computer science
illuminated

High-Level Programming Languages

Nell Dale & John Lewis
(adaptation by Michael Goldwasser)

Overview

- **Programming in a low-level language is tedious!**

  (have we made that clear yet?)

- **High-level languages offer:**
  - **better portability (the program runs on many CPU types)**
  - **data structures and other memory management**
  - **natural structures for expressing the flow of control**
  - **much better support for software maintenance and reuse**

Compilers

- **Compiler:** a program that translates a high-level language program into machine code (similar role as an assembler).
- High-level languages provide a richer set of instructions that makes the programmer's life even easier

Compilers

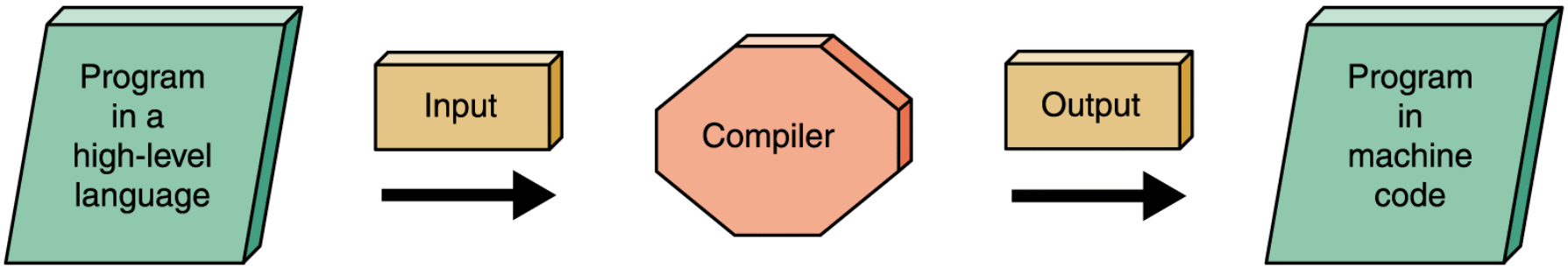


Figure 8.1 **Compilation process**

Interpreters

- **Interpreter:** a translating program that translates and executes the statements in sequence
  - Unlike an assembler or compiler which must be run in advance.
  - An interpreter translates a statement and then immediately executes the statement
  - Interpreters can be viewed as *simulators*

