C ++ Programming

2023 Spring; week 2

Instructor: Cheng-Chun Chang (張正春)

Department of Electrical Engineering

Textbook: P. Deitel and H. Deitel, C HOW TO PROGRAM 8/E, PEARSON, 2016

課程助教

第一排:
第二排:
第三排:
第四排:
第五排:
第六排:

打游擊:

© 2016 Pearson Education, Ltd. All rights reserved.

Programming: it's all about format …analogous to 唐詩宋詞

□低頭吃便當….

□處處蚊子咬….

KEY: 1) 用法 2) 用法 3)用法, and then you can modify it.

Chapter 15
C++ as a Better C; Introducing Object Technology

C How to Program, 8/e, GE

© 2016 Pearson Education, Ltd. All rights reserved.

15.1 Introduction

- The first 14 chapters presented a thorough treatment of procedural programming and top-down program design with C.
- The C++ section introduces two additional programming paradigms—object-oriented programming (with classes, encapsulation, objects, operator overloading, inheritance and polymorphism 類別、封裝、物件、運算子多載、繼承和多型) and generic programming **泛型程式設計** (with function templates and class templates).
- These chapters emphasize “crafting valuable classes” to create reusable software componentry.

© 2016 Pearson Education, Ltd. All rights reserved.

15.2 C++

- C++ improves on many of C's features and provides object-oriented-programming (OOP) capabilities that increase software productivity, quality and reusability.
- When a programming language becomes as entrenched as C, new requirements demand that the language evolve rather than simply be displaced by a new language.
- C++ was developed by Bjarne Stroustrup at Bell Laboratories and was originally called "C with classes."
- The name C++ includes C's increment operator (++) to indicate that C++ is an enhanced version of C.

© 2016 Pearson Education, Ltd. All rights reserved.

15.2 C++ (Cont.)

- The remaining provide an introduction to the version of C++ standardized in the United States through the American National Standards Institute (ANSI) and worldwide through the International Standards Organization (ISO).

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers

- This section revisits the addition program of Fig. 2.5 and illustrates several important features of the C++ language as well as some differences between C and C++.
- C file names have the `.c` (lowercase) extension.
- C++ file names can have one of several extensions, such as `.cpp`, `.cxx` or `.C` (uppercase).
- We use the extension `.cpp`.
- Figure 15.1 uses C++-style input and output to obtain two integers typed by a user at the keyboard, computes the sum of these values and outputs the result.
- Lines 1 and 2 each begin with //, indicating that the remainder of each line is a comment.

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- C++ allows you to begin a comment with // and use the remainder of the line as comment text.
- A // comment is a maximum of one line long.
- C++ programmers may also use /*...*/ C-style comments, which can be more than one line long.

© 2016 Pearson Education, Ltd. All rights reserved.

Programming codes 1

```
1  // Fig. 15.1: fig15_01.cpp
2  // Addition program that displays the sum of two numbers.
3  #include <iostream> // allows program to perform input and output
4
5  int main()
6  {
7     std::cout << "Enter first integer: "; // prompt user for data
8     int number1;
9     std::cin >> number1; // read first integer from user into number1
10
11    std::cout << "Enter second integer: "; // prompt user for data
12    int number2;
13    std::cin >> number2; // read second integer from user into number2
14    int sum = number1 + number2; // add the numbers; store result in sum
15    std::cout << "Sum is " << sum << std::endl; // display sum; end line
16 }
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Fig. 15.1 | Addition program that displays the sum of two numbers.

© 2016 Pearson Education, Ltd. All rights reserved.

Programming code 1

```
#include <iostream>

int main()
{
	std::cout << "Enter first integer: ";
	int number1;
	std::cin >> number1;

	std::cout << "Enter second integer: ";
	int number2;
	std::cin >> number2;
	int sum = number1 + number2;
	std::cout << "Sum is " << sum << std::endl;

