CMPS1134
Fundamentals of Computing

Programming Languages 1

Computer Science: An Overview
Eleventh Edition

J. Glenn Brookshear

Chapter 6

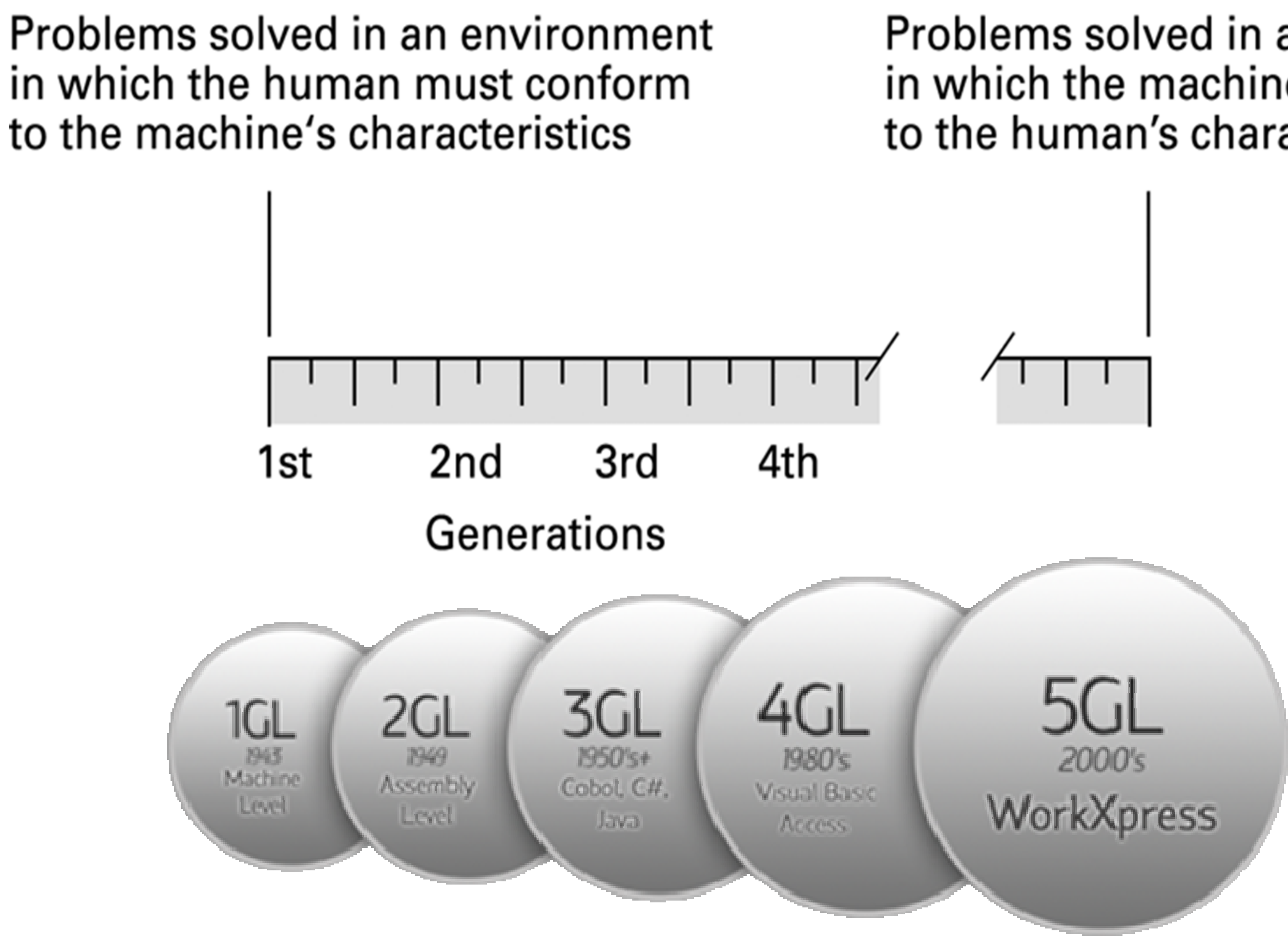
Chapter 6: Programming Languages

- **Historical Perspective**
- **Traditional Programming Concepts**
- **Procedural Units**

- Language Implementation
- Object Oriented Programming
- Programming Concurrent Activities
- Declarative Programming

Historical Perspective

Figure 6.1 **Generations of programming languages**



Historical Perspective
First-generation: Machine Language

- CPUs designed to recognize instructions encoded as bit patterns
- An instruction expressed in this language is a **machine instruction**
- Example:

156C	Load R5 with 6C
166D	Load R6 with 6D
5056	Add R5 & R6 into R0
30CE	Store R0 into CE
C000	Halt

Second-generation: Assembly language

- Developed in the 1940s
- Notational system that simplified programming process
- A **mnemonic** system for representing machine instructions
  - Mnemonic names for **op-codes**
  - Program **variables** or **identifiers**: Descriptive names for memory locations, chosen by the programmer

Historical Perspective

Assembly Language Characteristics

- One-to-one correspondence between machine instructions and assembly instructions
  - Programmer must think like the machine
- Inherently machine-dependent
- Converted to machine language by a program called an **assembler**

Historical Perspective
Program Example

Machine language	Assembly language	
156C	**LD**	R5, Price
166D	**LD**	R6, ShipCharge
5056	**ADDI**	R0, R5 R6
30CE	**ST**	R0, TotalCost
C000	**HLT**	

