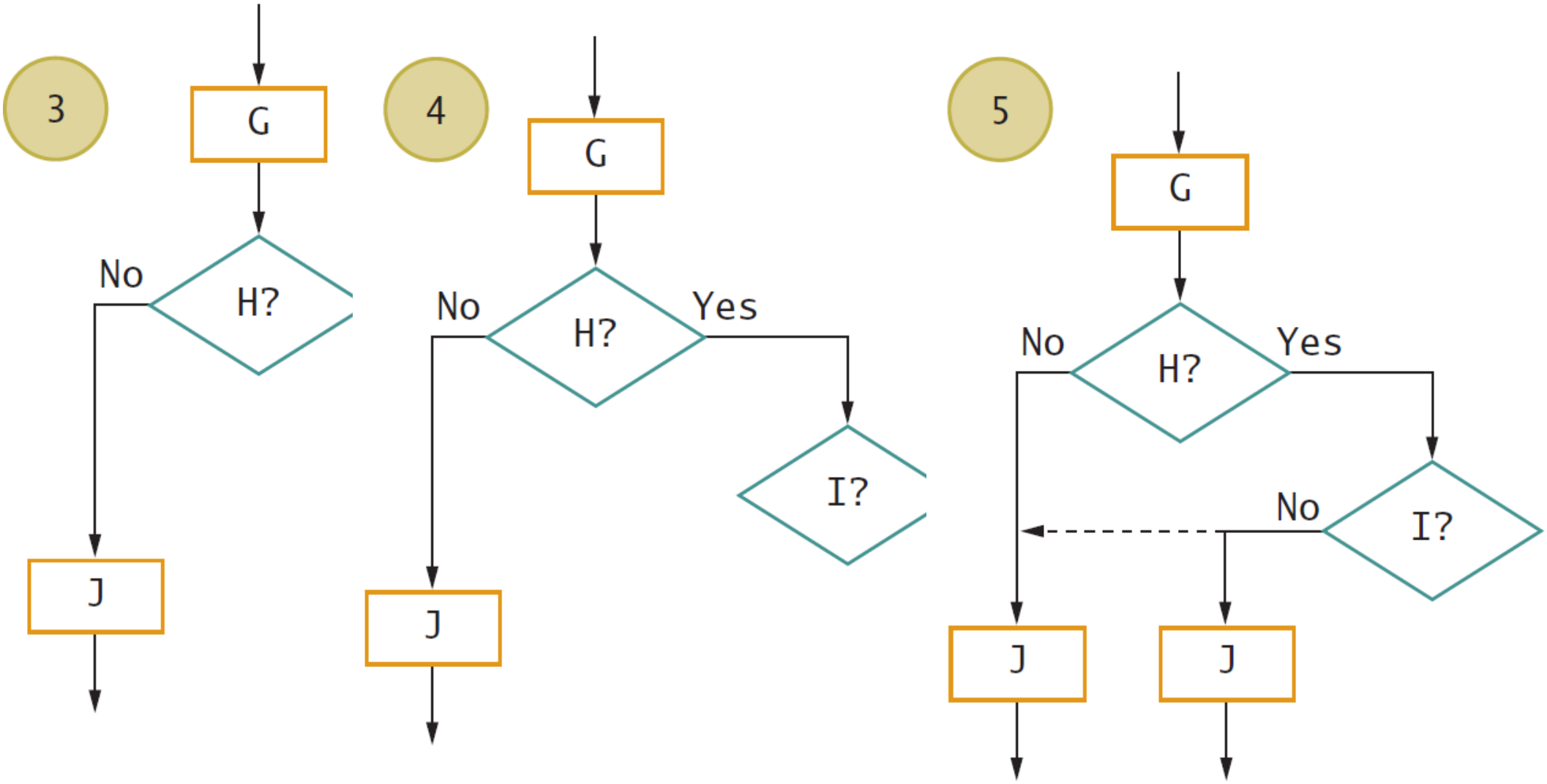
The “spaghetti bowl” Method



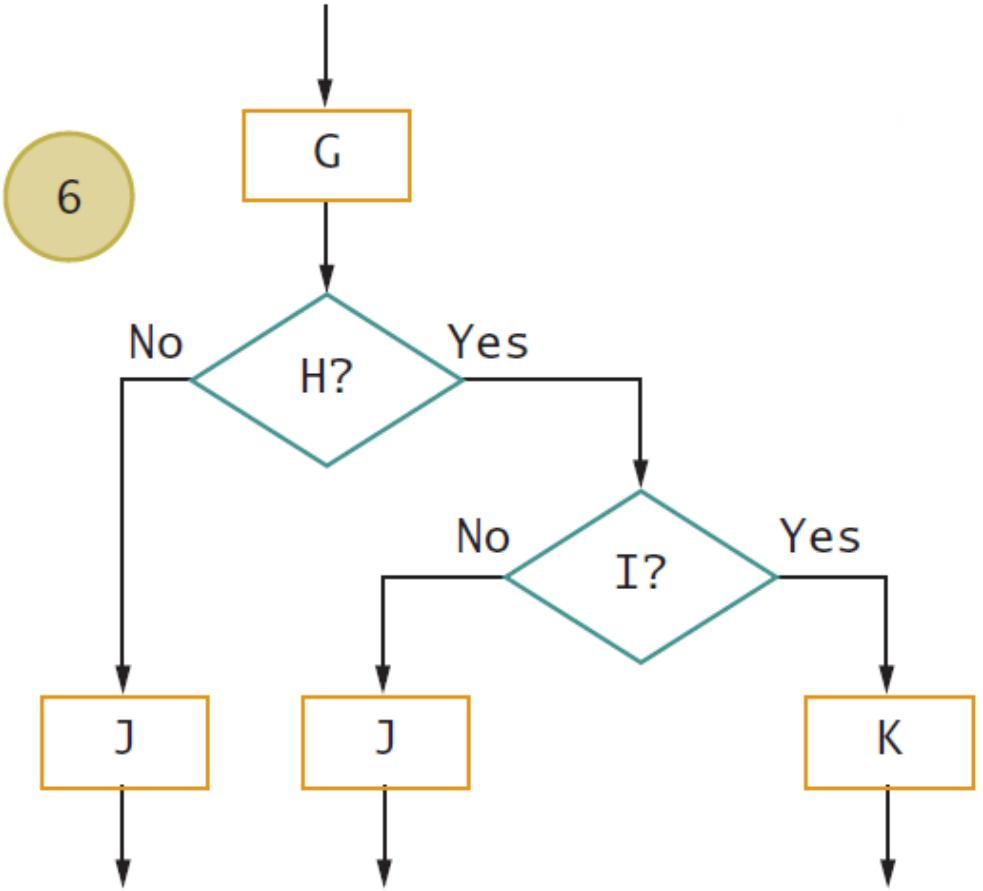