}
```

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- The C++ preprocessor directive in line 3 exhibits the standard C++ style for including header files from the standard library.
- This line tells the C++ preprocessor to include the contents of the input/output stream header file `<iostream>`.
- This file must be included for any program that outputs data to the screen or inputs data from the keyboard using C++-style stream input/output.
- We discuss `iostream`'s many features in detail in Chapter 2, Stream Input/Output.
- As in C, every C++ program begins execution with function `main` (line 5).

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- Keyword `int` to the left of main indicates that main returns an integer value.
- C++ requires you to specify the return type, possibly `void`, for all functions.
- In C++, specifying a parameter list with empty parentheses is equivalent to specifying a `void` parameter list in C.
- In C, using empty parentheses in a function definition or prototype is dangerous.
- It disables compile-time argument checking in function calls, which allows the caller to pass any arguments to the function.
- This could lead to runtime errors.

© 2016 Pearson Education, Ltd. All rights reserved.

Common Programming Error 15.1

Omitting the return type in a C++ function definition is a syntax error.

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- Line 7 is a familiar variable declaration.
- Declarations can be placed almost anywhere in a C++ program, but they must appear before their corresponding variables are used in the program.
- For example, in Fig. 15.1, the declaration in line 7 could have been placed immediately before line 10, the declaration in line 12 could have been placed immediately before line 16 and the declaration in line 13 could have been placed immediately before line 17.

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- Line 9 uses the standard output stream object—`std::cout`—and the stream insertion operator, `<<`, to display the string `"Enter first integer: "`.
- Output and input in C++ are accomplished with streams of characters.
- Thus, when line 9 executes, it sends the stream of characters `"Enter first integer: "` to `std::cout`, which is normally "connected" to the screen.
- We like to pronounce the preceding statement as "`std::cout` gets the character string `"Enter first integer: "`."
- **Line 10 uses the standard input stream object—`std::cin`—and the stream extraction operator, `>>`, to obtain a value from the keyboard.**

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- **Using the stream extraction operator with `std::cin` takes character input from the standard input stream, which is usually the keyboard.**
- We like to pronounce the preceding statement as, "`std::cin` gives a value to `number1`" or simply "`std::cin` gives `number1`."
- When the computer executes the statement in line 10, it waits for the user to enter a value for variable `number1`.
- The user responds by typing an integer (as characters), then pressing the *Enter* key.
- The computer converts the character representation of the number to an integer and assigns this value to the variable `number1`.

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- Line 15 displays `"Enter second integer: "` on the screen, prompting the user to take action.
- Line 16 obtains a value for variable `number2` from the user.
- The assignment statement in line 17 calculates the sum of the variables `number1` and `number2` and assigns the result to variable `sum`.
- **Line 18 displays the character string `Sum is` followed by the numerical value of variable `sum` followed by `std::endl`—a so-called stream manipulator.**
- **The name `endl` is an abbreviation for "end line."**
- The `std::endl` stream manipulator outputs a newline, then "flushes the output buffer."

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- This simply means that, on some systems where outputs accumulate in the machine until there are enough to "make it worthwhile" to display on the screen, **`std::endl` forces any accumulated outputs to be displayed at that moment**.
- This can be important when the outputs are prompting the user for an action, such as entering data.
- **We place `std::` before `cout`, `cin` and `endl`.**
- **This is required when we use standard C++ header files.**
- **The notation `std::cout` specifies that we're using a name, in this case `cout`, that belongs to "namespace" `std`.**
- Namespaces are an advanced C++ feature that we do not discuss in these introductory C++ chapters.

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- For now, you should simply remember to include `std::` before each mention of `cout`, `cin` and `endl` in a program.
- This can be cumbersome—in Fig. 15.3, we introduce the `using` statement, which will enable us to avoid placing `std::` before each use of a namespace `std` name.
- The statement in line 18 outputs values of different types.
- The stream insertion operator "knows" how to output each type of data.
- **Using multiple stream insertion operators (`<<`) in a single statement is referred to as concatenating, chaining or cascading stream insertion operations.**

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- Calculations can also be performed in output statements.
- We could have combined the statements in lines 17 and 18 into the statement
  - `std::cout << "Sum is " << number1 + number2 << std::endl;`

  thus eliminating the need for the variable `sum`.
- You'll notice that we did not have a `return 0;` statement at the end of main in this example.

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- **According to the C++ standard, if program execution reaches the end of `main` without encountering a `return` statement, it's assumed that the program terminated successfully—exactly as when the last statement in main is a `return` statement with the value `0`.**
- For that reason, we omit the `return` statement at the end of `main` in our C++ programs.

© 2016 Pearson Education, Ltd. All rights reserved.

15.3 A Simple Program: Adding Two Integers (Cont.)

- **A powerful C++ feature is that users can create their own types called classes (we introduce this capability in Chapter 16 and explore it in depth in Chapters 17–18).**
- **Users can then "teach" C++ how to input and output values of these new data types using the >> and << operators (this is called operator overloading 運算子多載—a topic we explore in Chapter 19).**

© 2016 Pearson Education, Ltd. All rights reserved.

15.4 C++ Standard Library

- **C++ programs consist of pieces called classes and functions.**
- **You can program each piece you need to form a C++ program.**
- **Instead, most C++ programmers take advantage of the rich collections of existing classes and functions in the C++ Standard Library.**
- **Thus, there are really two parts to learning the C++ "world."**
- **The first is learning the C++ language itself; the second is learning how to use the classes and functions in the C++ Standard Library.**
- **Many special-purpose class libraries are supplied by independent software vendors.**

© 2016 Pearson Education, Ltd. All rights reserved.

Software Engineering Observation 15.1

Use a "building-block" approach to create programs. Avoid reinventing the wheel. Use existing pieces wherever possible. Called ***software reuse****, this practice is central to object-oriented programming.*

Software Engineering Observation 15.2

When programming in C++, you typically will use the following building blocks: classes and functions from the C++ Standard Library, classes and functions you and your colleagues create and classes and functions from various popular third-party libraries.

15.4 C++ Standard Library (Cont)

- The advantage of creating your own functions and classes is that you'll know exactly how they work.
- You'll be able to examine the C++ code.
- The disadvantage is the time-consuming and complex effort that goes into designing, developing and maintaining new functions and classes that are correct and that operate efficiently.

