Chapter 6

Pearson International Edition

J. Glenn Brookshear

Computer Science

An Overview

Programming Languages

J. Glenn Brookshear
蔡文能

Agenda

Review

6.1 Historical perspective

6.2 Traditional programming concepts

6.3 Procedural Units

6.4 Language Implementation

6.5 Object-oriented Programming

6.6 Programming Concurrent Activities

6.7 Declarative programming (Prolog)

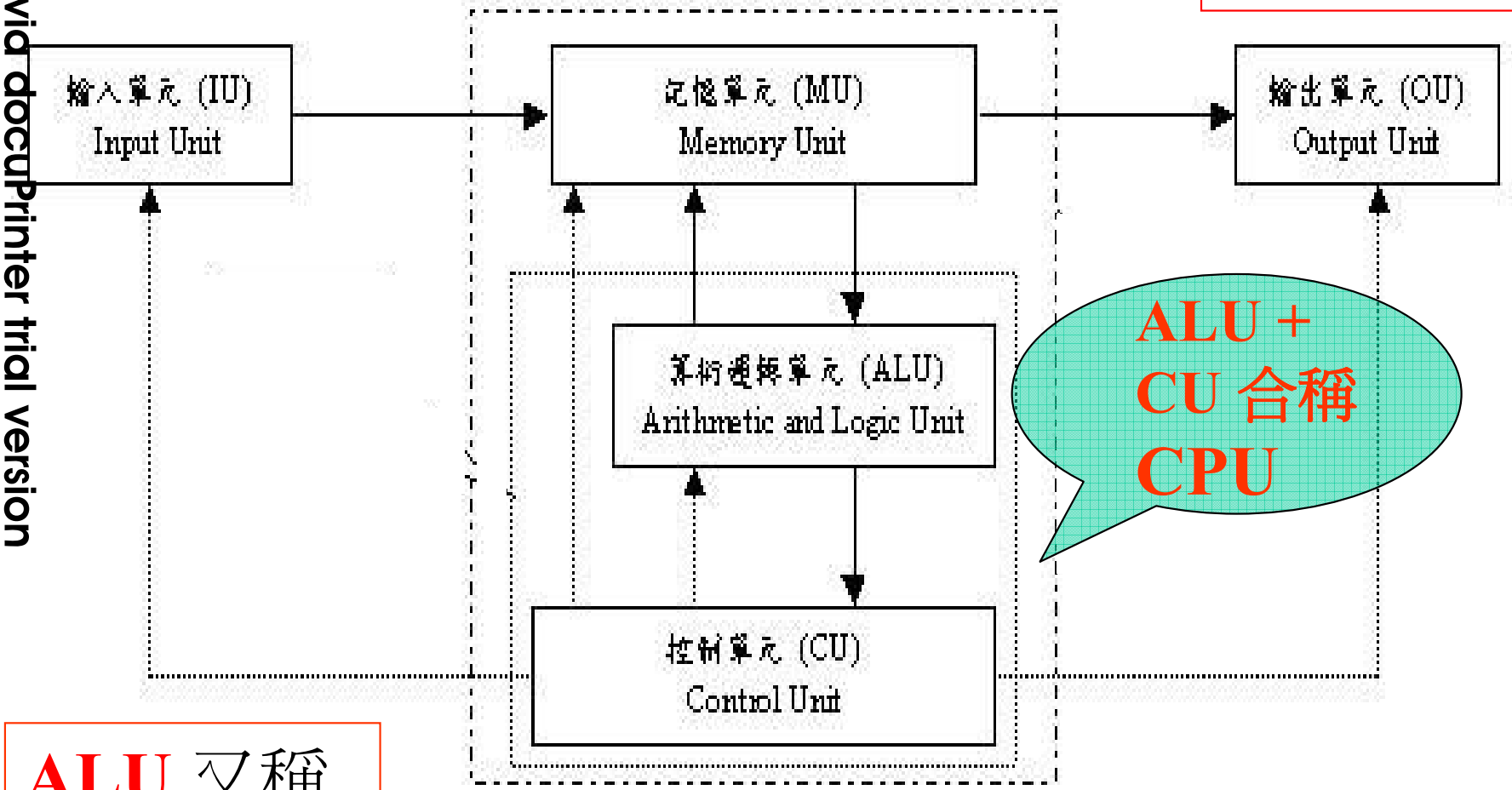
電腦硬體五大單元

計算 2+3=?

人會如何做?

電腦如何做?

單晶片微電腦 = CPU + Memory

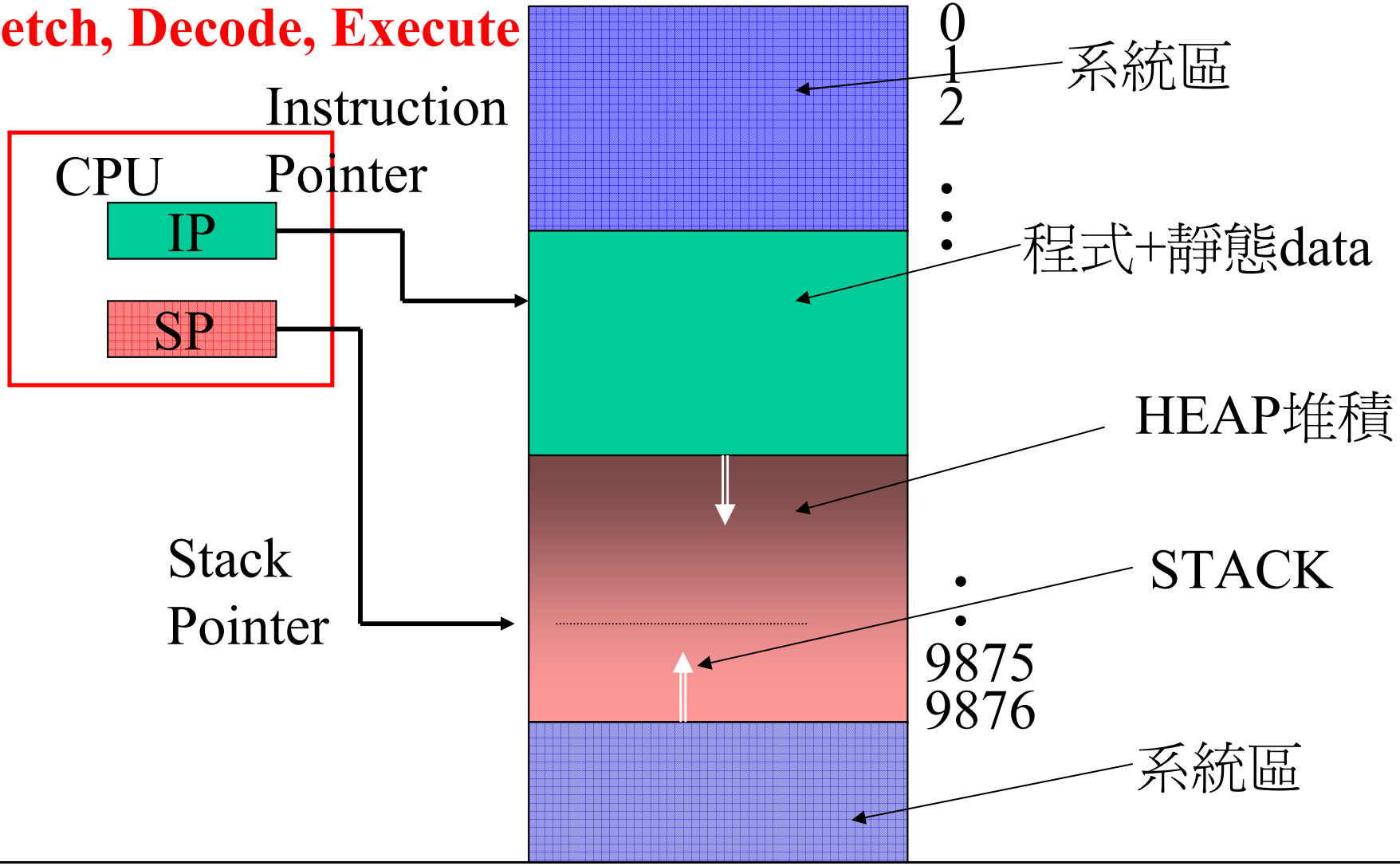


ALU 又稱 Data Path

電腦如何運作? How it works?