The “spaghetti bowl” Method

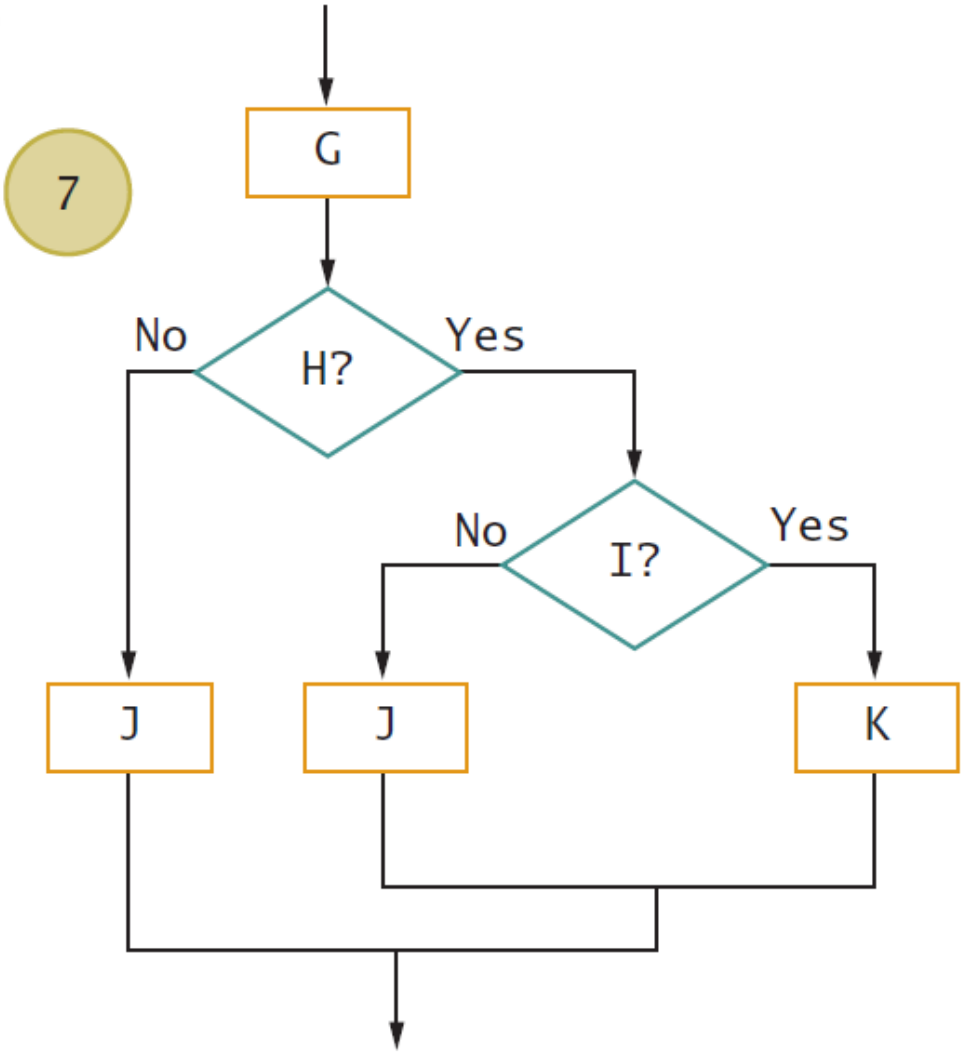