A Compiled Language

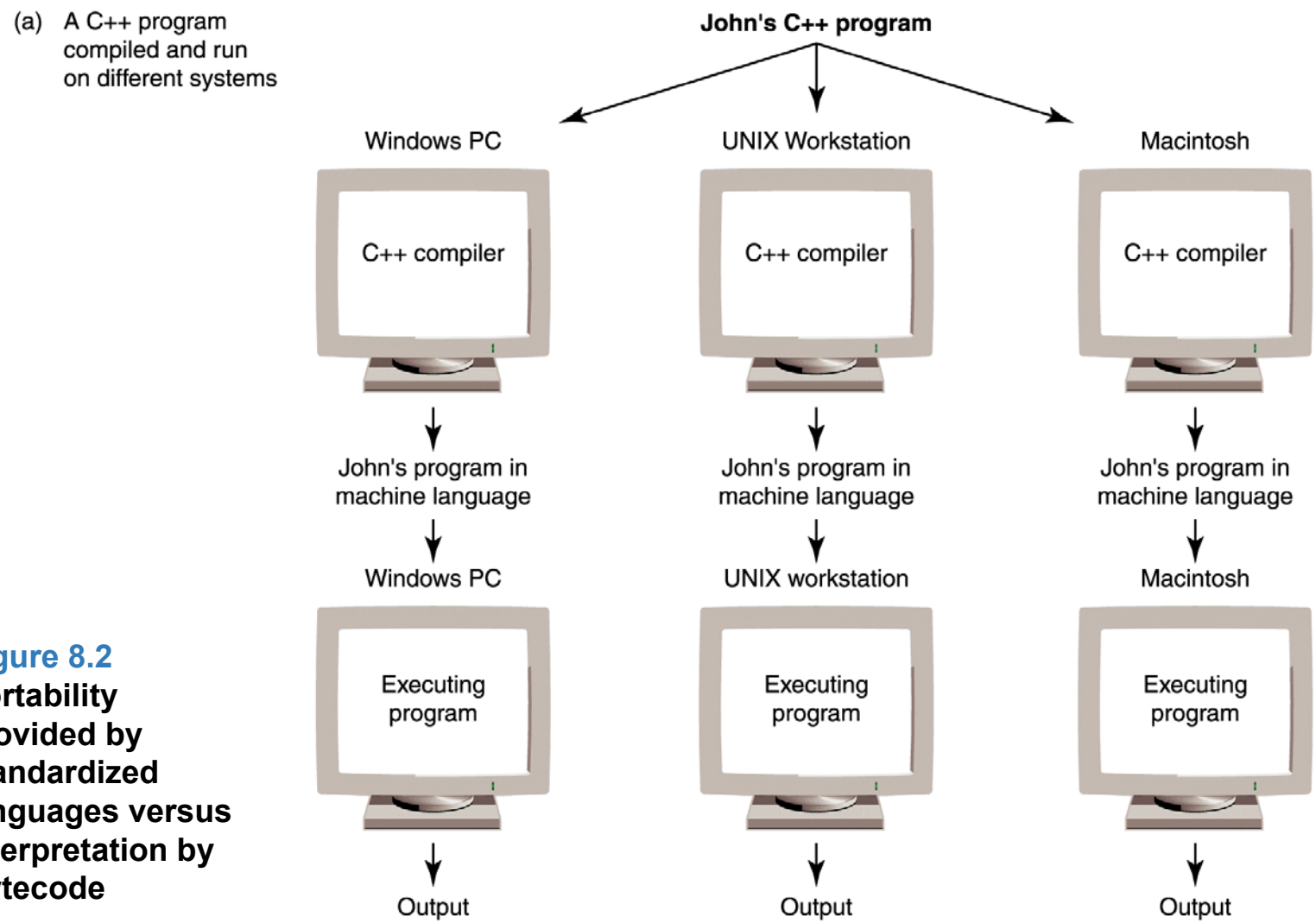


Figure 8.2
Portability provided by standardized languages versus interpretation by Bytecode

An Interpreted Language

(b) Java program compiled into Bytecode and run on different systems

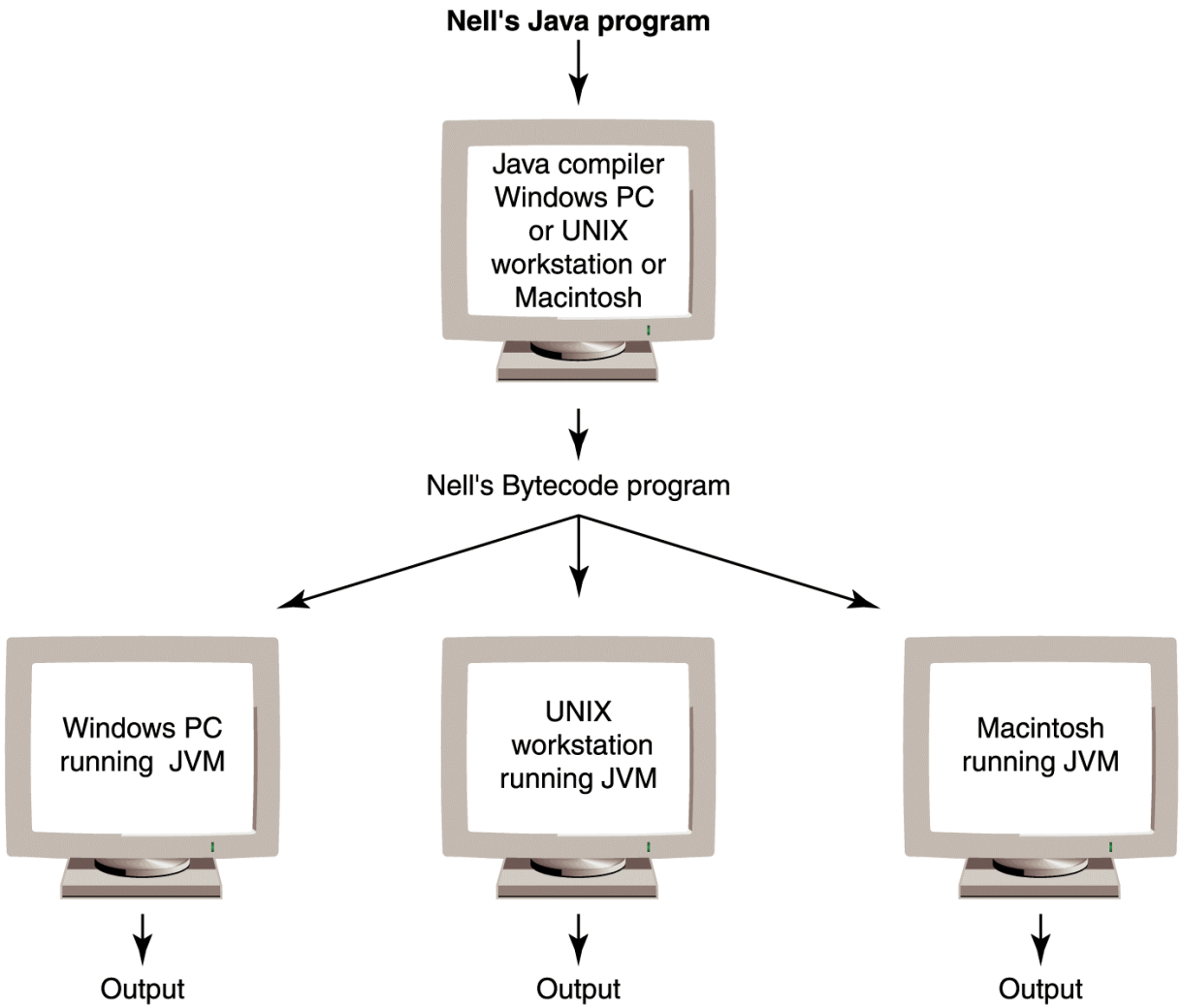


Figure 8.2
Portability provided by standardized languages versus interpretation by Bytecode

Programming Language Paradigms

- A *paradigm*?