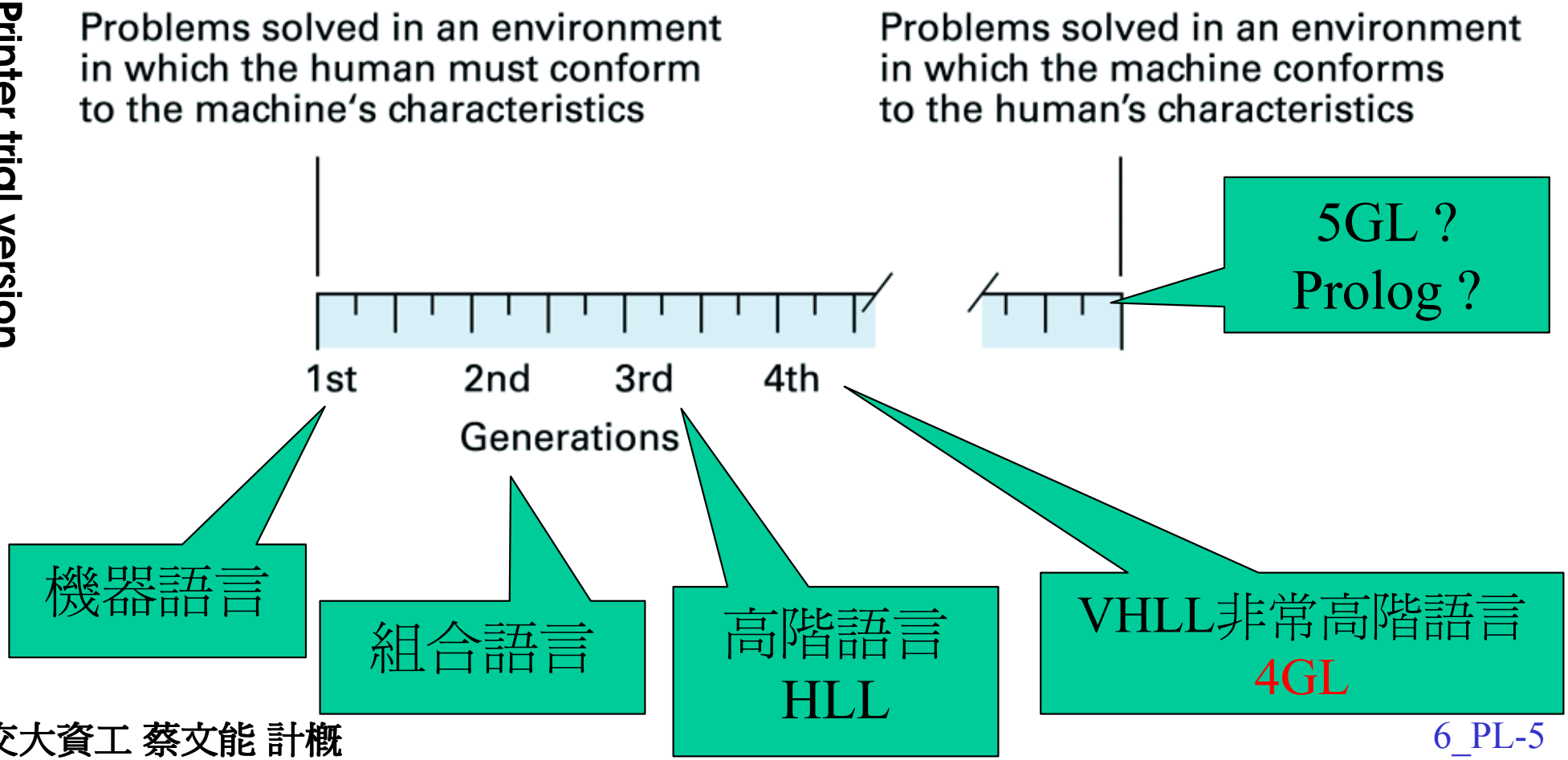
- **Auto 變數就是沒寫 static 的 Local 變數**

Fetch, Decode, Execute



Auto 變數佔用STACK區memory

Generations of programming languages



1st-generation :Machine Language

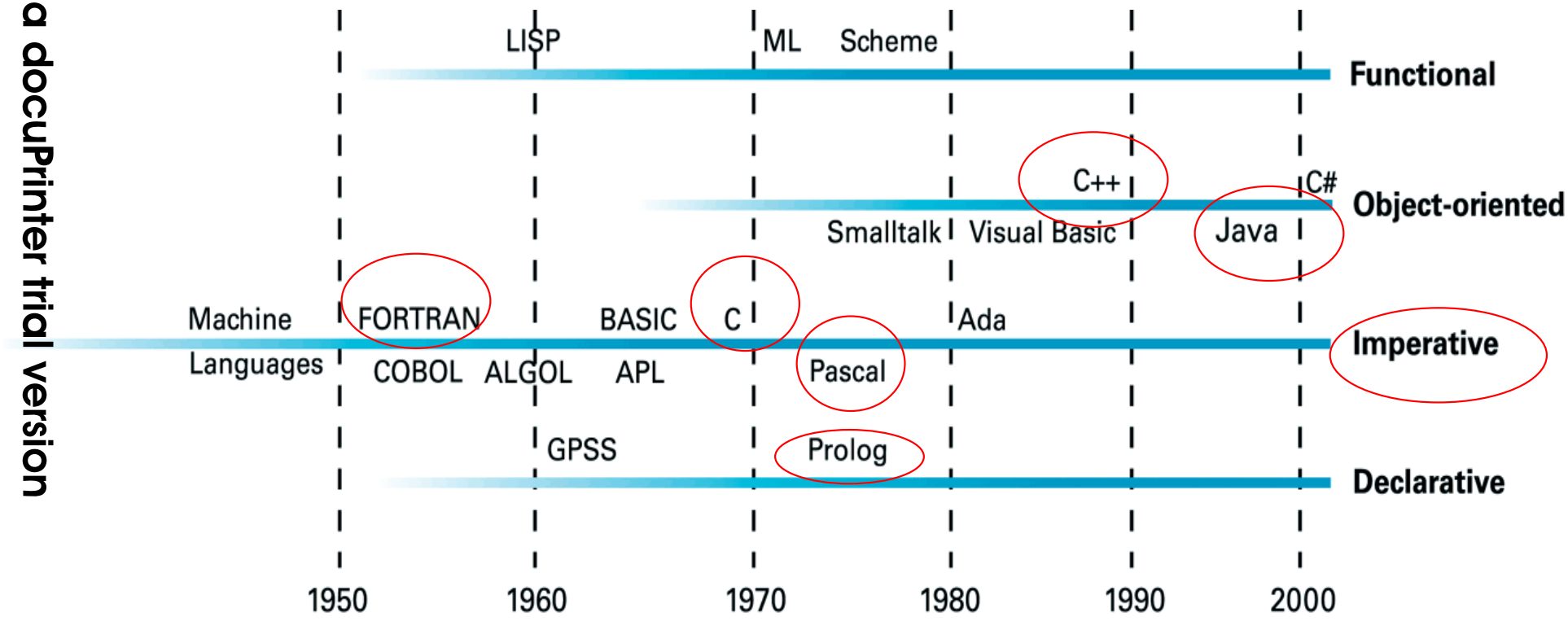
- 01010010010100100101001001010010001
- 0010
- 01001001
- 01001010 10100110
- 4a a6
- 80 20 2f
- 3a 9d 81 3e

2nd-generation : Assembly Language

- Load 100
- Loadi 100
- Load [100]
- Load R5, 100
- Load R5, [100]
- Store R5, 100
- Move AX,[100]
- Move [100],ax

- ADD AX,BX
- Sub AX,BX
- OR ax,dx
- JLT there
- JMP there
- CMP AX, 200
- JGE there
- Push AX

The evolution of programming paradigms



VB 6.0 and before: Object-Based

VB dot Net : Object Oriented

Imperative Paradigm

- Procedural paradigm
- Develops a sequence of commands that when followed, manipulate data to produce the desired result
- Approaches a problem by trying to find an algorithm for solving it

Object-Oriented Paradigm

- Grouping/classifying entities in the program
  - Entities are the objects
  - Groups are the classes
  - Objects of a class share certain properties
  - Properties are the variables or methods
- Encapsulation of data and procedures
  - Lists come with sorting functions
- Natural modular structure and program reuse
  - Inheriting from mother class definitions
- Many large-scale software systems are developed in the object oriented fashion

Declarative Paradigm

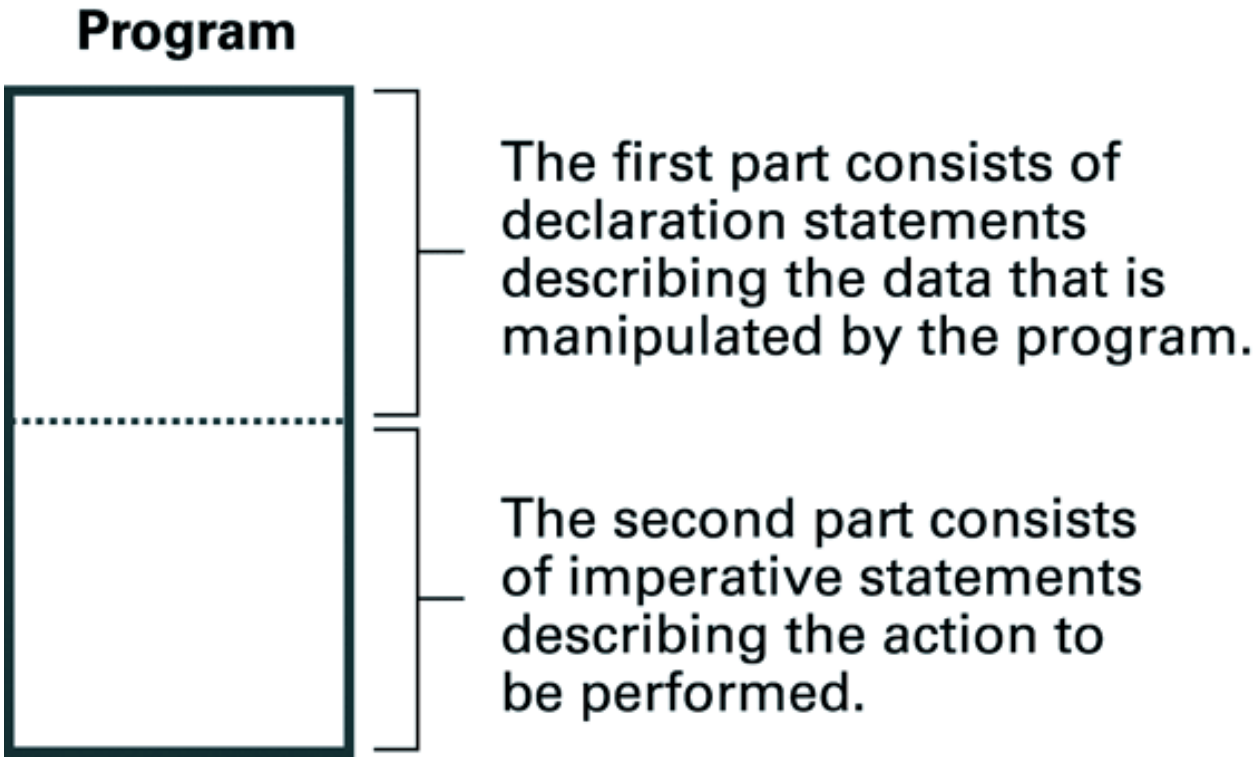
- Emphasizes
  - "What is the problem?"
  - Rather than "What algorithm is required to solve the problem?"
- Implemented a general problem-solving algorithm
- Develops a statement of the problem compatible with the algorithm and then applies the algorithm to solve it

Functional Paradigm

- Views the process of program development as connecting predefined “black boxes,” each of which accepts inputs and produces outputs
- Mathematicians refer to such “boxes” as functions
- Constructs functions as nested complexes of simpler functions

Chapter 6: Programming Languages

The composition of a typical **imperative** program or program unit



The same **variable** declarations in different languages

a. Variable declarations in Pascal

```
var
  Length, Width:      real;
  Price, Tax, Total: integer;
  Symbol:             char;
```

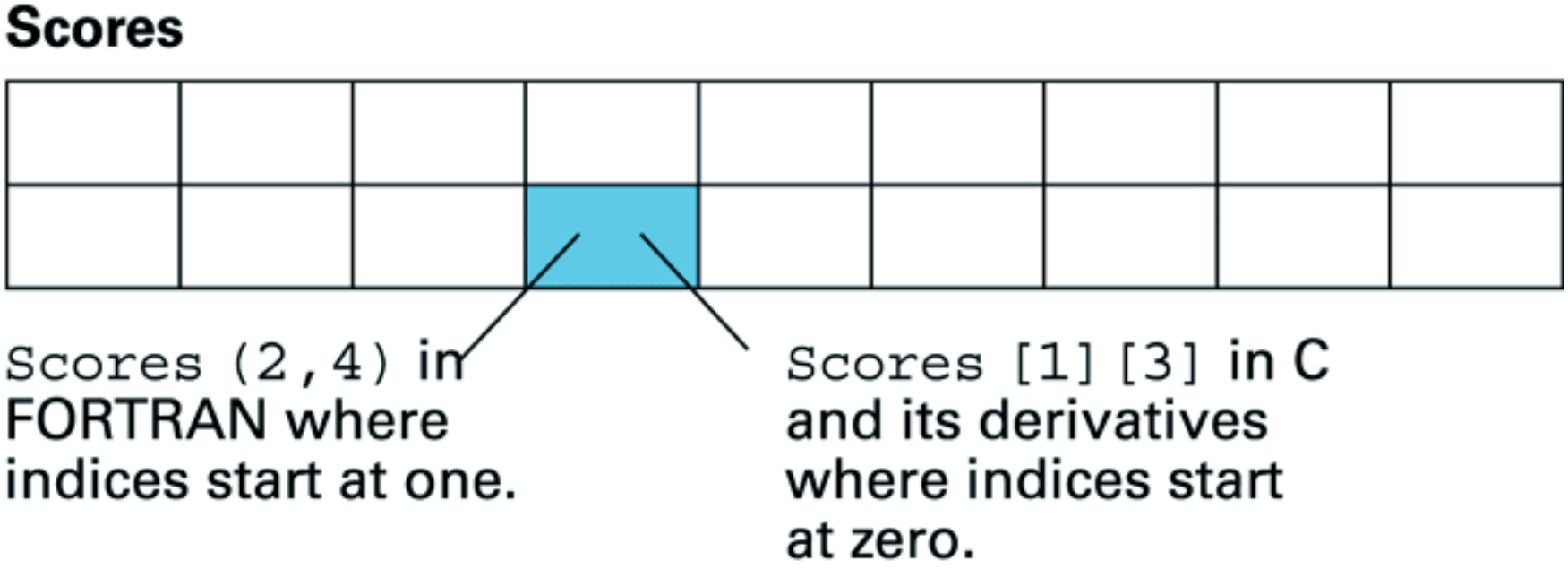
b. Variable declarations in C, C++, C#, and Java

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

c. Variable declarations in FORTRAN

```
REAL Length, Width
INTEGER Price, Tax, Total
CHARACTER Symbol
```

Figure 6.5: A two-dimensional array with two rows and nine columns



INTEGER Scores(2,9)

Column major

int Scores[2][9];

Row major

Figure 6.6a: Declaration of heterogeneous arrays in Pascal and C

異質性的

a. The array declaration in Pascal

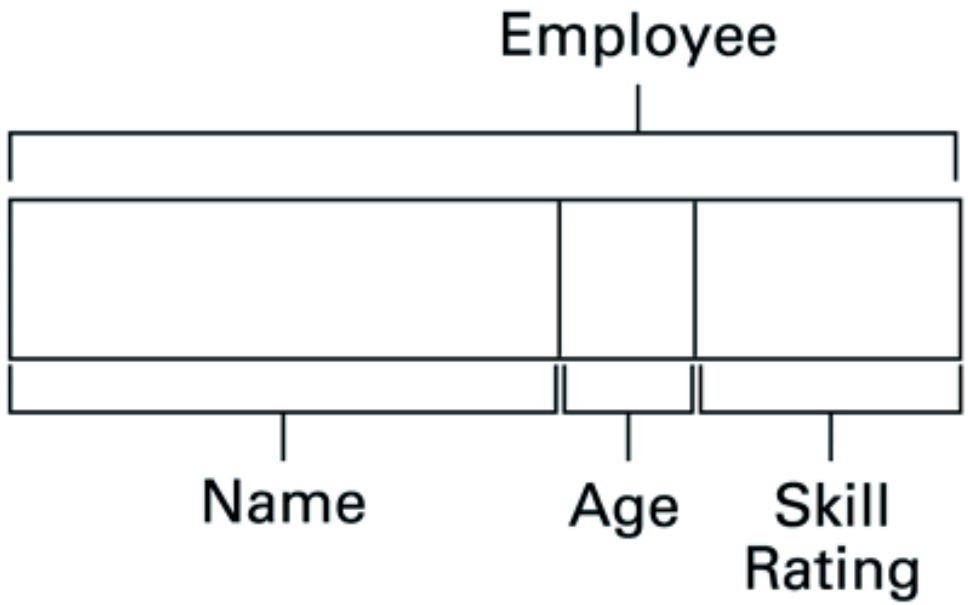
```
var
  Employee: record
              Name: packed array[1. . 8] of char;
              Age: integer;
              SkillRating: real
            end
```

b. The array declaration in C

```
struct
{ char Name [8];
  int Age;
  float SkillRating;
} Employee;
```

Figure 6.6b: Declaration of heterogeneous arrays in Pascal and C (continued)