The “spaghetti bowl” Method

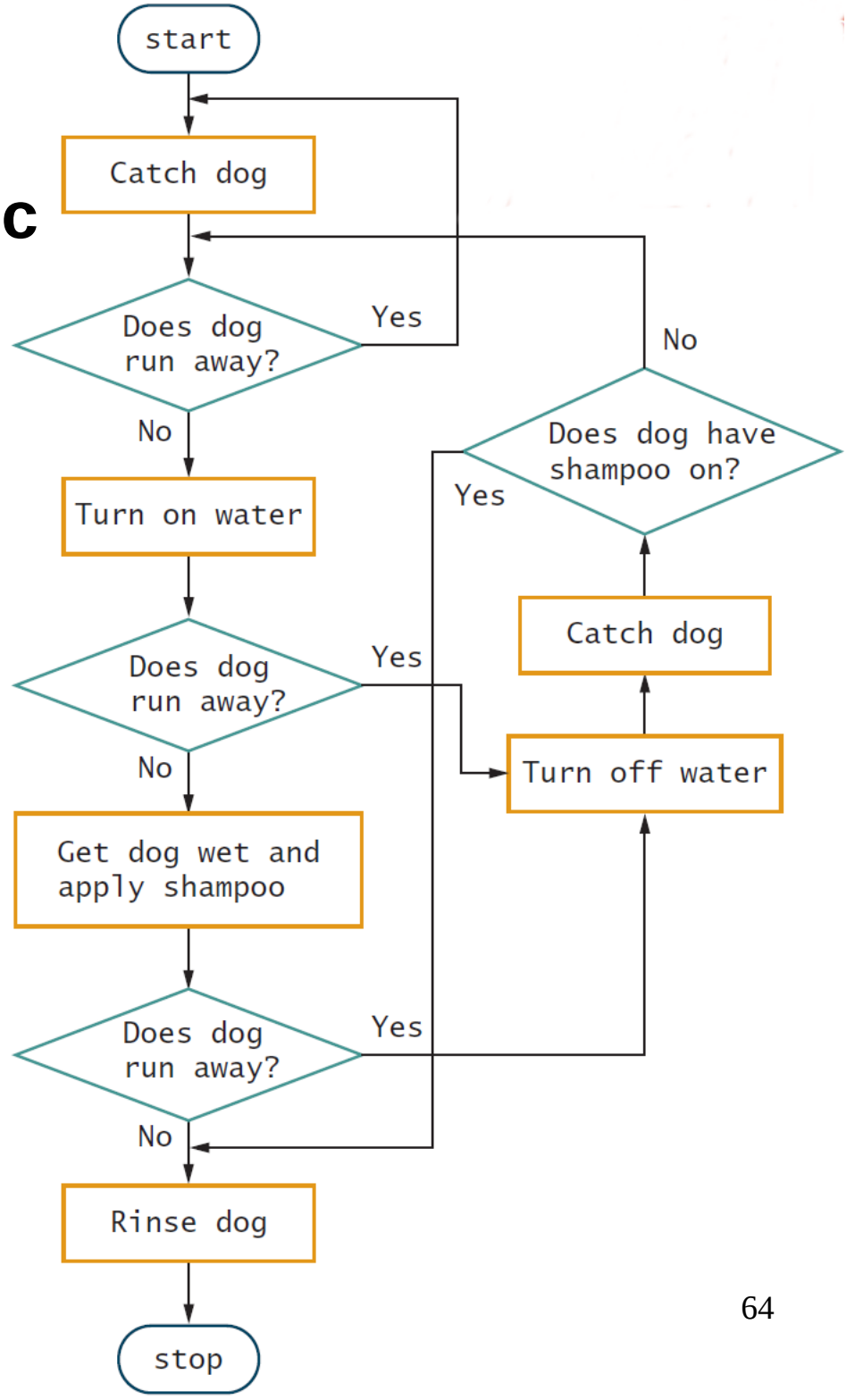


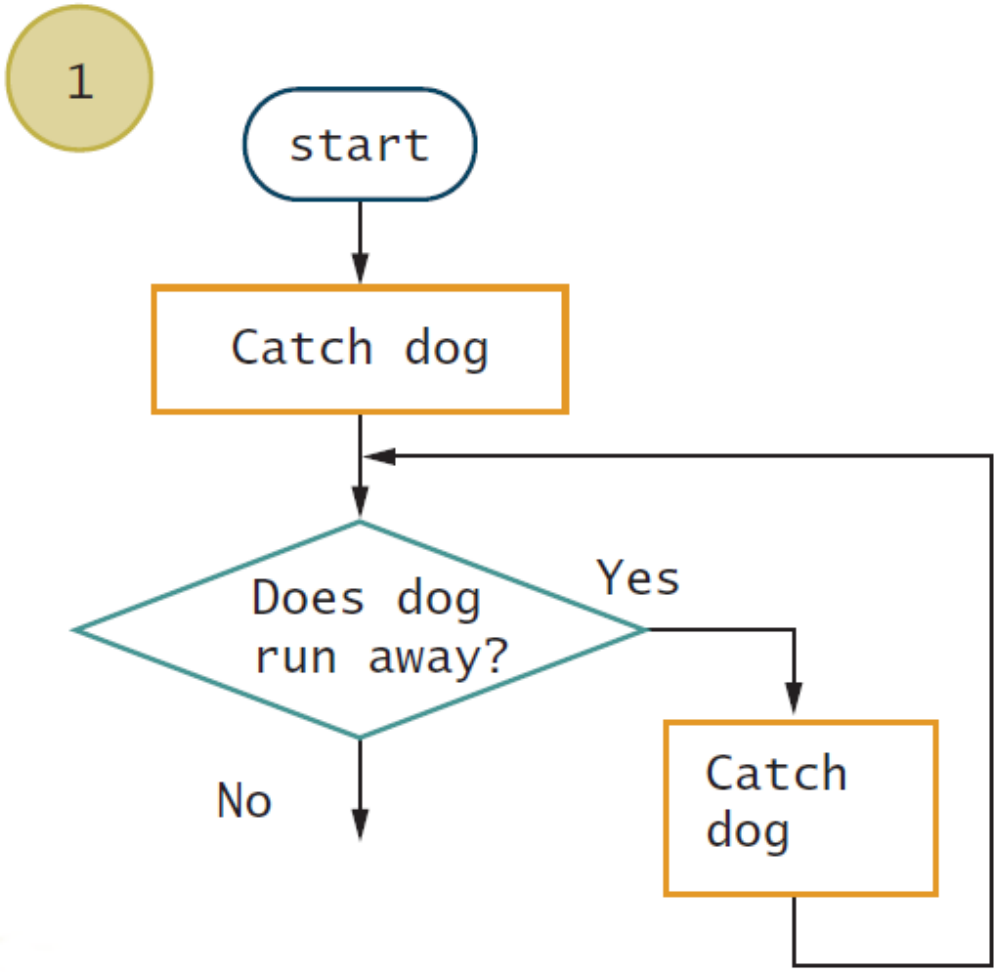