Historical Perspective
Third Generation Language

- Uses high-level primitives
  - Similar to our pseudocode in Chapter 5
- Machine independent (mostly)
- Examples: FORTRAN, COBOL
- Each primitive corresponds to a sequence of machine language instructions
- Converted to machine language by programs called **compilers** and **interpreters**

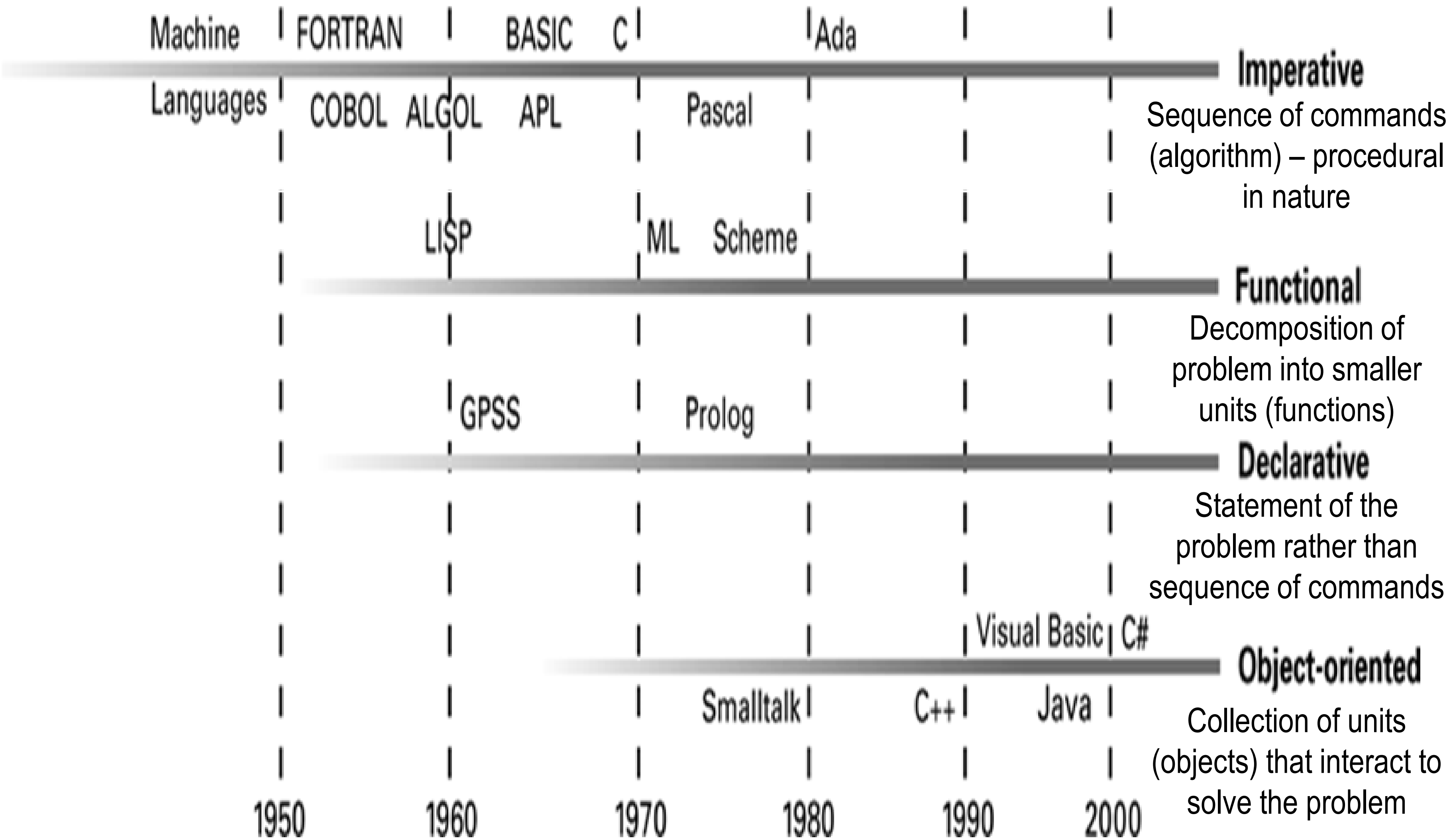
Fourth Generation Language

- **Non-procedural language** that enables users to access data in a database
- Data is accessed using English-like instructions or interaction with a graphical environment
- Easier to use than procedural languages
- Example: SQL

Historical Perspective
Fifth Generation Language

- True machine independence which will eliminate cross-platform incompatibilities
- Designed to make the computer solve the problem for you
- Used mainly in artificial intelligence research
- May be based on declarative programming, or graphical or visual tools to construct programs
- The goal of fifth-generation computing is to develop devices that respond to natural language input and are capable of learning and self-organization