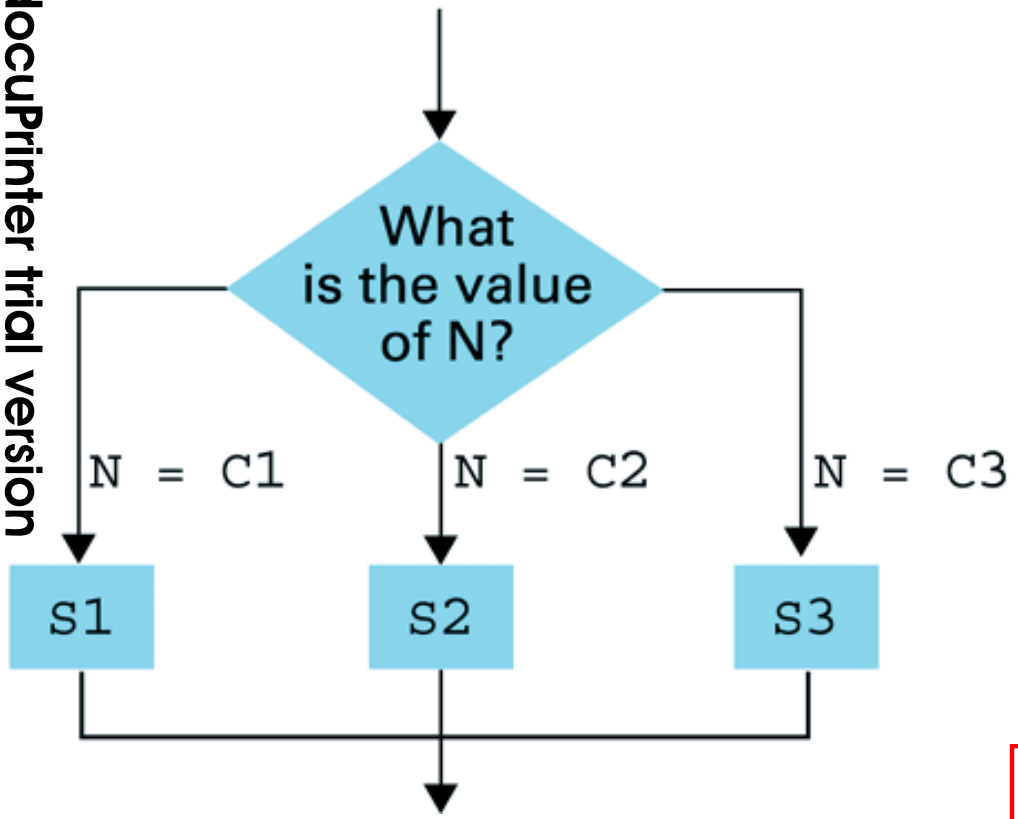
c. The conceptual organization of the array



User defined data structure

Figure 6.7a: Control structures and their representations in C, C++, C#, and Java

switch .. case



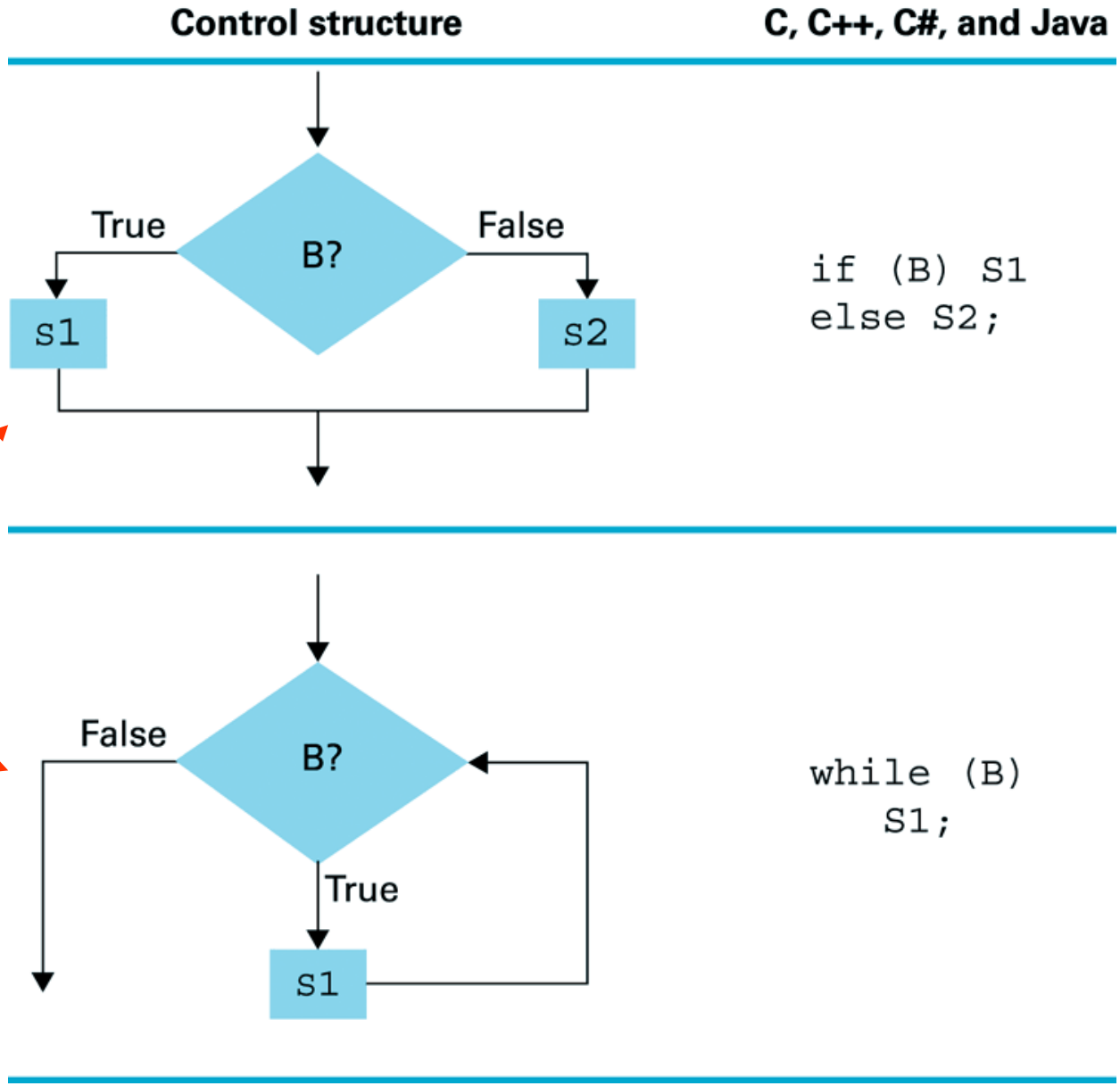

```
switch (N)
{ case C1: S1; break;
  case C2: S2; break;
  case C3: S3; break;
};
```

Falling through !

Figure 6.7b: Control structures and their representations in C, C++, C#, and Java (continued)

if-then-else

while loop



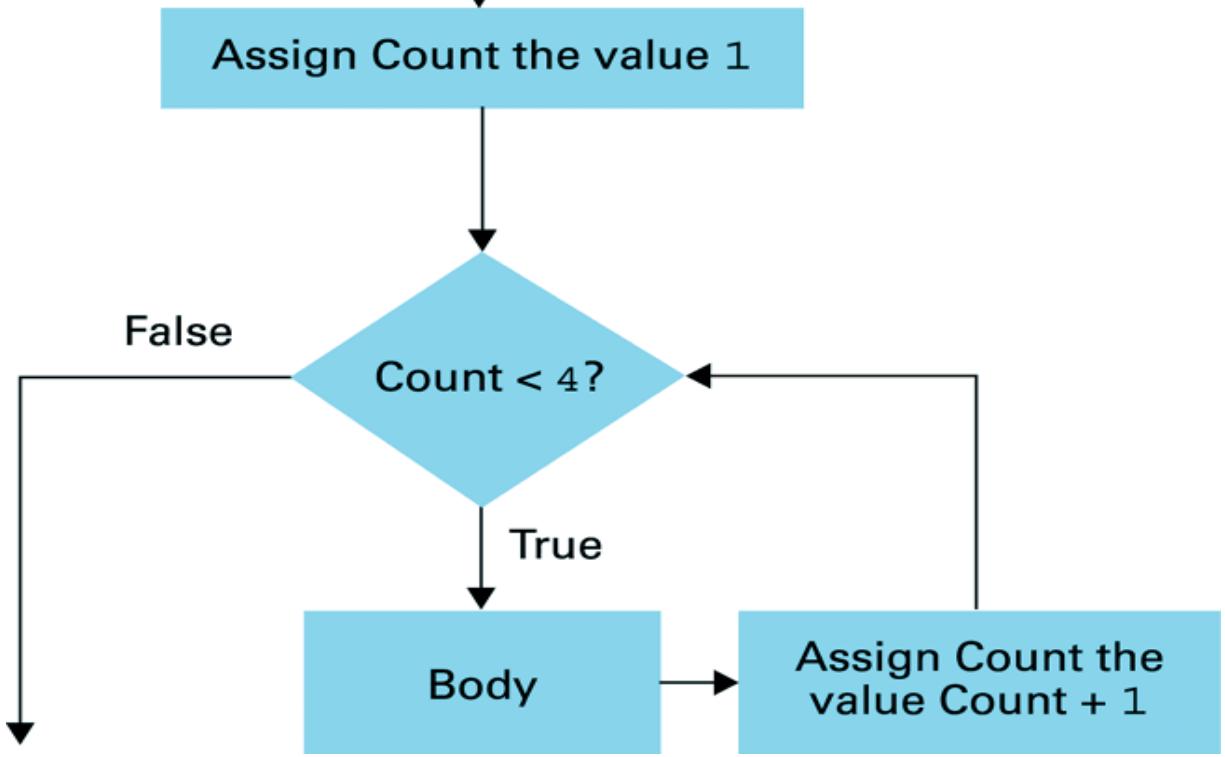
The **for loop** structure and its representation in Pascal/Delphi, C++, C#, and Java

Pascal

```
for Count := 1 to 3 do
begin
     body
end;
```

C++, C#, and Java

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



for Loop vs. while Loop

```
for(i=1 ; i<= 9 ; i++)
{
    /* Loop body */
}
```

這三個寫法意義完全一樣

```
i=1;
for( ; i<= 9 ; )
{
    /* Loop body */
    i++;
}
```

```
i=1;
while( i<= 9 )
{
    /* Loop body */
    i++;
}
```

for Loop vs. while Loop

```
for(❶ ; ❷ ; ❸) ❹;
```

```
❶;
while(❷ )
{
    ❹ ;
    ❸;
}
```