The “spaghetti bowl” Method

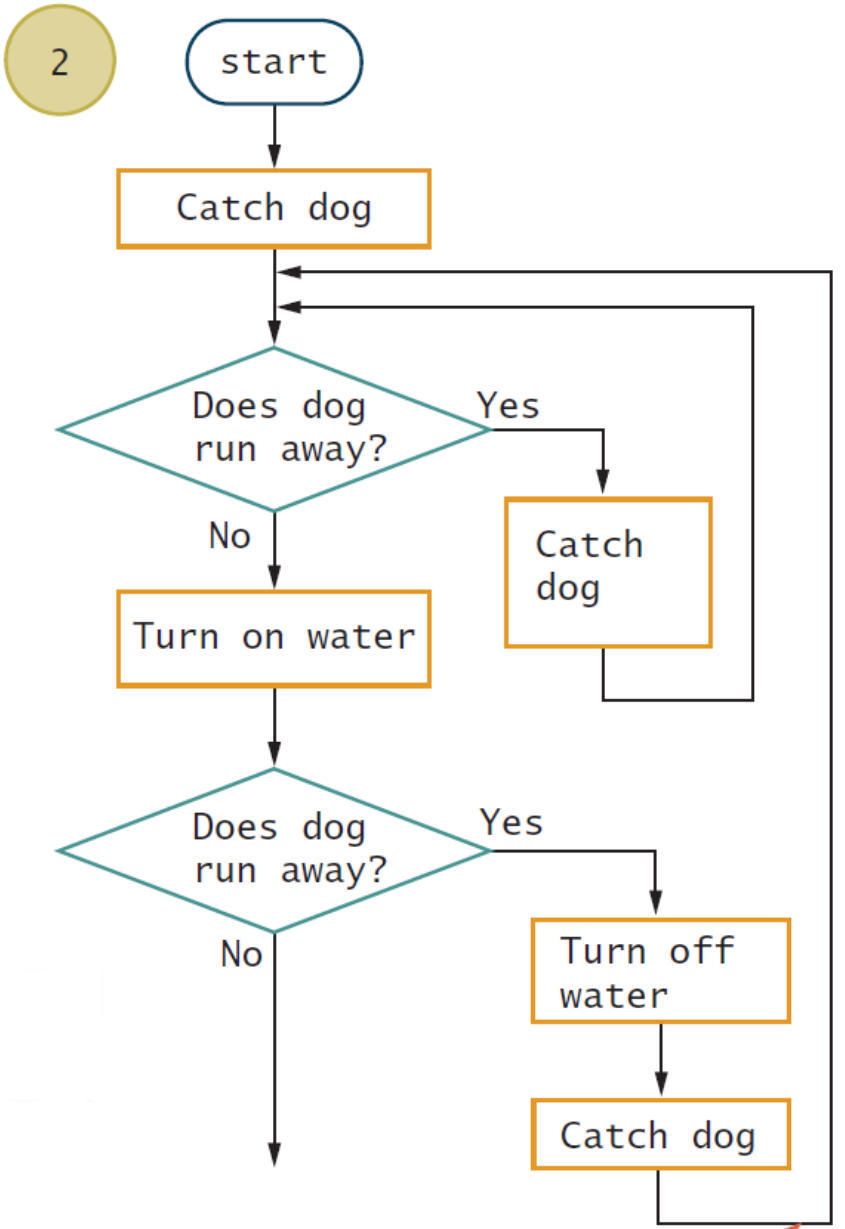




