Chapter 3:
Modules, Hierarchy Charts, and Documentation

Programming Logic and Design, 4th Edition Introductory

Objectives

- **After studying Chapter 3, you should be able to:**
- **Describe the advantages of modularization**
- **Modularize a program**
- **Understand how a module can call another module**
- **Explain how to declare variables**
- **Create hierarchy charts**

Objectives (continued)

- Understand documentation
- Create print charts
- Interpret file descriptions
- Understand the attributes of complete documentation

Modules, Subroutines, Procedures, Functions, or Methods

- Programmers seldom write programs as one long series of steps
- Instead, they break the programming problem down into reasonable units, and tackle one small task at a time
- These reasonable units are called **modules**
- Programmers also refer to them as **subroutines, procedures, functions,** or **methods**

Modules, Subroutines, Procedures, Functions, or Methods (continued)

- **The process of breaking a large program into modules is called modularization**
  - **Provides abstraction**
  - **Allows multiple programmers to work on a problem**
  - **Allows you to reuse your work**
  - **Makes it easier to identify structures**

Modularization Provides Abstraction

- **Abstraction:**
  - **Process of paying attention to important properties while ignoring nonessential details (selective ignorance)**
  - **Makes complex tasks look simple**
  - **Some level occurs in every computer program**

Modularization Provides Abstraction

- **Fifty years ago, an understanding of low-level circuitry instructions was necessary**
- **Now, newer high-level programming languages allow you to use English-like vocabulary in which one broad statement corresponds to dozens of machine instructions**
- **Modules or subroutines provide another way to achieve abstraction**

Modularization Allows Multiple Programmers to Work on a Problem

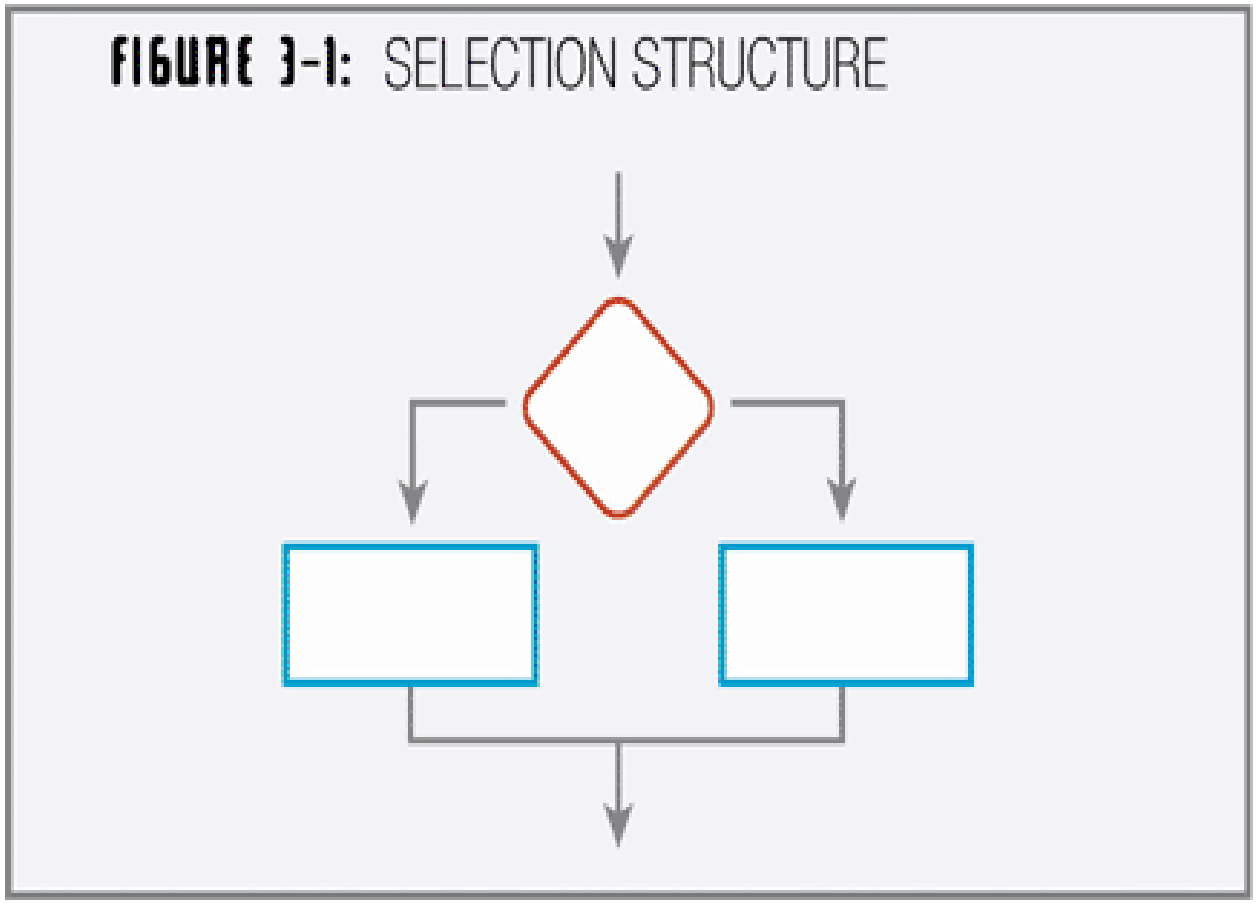
- When you dissect any large task into modules, you gain the ability to divide the task among various people
- Rarely does a single programmer write a commercial program that you buy off the shelf
- Modularization thus allows professional software developers to write new programs in weeks or months, instead of years