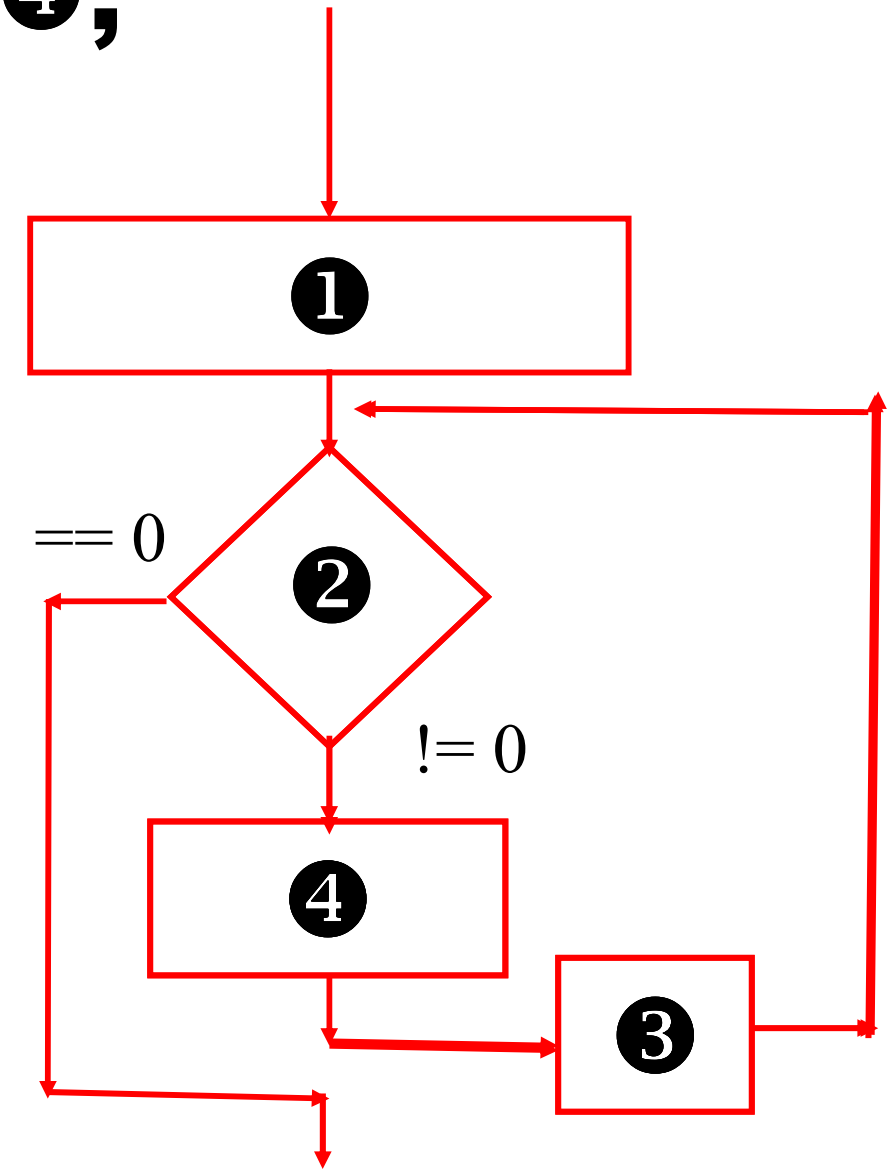
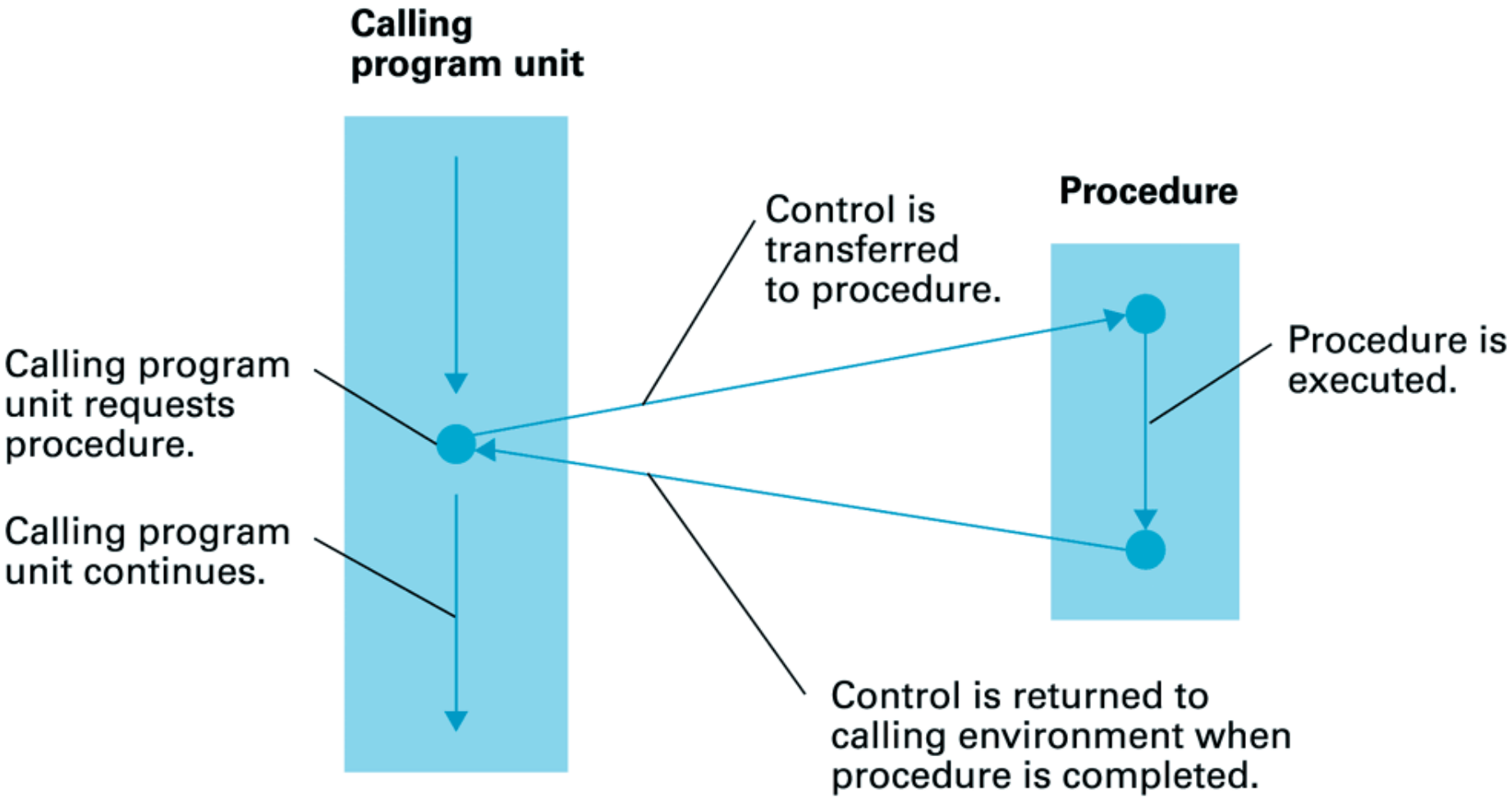


Figure 6.8 The flow of **control** involving a **procedure** or **function**



The procedure Project Population written in the programming language C

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;
```

This declares a local variable named Year.

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

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

Executing the procedure Demo and passing **parameters** by value

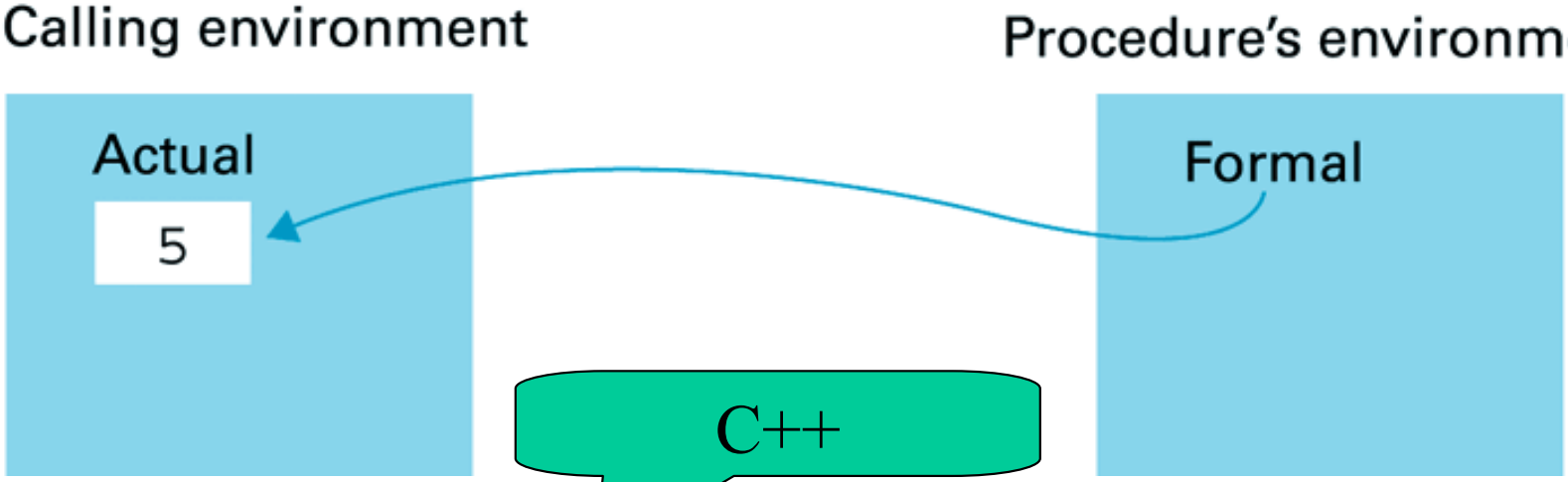
a. When the procedure is called, a copy of the data is given to the procedure

b. and the procedure manipulates its copy.

c. Thus, when the procedure has terminated, the calling environment has not been changed.

Executing the procedure Demo and passing parameters **by reference** (1/2)

a. When the procedure is called, the formal parameter becomes a reference to the actual parameter.



C++

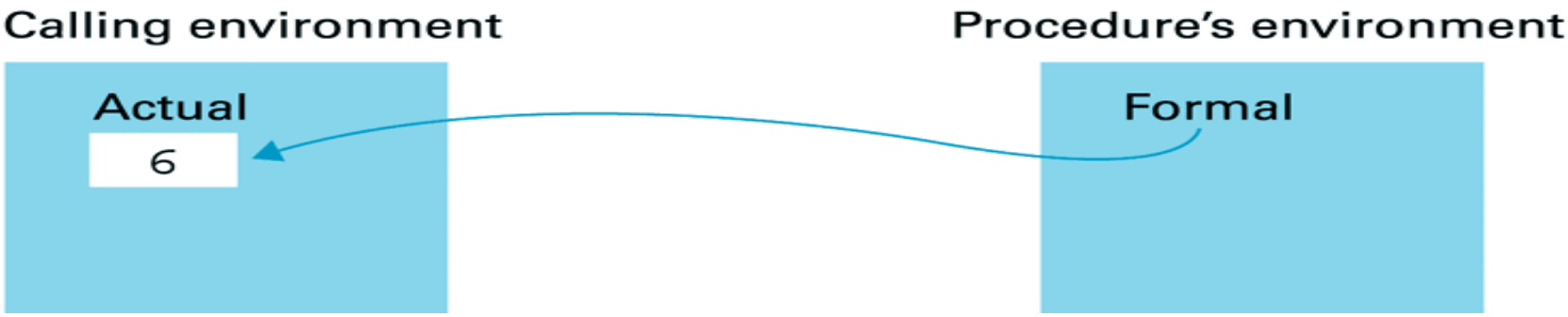
```
void swap(int& x, int& y);
```

Pascal/Delphi