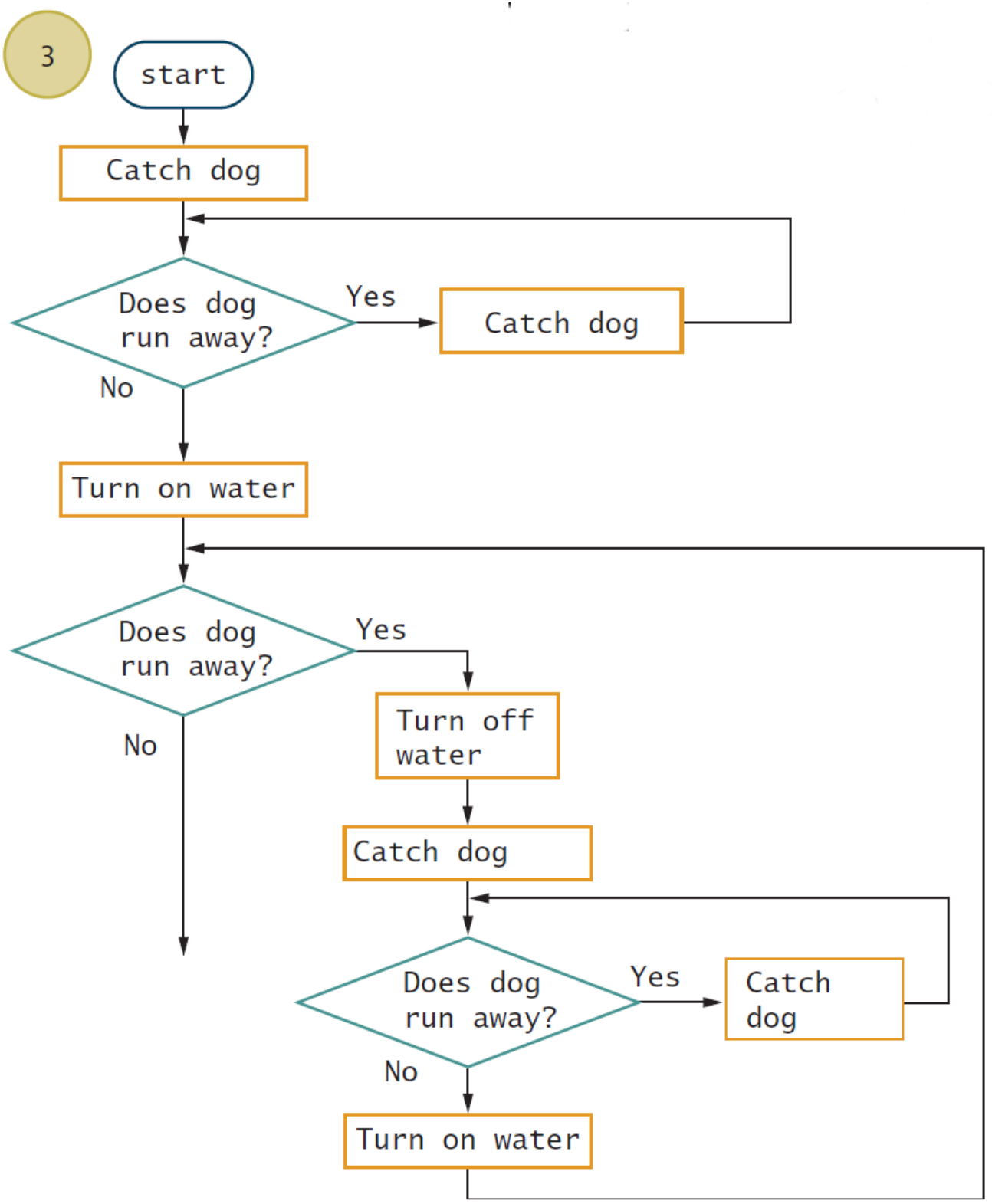
3.6 Structuring and Modularizing Unstructured Logic