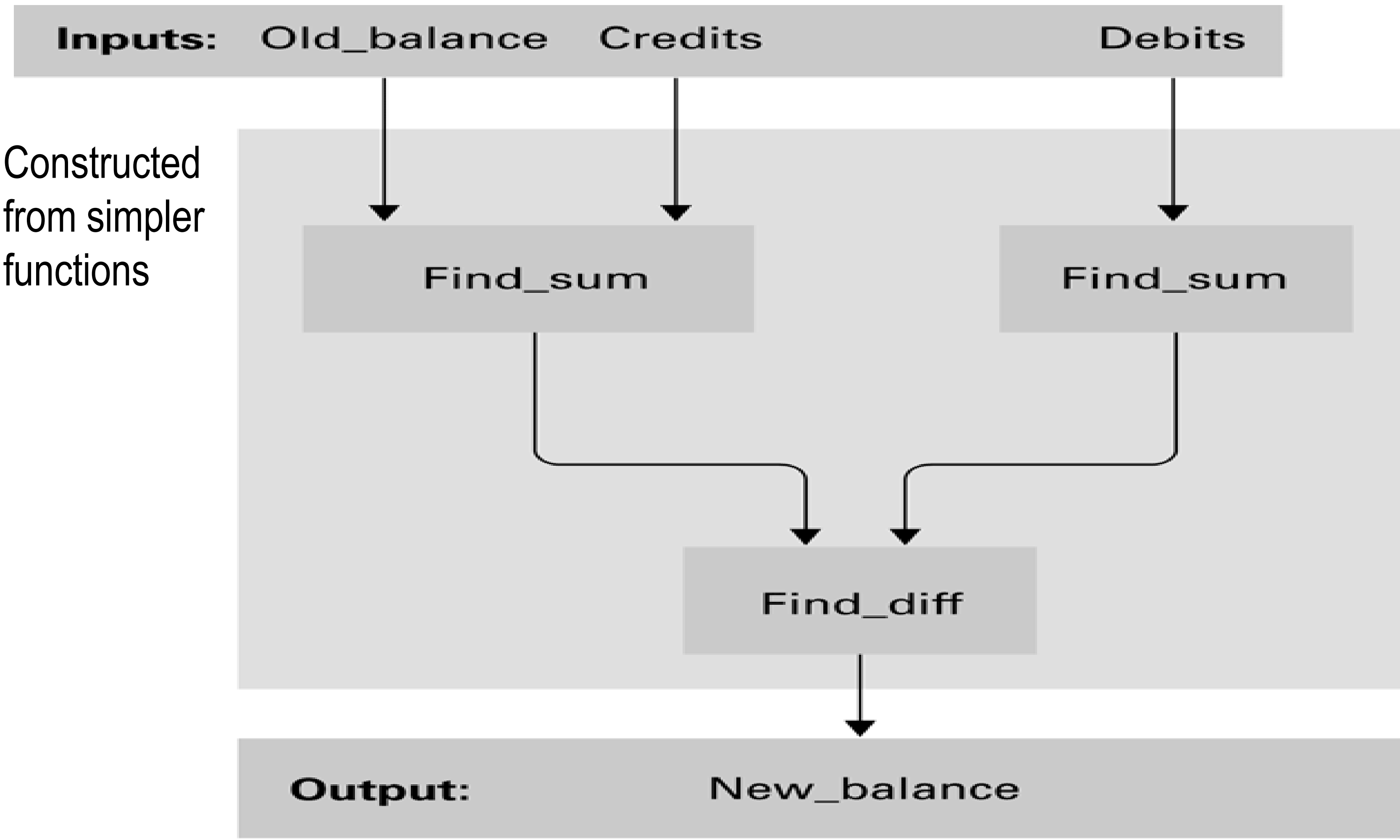
Historical Perspective
Evolution of programming paradigms (Fig 6.2)



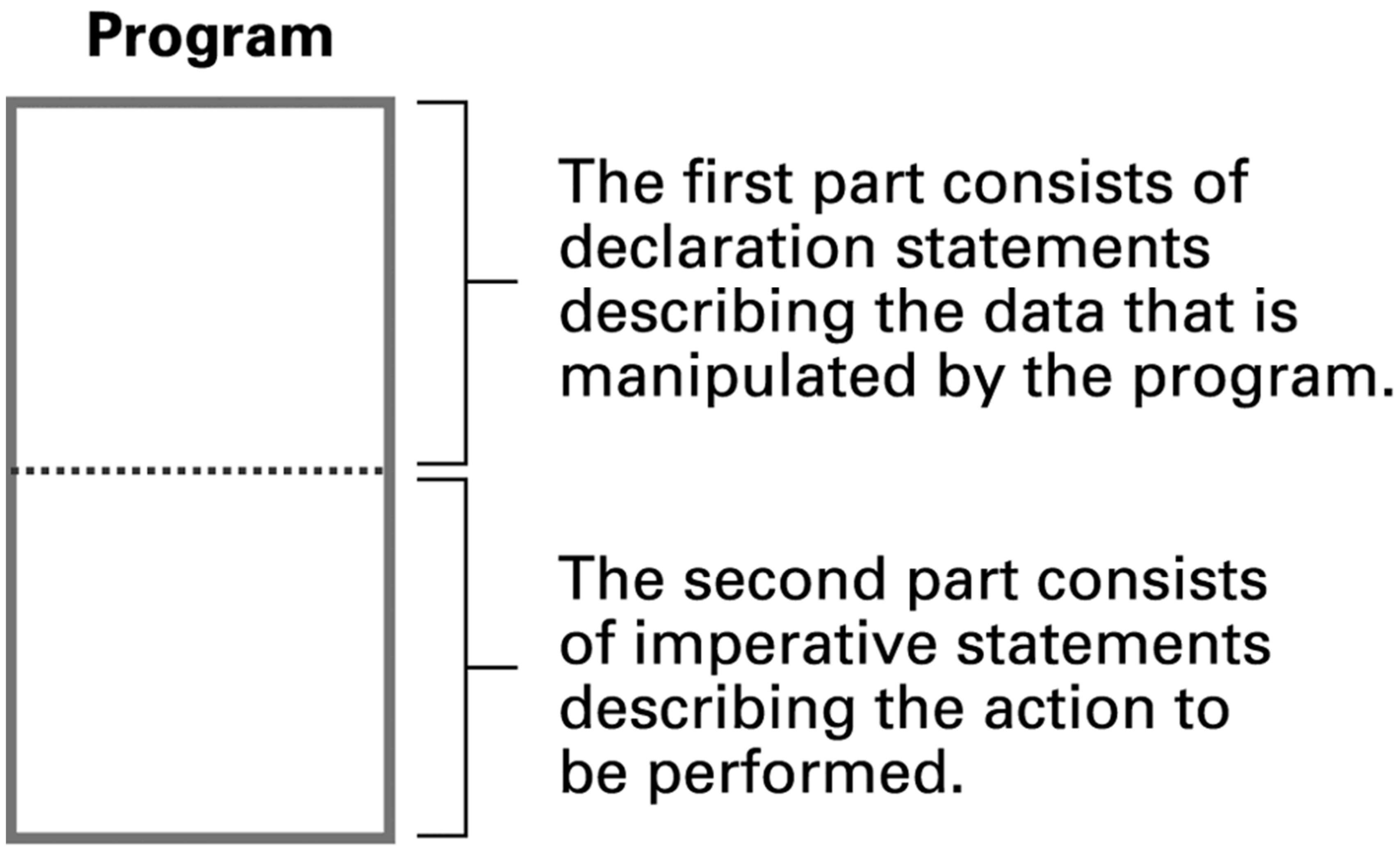
Historical Perspective: Functional Paradigm