- "A set of assumptions, concepts, values, and practices that constitute a way of viewing reality for the community that shares them"

  – American Heritage Dictionary

Programming Language Paradigms

- Imperative or procedural model
  - FORTRAN, COBOL, BASIC, C, Pascal, Ada, and C++
- Functional model
  - LISP, Scheme (a derivative of LISP), and ML
- Logic Programming
  - PROLOG

Programming Language Paradigms (cont.)

- Object-oriented paradigm
  - SIMULA and Smalltalk
  - C++ is as an imperative language with some object-oriented features
  - Java is an object-oriented language with some imperative features

Data Types

- Integer numbers
- Real numbers (actually, floating point)
- Characters
- Strings
- Boolean values (i.e., a single bit, usually interpreted as a true/false value)
- How was data handled by machine code?

Boolean Expressions

- **Boolean expression:** a sequence of identifiers, separated by compatible operators, that evaluates to *true* or *false*
- Boolean expression can be
  - A Boolean variable
  - An arithmetic expression followed by a relational operator followed by an arithmetic expression
  - A Boolean expression followed by a Boolean operator followed by a Boolean expression

Boolean Expressions

- A relational operator between two arithmetic expressions is asking if the relationship exists between the two expressions
- For example: *xValue* < *yValue*

Relationship	Symbol
equal to	= or ==
not equal to	<> or != or /=
less than or equal to	<=
greater than or equal to	>=
less than	<
greater than	>

Page 233

Strong Typing

- **Strong typing:** the requirement that only a value of the proper type can be stored into a variable
- A **data type** is a description of the set of values and the basic set of operations that can be applied to values of the type

Declarations

- A **declaration** is a statement that associates an identifier with a variable, an action, or some other entity within the language that can be given a name so that the programmer can refer to that item by name

Declarations (cont.)

Language	Variable Declaration

Ada	`sum : Float := 0; --set up word with 0 as contents`
	`num1: Integer;  --set up a two-byte block for num1`
	`num2: Integer;  --set up a two-byte block for num2`
	`num3: INTEGER;  --set up a two-byte block for num3`
	`...`
	`num1:= 1;`

VB.NET	`Dim sum As Single = 0.0F ' set up word with 0 as contents`
	`Dim num1 As Integer ' set up a two-byte block for num1`
	`Dim num2 As Integer ' set up a two-byte block for num2`
	`Dim num3 As Integer ' set up a two-byte block for num3`
	`...`
	`num1 = 1`

C++/Java	`float sum = 0.0;  // set up word with 0 as contents`
	`int num1:  // set up a two-byte block for num1`
	`int num2:  // set up a two-byte block for num2`
	`int num3:  // set up a two-byte block for num3`
	`num1 = 1;`

Assignment statement

- **Assignment statement:** an action statement (not a declaration) that says to evaluate the expression on the right-hand side of the symbol and store that value into the place named on the left-hand side
- **Named constant:** A location in memory, referenced by an identifier, that contains a data value that cannot be changed

Assignment Statement

	Constant Declaration

Ada	`Comma : constant Character := ',';`
	`Message : constant String := "Hello";`
	`Tax_Rate : constant Float := 8.5;`

VB.NET	`Const WORD1 As Char = ","c`
	`Const MESSAGE As String = "Hello"`
	`Const TaxRate As Double = 8.5`

C++	`const char COMMA = ',';`
	`const string MESSAGE = "Hello";`
	`const double TAX_RATE = 8.5;`

Java	`final char COMMA = ',';`
	`final String MESSAGE = "Hello";`
	`final double TAX_RATE = 8.5;`

Composite Data Types

- Records
  - A record is a named *heterogeneous* collection of items in which individual items are accessed by name
  - The elements in the collection can be of various types

Arrays

- An array is a named collection of homogeneous items in which individual items are accessed by their place within the collection
  - The place within the collection is called an *index*

Language	Array Declaration
Ada	`type Index_Range is range 1..10;` `type Ten_Things is array (Index_Range) of Integer;`
VB.NET	`Dim TenThings(10) As Integer`
C++/Java	`int tenThings[10];`