```
procedure swap(var x:integer; var y:integer);
```

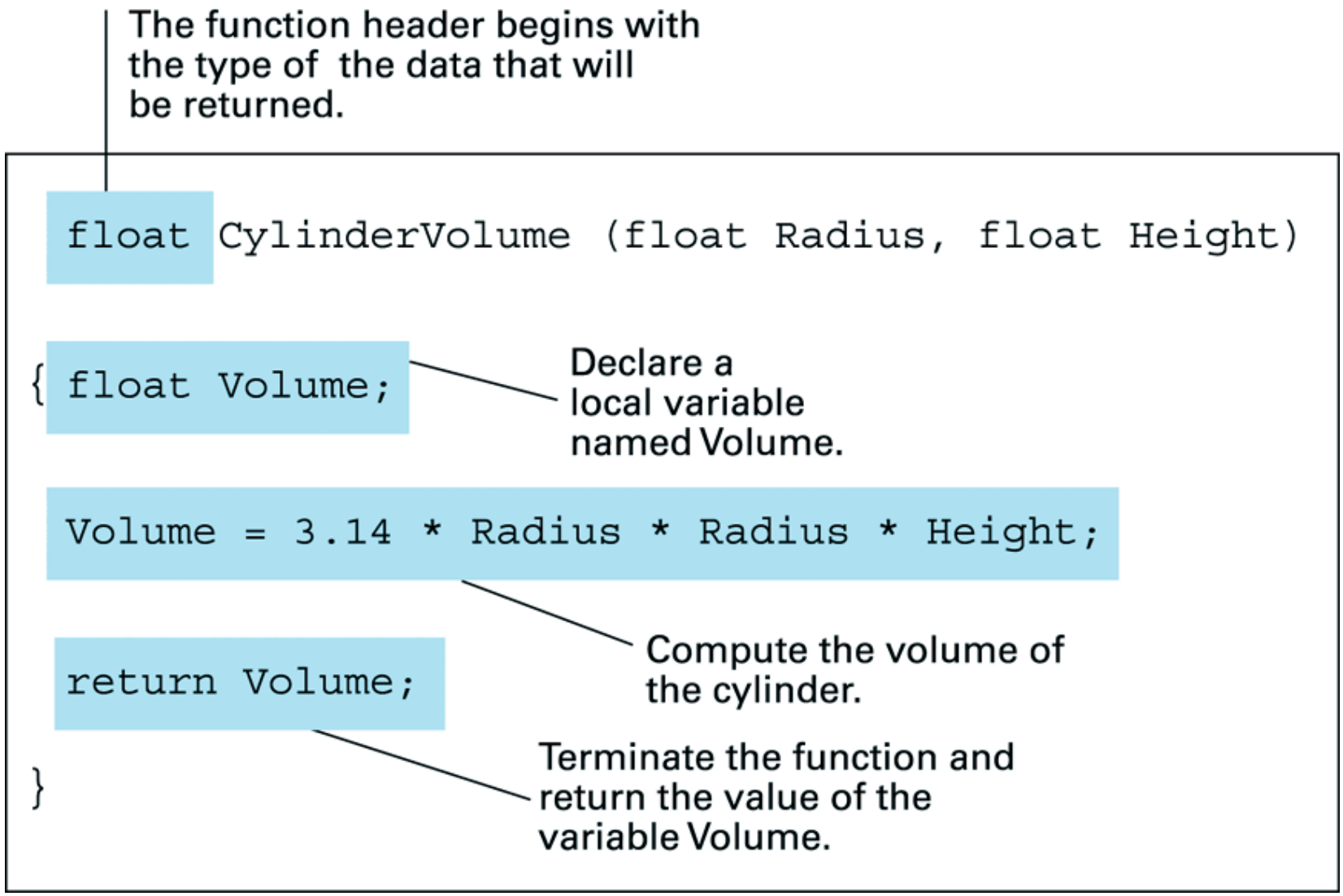
Executing the procedure Demo and passing parameters **by reference** (2/2)

b. Thus, changes directed by the procedure are made to the actual parameter

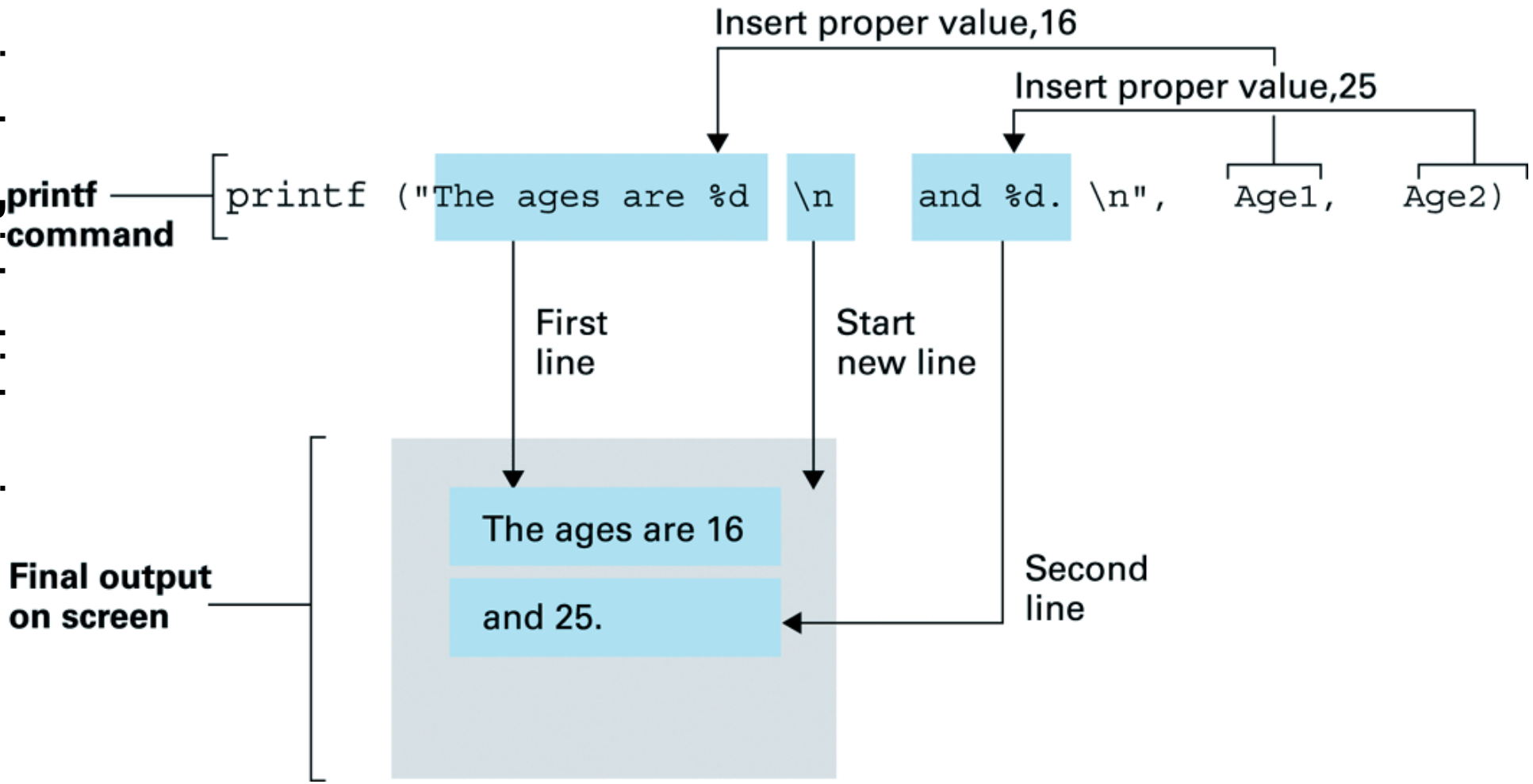


c. and are, therefore, preserved after the procedure has terminated.

The function CylinderVolume written in the programming language C

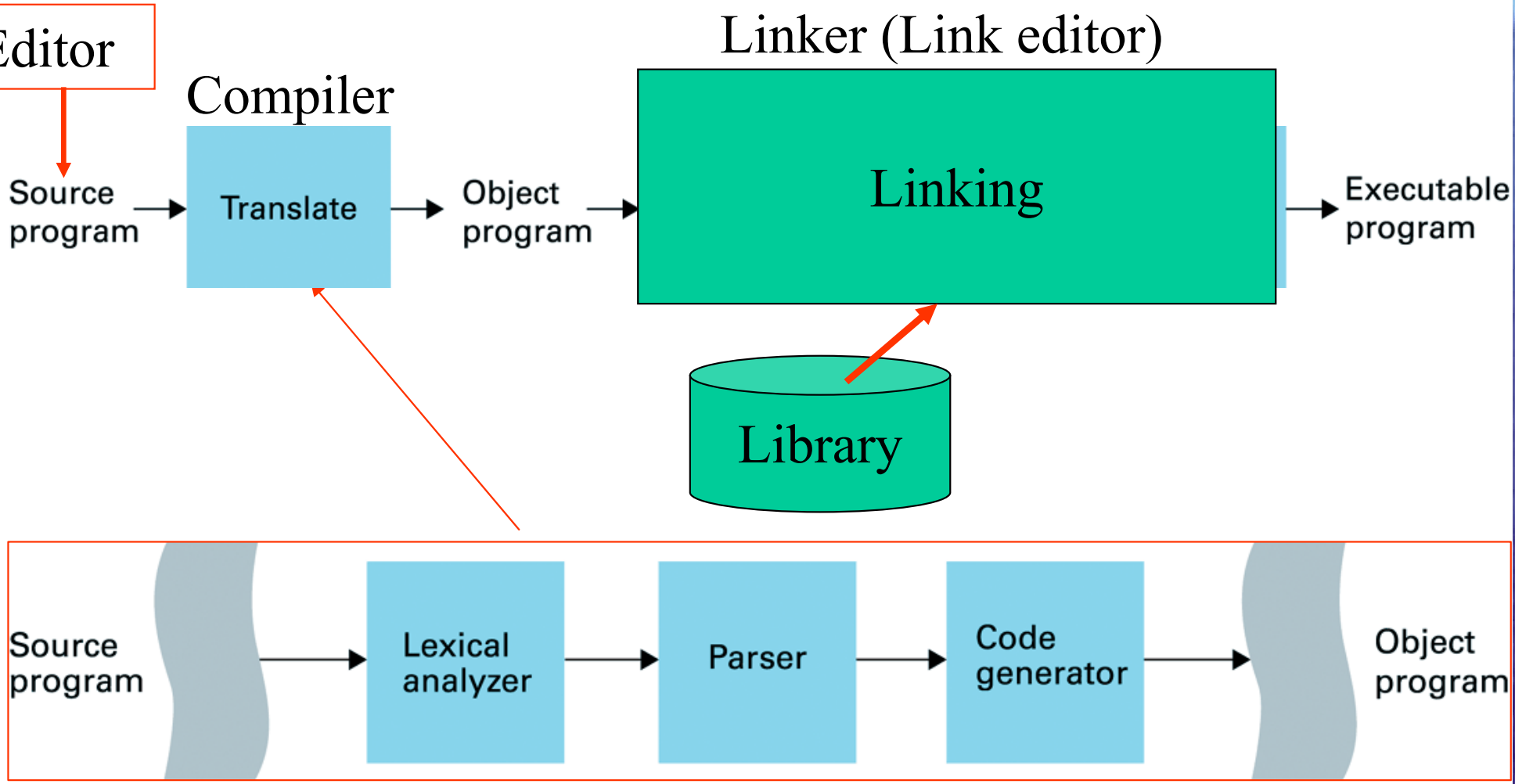


An example of formatted output in C/C++ Language



Java JDK 1.5 (5.0) and above support printf( )

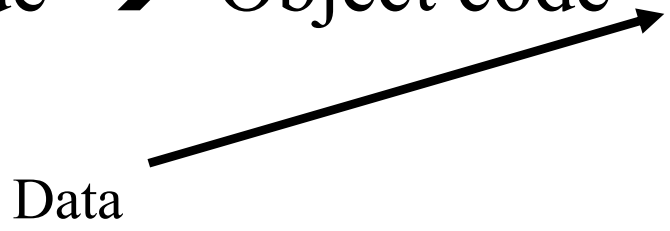
The complete program preparation process



These tasks are What the **compiler** doing.

Compiler vs. Interpreter

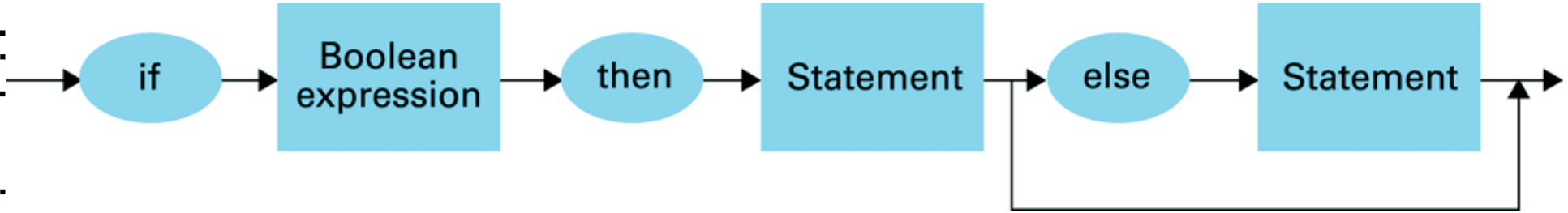
- Compiler 編譯器 (翻譯完就不管了)
  - Source code ➔ Object code
- Interpreter 解譯器 (負責做出答案或結果)
  - Source code ➔ Object code ➔ Result



A syntax diagram of the if-then-else pseudocode statement

if (boolean-expression) then statement-1

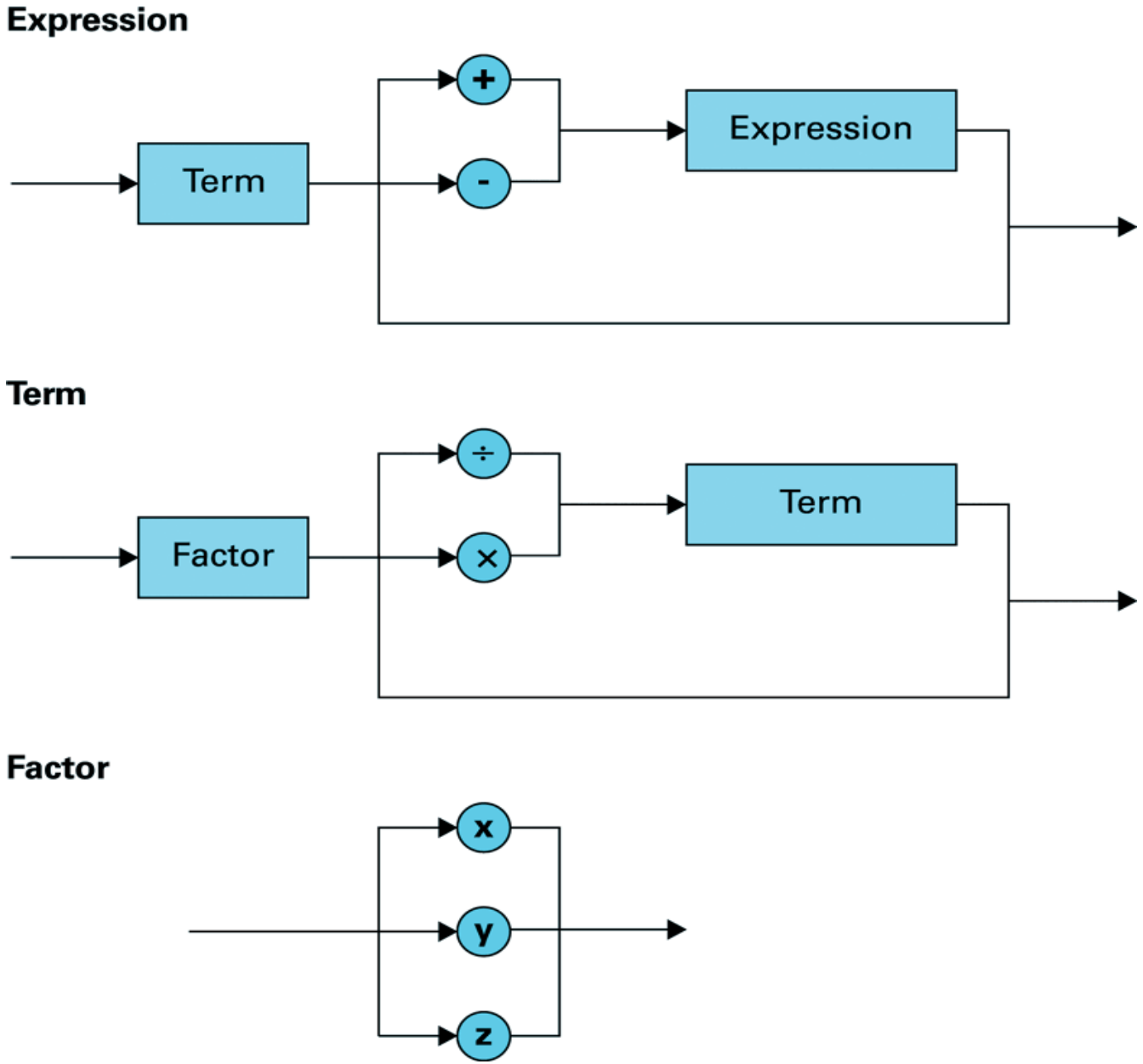
else statement-2;



Similar to the if-statement in Pascal Language

C/C++/Java 的 if statement 沒有 “then”
且 else 之前有分號 “;”

Syntax diagrams describing the structure of a simple algebraic expression