Checkbook balancing function (Fig.6.3)

Constructed from simpler functions



The composition of a typical imperative program or program unit (Figure 6.4)



Variables and Data Types

Variables

Locations in main memory referenced by descriptive names rather than by numeric addresses

Data Types

Type of data that will be stored at the memory location associated with a variable:

- Integer: Whole numbers
- Real (float): Numbers with fractions
- Character: Symbols
- Boolean: True/false

Traditional Programming Concepts
Variables and Data Types Examples

```
float Length, Width;
int   Price, Total, Tax;
char  Symbol;

int   WeightLimit = 100;
```

Traditional Programming Concepts

Data Structure

- Conceptual shape or arrangement of data
- A common data structure is the array
  - In C

```
int Scores[2][9];
```

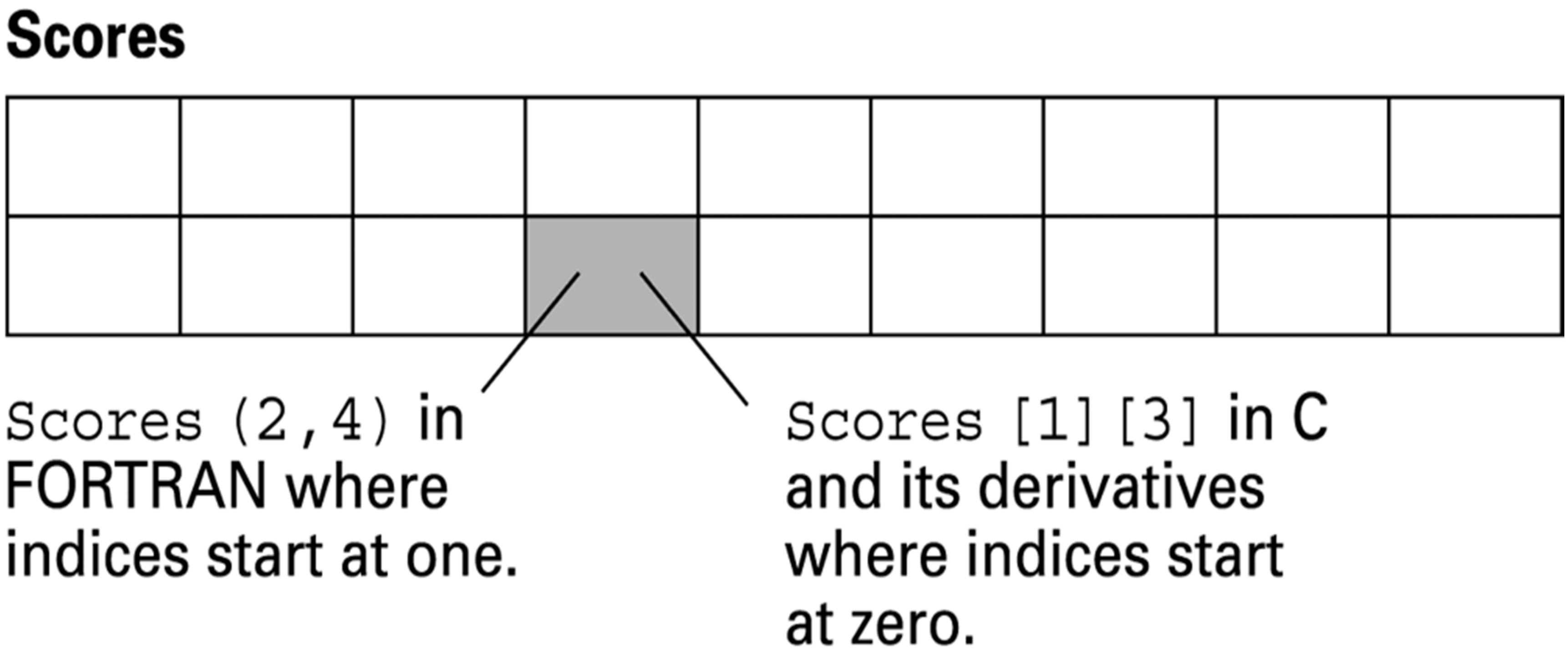
  - In FORTRAN

```
INTEGER Scores(2,9)
```

