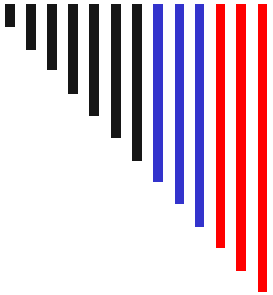

Chapter 1
Introduction to Computers

Objectives

- To review basic computer systems concepts
- To be able to understand the different computing environments and their components
- To review the history of computer languages
- To be able to list and describe the classifications of computer languages
- To understand the steps in the development of a computer program
- To review the system development life cycle

1-1 Computer Systems

Today computer systems are found everywhere. Computers have become almost as common as televisions. But what is a computer? A computer is a system made of two major components: hardware and software.

Topics discussed in this section:

Computer Hardware

Computer Software

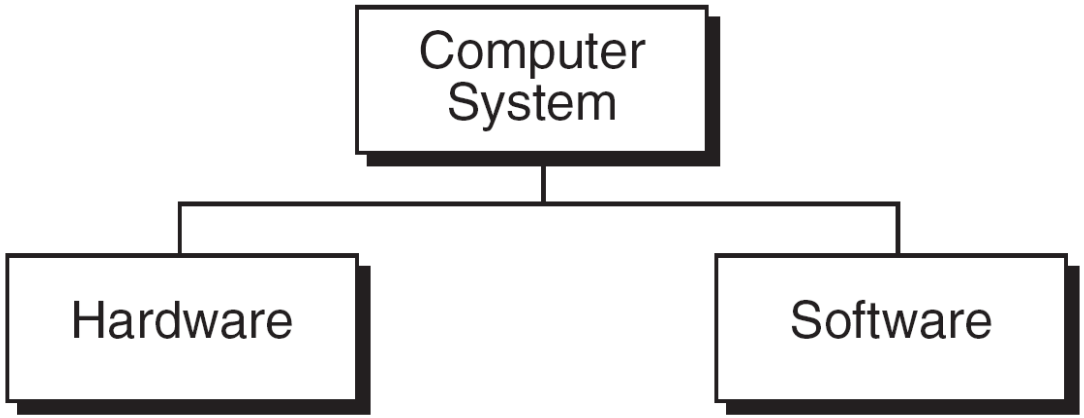

FIGURE 1-1 A Computer System

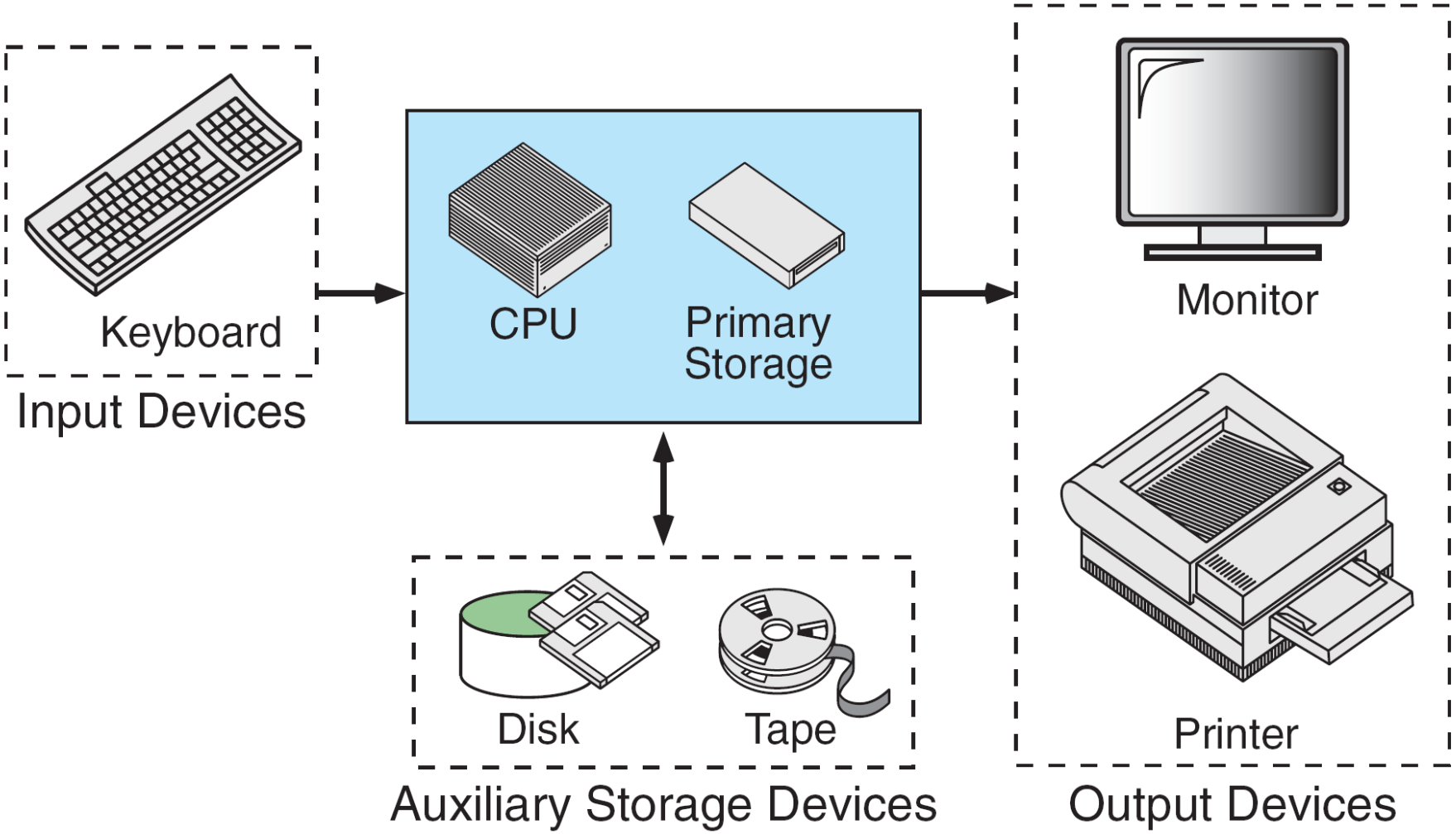

FIGURE 1-2 Basic Hardware Components

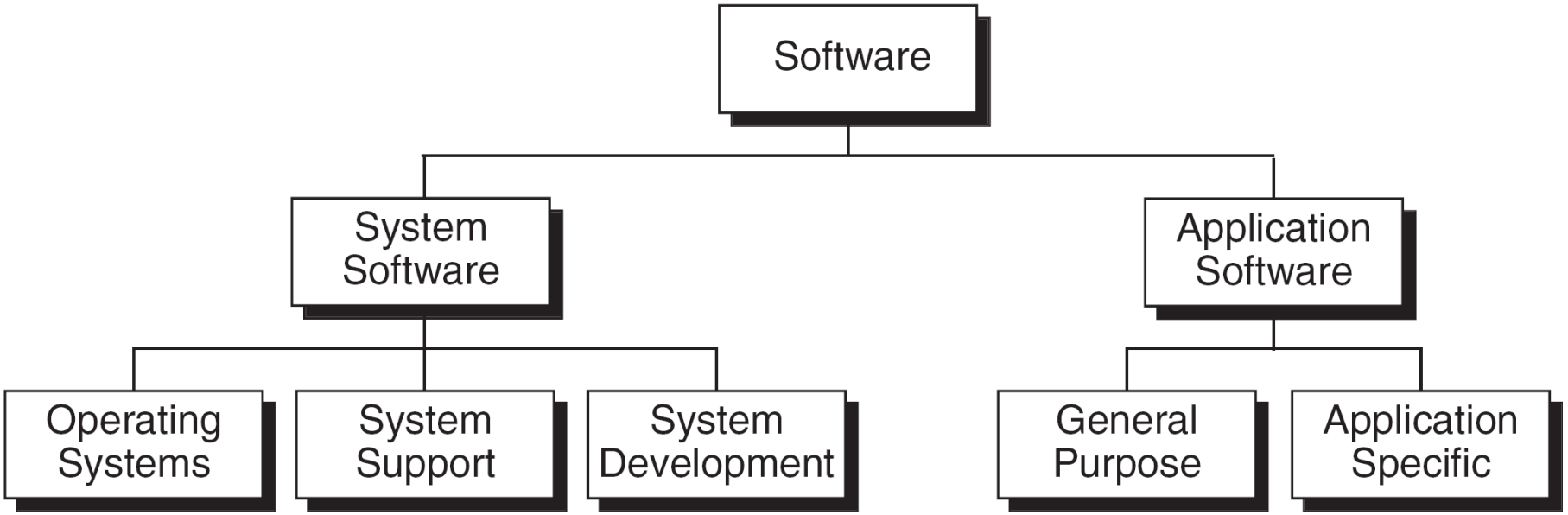

FIGURE 1-3 Types of Software

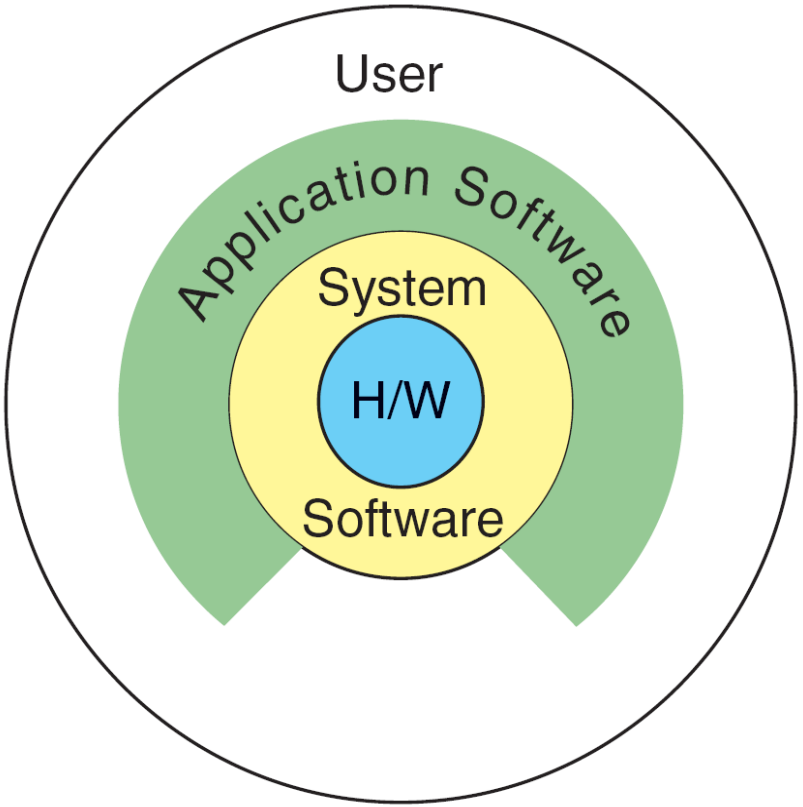

FIGURE 1-4 Relationship between system and application software

1-2 Computing Environments

In the early days of computers, there was only one environment, the mainframe computer hidden in a central computing department. With the advent of minicomputers and personal computers, the environment changed with computers on virtually every desktop. In this section we describe several different environments.