© 2016 Pearson Education, Ltd. All rights reserved.

Performance Tip 15.1

Using C++ Standard Library functions and classes instead of writing your own versions can improve program performance, because they are written to perform efficiently. This technique also shortens program development time.

Portability Tip 15.1

Using C++ Standard Library functions and classes instead of writing your own improves program portability, because they are included in every C++ implementation.

15.5 Header Files

- The C++ Standard Library is divided into many portions, each with its own header file.
- **The header files contain the function prototypes for the related functions that form each portion of the library.**
- **The header files also contain definitions of various class types and functions, as well as constants needed by those functions.**
- **A header file "instructs" the compiler on how to interface with library and user-written components.**
- Figure 15.2 lists common C++ Standard Library header files.
- Header file names ending in `.h` are "old-style" headers that have been superceded by C++ Standard Library headers..

© 2016 Pearson Education, Ltd. All rights reserved.

C++ Standard Library header file	Explanation
`<iostream>`	Contains function prototypes for the C++ standard input and output functions, introduced in Section 15.3, and is covered in more detail in Chapter 21, Stream Input/Output: A Deeper Look.
`<iomanip>`	Contains function prototypes for stream manipulators that format streams of data. This header is first used in Section 15.15 and is discussed in more detail in Chapter 21.
`<cmath>`	Contains function prototypes for the math library functions.
`<cstdlib>`	Contains function prototypes for conversions of numbers to text, text to numbers, memory allocation, random numbers and various other utility functions. Portions of the header are covered in Chapter 18, Operator Overloading; Class `string` and Chapter 22, Exception Handling: A Deeper Look.
`<ctime>`	Contains function prototypes and types for manipulating the time and date.

Fig. 15.2 | C++ Standard Library header files. (Part 1 of 4.)

© 2016 Pearson Education, Ltd. All rights reserved.

C++ Standard Library header file	Explanation
`<array>`, `<vector>`, `<list>`, `<forward_list>`, `<deque>`, `<queue>`, `<stack>`, `<map>`, `<unordered_map>`, `<unordered_set>`, `<set>`, `<bitset>`	These headers contain classes that implement the C++ Standard Library containers. Containers store data during a program's execution. The `<vector>` header is first introduced in Section 15.15.
`<cctype>`	Contains function prototypes for functions that test characters for certain properties (such as whether the character is a digit or a punctuation), and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.
`<cstring>`	Contains function prototypes for C-style string-processing functions. This header is used in Chapter 18.
`<typeinfo>`	Contains classes for runtime type identification (determining data types at execution time). This header is discussed in Section 20.8.

Fig. 15.2 | C++ Standard Library header files. (Part 2 of 4.)

© 2016 Pearson Education, Ltd. All rights reserved.

C++ Standard Library header file	Explanation
`<exception>`, `<stdexcept>`	These headers contain classes that are used for exception handling (discussed in Chapter 22).
`<memory>`	Contains classes and functions used by the C++ Standard Library to allocate memory to the C++ Standard Library containers. This header is used in Chapter 22.
`<fstream>`	Contains function prototypes for functions that perform input from and output to files on disk.
`<string>`	Contains the definition of class `string` from the C++ Standard Library.
`<sstream>`	Contains function prototypes for functions that perform input from strings in memory and output to strings in memory.
`<functional>`	Contains classes and functions used by C++ Standard Library algorithms.
`<iterator>`	Contains classes for accessing data in the C++ Standard Library containers.
`<algorithm>`	Contains functions for manipulating data in C++ Standard Library containers.
`<cassert>`	Contains macros for adding diagnostics that aid program debugging.

Fig. 15.2 | C++ Standard Library header files. (Part 3 of 4.)

© 2016 Pearson Education, Ltd. All rights reserved.

C++ Standard Library header file	Explanation
`<cfloat>`	Contains the floating-point size limits of the system.
`<climits>`	Contains the integral size limits of the system.
`<cstdio>`	Contains function prototypes for the C's standard I/O functions.
`<locale>`	Contains classes and functions normally used by stream processing to process data in the natural form for different languages (e.g., monetary formats, sorting strings, character presentation, etc.).
`<limits>`	Contains classes for defining the numerical data type limits on each computer platform.
`<utility>`	Contains classes and functions that are used by many C++ Standard Library headers.

Fig. 15.2 | C++ Standard Library header files. (Part 4 of 4.)

© 2016 Pearson Education, Ltd. All rights reserved.

15.5 Header Files (Cont.)

- **You can create custom header files.**
- **Programmer-defined header files should end in `.h`.**
- **A programmer-defined header file can be included by using the `#include` preprocessor directive.**
- **For example, the header file `square.h` can be included in a program by placing the directive `#include "square.h"` at the beginning of the program.**

© 2016 Pearson Education, Ltd. All rights reserved.

15.6 Inline Functions

- **Implementing a program as a set of functions is good from a software engineering standpoint, but function calls involve execution-time overhead.**
- **C++ provides inline functions to help reduce function call overhead—especially for small functions.**
- **Placing the qualifier `in-line` before a function's return type in the function definition "advises" the compiler to generate a copy of the function's code in place (when appropriate) to avoid a function call.**

© 2016 Pearson Education, Ltd. All rights reserved.

15.6 Inline Functions (Cont.)

- The trade-off is that multiple copies of the function code are inserted in the program (often making the program larger) rather than there being a single copy of the function to which control is passed each time the function is called.
- The compiler can ignore the `inline`-qualifier and typically does so for all but the smallest functions.

© 2016 Pearson Education, Ltd. All rights reserved.

Software Engineering Observation 15.3

Changing to an `inline` *function could require clients of the function to be recompiled. This can be significant in program development and maintenance situations.*

Performance Tip 15.2

Using `inline` *functions can reduce execution time but may increase program size.*

Software Engineering Observation 15.4

The `inline` *qualifier should be used only with small, frequently used functions.*

15.6 Inline Functions (Cont.)

- Figure 15.3 uses `inline` function `cube` (lines 11–14) to calculate the volume of a cube of side length `side`.
- Keyword `const` in the parameter list of function `cube` tells the compiler that the function does not modify variable `side`.
- This ensures that the value of `side` is not changed by the function when the calculation is performed.
- Notice that the complete definition of function `cube` appears before it's used in the program.
- This is required so that the compiler knows how to expand a `cube` function call into its inlined code.
- For this reason, reusable inline functions are typically placed in header files, so that their definitions can be included in each source file that uses them.

© 2016 Pearson Education, Ltd. All rights reserved.

Software Engineering Observation 15.5

The `const` qualifier should be used to enforce the principle of least privilege. Using the principle of least privilege to properly design software can greatly reduce debugging time and improper side effects, and can make a program easier to modify and maintain.

© 2016 Pearson Education, Ltd. All rights reserved.

Programming codes 2