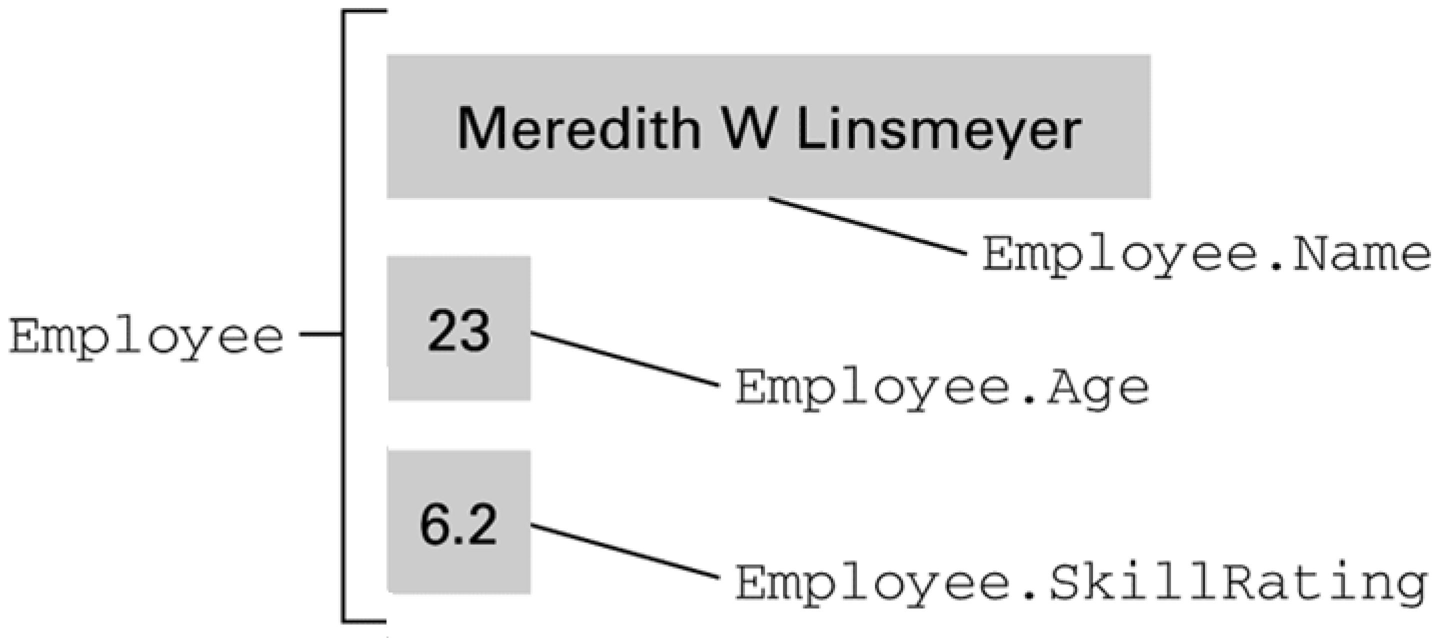
A two-dimensional array (Figure 6.5)

This array named **Scores** has two dimensions:

- two-(2) rows
- nine-(9) columns



Traditional Programming Concepts
The conceptual structure of the aggregate type Employee (Figure 6.6)




```
struct {   char  Name[25];
           int   Age;
           float SkillRating;
        } Employee;
```

Traditional Programming Concepts
Assignment Statements

- In C, C++, C#, Java

```
z = x + y;
```

- In Ada

```
z := x + y;
```

- In APL (A Programming Language)

```
z ← x + y
```

Control Statements

- Go to statement

```
        goto 40
   20   Evade()
        goto 70
   40   if (KryptoniteLevel < LethalDose) then goto 60
        goto 20
   60   RescueDamsel()
   70   ...
```

- As a single statement

```
if (KryptoniteLevel < LethalDose):
    RescueDamsel()
else:
    Evade()
```

Control Statements (continued)

- If in Python

```
if (condition):
    statementA
else:
    statementB
```

- In C, C++, C#, and Java

```
if (condition) statementA; else statementB;
```

- In Ada

```
IF condition THEN
    statementA;
ELSE
    statementB;
END IF;
```

Control Statements (continued)

- While in Python

```
while (condition):
    body
```

- In C, C++, C#, and Java

```
while (condition)
{ body }
```

- In Ada

```
WHILE condition LOOP
    body
END LOOP;
```

Control Statements (continued)