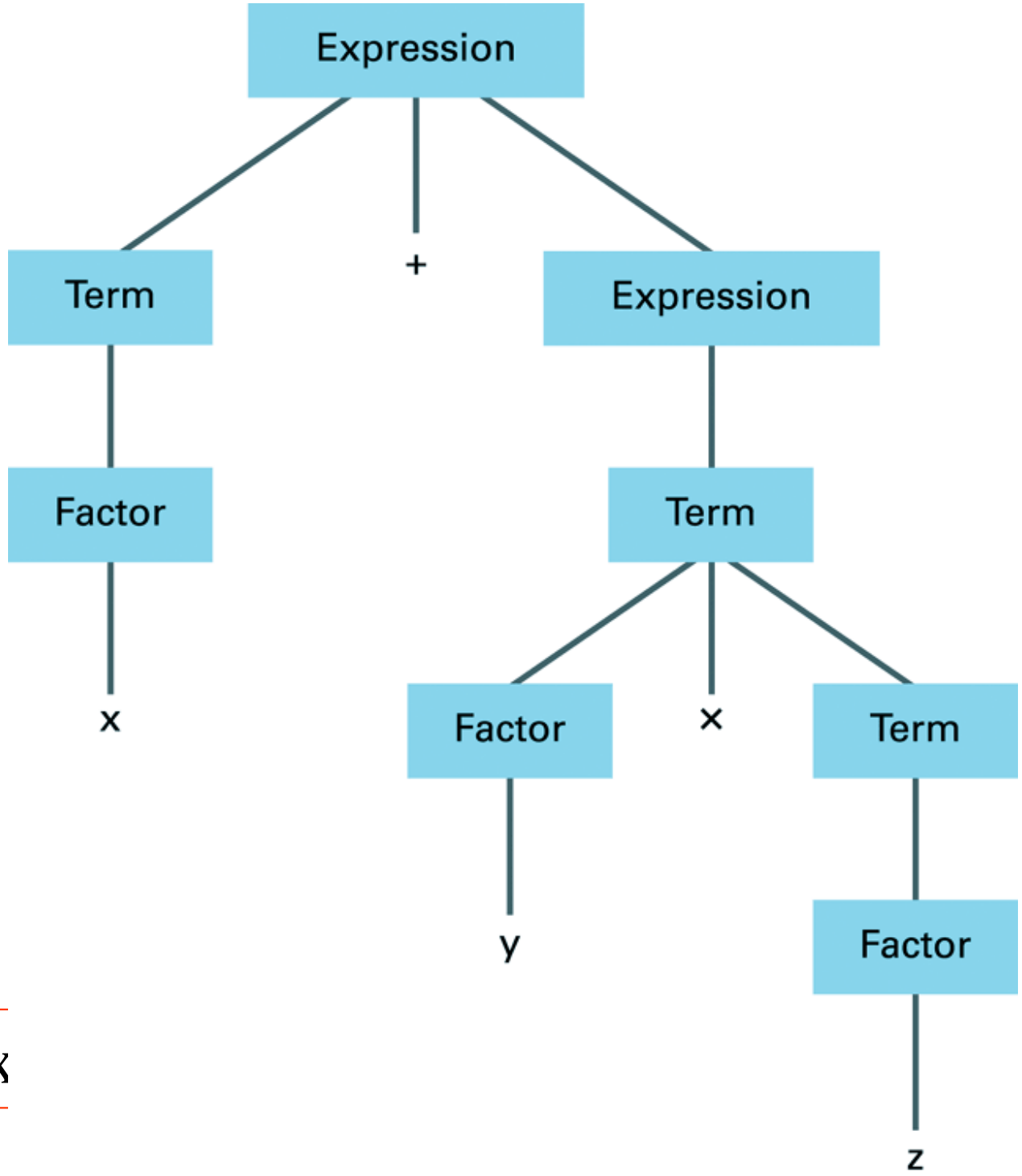


Wirth, Niklaus

(Pascal 的發明者)
發明的

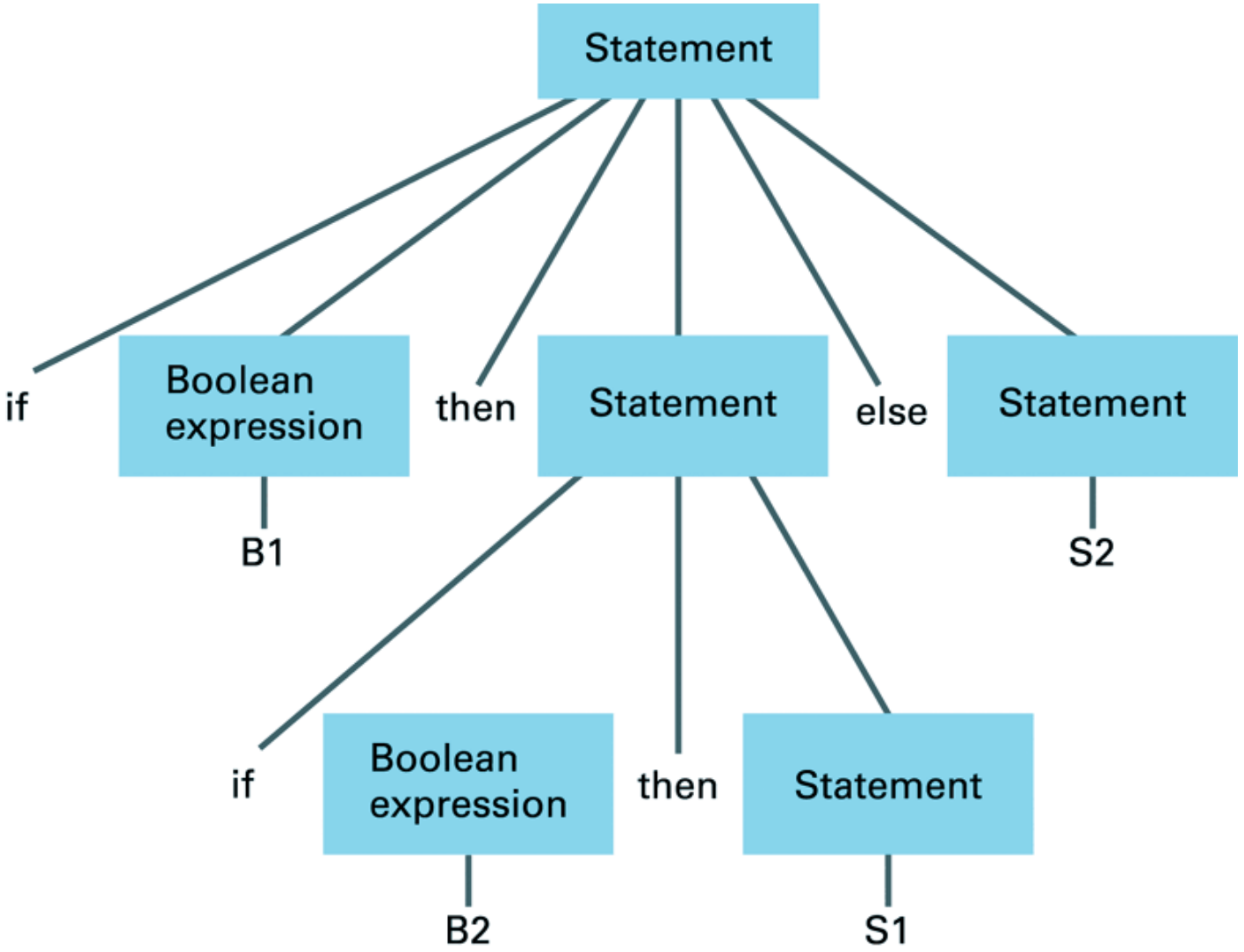
Syntax diagram is invented by Niklaus Wirth

The **parse tree** for the string x + y * z based on the syntax diagrams in previous Figure

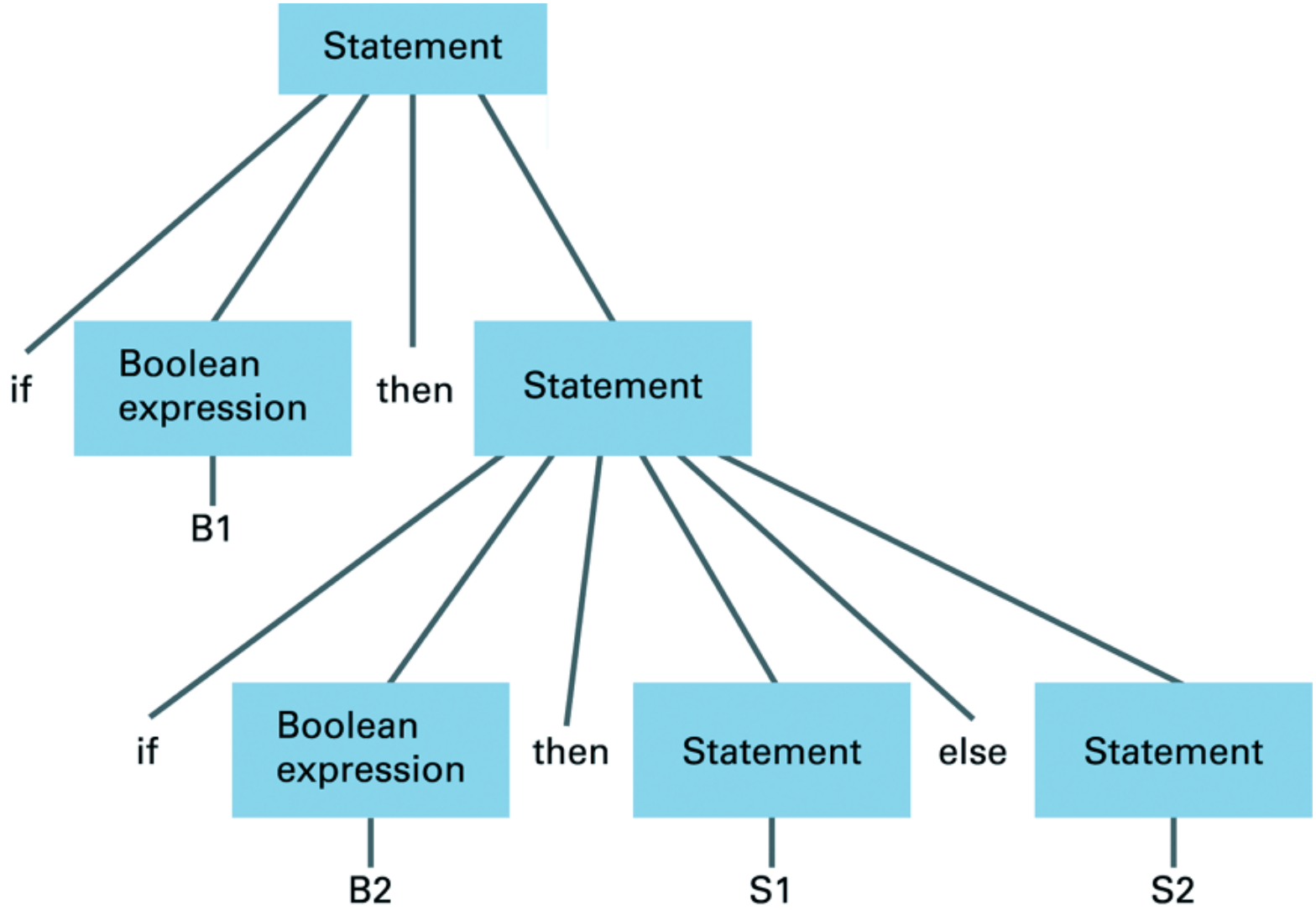


parse tree != Ex

Two distinct parse trees for the statement **if B1 then if B2 then S1 else S2** (1/2)



Two distinct parse trees for the statement **if B1 then if B2 then S1 else S2** (2/2)



Chapter 6: Programming Languages

Objects and Classes

- **Object**
  - Active program unit containing both data and procedures (methods or functions)
- **Class**
  - A template for all objects of the same type

An **Object** is often called an **instance** of the class.

Object 就是以前的變數; class 是一種 type

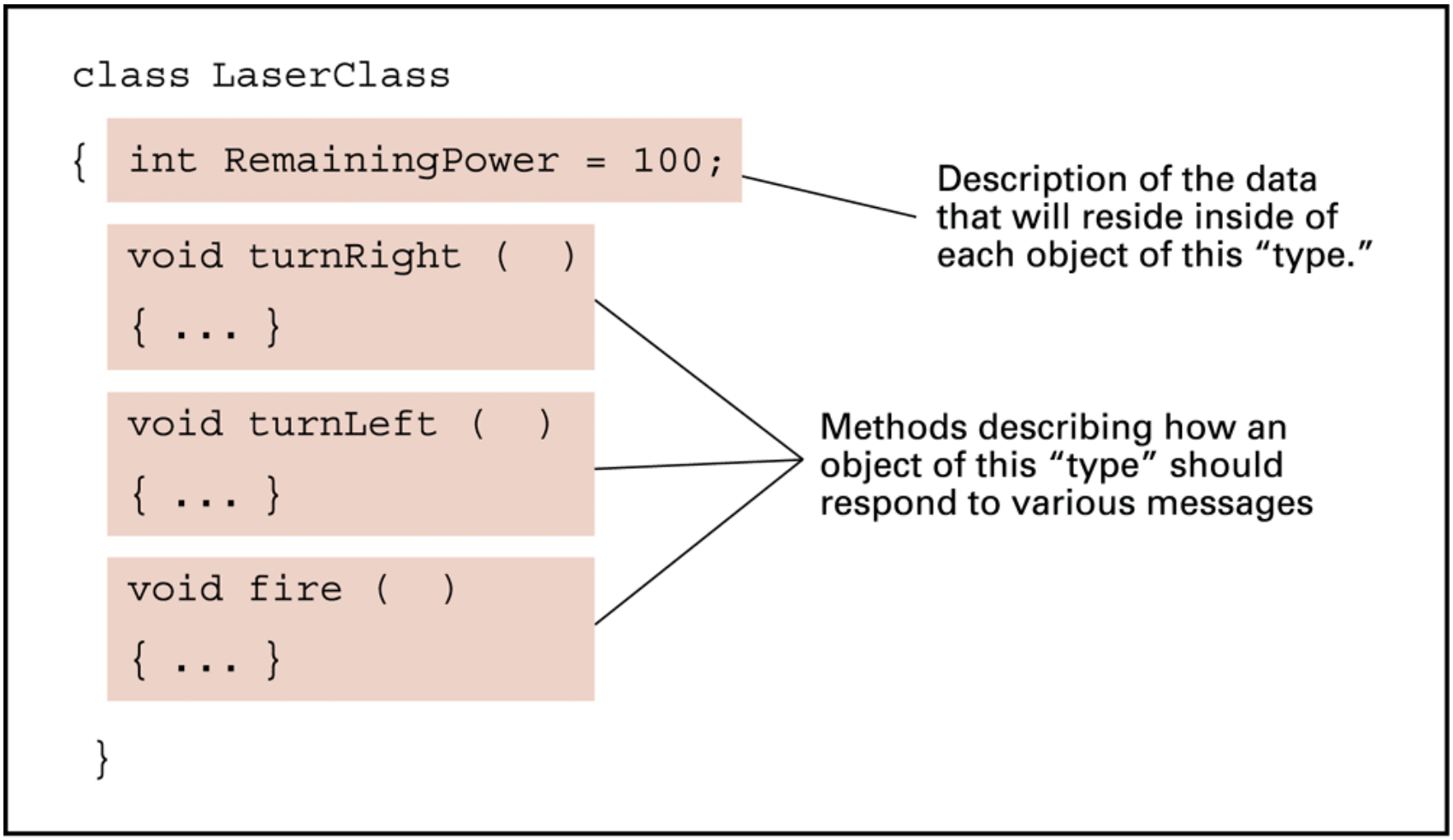
Components of an object

- **Instance variable (attribute)**
  - Variable within an object
- **Method**
  - **Function** or procedure within an object
  - Can manipulate the object's instance variables
- **Constructor**
  - Special method to initialize a new object instance

Class is an ADT

ADT : Abstract Data Type

Class Example



Constructor Example