Modularization Allows You to Reuse Your Work

- **If a subroutine or function is useful and well-written, you may want to use it more than once within a program or in other programs**
- **You can find many real-world examples of reusability where systems with proven designs are incorporated, rather than newly invented, by individuals beginning a certain task**

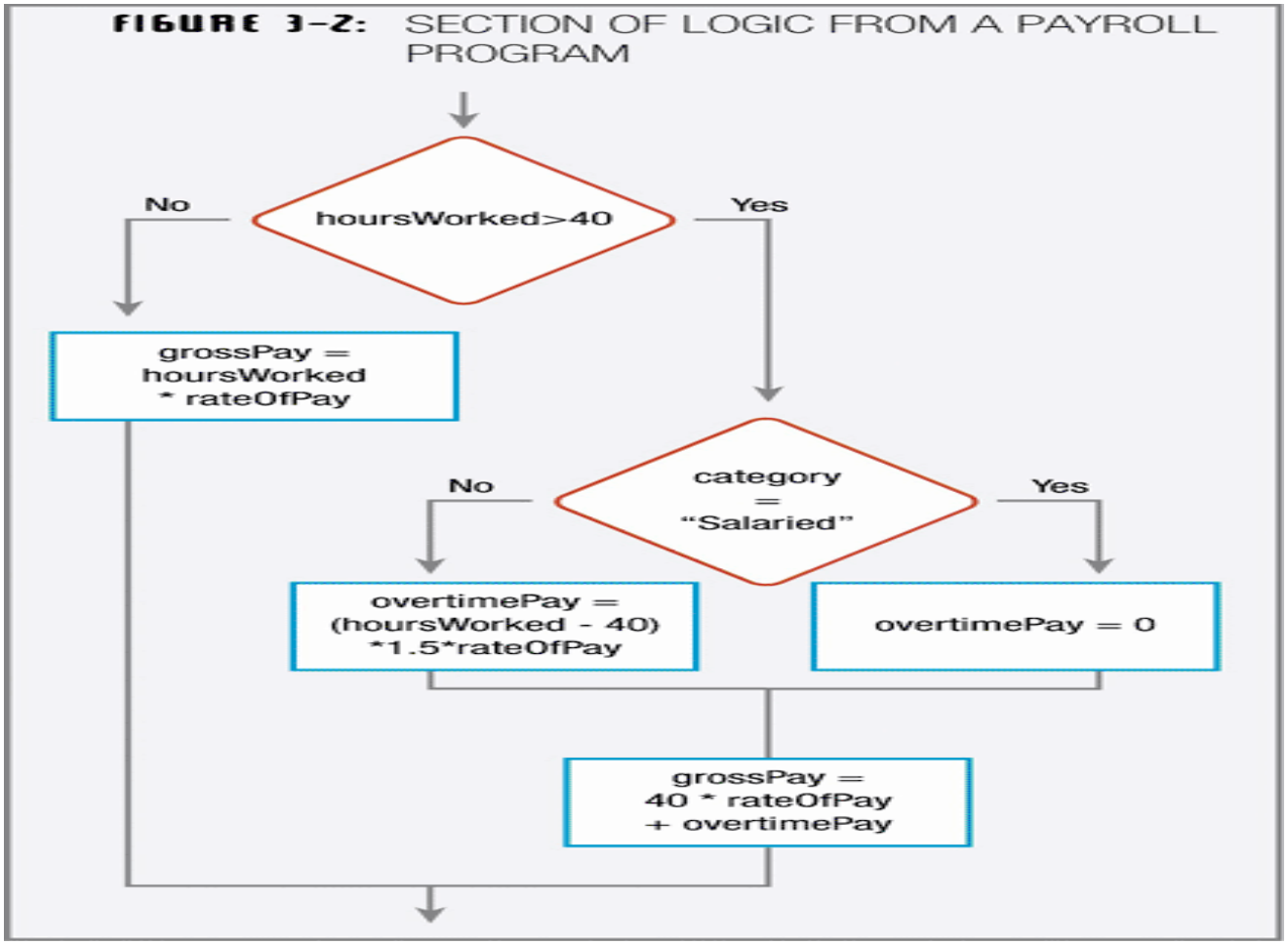
Modularization Makes It Easier to Identify Structures