- Switch statement in C, C++, C#, and Java

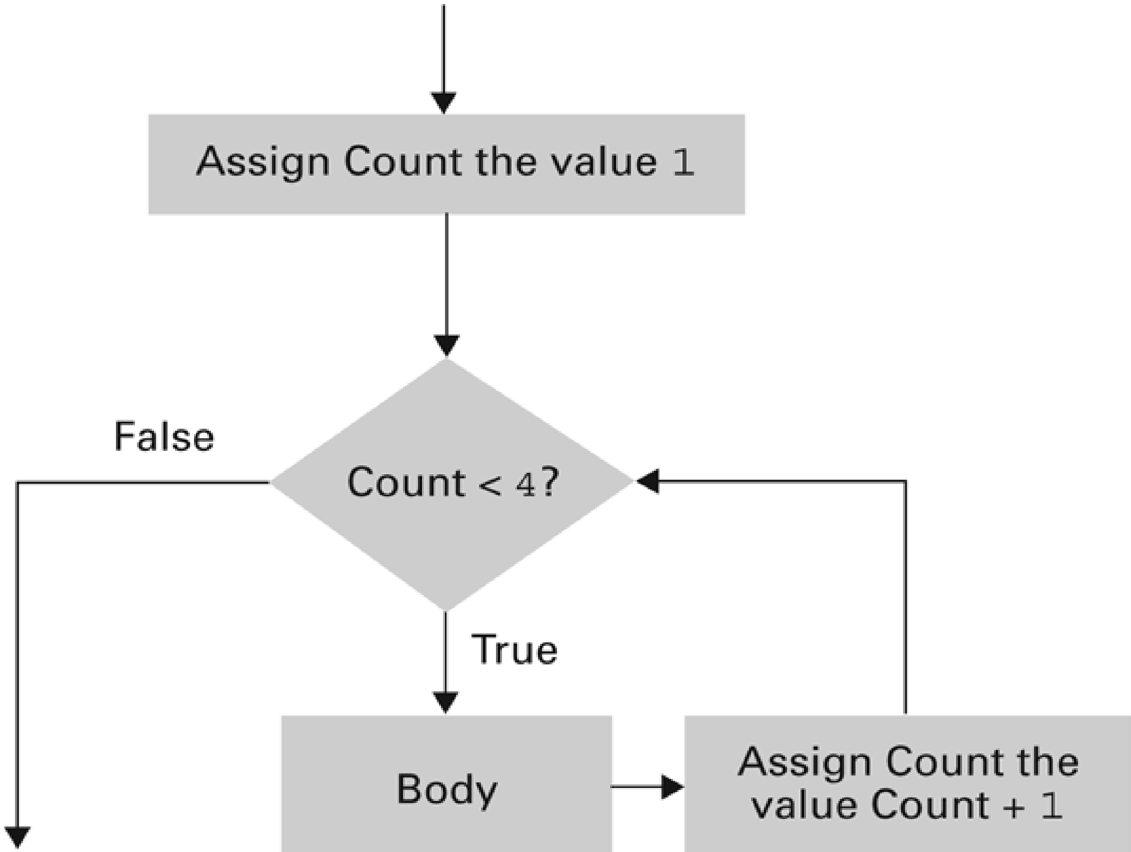
```
switch (variable) {
    case 'A': statementA; break;
    case 'B': statementB; break;
    case 'C': statementC; break;
    default:  statementD; }
```

- In Ada

```
CASE variable IS
    WHEN 'A'=> statementA;
    WHEN 'B'=> statementB;
    WHEN 'C'=> statementC;
    WHEN OTHERS=> statementD;
END CASE;
```

Traditional Programming Concepts

The for loop structure and its representation in C++, C#, and Java (Figure 6.7)




```
for (int Count = 1; Count < 4; Count++)
     body ;
```

Comments

- Explanatory statements within a program
- Helpful when a human reads a program
- Ignored by the compiler

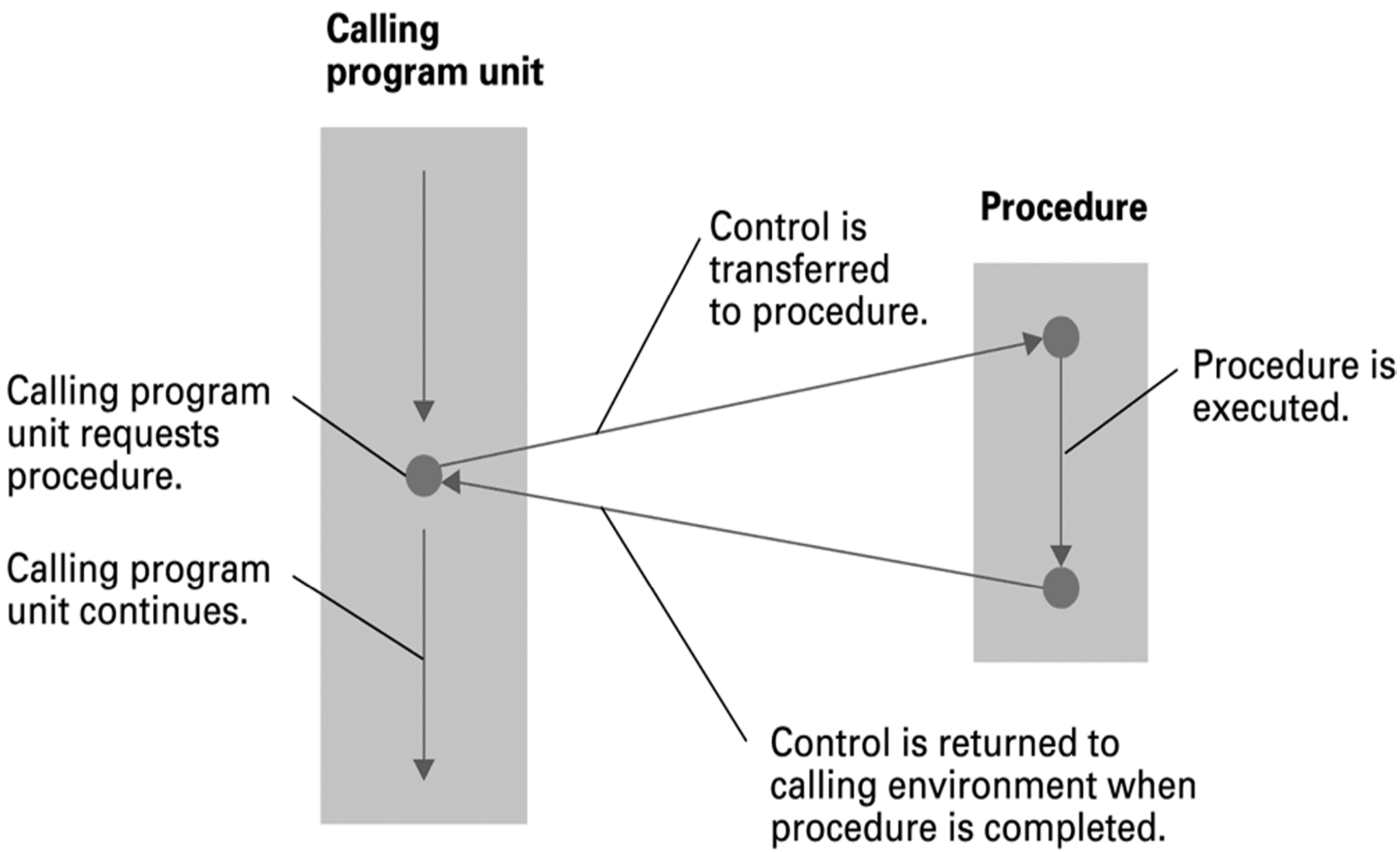
```
/* This is a comment. */

// This is a comment
```