Page 260

Arrays

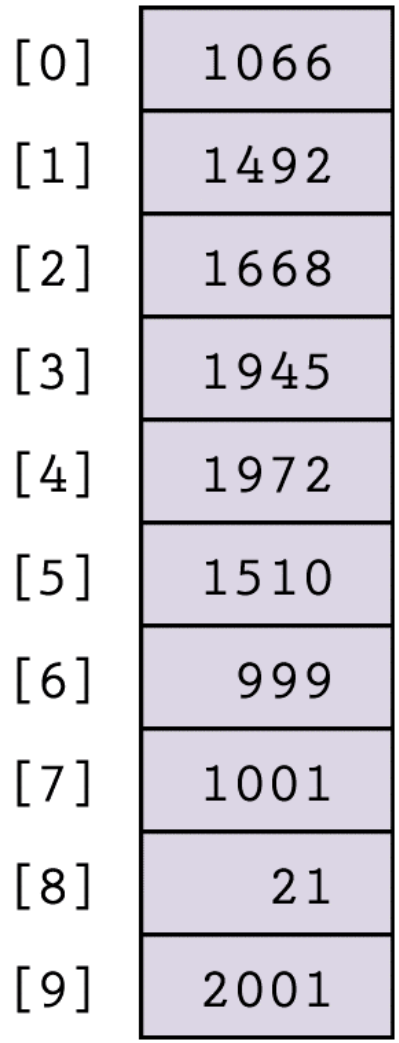


Figure 8.8
Array variable *tenThings* **accessed from** *0..9*

Control Structures

- **Control structure:** an instruction that determines the order in which other instructions in a program are executed
- Structured programming
  - each logical unit of a program should have just one entry and one exit
- These constructs are selection statements, looping statements, and subprogram statements
  - Statements are executed in sequence until an instruction is encountered that changes this sequencing

Selection Statements

- The ***if*** statement allows the program to test the state of the program variables using a Boolean expression

Language	if Statement

Ada	`if Temperature > 75 then`
	`  Put(Item => "No jacket is necessary")`
	`else`
	`  Put (Item => "A light jacket is appropriate");`
	`end if;`

VB.NET	`if (Temperature > 75) Then`
	`  MsgBox("No jacket is necessary")`
	`Else`
	`  MsgBox("A light jacket is appropriate")`
	`End if`

C++	`if (temperature > 75)`
	`  cout << "No jacket is necessary";`
	`else`
	`  cout << "A light jacket is appropriate";`

Java	`if (temperature > 75)`
	`  System.out.print("No jacket is necessary");`
	`else`
	`  System.out.print("A light jacket is appropriate");`