Topics discussed in this section:

Personal Computing Environment
Time-Sharing Environment
Client/Server Environment
Distributed Computing

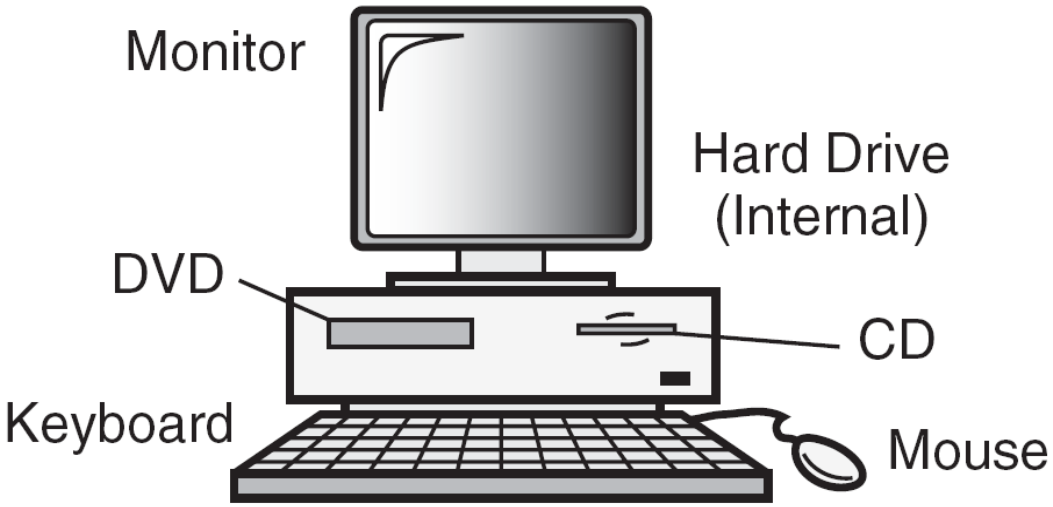

FIGURE 1-5 Personal Computing Environment

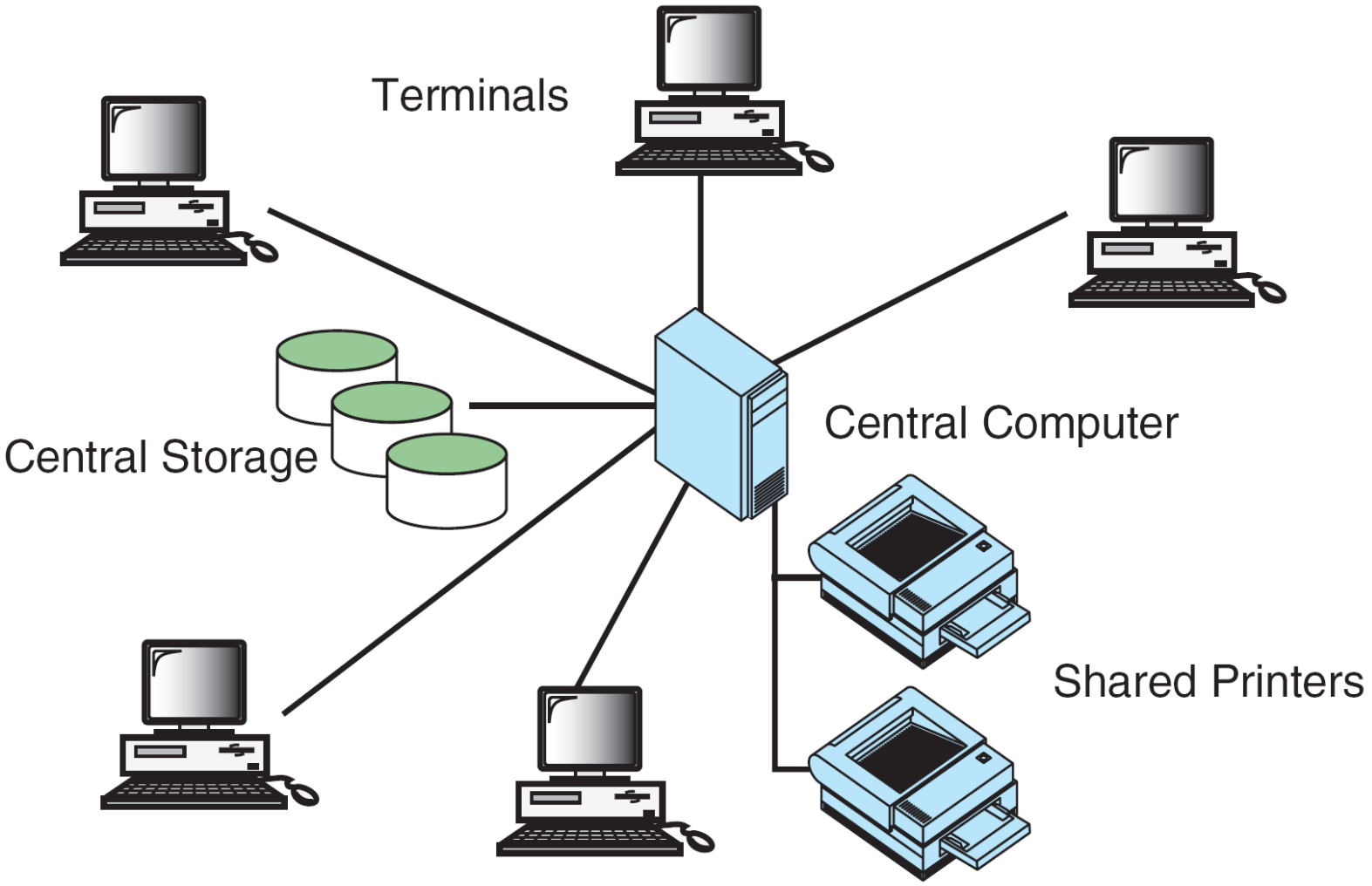

FIGURE 1-6 Time-sharing Environment

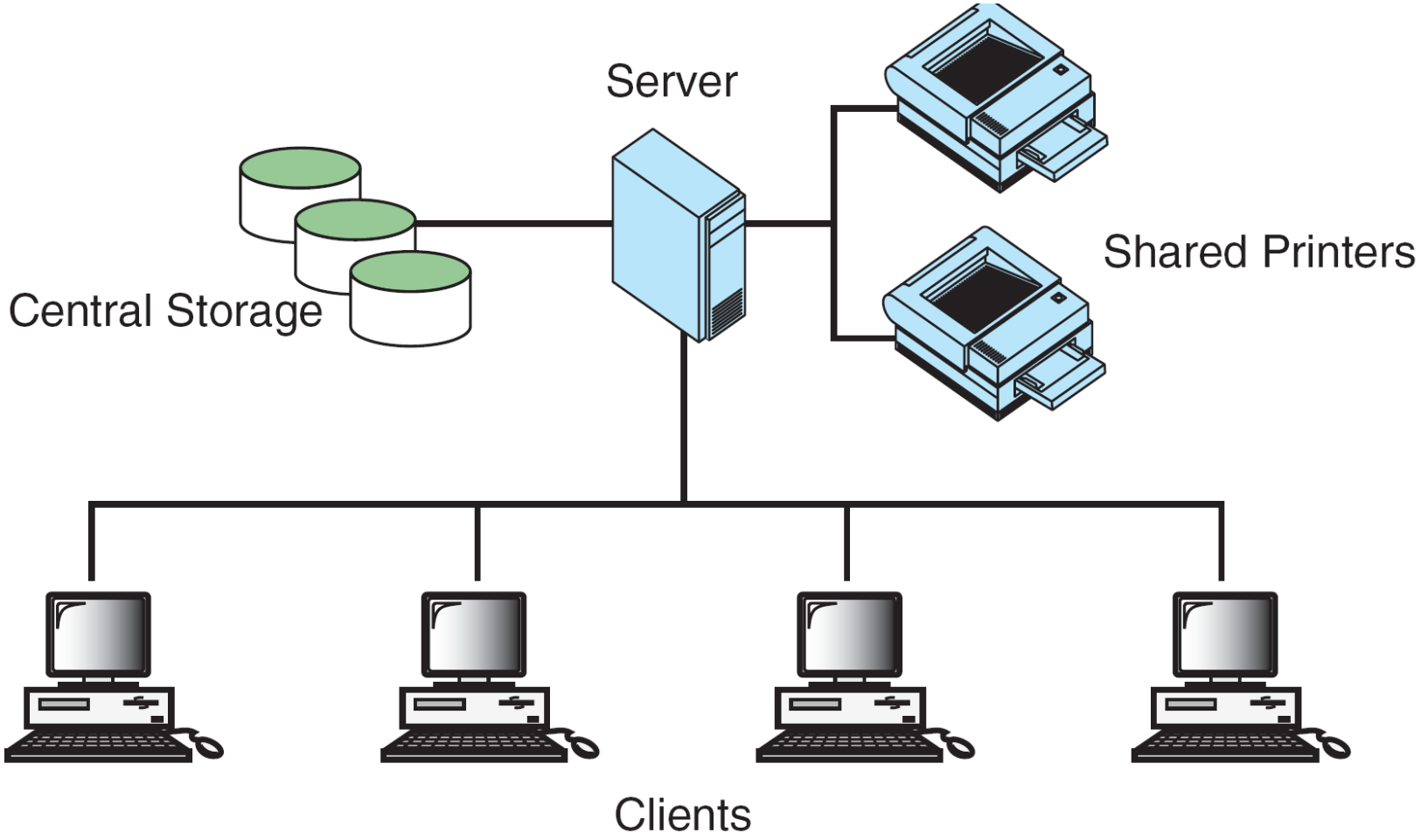

FIGURE 1-7 The Client/Server Environment

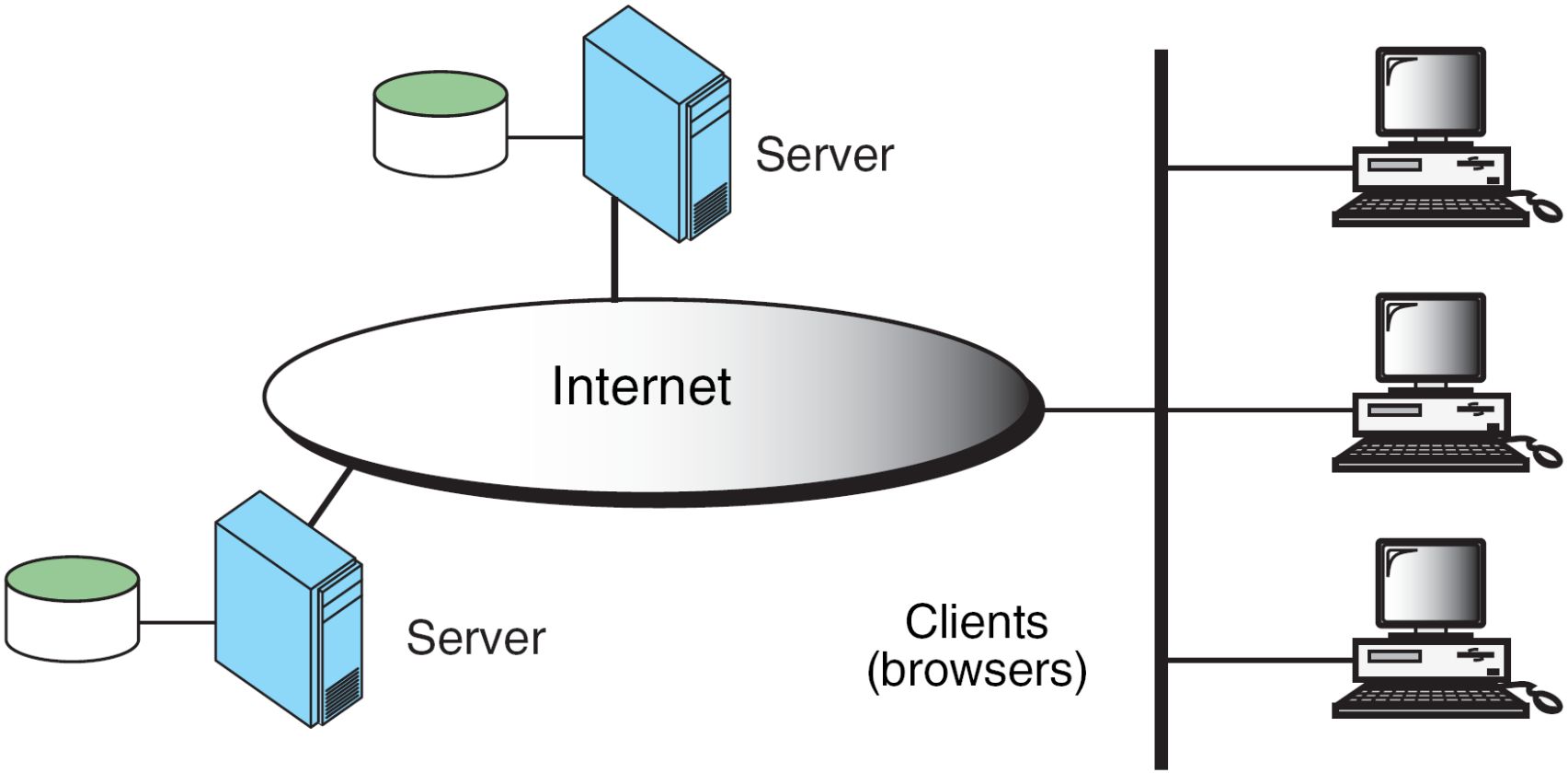

FIGURE 1-8 Distributed Computing

1-3 Computer Languages

To write a program for a computer, we must use a computer language. Over the years computer languages have evolved from machine language to natural languages.

Topics discussed in this section:

Machine Languages
Symbolic Languages
High-Level Languages

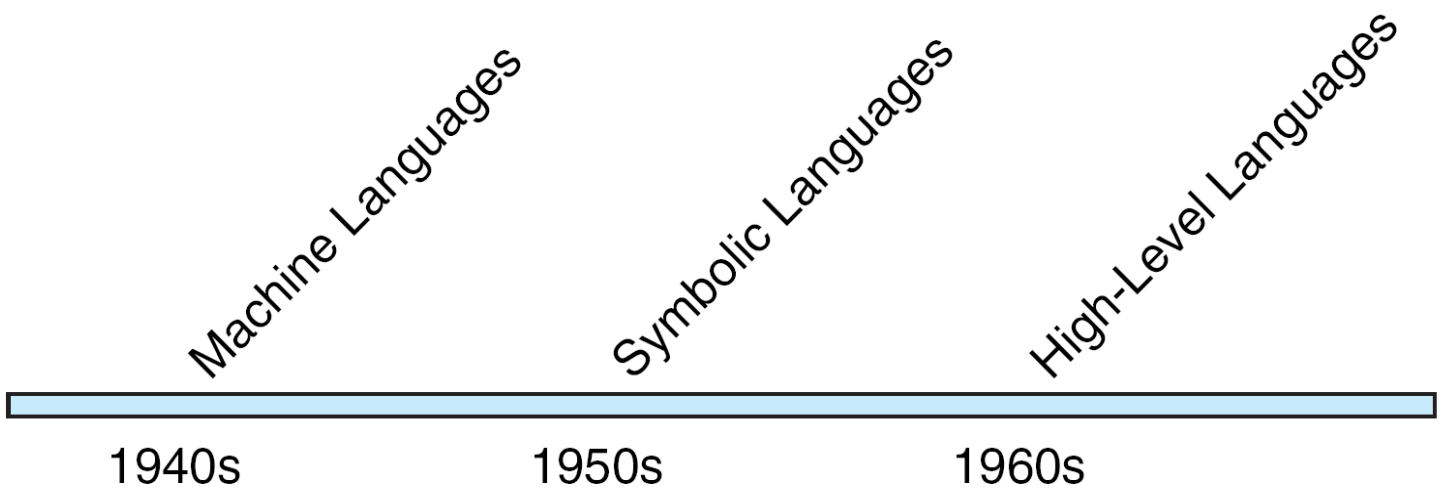