Procedural Units

- Many terms for this concept:
  - Subprogram, subroutine, procedure, method, function
- Unit begins with the function’s header
- **Local** versus **Global** variables
- **Forma**l versus **Actual** Parameters
- Passing parameters:

  by value versus **by reference**

Procedural Units
Flow of control involving a procedure (Fig 6.8)



Procedural Units

PROCEDURES: Procedure ProjectPopulation written in C (Figure 6.9)

Starting the head with the term "void" is the way that a C programmer specifies that the program unit is a procedure rather than a function. We will learn about functions shortly.

The formal parameter list. Note that C, as with many programming languages, requires that the data type of each parameter be specified.

```
void ProjectPopulation (float GrowthRate)

{ int Year;

  Population[0] = 100.0;
  for (Year = 0; Year =< 10; Year++)
  Population[Year+1] = Population[Year] + (Population[Year] * GrowthRate);
}
```

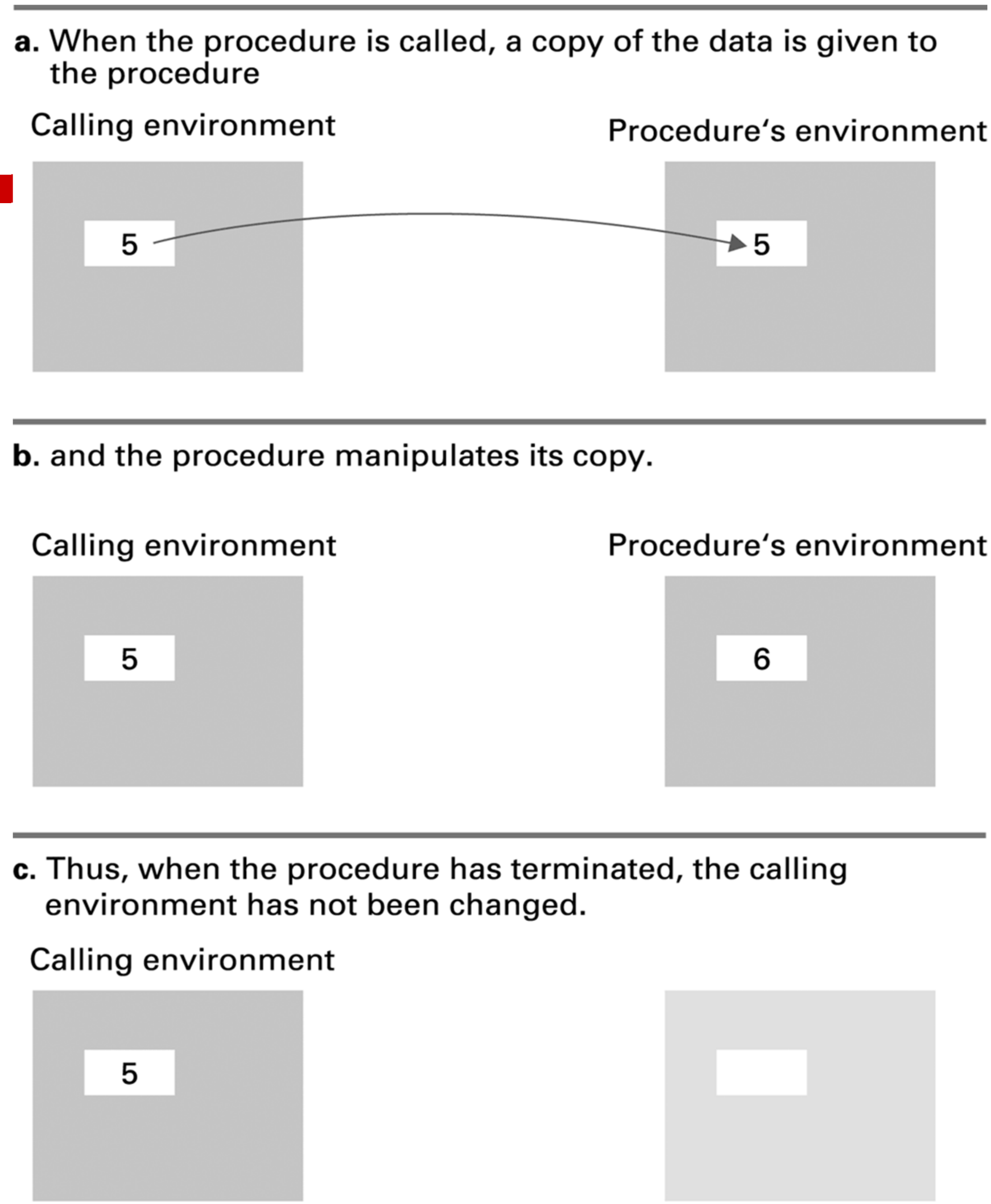
This declares a local variable named Year.