```
class  LaserClass
{ int RemainingPower;

{ LaserClass (InitialPower)
  { RemainingPower = InitialPower;
  }

  void turnRight (  )
  { ... }

  void turnLeft (  )
  { ... }

  void fire (  )
  { ... }

}
```

Constructor assigns a value to Remaining Power when an object is created.

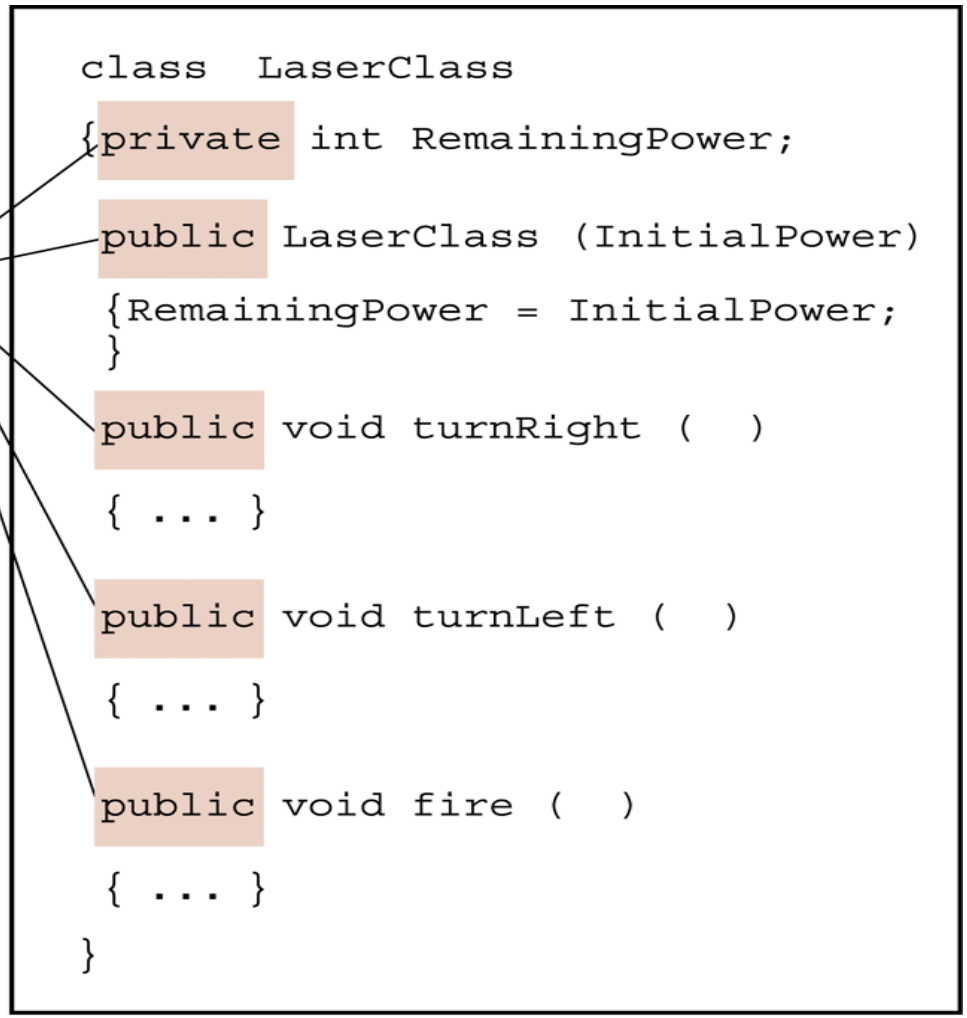
Encapsulation 封藏

- **Encapsulation**
  - A way of restricting access to the internal components of an object
  - **Private**
  - **Public**
  - **Protected**
  - Package available (default access in Java)

Encapsulation Example

The structure of a class describing a laser weapon in a computer game

Components in the class are designated public or private depending on whether they should be accessible from other program units.



```
class  LaserClass
{private int RemainingPower;
 public LaserClass (InitialPower)
 {RemainingPower = InitialPower;
 }
 public void turnRight (  )
 { ... }
 public void turnLeft (  )
 { ... }
 public void fire (  )
 { ... }
}
```

LaserClass definition using encapsulation as it would appear in a Java or C# program

Components in the class are designated public or private depending on whether they should be accessible from other program units.

```
class  LaserClass
{private int RemainingPower;