FIGURE 1-9 Computer Language Evolution

PROGRAM 1-1 The Multiplication Program in Machine Language

```
 1            00000000 00000100 0000000000000000
 2  01011110  00001100 11000010 0000000000000010
 3            11101111 00010110 0000000000000101
 4            11101111 10011110 0000000000001011
 5  11111000  10101101 11011111 0000000000010010
 6            01100010 11011111 0000000000010101
 7  11101111  00000010 11111011 0000000000010111
 8  11110100  10101101 11011111 0000000000011110
 9  00000011  10100010 11011111 0000000000100001
10  11101111  00000010 11111011 0000000000100100
11  01111110  11110100 10101101
12  11111000  10101110 11000101 0000000000101011
13  00000110  10100010 11111011 0000000000110001
14  11101111  00000010 11111011 0000000000110100
15            01010000 11010100 0000000000111011
16                     00000100 0000000000111101
```

Note

The only language understood by computer hardware is machine language.

PROGRAM 1-2 The Multiplication Program in Symbolic Language

```
 1        entry   main,^m<r2>
 2        subl2   #12,sp
 3        jsb     C$MAIN_ARGS
 4        movab   $CHAR_STRING_CON
 5
 6        pushal  -8(fp)
 7        pushal  (r2)
 8        calls   #2,SCANF
 9        pushal  -12(fp)
10        pushal  3(r2)
11        calls   #2,SCANF
12        mull3   -8(fp),-12(fp),-
13        pusha   6(r2)
14        calls   #2,PRINTF
15        clrl    r0
16        ret
```

Note

Symbolic language uses symbols, or mnemonics, to represent the various machine language instructions.