```
1  // Fig. 15.3: fig15_03.cpp
2  // inline function that calculates the volume of a cube.
3  #include <iostream>
4  using std::cout;
5  using std::cin;
6  using std::endl;
7
8  // Definition of inline function cube. Definition of function appears
9  // before function is called, so a function prototype is not required.
10 // First line of function definition acts as the prototype.
11 inline double cube( const double side )
12 {
13    return side * side * side; // calculate the cube of side
14 }
15
16 int main()
17 {
18    double sideValue; // stores value entered by user
19
```

Fig. 15.3 | `inline` function that calculates the volume of a cube. (Part 1 of 2.)

© 2016 Pearson Education, Ltd. All rights reserved.

```
20      for ( int i = 1; i <= 3; i++ )
21      {
22         cout << "\nEnter the side length of your cube: ";
23         cin >> sideValue; // read value from user
24
25         // calculate cube of sideValue and display result
26         cout << "Volume of cube with side "
27            << sideValue << " is " << cube( sideValue ) << endl;
28      }
29   }
```

```
Enter the side length of your cube: 1.0
Volume of cube with side 1 is 1

Enter the side length of your cube: 2.3
Volume of cube with side 2.3 is 12.167

Enter the side length of your cube: 5.4
Volume of cube with side 5.4 is 157.464
```

Fig. 15.3 | `inline` function that calculates the volume of a cube. (Part 2 of 2.)

© 2016 Pearson Education, Ltd. All rights reserved.

Programming code 2

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
inline double cube(const double side)
{
        return side * side*side;
}

int main()
{
    double sideValue;
    for (int i = 1; i <= 3; i++)
    {
        cout << "\nEnter the side length of your cube: ";
        cin >> sideValue;
        cout << "Value of cube with side "
        << sideValue << " is " << cube(sideValue) <<
        endl;
    }

}
```

© 2016 Pearson Education, Ltd. All rights reserved.