Page 243

Selection Statements

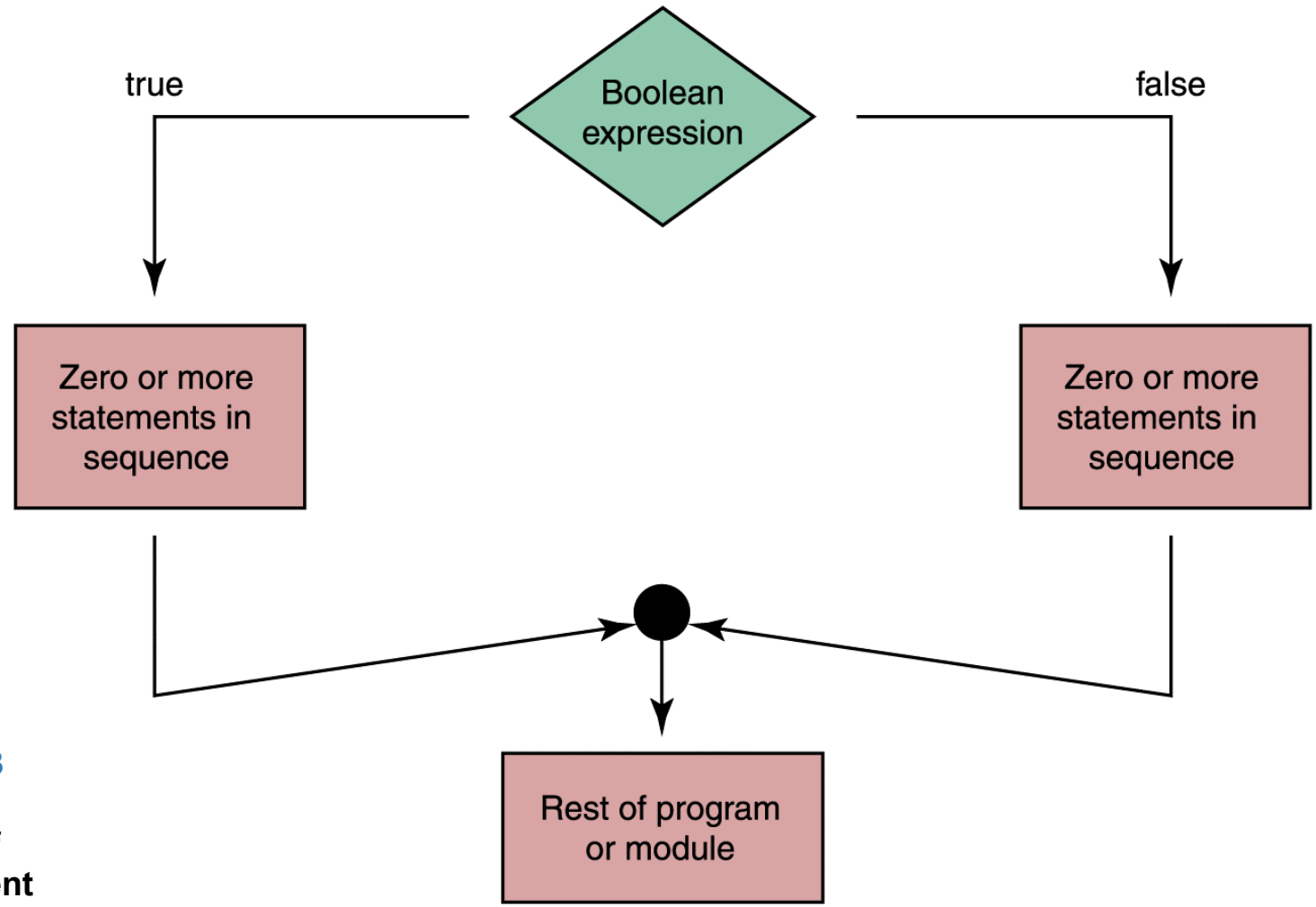


Figure 8.3
Flow of control of *if* statement

Selection Statements

```
If (temperature > 90)
        Write "Texas weather: wear shorts"
Else If (temperature > 70)
        Write "Ideal weather: short sleeves are fine"
Else if (temperature > 50)
        Write "A little chilly: wear a light jacket"
Else If (temperature > 32)
        Write "Philadelphia weather: wear a heavy coat"
Else
        Write "Stay inside"
```

Page 245

case Statement

- For convenience, many high-level languages include a case (or switch) statement
- Allows us to make multiple-choice decisions easier, provided the choices are discrete

```
CASE operator OF
'+'  : Set answer to one + two
'-'  : Set answer to one – two
'*'  : Set answer to one * two
'/'  : Set answer to one / two
```

Page 246

Looping Statements

- The ***while*** statement is used to repeat a course of action
- Let's look at two distinct types of repetitions

Looping Statements

- *Count-controlled loops*
  - Repeat a specified number of times
  - Use of a special variable called a loop control variable

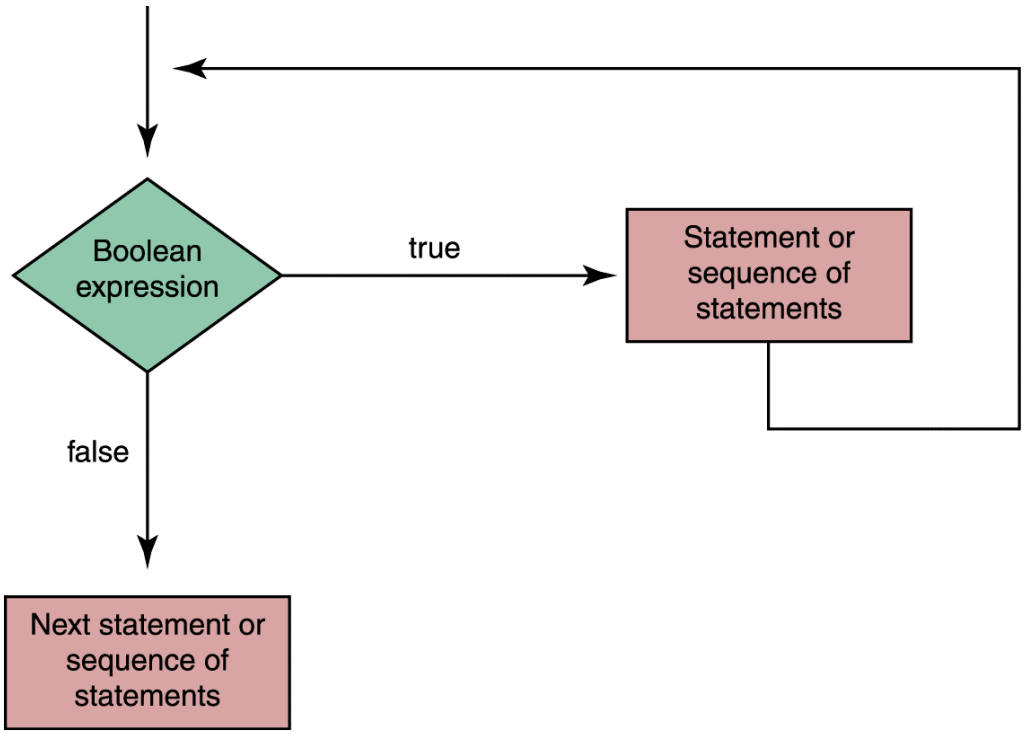


Figure 8.4
Flow of control of *while* statement

Looping Statements

- *Count-controlled* loops

Language	Count-Controlled Loop with a while Statement

Ada	`Count = 1;`
	`while Count <= Limit loop`
	`  ...`
	`  Count = Count + 1;`
	`end loop;`

VB.NET	`Count = 1`
	`While (count <= limit)`
	`  ...`
	`  count = count + 1`
	`End While`