- **When you combine several programming tasks into modules, it may be easier for you to identify structures**



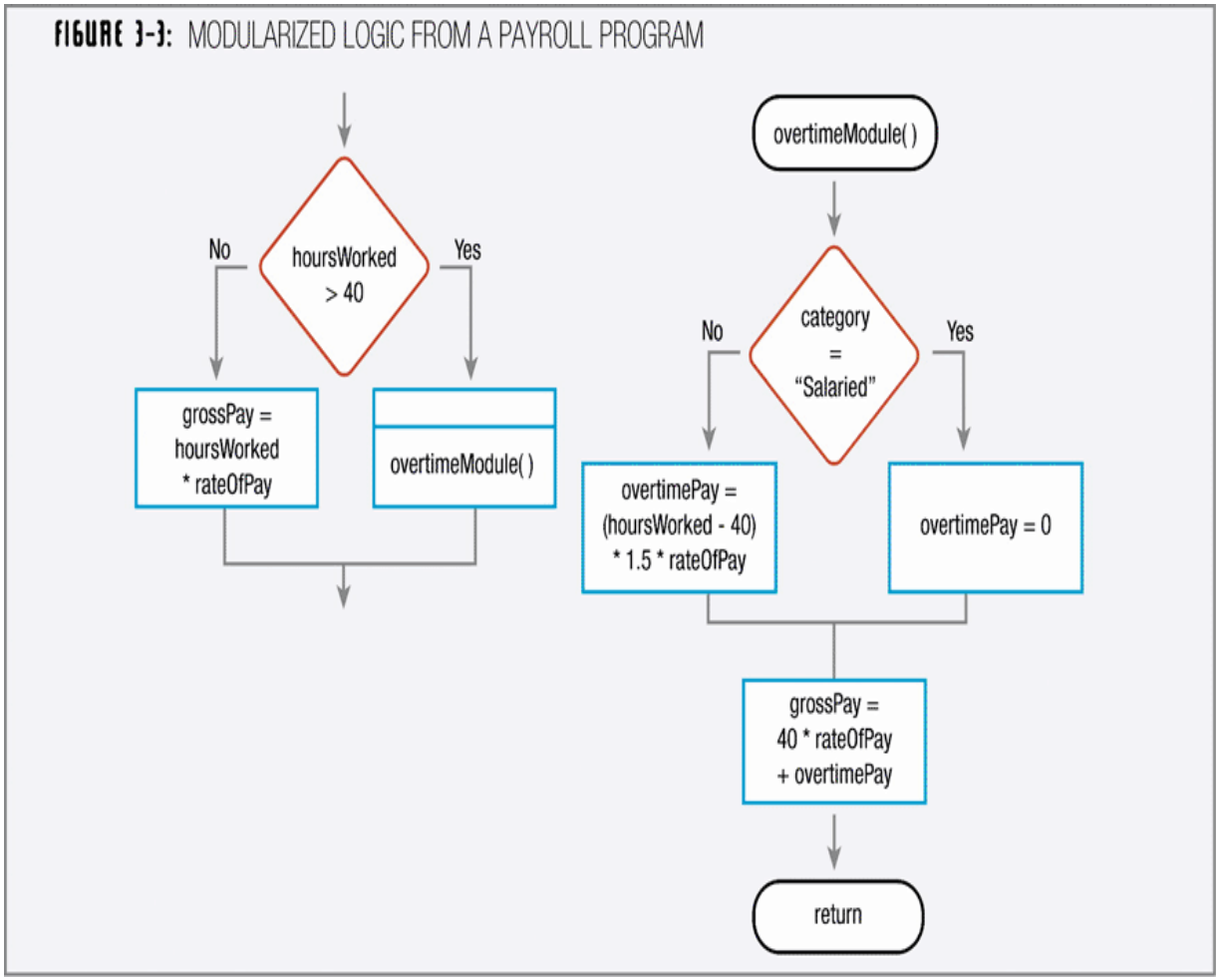
Modularization Makes It Easier to Identify Structures (continued)