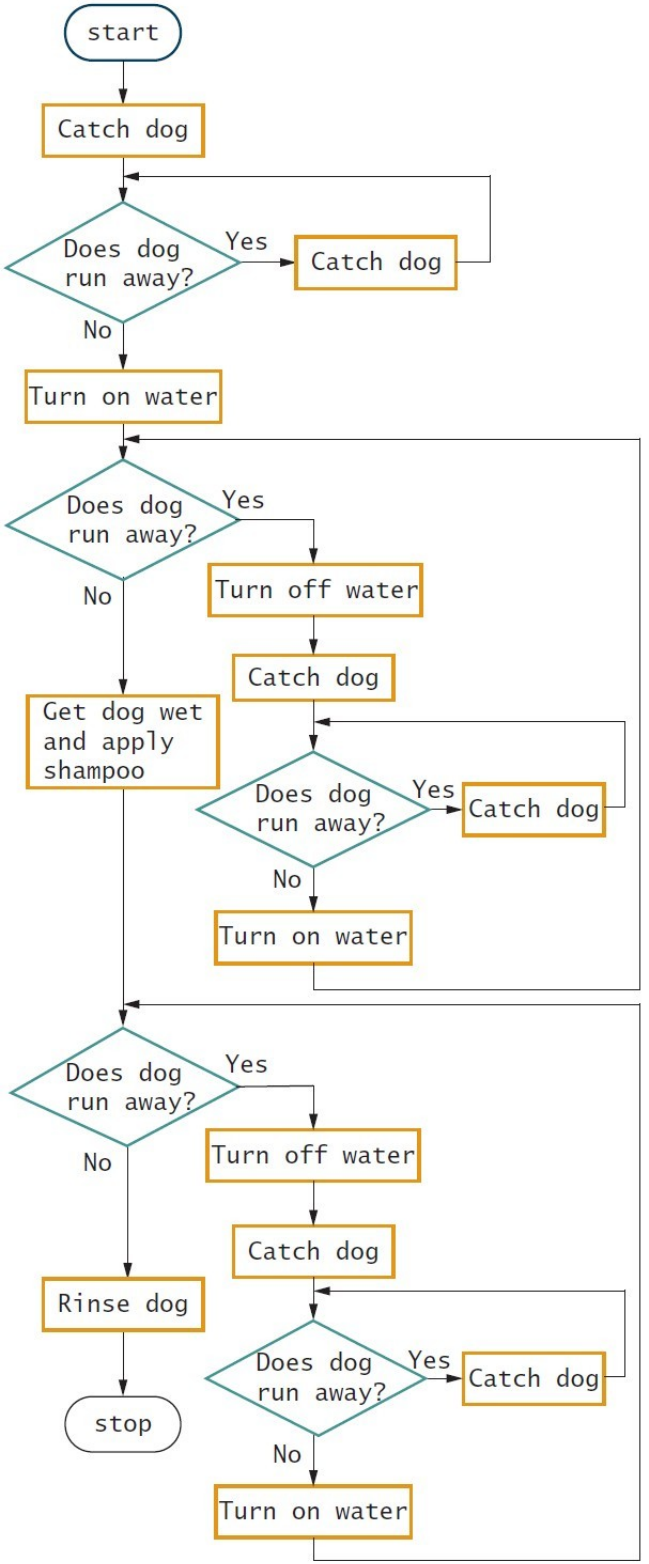




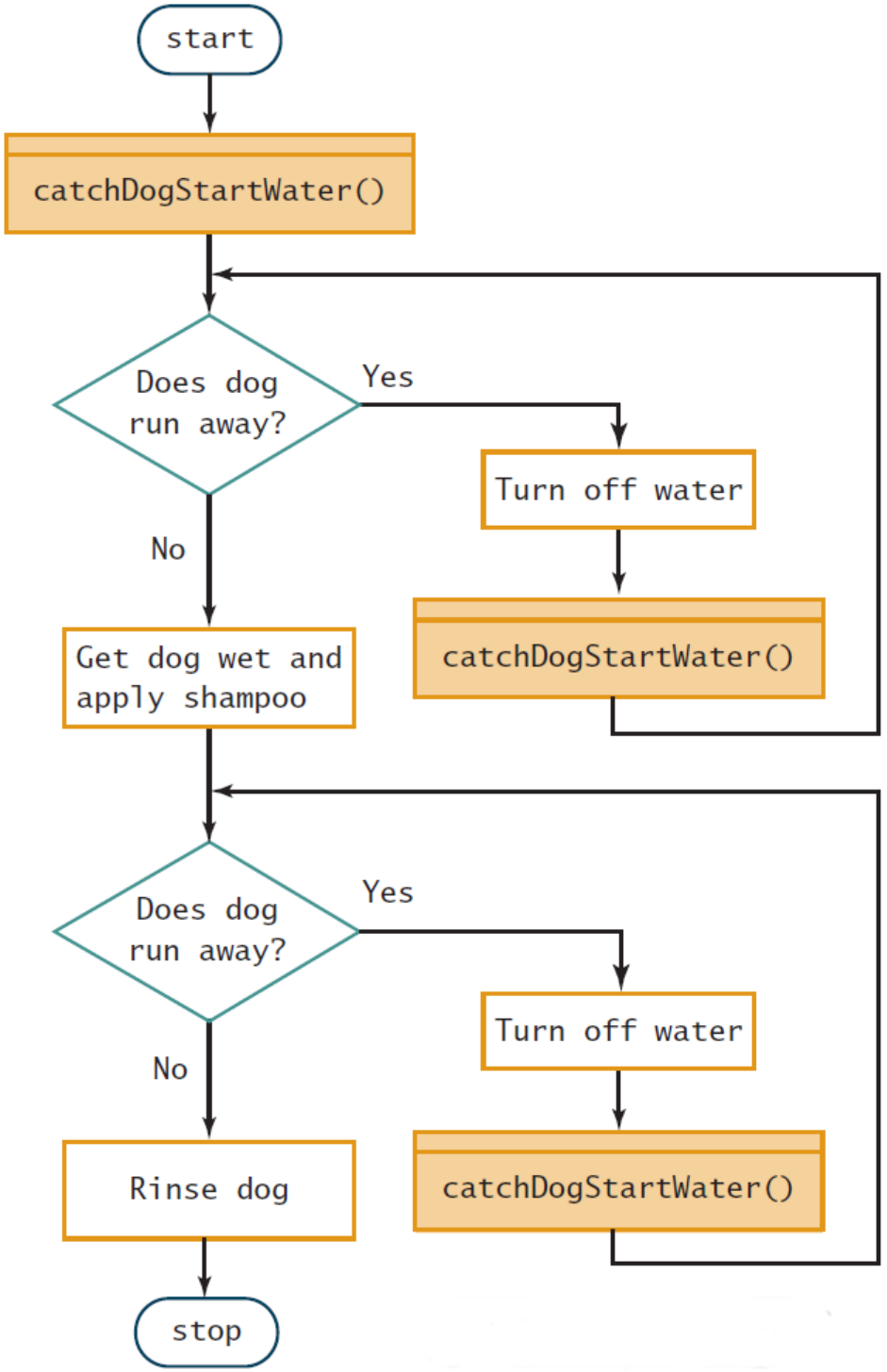


Don't Do It
This loop is not structured because its logic does not return to the question after its body executes.


3
start
Catch dog
Does dog run away?
Yes
Catch dog
No
Turn on water
Does dog run away?
Yes
Turn off water
No
Catch dog
Does dog run away?
Yes
Catch dog
No
Turn on water

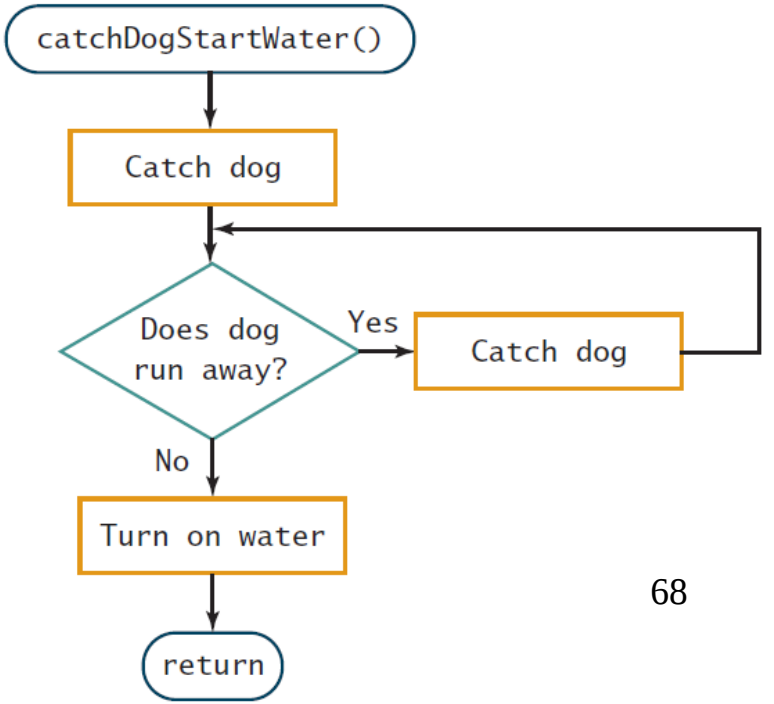



```
start
   Catch dog
   while dog runs away
      Catch dog
   endwhile
   Turn on water
   while dog runs away
      Turn off water
      Catch dog
      while dog runs away
         Catch dog
      endwhile
      Turn on water
   endwhile
   Get dog wet and apply shampoo
   while dog runs away
      Turn off water
      Catch dog
      while dog runs away
         Catch dog
      endwhile
      Turn on water
   endwhile
   Rinse dog
stop
```




```
start
   catchDogStartWater()
   while dog runs away
      Turn off water
      catchDogStartWater()
   endwhile
   Get dog wet and apply shampoo
   while dog runs away
      Turn off water
      catchDogStartWater()
   endwhile
   Rinse dog
stop

catchDogStartWater()
   Catch dog
   while dog runs away
      Catch dog
   endwhile
   Turn on water
return
```