C++/Java	`count = 1;`
	`while (count <= limit)`
	`{`
	`  ...`
	`  count++;`
	`}`

Looping Statements

- *Event-controlled* loops
  - The number of repetitions is controlled by an event that occurs within the body of the loop itself

```
Read a value                Initialize event
While (value >= 0)          Test event
    ...                     Body of loop
    Read a value            Update event
...                         Statement(s) following loop
```

Page 249

Looping Statements

– *Event-controlled* loops

```
Set sum to 0                          Initialize sum to zero
Set posCount to 0                     Initialize event
While (posCount <= 10)                Test event
      Read a value
      If (value > 0)                  Test to see if event should be updated
            Set posCount to posCount + 1    Update event
            Set sum to sum + value          Add value into sum
...                                   Statement(s) following loop
```

Page 249

Subprogram Statements

- We can give a section of code a name and use that name as a statement in another part of the program
- When the name is encountered, the processing in the other part of the program halts while the named code is executed

Subprogram Statements

- There are times when the calling unit needs to give information to the subprogram to use in its processing
- A **parameter list** is a list of the identifiers with which the subprogram is to work, along with the types of each identifier placed in parentheses beside the subprogram name

Subprogram Statements

(a) Subprogram A does its task and calling unit continues with next statement

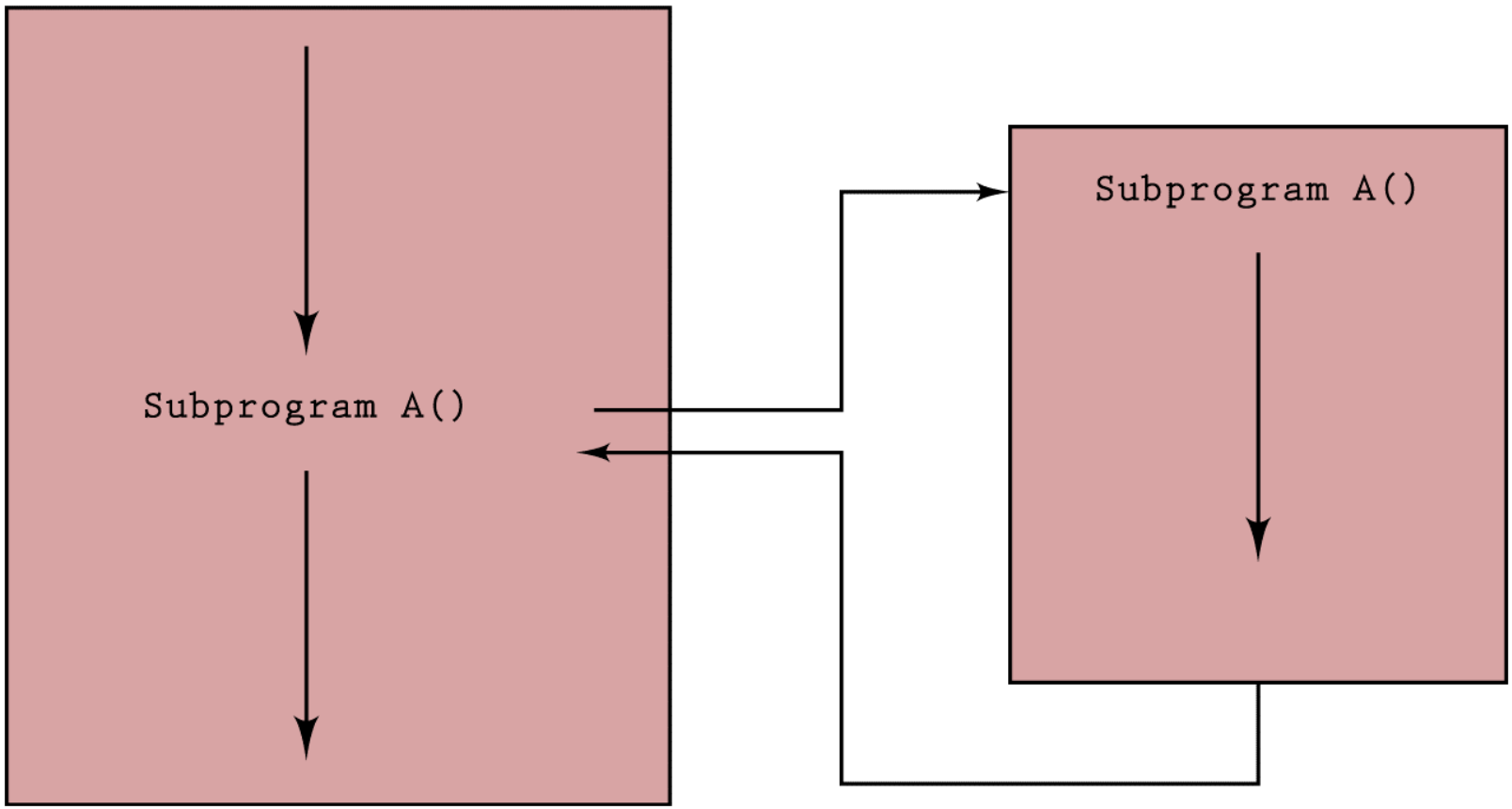


Figure 8.5 **Subprogram flow of control**

Subprogram Statements

(b) Subprogram B does its task and returns a value that is added to 5 and stored in `x`