public LaserClass (InitialPower)
 {RemainingPower = InitialPower;
 }

public void turnRight (  )
 { ... }

public void turnLeft (  )
 { ... }

public void fire (  )
 { ... }
}
```

Additional Concepts

- Inheritance 繼承 (extends 擴充)
  - Allows new classes to be defined in terms of previously defined classes
- Polymorphism 多型; 同名異式
  - Allows method calls to be interpreted by the object that receives the call dynamically

Function name Overloading : static binding

Polymorphism : dynamic binding

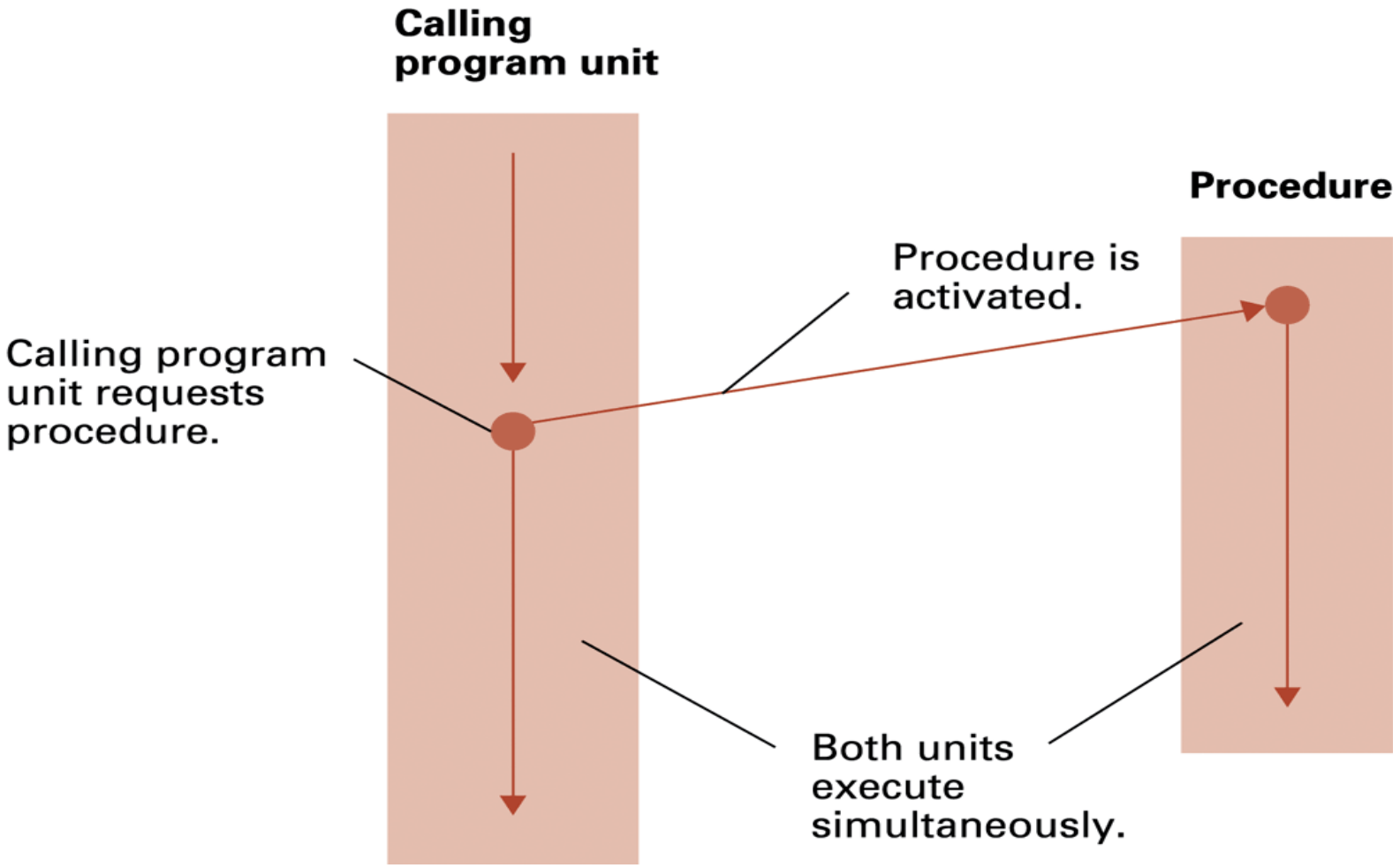
Chapter 6: Programming Languages

Program Concurrent Activities

- **Parallel** or **concurrent** processing
- Simultaneous execution of multiple processes
- True concurrent processing requires multiple CPUs (true concurrent means Simultaneous)
- Can be simulated using time-sharing with a single CPU
- Examples: Ada task and **Java thread**

Concurrent != Simultaneous

Parallel Processing



Basic Idea

- Creating new process (or thread)
- Handling communication between processes

- Mutually exclusive access over **critical regions**
  - Data accessed by only one process at a time
- Monitor
  - A data item augmented with the ability to control access to itself

- (以後在 Java 課討論)

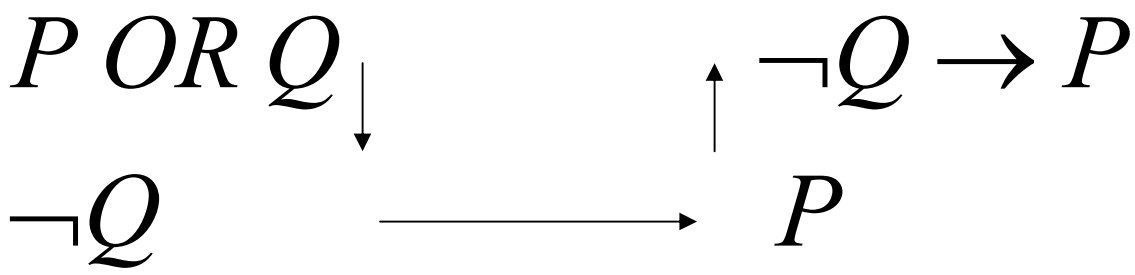
Declarative Programming

- **Resolution**
  - Combining two or more statements to produce a new, logically equivalent statement
- **Resolvent**
  - a new statement deduced by resolution

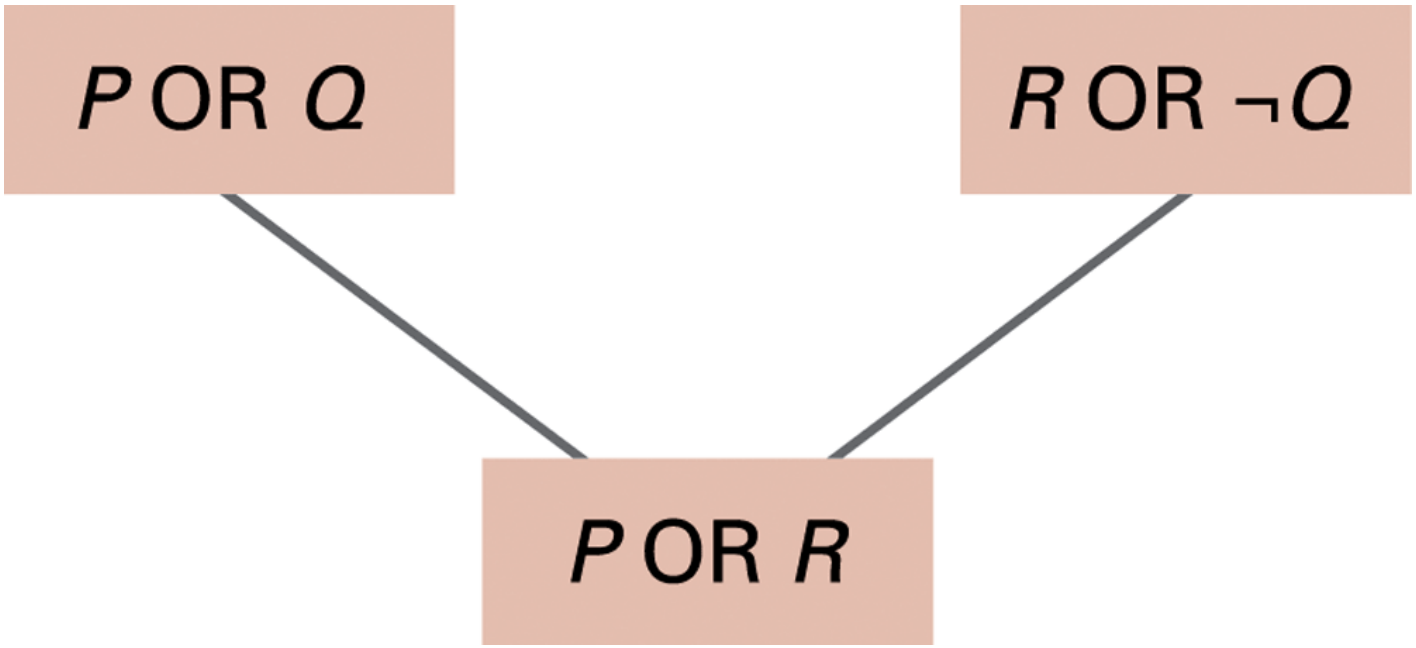
- Example: (P OR Q) AND (R OR ¬Q)
  resolves to (P OR R)

Logical Deduction

- Either Kermit is on stage (Q) or Kermit is sick (P)
- Kermit is not on stage (not Q)
- Kermit is sick (P)
- Resolution Principle



Resolving (P OR Q) and (R OR ¬ Q) to Produce (P OR R)



Truth Table

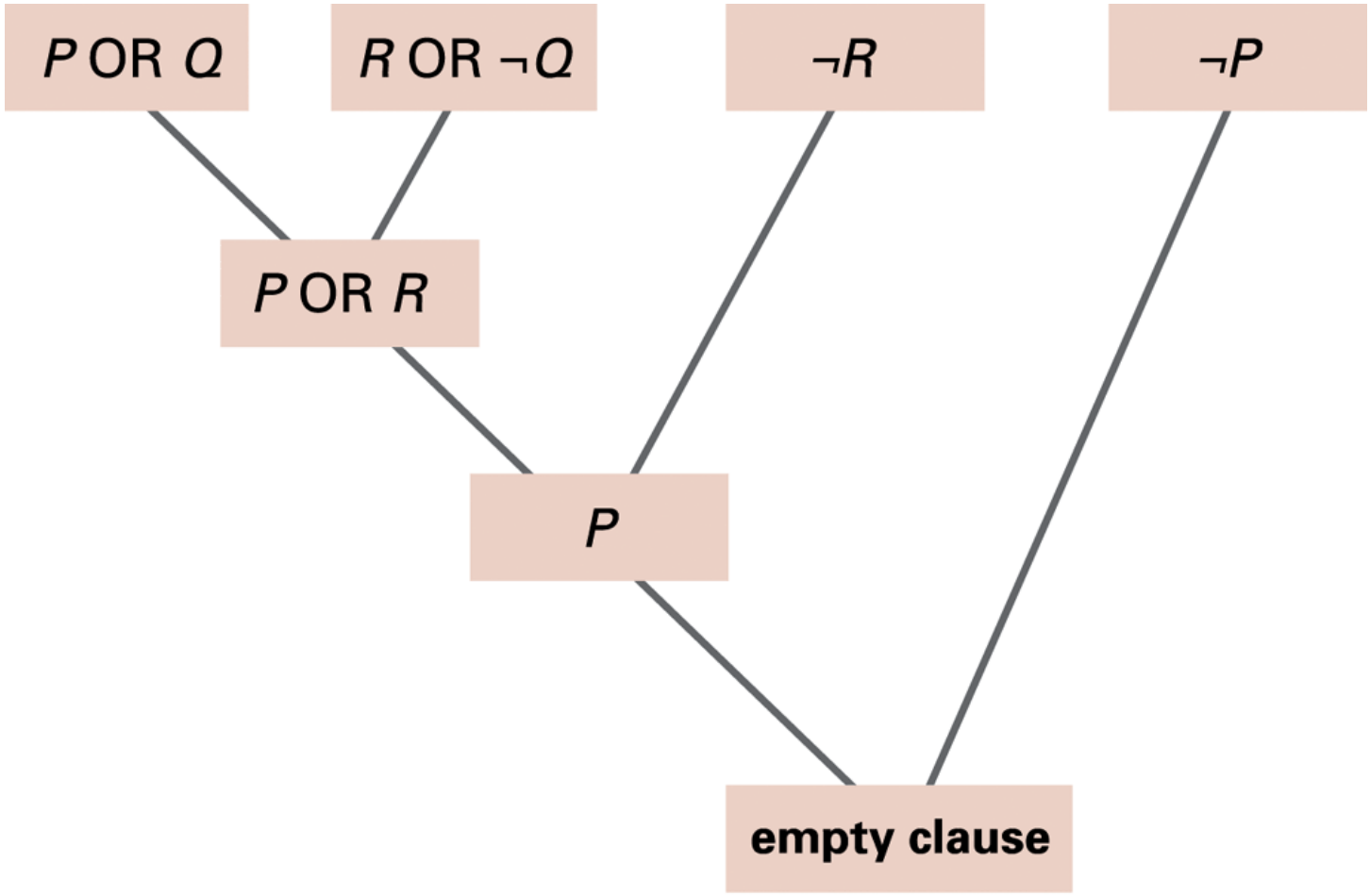
P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

Clause Form

- **First-order predicate logic** vs. Higher-order predicate logic
- Clause form

$$(P_1\ OR\ Q_1)\ AND\ (P_2\ OR\ Q_2)\ AND\cdots AND\ (P_N\ OR\ Q_N)$$

Confirming the Inconsistency of a Set of Inconsistent Clauses



Interpretation

- P: Kermit is on stage
- Q: Kermit has got SARS for two weeks
- R: Kermit coughs

$$P\ OR\ Q \equiv \neg Q \rightarrow P$$

$$R\ OR\ \neg Q \equiv \neg R \rightarrow \neg Q$$

$$P\ OR\ R \equiv \neg R \rightarrow P$$

Automatic Reasoning using reduction ad absurdum

- Want to confirm a collection of statements implies the statement P
- Same as contradicting the statement $\neg P$
- Apply resolution to the original statements and the statement $\neg P$ until an empty clause occurs
- With $\neg P$ inconsistent with the original statements, the original statements must imply P

Unification

- The process of assigning values to variables so that resolution can be performed

$$(Mary\ is\ at\ X) \rightarrow (Mary's\ lamb\ is\ at\ X)$$

$$Mary\ is\ at\ home$$

$$\neg(Mary\ is\ at\ X) OR (Mary's\ lamb\ is\ at\ X)$$

$$(Mary\ is\ at\ home)$$

$$\neg(Mary\ is\ at\ home) OR (Mary's\ lamb\ is\ at\ home)$$

$$(Mary\ is\ at\ home)$$

$$(Mary's\ lamb\ \ is\ \ at\ home)$$

Prolog

- **PRO**gramming in **LOG**ic
- A Prolog program consists of a collection of initial statements upon which the underlying algorithm bases its deductive reasoning

Prolog Syntax

- **Fact**
  - *predicateName*(*arguments*).
  - Example: `parent(bill, mary).`
- **Rule**
  - *conclusion* :- *premise.*
  - :- means “if”
  - Example: `wise(X) :- old(X).`
  - Example: `faster(X,Z) :- faster(X,Y), faster(Y,Z).`
- All statements must be fact or rules.

Sample Prolog program (1/2)

```
bsd2% cat hom.pl
:-dynamic loves/2.
loves(kmy,mary).
loves(mary,kmy).
loves(chang3,mary).
samesex(X,Y):- male(X), male(Y).
samesex(X,Y):- female(X), female(Y).
male(chang3).
female(mary).
female(kmy).
hom(X,Y):- loves(X,Y), loves(Y,X), samesex(X,Y).
shuey(X) :- loves(X,Y), hom(Y,Z), not(Z=X).
bsd2%
```

Sample Prolog program (2/2)

```
bsd2% pl
Welcome to SWI-Prolog …
1 ?- [hom].
hom compiled, 0.00 sec, 1,672 bytes.
Yes
2 ?- listing.
  …
3 ?- hom(kmy,mary).
Yes
4 ?- shuey(chang3).
Yes
5 ?- listing(loves).
loves(kmy, mary).
loves(mary, kmy).
loves(chang3, mary).
Yes
6 ?-   assert(loves(lee4,kmy)).
```

A GCD program in Prolog

```
bsd2% cat -n gcd.pl
     1  gcd(A,0,A).
     2  gcd(A,B,D):-(A>B), (B>0),
     3              R is A mod B, gcd(B,R,D).
     4  gcd(A,B,D):- A<B, gcd(B,A,D).
     5  run:- write('first number='),read(X),
     6        write('second number='), read(Y),
     7        gcd(X,Y,ANS), write('The gcd of '),
     8         write(X), write(' and '), print(Y),
     9          put(' '),print(is), put(' '), write(ANS).
bsd2 % pl
Welcome to SWI-Prolog (  …
1 ?- [gcd].
```

Run the GCD prolog program

```
2 ?- gcd(4,6,ANS).
ANS = 2
Yes
3 ?- run.
first number=6.
second number=9.
The gcd of 6 and 9 is 3
Yes
4 ?- listing.
 …
```

A Factorial(N) program in Prolog

```
bsd2% cat –n factn.pl
     1  fact_aux(N, FactN, I, P) :-
     2     I =< N, !,
     3     NewP is P * I,
     4     NewI is  I + 1,
     5     fact_aux(N, FactN, NewI, NewP).
     6  fact_aux(N, FactN, I, FactN) :-
     7     I > N.
     8  fact(N, FactN) :-
     9     fact_aux(N, FactN, 1, 1).
    10   run:- write('Give me a positive number: '),read(X),
    11         fact(X,ANS),
    12         write('The Factorial of '),
    13         write(X), write(' is '),
    14         write(ANS).
bsd2%
```

Run the Factorial(N) Prolog program

```
bsd2 % pl
Welcome to SWI-Prolog (  …
1 ?- [factn].
Yes
2 ?- run.
Give me a positive number: 5.
The Factorial of 5 is 120
Yes
3 ?- run.
Give me a positive number: 6.
The Factorial of 6 is 720
Yes
4 ?- listing(fact).
```

Thank You!

Programming Languages

謝謝捧場

tsaiwn@csie.nctu.edu.tw

蔡文能