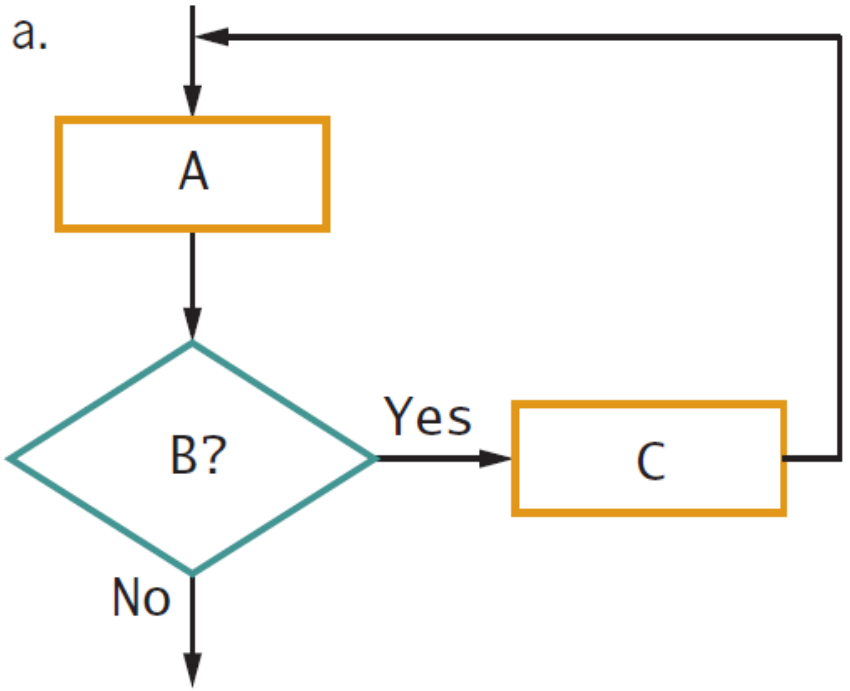
3.7 Summary

- Spaghetti code is the descriptive name for unstructured program statements that do not follow the rules of structured logic.
- Clearer programs can be constructed using only three basic structures: sequence, selection, and loop.
- A priming input is the statement that gets the first input value prior to starting a structured loop.
- Programmers use structured techniques to promote clarity, professionalism, efficiency, and modularity.
- One way to order an unstructured flowchart segment is to imagine it as a bowl of spaghetti that you must untangle.

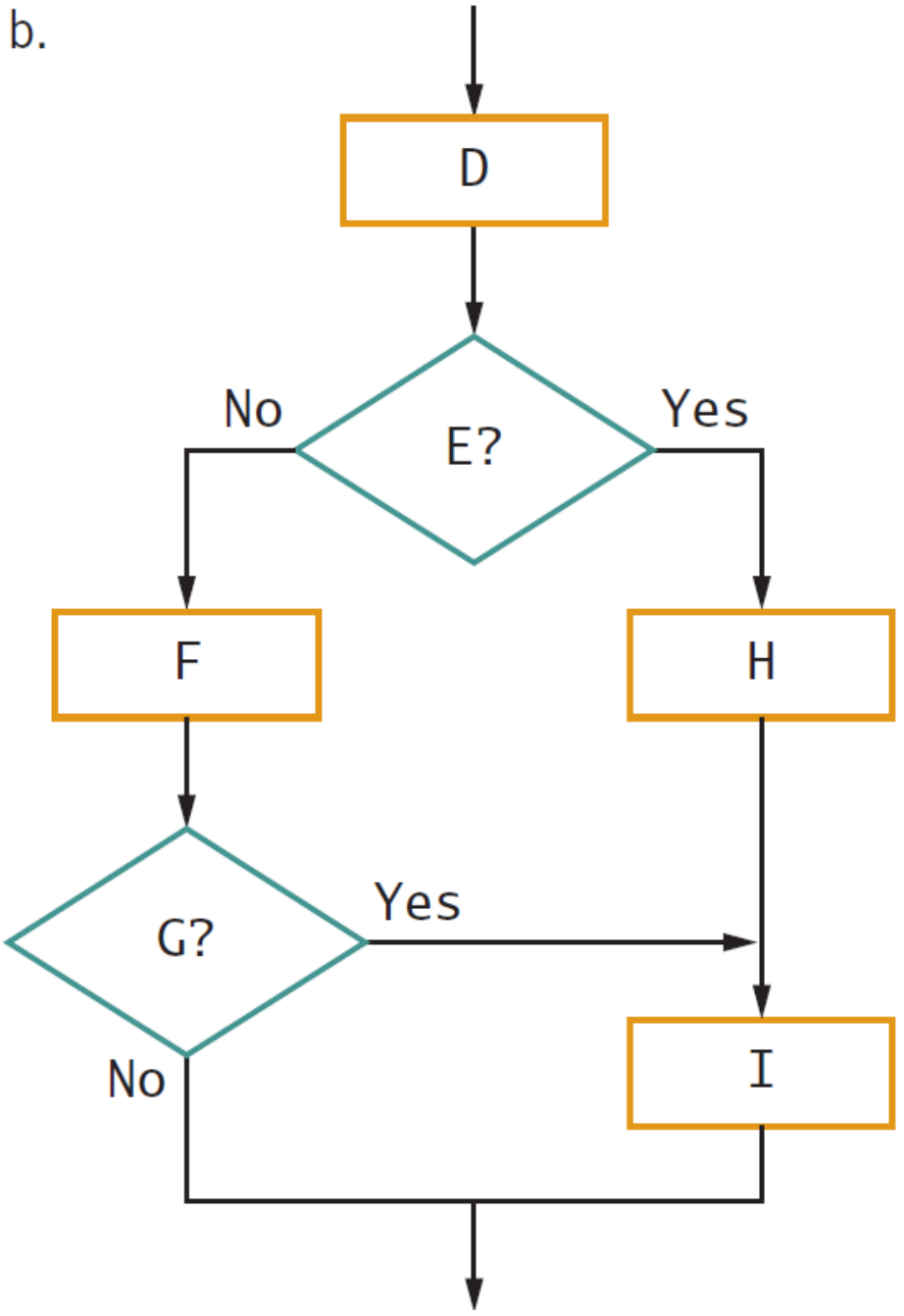
3.8 Exercises

- Redraw each segment so that it does the same thing but is structured.

a.



b.



c.