- When you work with a program segment that looks like Figure 3-2, you may question whether it is structured



Modularization Makes It Easier to Identify Structures (continued)

- If you can modularize some of the statements and give them a more abstract group name, as in Figure 3-3, easier to see
  - that the program involves a major selection
  - that the program segment is structured



Modularizing a Program

- **When you create a module or subroutine, you give it a name**
- **In this text, module names follow the same two rules used for variable names:**
  - **Must be one word**
  - **Should have some meaning**

Modularizing a Program (continued)

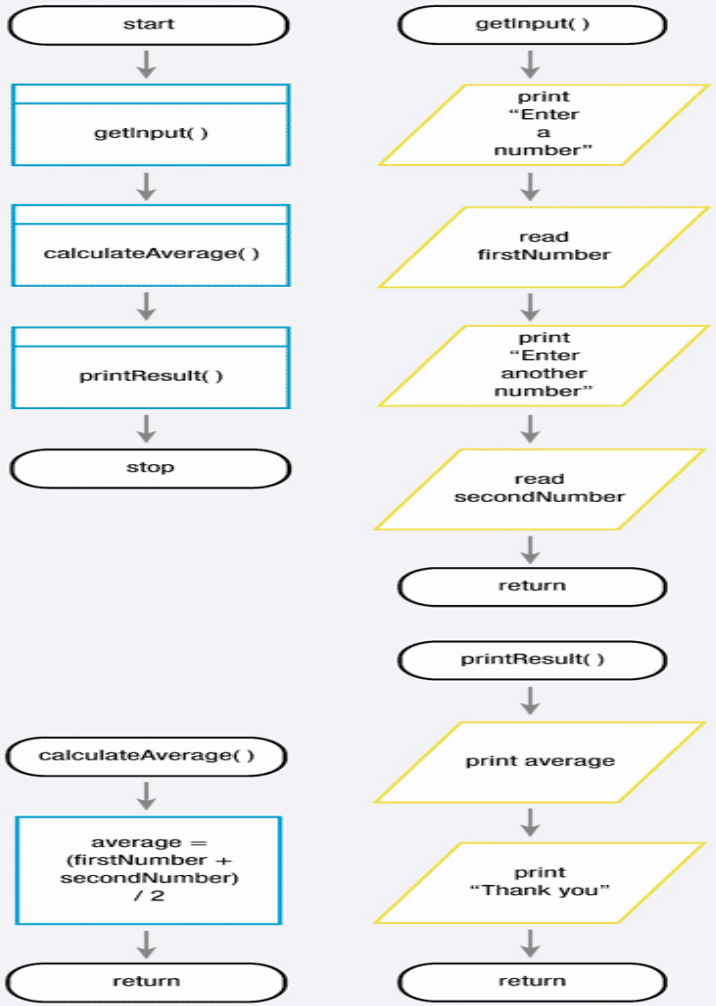
- When a program uses a module, you can refer to the main program as the calling program
- Whenever a main program calls a module, the logic transfers to the module
- When the module ends, the logical flow transfers back to the main calling program and resumes where it left off

Modularizing a Program (continued)

- **Draw each module separately with its own sentinel symbols**
- **The symbol that is equivalent of the start symbol in a program contains the nameOfModule**
  - **This name must be identical to the name used in the calling program.**
- **The symbol that is equivalent of the end symbol in a program contains return**

Modularizing a Program (continued)