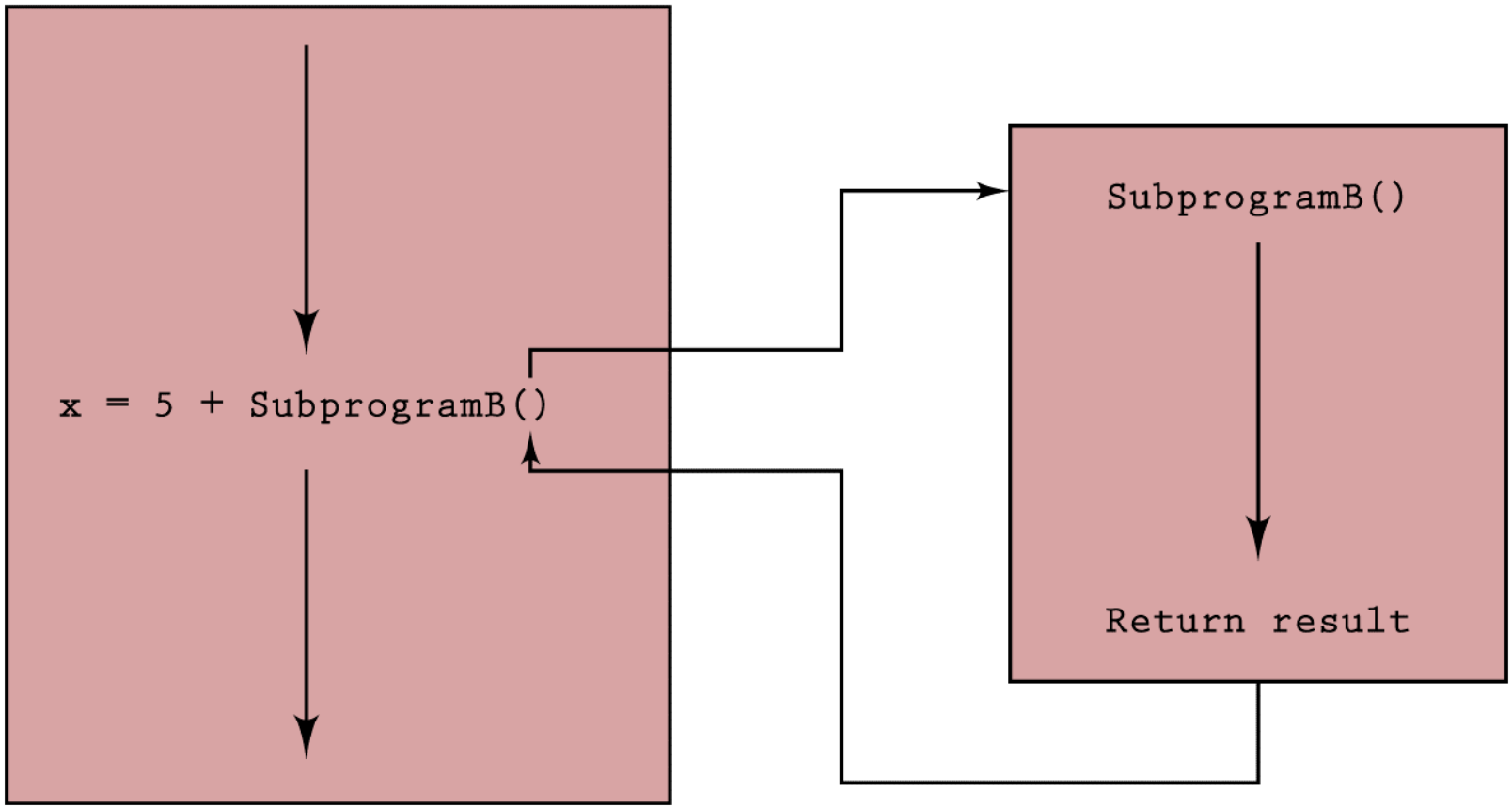


Figure 8.5 **Subprogram flow of control**

Subprogram Statements

- **Value parameter:** a parameter that expects a copy of its argument to be passed by the calling unit
- **Reference parameter:** a parameter that expects the address of its argument to be passed by the calling unit
- Review of Machine Language: immediate operands vs. direct addressing

Subprogram Statements

Language	Subprogram Declaration

VB.NET	`Public Sub Example(ByVal one As Integer,`
	`  ByVal two As Integer,`
	`  ByRef three As Single)`
	`  ...`
	`End Sub`