PROGRAM 1-3 The Multiplication Program in C

```
 1  /* This program reads two integers from the keyboard
 2     and prints their product.
 3        Written by:
 4        Date:
 5  */
 6  #include <stdio.h>
 7
 8  int main (void)
 9  {
10  // Local Definitions
11     int number1;
12     int number2;
13     int result;
14
15  // Statements
16     scanf  ("%d", &number1);
```

continued

PROGRAM 1-3 The Multiplication Program in C (*continued*)

```
17      scanf  ("%d", &number2);
18      result = number1 * number2;
19      printf ("%d", result);
20      return 0;
21  }   // main
```

1-4 Creating and Running Programs

In this section, we explain the procedure for turning a program written in C into machine language. The process is presented in a straightforward, linear fashion, but you should recognize that these steps are repeated many times during development to correct errors and make improvements to the code.

Topics discussed in this section:

Writing and Editing Programs
Compiling Programs
Linking Programs
Executing Programs

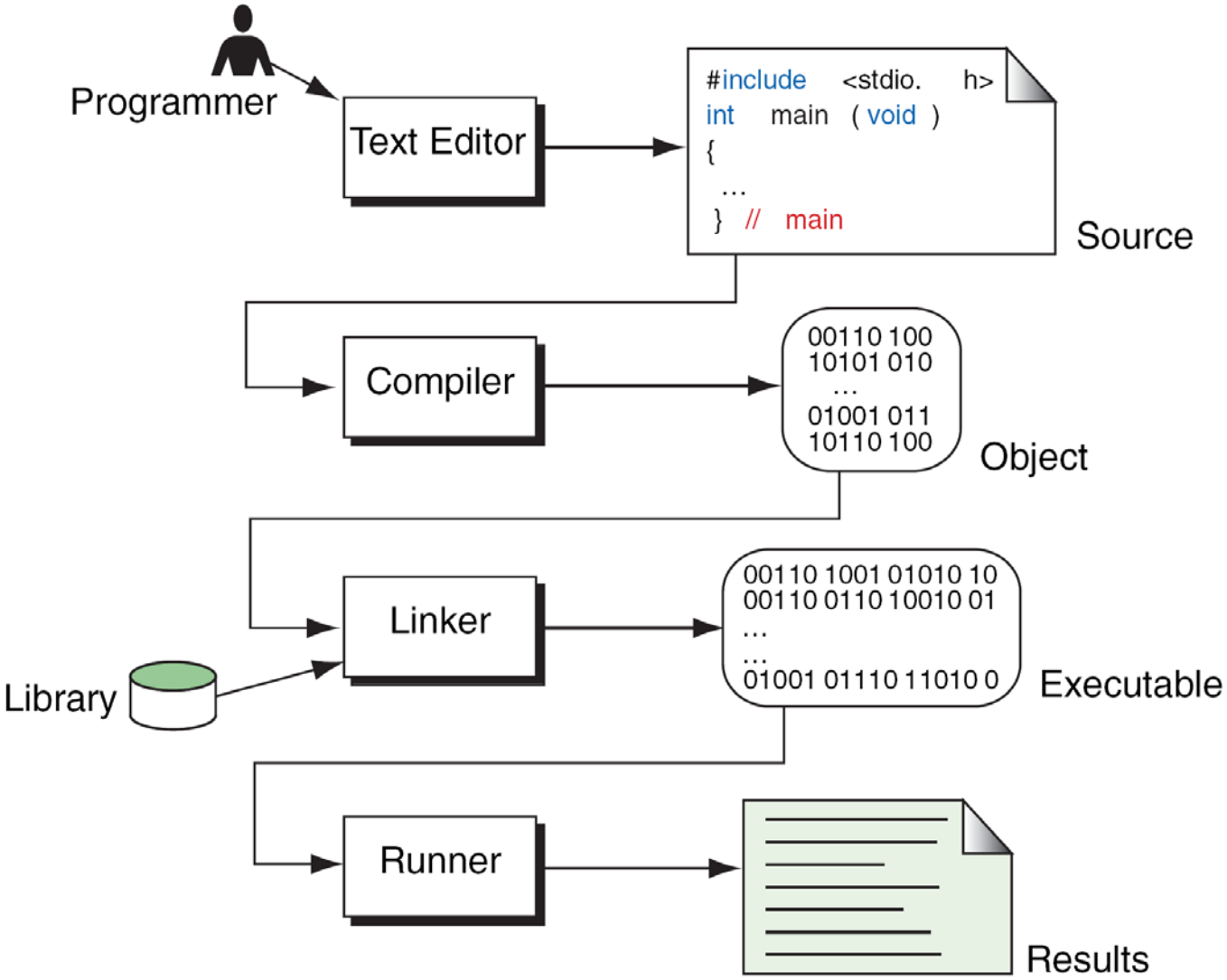

FIGURE 1-10 Building a C Program

1-5 System Development

We've now seen the steps that are necessary to build a program. In this section, we discuss how we go about developing a program. This critical process determines the overall quality and success of our program. If we carefully design each program using good structured development techniques, our programs will be efficient, error-free, and easy to maintain.