15.6 Inline Functions (Cont.)

- Lines 4–6 are `using` statements that help us eliminate the need to repeat the `std::` prefix.
- Once we include these `using` statements, we can write `cout` instead of `std::cout`, `cin` instead of `std::cin` and `endl` instead of `std::endl`, in the remainder of the program.
- From this point forward, each C++ example contains one or more `using` statements.
- **In place of lines 4–6, many programmers prefer to use**
  - `using namespace std;`

  which enables a program to use all the names in any standard C++ header file (such as `<iostream>`) that a program might include.

© 2016 Pearson Education, Ltd. All rights reserved.

15.6 Inline Functions (Cont.)

- From this point forward in our C++ programs, we'll use the preceding declaration in our programs.
- The `for` statement's condition (line 20) evaluates to either 0 (false) or nonzero (true).
- This is consistent with C.
- C++ also provides type `bool` for representing boolean (true/false) values.
- The two possible values of a `bool` are the keywords `true` and `false`.
- **When `true` and `false` are converted to integers, they become the values 1 and 0, respectively**.

© 2016 Pearson Education, Ltd. All rights reserved.

15.6 Inline Functions (Cont.)

- **When non-boolean values are converted to type `bool`, non-zero values become `true`, and zero or null pointer values become `false`.**

© 2016 Pearson Education, Ltd. All rights reserved.

15.7 C++ Keywords

- **Figure 15.4 lists the keywords common to C and C++ and the keywords unique to C++.**

© 2016 Pearson Education, Ltd. All rights reserved.

C++ Keywords

Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++-only keywords

and	and_eq	asm	bitand	bitor
bool	catch	class	compl	const_cast
delete	dynamic_cast	explicit	export	false
friend	inline	mutable	namespace	new
not	not_eq	operator	or	or_eq
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	try
typeid	typename	using	virtual	wchar_t
xor	xor_eq			

Fig. 15.4 | C++ keywords. (Part 1 of 2.)

© 2016 Pearson Education, Ltd. All rights reserved.

C++ Keywords				
C++11 keywords				
`alignas`	`alignof`	`char16_t`	`char32_t`	`constexpr`
`decltype`	`noexcept`	`nullptr`	`static_assert`	`thread_local`

Fig. 15.4 | C++ keywords. (Part 2 of 2.)

© 2016 Pearson Education, Ltd. All rights reserved.

15.8 References and Reference Parameters

- **Two ways to pass arguments to functions in many programming languages are pass-by-value and pass-by-reference.**
- **When an argument is passed by value, a *copy* of the argument's value is made and passed (on the function call stack) to the called function.**
- **Changes to the copy do not affect the original variable's value in the caller.**
- **This prevents the accidental side effects that so greatly hinder the development of correct and reliable software systems.**

Performance Tip 15.3

One disadvantage of pass-by-value is that, if a large data item is being passed, copying that data can take a considerable amount of execution time and memory space.

15.8 References and Reference Parameters (Cont.)

- This section introduces reference and reference parameters—the first of two means that C++ provides for performing pass-by-reference.
- **With pass-by-reference, the caller gives the called function the ability to access the caller's data directly, and to modify that data if the called function chooses to do so.**

© 2016 Pearson Education, Ltd. All rights reserved.

Performance Tip 15.4

Pass-by-reference is good for performance reasons, because it can eliminate the pass-by-value overhead of copying large amounts of data.

Software Engineering Observation 15.6

Pass-by-reference can weaken security; the called function can corrupt the caller's data.

15.8 References and Reference Parameters (Cont.)

- Later, we'll show how to achieve the performance advantage of pass-by-reference while simultaneously achieving the software engineering advantage of protecting the caller's data from corruption.
- **A reference parameter is an alias别名for its corresponding argument in a function call.**
- To indicate that a function parameter is passed by reference, simply follow the parameter's type in the function prototype by an ampersand (&); use the same notation when listing the parameter's type in the function header.

© 2016 Pearson Education, Ltd. All rights reserved.