FIGURE 3-4: FLOWCHART AND PSEUDOCODE FOR AVERAGING PROGRAM WITH MODULES




```
start
      perform getInput()
      perform calculateAverage()
      perform printResult()
stop

getInput()
      print "Enter a number"
      read firstNumber
      print "Enter another number"
      read secondNumber
return

calculateAverage()
      average = (firstNumber + secondNumber) / 2
return

printResult()
      print average
      print "Thank you"
return
```

Modules Calling Other Modules

- **Determining when to break down any particular module further into its own subroutines or submodules is an art**
- **Some companies may have arbitrary rules, such as:**
  - **“a subroutine should never take more than a page,” or**
  - **“a module should never have more than 30 statements in it,” or**
  - **“never have a method or function with only one statement in it”**

Modules Calling Other Modules (continued)

- A better policy is to place together statements that contribute to one specific task
- The more the statements contribute to the same job, the greater the functional cohesion of the module

Declaring Variables

- **The primary work of most modules in most programs you write is to manipulate data**
- **Many program languages require you to declare all variables before you use them**
- **Declaring a variable involves:**
  - **providing a name for the memory location where the computer will store the variable values, and**
  - **notifying the computer of what type of data to expect**

Declaring Variables (continued)

- Every programming language has specific rules for declaring variables, but all involve identifying at least two attributes for every variable:
  - Declaring a data type
  - Giving the variable a name
- In many modern programming languages, variables typically are declared within each module that uses them
  - Known as **local variables**

Declaring Variables (continued)

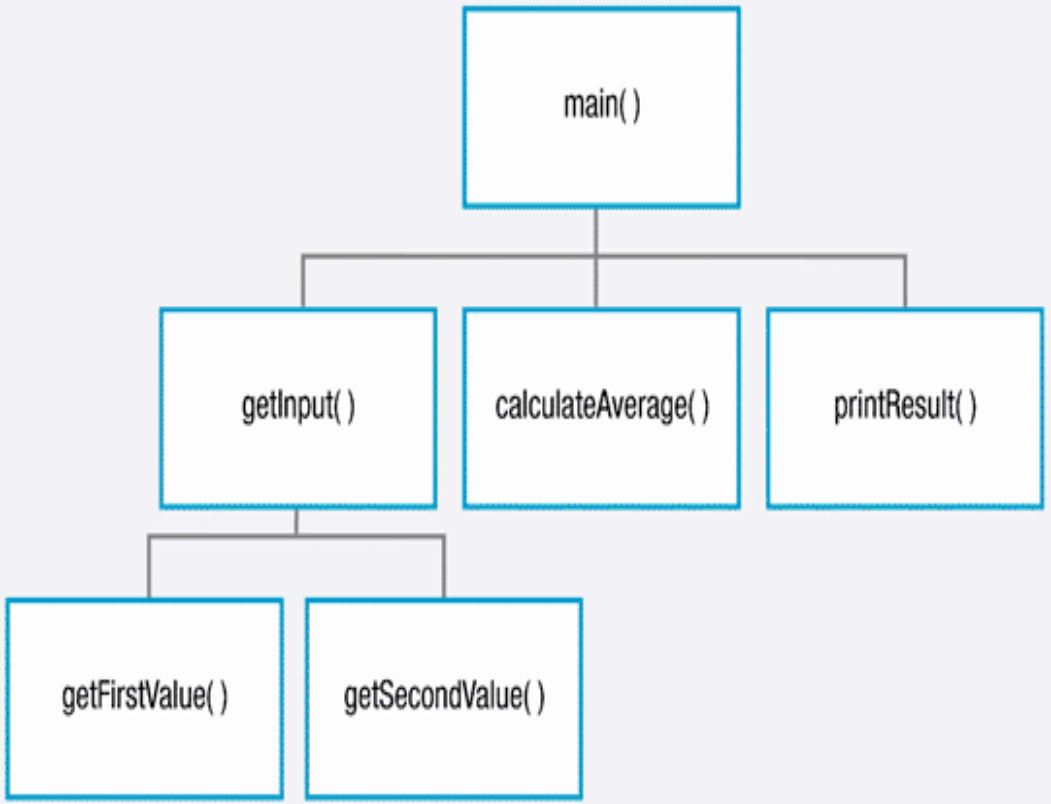
- **Global variables—variables given a type and name once, and then used in all modules of the program**
- **Annotation symbol or annotation box – an attached box containing notes**
  - **Use when you have more to write than can conveniently fit within a flowchart symbol**
- **Data dictionary — a list of every variable name used in a program, along with its type, size, and description**