Topics discussed in this section:

System Development Life Cycle
Program Development

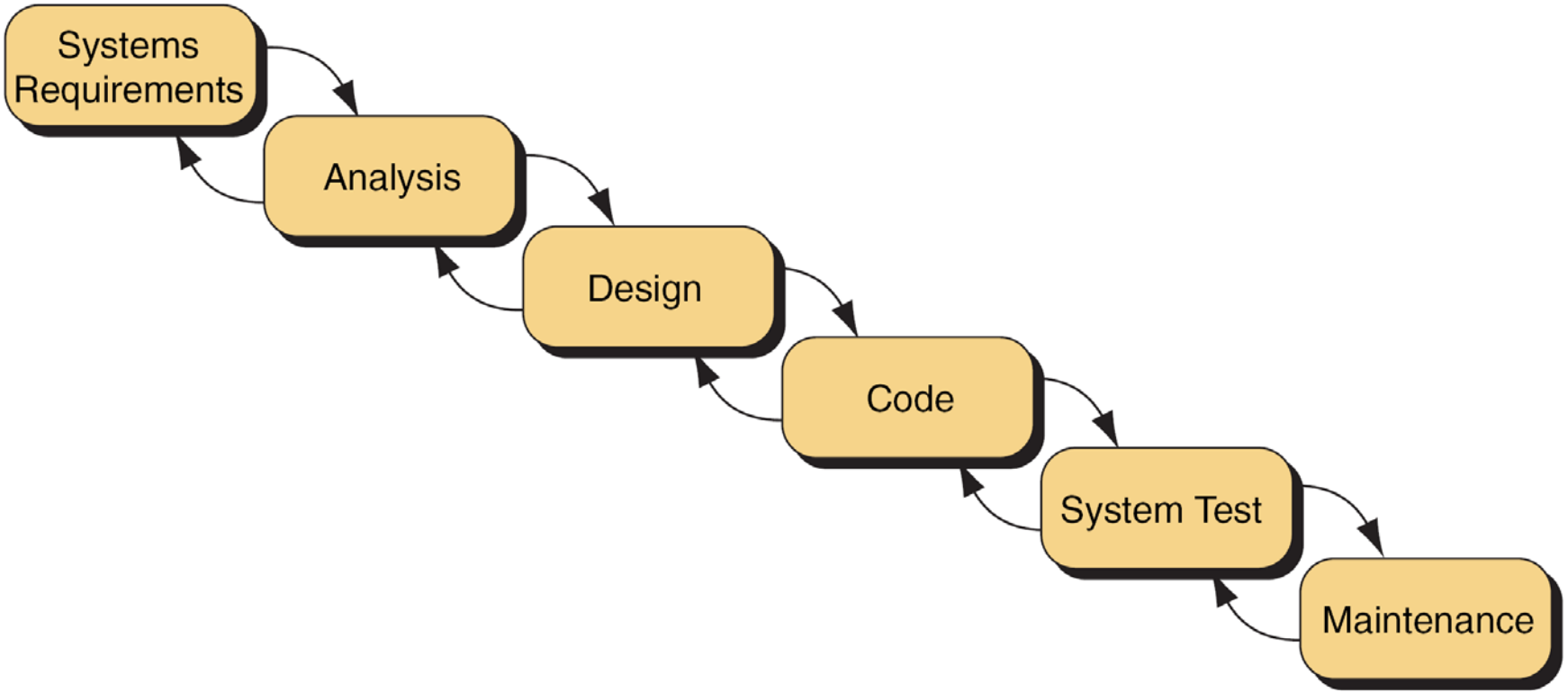

FIGURE 1-11 **Waterfall Model**

Note

An old programming proverb:
Resist the temptation to code.