15.8 References and Reference Parameters (Cont.)

- For example, the following declaration in a function header
  - `int &count`

  when read from right to left is pronounced "`count` is a reference to an `int`."
- In the function call, simply mention the variable by name to pass it by reference.
- Then, mentioning the variable by its parameter name in the body of the called function actually refers to the original variable in the calling function, and the original variable can be modified directly by the called function.
- As always, the function prototype and header must agree.

© 2016 Pearson Education, Ltd. All rights reserved.

15.8 References and Reference Parameters (Cont.)

Passing Arguments by Value and by Reference

- Figure 15.5 compares pass-by-value and pass-by-reference with reference parameters.
- The "styles" of the arguments in the calls to function `squareByValue` (line 17) and function `squareByReference` (line 22) are identical—both variables are simply mentioned by name in the function calls.
- Without checking the function prototypes or function definitions, it's not possible to tell from the calls alone whether either function can modify its arguments.

© 2016 Pearson Education, Ltd. All rights reserved.

15.8 References and Reference Parameters (Cont.)

- Because function prototypes are mandatory, however, the compiler has no trouble resolving the ambiguity.
- Recall that a function proto-type tells the compiler the type of data returned by the function, the number of parameters the function expects to receive, the types of the parameters, and the order in which they are expected.
- The compiler uses this information to validate function calls.
- In C, function prototypes are not required.
- Making them mandatory in C++ enables type-safe linkage, which ensures that the types of the arguments conform to the types of the pa-rameters.

© 2016 Pearson Education, Ltd. All rights reserved.

15.8 References and Reference Parameters (Cont.)

- Otherwise, the compiler reports an error.
- Locating such type errors at compile time helps prevent the runtime errors that can occur in C when arguments of incorrect data types are passed to functions.

© 2016 Pearson Education, Ltd. All rights reserved.

Programming codes 3

```
1  // Fig. 15.5: fig15_05.cpp
2  // Comparing pass-by-value and pass-by-reference with references.
3  #include <iostream>
4  using namespace std;
5
6  int squareByValue( int ); // function prototype (value pass)
7  void squareByReference( int & ); // function prototype (reference pass)
8
9  int main()
10 {
11    // demonstrate squareByValue
12    int x = 2;
13    cout << "x = " << x << " before squareByValue\n";
14    cout << "Value returned by squareByValue: "
15       << squareByValue( x ) << endl;
16    cout << "x = " << x << " after squareByValue\n" << endl;
17
18    // demonstrate squareByReference
19    int z = 4;
20    cout << "z = " << z << " before squareByReference" << endl;
21    squareByReference( z );
22    cout << "z = " << z << " after squareByReference" << endl;
23 }
24
```

Fig. 15.5 | Comparing pass-by-value and pass-by-reference with references. (Part 1 of 2.)

© 2016 Pearson Education, Ltd. All rights reserved.

```
25  // squareByValue multiplies number by itself, stores the
26  // result in number and returns the new value of number
27  int squareByValue( int number )
28  {
29     return number *= number; // caller's argument not modified
30  }
31
32  // squareByReference multiplies numberRef by itself and stores the result
33  // in the variable to which numberRef refers in the caller
34  void squareByReference( int &numberRef )
35  {
36     numberRef *= numberRef; // caller's argument modified
37  }
```

```
x = 2 before squareByValue
Value returned by squareByValue: 4
x = 2 after squareByValue

z = 4 before squareByReference
z = 16 after squareByReference
```

Fig. 15.5 | Comparing pass-by-value and pass-by-reference with references. (Part 2 of 2.)

© 2016 Pearson Education, Ltd. All rights reserved.

Programming code 3 (1)

```
#include <iostream>
using namespace std;

int squareByValue( int );
void squareByReference(int & );

int main()
{
      int x = 2;
      cout << "x= " << x << " before squareByValue\n";
      cout << "Value returned by squareByValue: "
      << squareByValue(x) << endl;
      cout << "x = " << x << " after squareByValue\n" <<
      endl;

      int z = 4;
      cout << "z = " << z << " before squareByReference" <<
      endl;
      squareByReference(z);
      cout << "z= " << z << " after squareByReference" <<
      endl;
}
```

© 2016 Pearson Education, Ltd. All rights reserved.

Programming code 3 (2)

```
int squareByValue(int number)
{
        return number *= number;
}

void squareByReference(int &numberRef)
{
        numberRef *= numberRef;
}
```

© 2016 Pearson Education, Ltd. All rights reserved.