Creating Hierarchy Charts

- **You can use a hierarchy chart to illustrate modules' relationships**
  - **Does not tell you what tasks are to be performed within a module**
  - **Does not tell you *when* or *how* a module executes**
  - **Rather, identifies which routines exist within a program and which routines call which other routines**
- **The hierarchy chart for the last version of the number-averaging program looks like Figure 3-7, and shows which modules call which others**

Creating Hierarchy Charts (continued)

FIGURE 3-1: HIERARCHY CHART FOR NUMBER-AVERAGING PROGRAM IN FIGURE 3-6



Understanding Documentation

- **Documentation refers to all supporting material that goes with a program**
- **Two broad categories:**
  - **Documentation intended for users**
  - **documentation intended for programmers**
- **People who use computer programs are called end users, or users for short**

Understanding Documentation (continued)

- **Programmers require instructions known as program documentation to plan the logic of or modify a computer program**
- **End users never see program documentation**
- **Divided into internal and external**

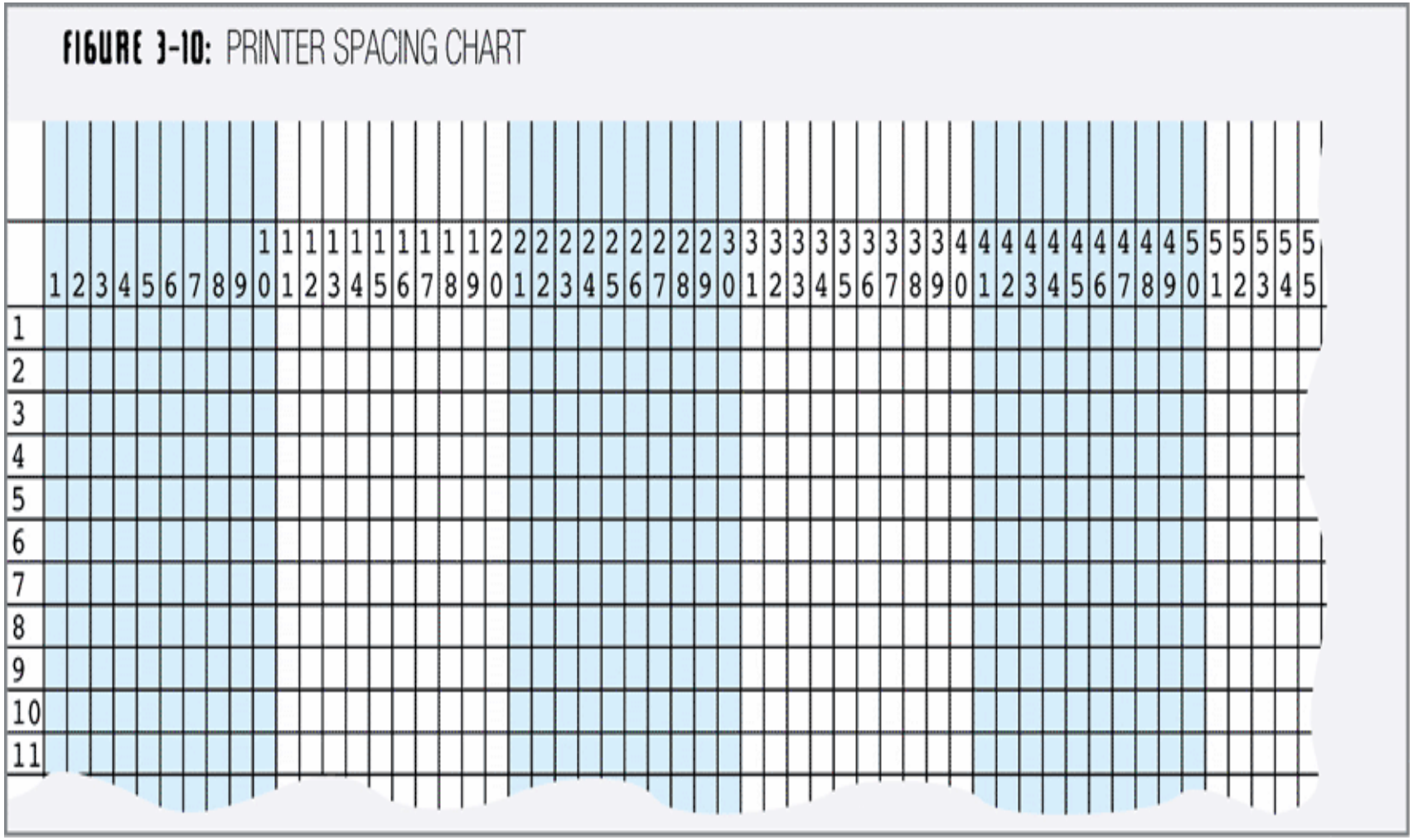
Understanding Documentation (continued)