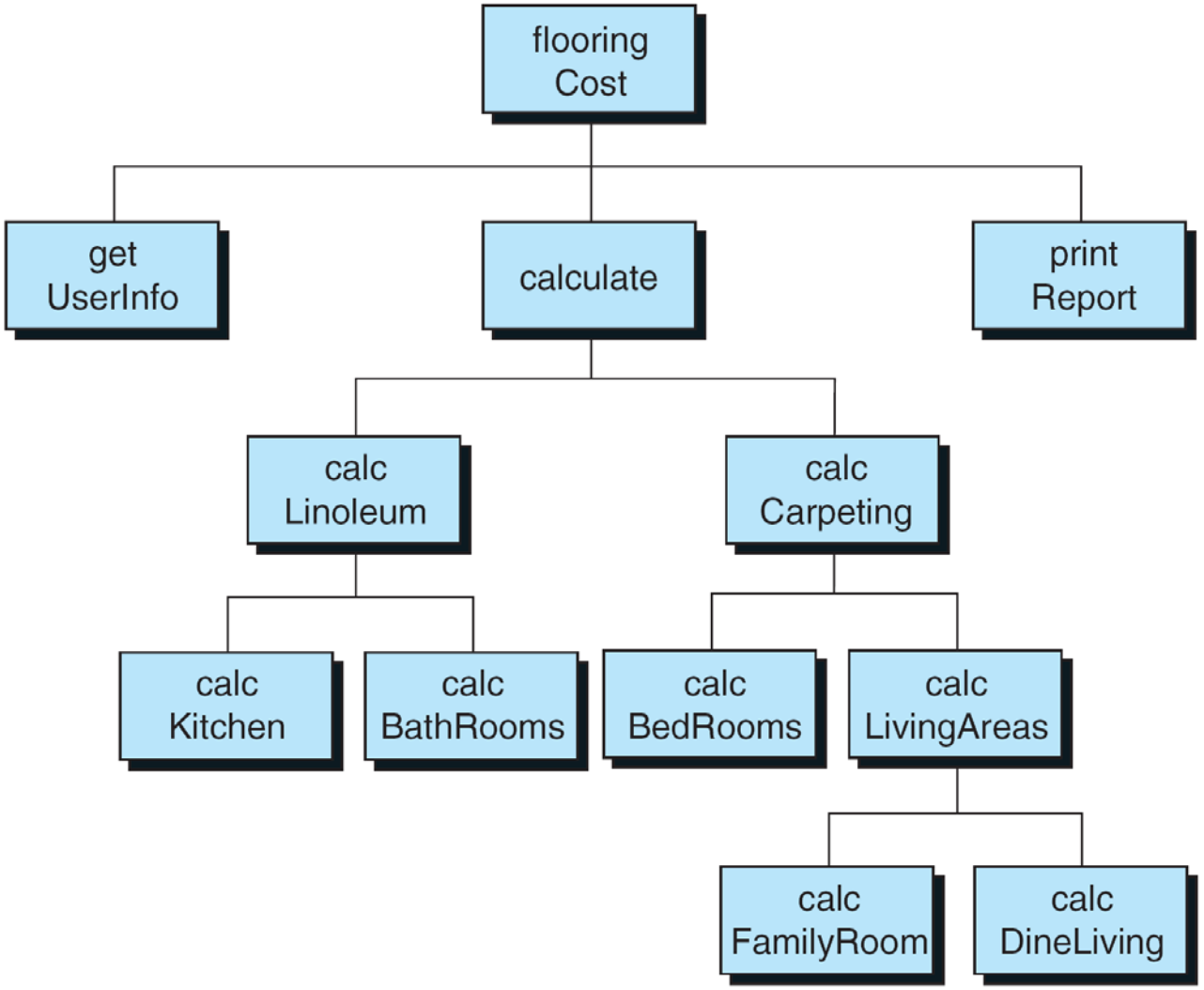

FIGURE 1-12 **Structure Chart for Calculating Square Footage**

Note

Pseudocode
English-like statements that follow a loosely defined syntax and are used to convey the design of an algorithm.

ALGORITHM 1-1 Pseudocode for Calculate Bathrooms

```
Algorithm Calculate BathRooms
 1 prompt user and read linoleum price
 2 prompt user and read number of bathrooms
 3 set total bath area and baths processed to zero
 4 while ( baths processed < number of bathrooms )
   1  prompt user and read bath length and width
   2  total bath area =
   3          total bath area + bath length * bath width
   4  add 1 to baths processed
 5 bath cost = total bath area * linoleum price
 6 return bath cost
end Algorithm Calculate BathRooms
```

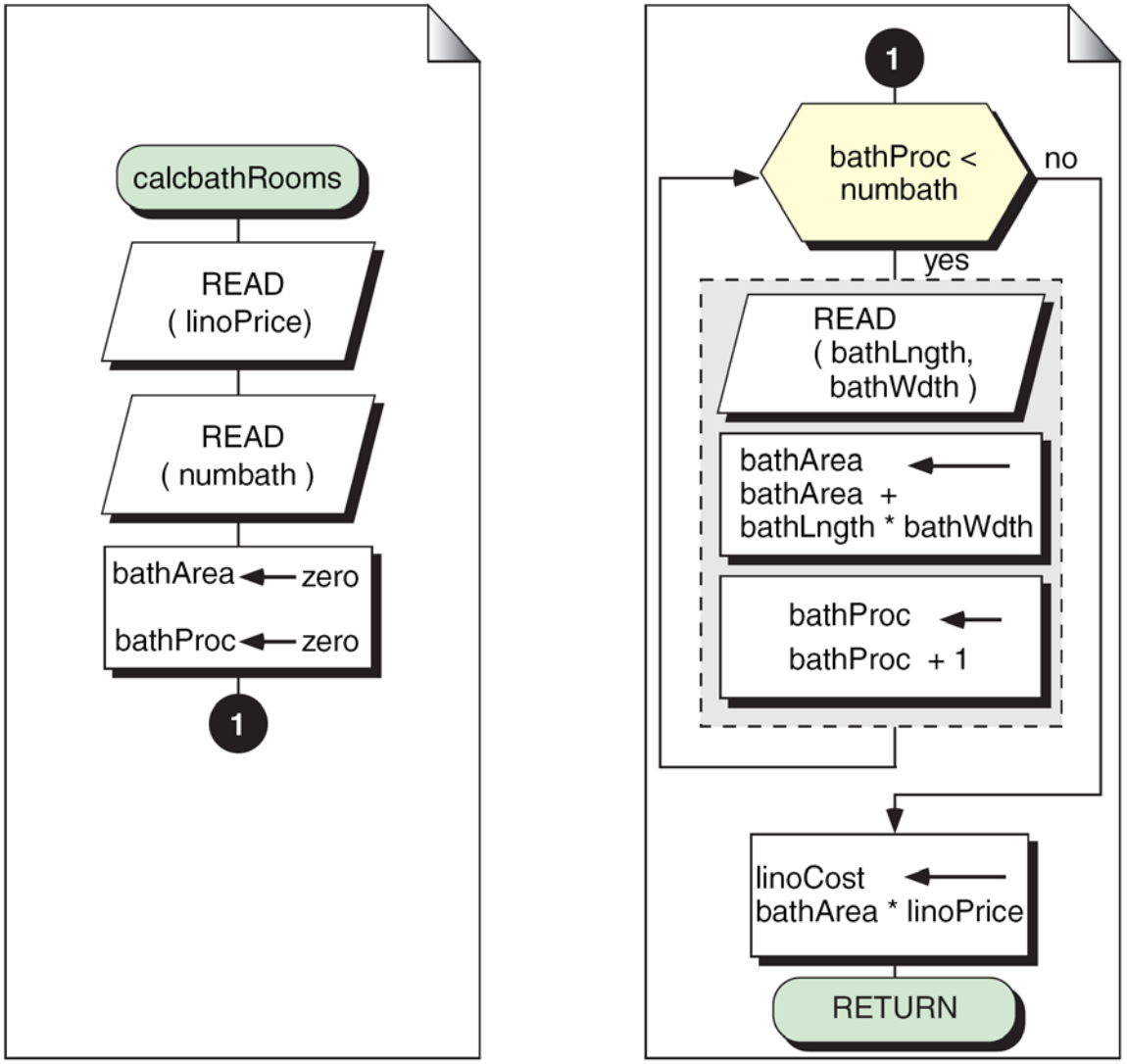

FIGURE 1-13 Flowchart for Calculate Bathrooms

Note

Except for the most simple program, one set of test data will not completely validate a program.

1-6 Software Engineering

Software engineering is the establishment and use of sound engineering methods and principles to obtain software that is reliable and that works on real machines.

This definition, from the first international conference on software engineering in 1969, was proposed 30 years after the first computer was built. During that period, software was more of an art than a science.