These statements describe how the populations are to be computed and stored in the global array named Population.

Procedural Units

Executing a procedure passing parameters by value

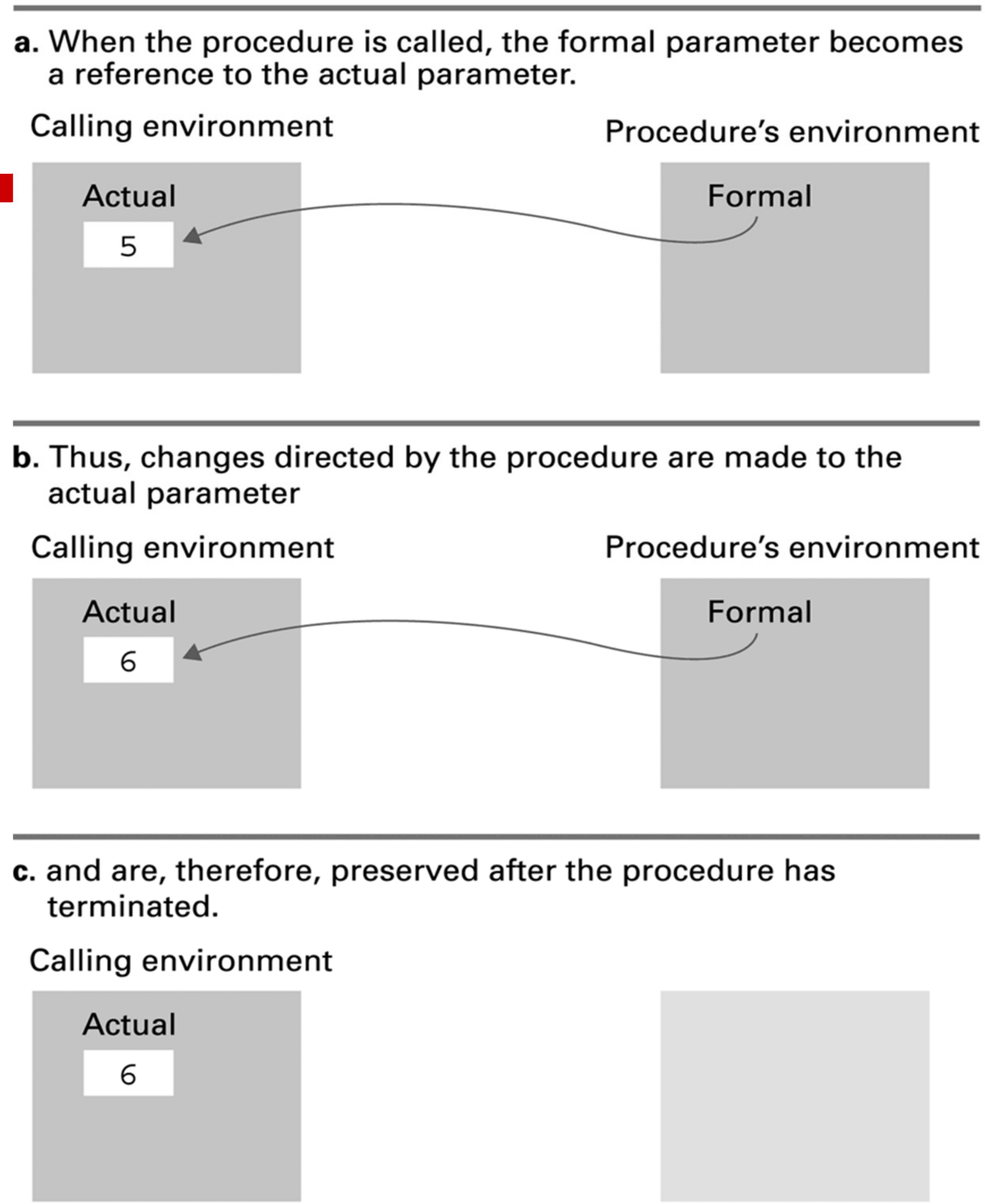
(Figure 6.10)



Procedural Units

Executing a procedure by passing parameters by reference

(Figure 6.11)



Procedural Units

FUNCTIONS: Function CylinderVolume written in C (Figure 6.12)

The function header begins with the type of the data that will be returned.

```
float CylinderVolume (float Radius, float Height)
{ float Volume;
  Volume = 3.14 * Radius * Radius * Height;
  return Volume;
}
```

Declare a local variable named Volume.

Compute the volume of the cylinder.

Terminate the function and return the value of the variable Volume.