- **Internal program documentation consists of program comments, or nonexecuting statements that programmers place within their code to explain program statements in English**
- **External program documentation includes all the supporting paperwork that programmers develop before they write a program**
- **Because most programs have input, processing, and output, usually there is documentation for all these functions**

Output Documentation

- Usually the first to be written
- A very common type of output is a printed report
- You can design a printed report on a printer spacing chart, which is also referred to as a print chart or a print layout
- Figure 3-10 shows a printer spacing chart, which basically looks like graph paper

Output Documentation (continued)



Output Documentation (continued)

- Not all program output takes the form of printed reports
- If your program's output will appear on a monitor screen, particularly if you are working in a GUI, or graphical user interface environment like Windows, your design issues will differ
- In a GUI program, the user sees a screen, and can typically make selections using a mouse or other pointing device

Output Documentation (continued)

- **Instead of a print chart, your output design might resemble a sketch of a screen**
- **Figure 3-21 shows how inventory records might be displayed in a graphical environment**

FIGURE 3-21: INVENTORY RECORDS DISPLAYED IN A GUI ENVIRONMENT

Input Documentation

- **Once you have planned the design of the output, you need to know what input is available to produce this output**
- **If you are producing a report from stored data, you frequently will be provided with a file description that describes the data contained in a file**
- **You usually find a file's description as part of an organization's information systems documentation**

Input Documentation (continued)

FIGURE 3-22: INVENTORY FILE DESCRIPTION

INVENTORY FILE DESCRIPTION

File name: INVTRY

FIELD DESCRIPTION	POSITIONS	DATA TYPE	DECIMALS
Name of item	1–15	Character	
Price of item	16–20	Numeric	2
Quantity in stock	21–24	Numeric	0

Input Documentation (continued)

- A byte is a unit of computer storage that can contain any of 256 combinations of 0s and 1s that often represent a character
- The input description in Figure 3-22 shows that two of the positions in the price are reserved for decimal places
- Typically, decimal points themselves are not stored in data files; they are implied, or assumed
- Also, typically, numeric data are stored with leading zeroes so that all allotted positions are occupied

Input Documentation (continued)

- Typically, programmers create one program variable for each field that is part of the input file
- In addition to the field descriptions contained in the input documentation, the programmer might be given specific variable names to use for each field, particularly if such variable names must agree with the ones that other programmers working on the project are using
- In many cases, however, programmers are allowed to choose their own variable names

Input Documentation (continued)

- **Organizations may use different forms to relay the information about records and fields, but the very least the programmer needs to know is:**
  - **What is the name of the file?**
  - **What data does it contain?**
  - **How much room do the file and each of its fields take up?**
  - **What type of data can be stored in each field—character or numeric?**

Completing the Documentation

- **User documentation includes**
  - **all manuals or other instructional materials that non-technical people use, as well as**
  - **operating instructions that computer operators and data-entry personnel need**
- **Needs to be written clearly, in plain language, with reasonable expectations of the users' expertise**

Completing the Documentation (continued)

- **User documentation may address:**
  - **How to prepare input for the program**
  - **To whom the output should be distributed**
  - **How to interpret the normal output**
  - **How to interpret and react to any error message generated by the program**
  - **How frequently the program needs to run**

Summary

- **Programmers break programming problems down into smaller, reasonable units called modules, subroutines, procedures, functions, or methods**
- **When you create a module or subroutine, you give the module a name that a calling program uses when the module is about to execute**
- **A module can call other modules**

Summary

- **Declaring a variable involves providing a name for the memory location where the computer will store the variable value, and notifying the computer of what type of data to expect**
- **Documentation refers to all of the supporting material that goes with a program**
- **A file description lists the data contained in a file, including a description, size, and data type**