Common Programming Error 15.2

Because reference parameters are mentioned only by name in the body of the called function, you might inadvertently treat reference parameters as pass-by-value parameters. This can cause unexpected side effects if the original copies of the variables are changed by the function.

Performance Tip 15.5

For passing large objects efficiently, use a constant reference parameter to simulate the appearance and security of pass-by-value and avoid the overhead of passing a copy of the large object. The called function will not be able to modify the object in the caller.

Software Engineering Observation 15.7

Many programmers do not declare parameters passed by value as `const`*, even when the called function should not modify the passed argument. Keyword* `const` *in this context would protect only a copy of the original argument, not the original argument itself, which when passed by value is safe from modification by the called function.*

15.8 References and Reference Parameters (Cont.)

- To specify a reference to a constant, place the `const` qualifier before the type specifier in the parameter declaration.
- Note in line 35 of Fig. 15.5 the placement of `&` in the parameter list of function `squareByReference`.
- Some C++ programmers prefer to write `int& numberRef` with the ampersand abutting `int`—both forms are equivalent to the compiler.

© 2016 Pearson Education, Ltd. All rights reserved.

Software Engineering Observation 15.8

For the combined reasons of clarity and performance, many C++ programmers prefer that modifiable arguments be passed to functions by using pointers, small nonmodifiable arguments be passed by value and large nonmodifiable arguments be passed by using references to constants.

© 2016 Pearson Education, Ltd. All rights reserved.

15.8 References and Reference Parameters (Cont.)

References as Aliases within a Function

- **References can also be used as aliases 别名 for other variables within a function (although they typically are used with functions as shown in Fig. 15.5).**
- For ex-ample, the code
  - ```
    int count = 1; // declare integer variable count
    int &cRef = count; // create cRef as an alias for count
    cRef++; // increment count (using its alias cRef)
    ```

  increments variable `count` by using its alias `cRef`.

© 2016 Pearson Education, Ltd. All rights reserved.
    ```

15.8 References and Reference Parameters (Cont.)

- Reference variables must be initialized in their declarations, as we show in line 9 of both Fig. 15.6 and Fig. 15.7, and cannot be reassigned as aliases to other variables.
- **Once a reference is declared as an alias for a variable, all operations "performed" on the alias (i.e., the reference) are actually performed on the original variable.**
- **The alias is simply another name for the original variable.**

© 2016 Pearson Education, Ltd. All rights reserved.

15.8 References and Reference Parameters (Cont.)

- Taking the address of a reference and comparing references do not cause syntax errors-; rather, each operation occurs on the variable for which the reference is an alias.
- Unless it's a reference to a constant, a reference argument must be an *lvalue* 左值 (e.g., a variable name), not a constant or expression that returns an *rvalue* 右值(e.g., the result of a calculation).

© 2016 Pearson Education, Ltd. All rights reserved.

Programming codes 4

```
1  // Fig. 15.6: fig15_06.cpp
2  // Initializing and using a reference.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int x = 3;
9     int &y = x; // y refers to (is an alias for) x
10
11    cout << "x = " << x << endl << "y = " << y << endl;
12    y = 7; // actually modifies x
13    cout << "x = " << x << endl << "y = " << y << endl;
14 }
```

```
x = 3
y = 3
x = 7
y = 7
```

Fig. 15.6 | Initializing and using a reference.

© 2016 Pearson Education, Ltd. All rights reserved.

```
1  // Fig. 15.7: fig15_07.cpp
2  // Uninitialized reference is a syntax error.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int x = 3;
9     int &y; // Error: y must be initialized
10
11    cout << "x = " << x << endl << "y = " << y << endl;
12    y = 7;
13    cout << "x = " << x << endl << "y = " << y << endl;
14 }
```

Fig. 15.7 | Uninitialized reference is a syntax error. (Part 1 of 2.)

© 2016 Pearson Education, Ltd. All rights reserved.

Microsoft Visual C++ compiler error message:

```
fig15_07.cpp(9) : error C2530: 'y' :
   references must be initialized
```

GNU C++ compiler error message:

```
fig15_07.cpp:9: error: 'y' declared as a reference but not initialized
```

Xcode LLVM compiler error message:

```
Declaration of reference variable 'y' requires an initializer
```

Fig. 15.7 | Uninitialized reference is a syntax error. (Part 2 of 2.)

© 2016 Pearson Education, Ltd. All rights reserved.

Programming code 4

```
#include <iostream>
using namespace std;

int main()
{
	int x = 3;
	int &y = x;

	cout << "x = " << x << endl << "y = " << y << endl;
	y = 7;
	cout << "x = " << x << endl << "y = " << y << endl;