C++/Java	`void Example(int one; int two; float& three)`
	`{`
	`  ...`
	`}`

Page 253

Recursion

- **Recursion:** the ability of a subprogram to call itself
- Each recursive solution has at least two cases
  - *base case:* the one to which we have an answer
  - *general case:* expresses the solution in terms of a call to itself with a smaller version of the problem
- For example, the factorial of a number is defined as the number times the product of all the numbers between itself and 0:

$$N! = N * (N - 1)!$$

Functionality of Object-Oriented Languages

- Encapsulation
- Inheritance
- Polymorphism

Encapsulation

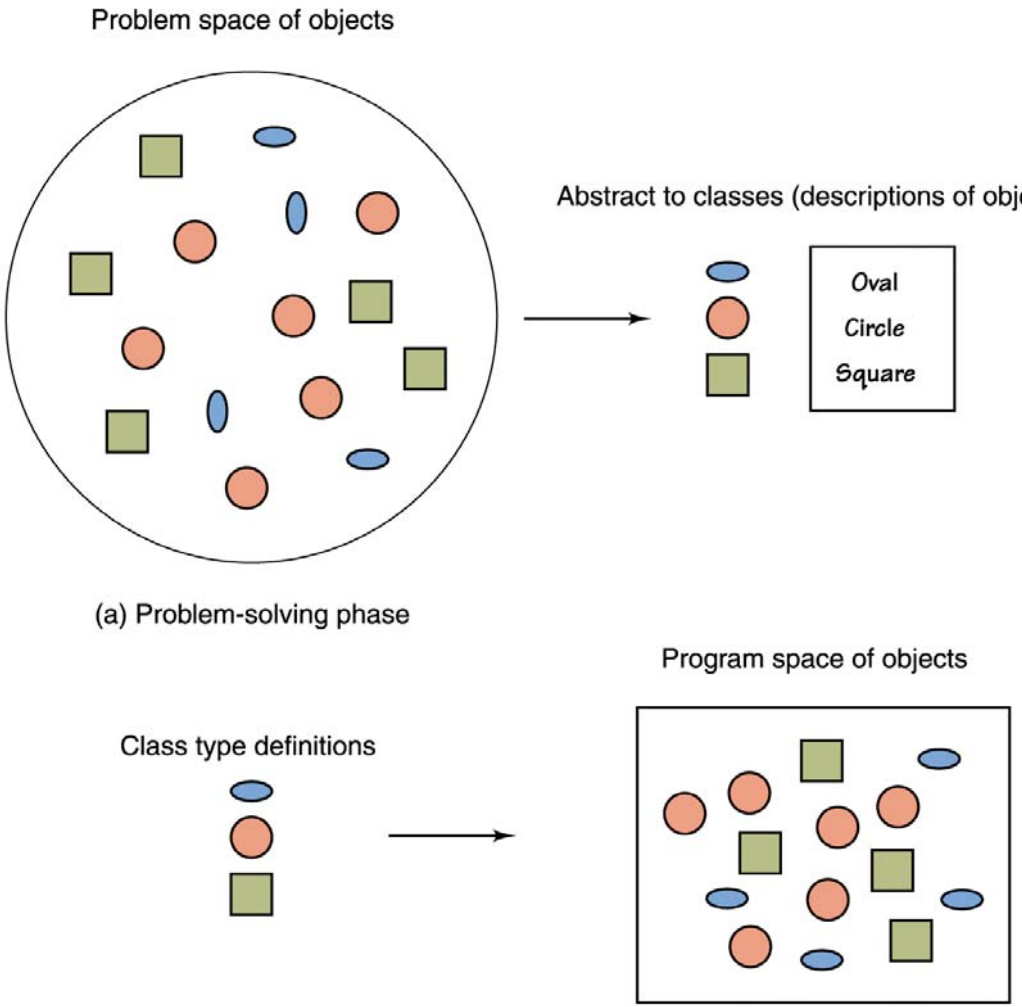
- **Encapsulation** is a language feature that enforces information hiding

- A **class** is a language construct that is a pattern for an object and provides a mechanism for encapsulating the properties and actions of the object class

Inheritance

- Inheritance fosters reuse by allowing an application to take an already-tested class and derive a class from it that inherits the properties the application needs
- **Polymorphism:** the ability of a language to have duplicate method names in an inheritance hierarchy and to apply the method that is appropriate for the object to which the method is applied

Inheritance



- Inheritance and polymorphism combined allow the programmer to build useful hierarchies of classes that can be reused in different applications

Figure 8.9
Mapping of problem into solution

Asynchronous Processing

- Asynchronous processing is the concept that input and output can be accomplished through windows on the screen
  - *Clicking* has become a major form of input to the computer
  - Mouse clicking is not within the sequence of the program
  - A user can click a mouse at any time during the execution of a program
  - This type of processing is called **asynchronous**

Ethical Issues: Hacking

- Hacking refers to the trespassing or accessing of a Web site without authorization
- Whether the hackers damage the content or leave the site untouched, their ability to infiltrate secure systems is powerful and disturbing
- One study asserts that 59% of all company-owned Web sites were hacked during 1997