}
```

© 2016 Pearson Education, Ltd. All rights reserved.

Programming code 4

```
#include <iostream>
using namespace std;

int main()
{
	int x = 3;
	int &y;

	cout << "x = " << x << endl << "y = " << y << endl;
	y = 7;
	cout << "x = " << x << endl << "y = " << y << endl;

}
```

© 2016 Pearson Education, Ltd. All rights reserved.

15.8 References and Reference Parameters (Cont.)

Returning a Reference from a Function

- **Returning references from functions can be dangerous.**
- **When returning a reference to a variable declared in the called function, the variable should be declared `static` within that function.**
- **Otherwise, the reference refers to an automatic variable that is discarded when the function terminates; such a variable is "undefined," and the program's behavior is unpredictable.**
- **References to undefined variables are called dangling references 懸置參考.**

© 2016 Pearson Education, Ltd. All rights reserved.

Common Programming Error 15.3

Not initializing a reference variable when it's declared is a compilation error, unless the declaration is part of a function's parameter list. Reference parameters are initialized when the function in which they're declared is called.

Common Programming Error 15.4

Attempting to reassign a previously declared reference to be an alias to another variable is a logic error. The value of the other variable is simply assigned to the variable for which the reference is already an alias.

Common Programming Error 15.5

Returning a reference to an automatic variable in a called function is a logic error. Some compilers issue a warning when this occurs.

15.8 References and Reference Parameters (Cont.)

Error Messages for Uninitialized References

- The C++ standard does not specify the error messages that compilers use to indicate particular errors.
- For this reason, we show in Fig. 15.7 the error messages produced by several compilers when a reference is not initialized.

© 2016 Pearson Education, Ltd. All rights reserved.

15.9 Empty Parameter Lists

- C++, like C, allows you to define functions with no parameters.
- **In C++, an empty parameter list is specified by writing either `void` or nothing at all in parentheses.**
- **The prototypes**
  - `void print();`
    `void print( void );`

  each specify that function `print` does not take arguments and does not return a value.
- **These prototypes are equivalent.**

© 2016 Pearson Education, Ltd. All rights reserved.

Portability Tip 15.2

The meaning of an empty function parameter list in C++ is dramatically different than in C. In C, it means all argument checking is disabled (i.e., the function call can pass any arguments it wants). In C++, it means that the function takes no arguments. Thus, C programs using this feature might cause compilation errors when compiled in C++.

Common Programming Error 15.6

It's a compilation error to specify default arguments in both a function's prototype and header.

- Exercise

Exercise 1

- Enter a number x, and use the “ Call by value” to calculate its cube value. Then enter another number and create an aliasing z, which will change with the value of x, and then calculate the cube values of z and x, using the function you created.

```
#include <iostream>
using namespace std;

int cubeByValue(int number)
{
    return number * number*number;
}
```

```
int main()
{
    int x;
    int &z = x;
    ....
}
```

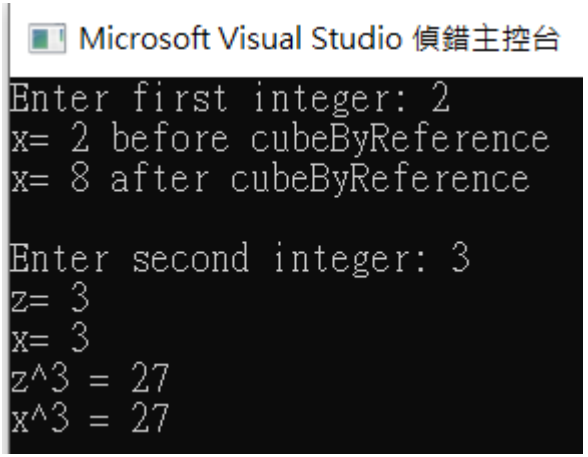
© 2016 Pearson Education, Ltd. All rights reserved.

Exercise 2

- Same to exercise 1 , but now use the “ Call by reference” to perform the result
- （ Hint:You can reference programming code3 ）

```
1
2   #include <iostream>
3   using namespace std;
4
5   void  cubeByReference(int &numberRef)
6   {
7       numberRef=numberRef*numberRef*numberRef;
8   }
9
```

```
11  int main()
12  {
13
14      int x;
15      int &z = x;
16
17
18
19        ....
20
21
22
23
24
25
26  }
27
```



© 2016 Pearson Education, Ltd. All rights reserved.

To be continued……

Instructor: Cheng-Chun Chang (張正春)
Department of Electrical Engineering

Exercise1.txt

Exercise2.txt

© 2016 Pearson Education, Ltd. All rights reserved.