EE1003
Introduction to Computer I

Chapter 2
Introduction to C++ Programming

Andy, Yu-Guang Chen
Assistant Professor, Department of EE
National Central University
andyygchen@ee.ncu.edu.tw

Learning Objectives

In this chapter you'll learn:

- To write simple computer programs in C++.
- To read data from the keyboard and write data to the screen.
- To use fundamental types.
- Basic computer memory concepts.
- To use arithmetic operators.
- Operator precedence.
- To write simple decision-making statements.

Outline

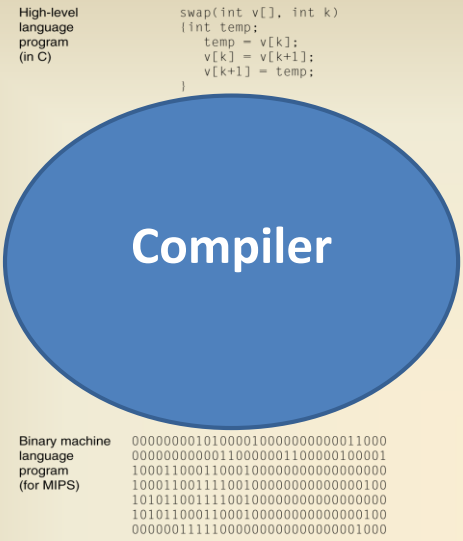
2.1 Introduction
2.2 First Program in C++: Printing a Line of Text
2.3 Modifying Our First C++ Program
2.4 Another C++ Program: Adding Integers
2.5 Memory Concepts
2.6 Arithmetic
2.7 Decision Making: Equality and Relational Operators
2.8 Wrap-Up

2.1 Introduction

- We now introduce C++ programming, which facilitates a disciplined approach to program design.
- Most of the C++ programs you'll study in this book process information and display results.

2.1 Introduction

◆C++ program execution process



2.2 First Program in C++: Printing a Line of Text

◆ Simple program that prints a line of text (Fig. 2.1).

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

```
Welcome to C++!
```

Fig. 2.1 | Text-printing program.

2.2 First Program in C++: Printing a Line of Text

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
```

- // indicates that the remainder of each line is a comment.
  - You insert comments to document your programs and to help other people read and understand them.
  - Comments are ignored by the C++ compiler and do not cause any machine-language object code to be generated.
- A comment beginning with // is called a single-line comment because it terminates at the end of the current line.
- You also may use C's style in which a comment—possibly containing many lines—begins with /* and ends with */.

2.2 First Program in C++: Printing a Line of Text

```
3  #include <iostream> // allows program to output data to the screen
```

- A preprocessor directive is a message to the C++ preprocessor.
- Lines that begin with # are processed by the preprocessor before the program is compiled.
- #include <iostream> notifies the preprocessor to include in the program the contents of the input/output stream header file <iostream>.
  - Must be included for any program that outputs data to the screen or inputs data from the keyboard using C++-style stream input/output.

2.2 First Program in C++: Printing a Line of Text

Good Programming Practice 2.1

Begin every program with a comment that describes the purpose of the program.

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

```
Welcome to C++!
```

Common Programming Error 2.1

Forgetting to include the `<iostream>` header file in a program that inputs data from the keyboard or outputs data to the screen causes the compiler to issue an error message, because the compiler cannot recognize references to the stream components (e.g., `cout`).

2.2 First Program in C++: Printing a Line of Text

- You use blank lines, space characters and tab characters (i.e., "tabs") to make programs easier to read.
  - Together, these characters are known as white space.
  - White-space characters are normally ignored by the compiler.

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

```
Welcome to C++!
```

Fig. 2.1 | Text-printing program.

2.2 First Program in C++: Printing a Line of Text

```
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

- `main` is a part of every C++ program.
- The parentheses after `main` indicate that main is a program building block called a function.
- C++ programs typically consist of one or more functions and classes.
- Exactly one function in every program must be named `main`.
- C++ programs begin executing at function `main`, even if `main` is not the first function in the program.
- The keyword `int` to the left of `main` indicates that `main` "returns" an integer (whole number) value.
  - A keyword is a word in code that is reserved by C++ for a specific use.
  - For now, simply include the keyword `int` to the left of `main` in each of your programs.

2.2 First Program in C++: Printing a Line of Text

```
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

- A left brace, {, must begin the body of every function.
- A corresponding right brace, }, must end each function's body.
- A statement instructs the computer to perform an action.
- A string is sometimes called a character string or a string literal.
- We refer to characters between double quotation marks simply as strings.
  - White-space characters in strings are NOT ignored by the compiler.
- A statement normally ends with a semicolon (;), also known as the statement terminator.
  - Preprocessor directives (like `#include`) do not end with a semicolon.

2.2 First Program in C++: Printing a Line of Text

```
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

- When a `cout` statement executes, it sends a stream of characters to the standard output stream object—std::cout—which is normally "connected" to the screen.
- The `std::` before `cout` is required when we use names that we've brought into the program by the preprocessor directive `#include <iostream>`.
  - The notation `std::cout` specifies that we are using a name, in this case `cout`, that belongs to "namespace" `std`.
  - The names `cin` (the standard input stream) and `cerr` (the standard error stream) also belong to namespace `std`.
- The << operator is referred to as the stream insertion operator.
  - The value to the operator's right, the right operand, is inserted in the output stream.

2.2 First Program in C++: Printing a Line of Text

```
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

- When a backslash (\) is encountered in a string of characters, the next character and the backslash form an escape sequence.
  - It indicates that a "special" character is to be output.
- The escape sequence `\n` means newline. (not printed)
  - Causes the cursor to move to the beginning of the next line on the screen.
- When the return statement is used at the end of `main`, the value `0` indicates that the program has terminated successfully.
- If program execution reaches the end of `main` without a `return` statement, it's assumed that the program terminated successfully
  - Similar to a `return` statement with the value `0`.

2.2 First Program in C++: Printing a Line of Text

Escape sequence	Description
\n	Newline. Position the screen cursor to the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line.
\a	Alert. Sound the system bell.
\\	Backslash. Used to print a backslash character.
\'	Single quote. Use to print a single quote character.
\"	Double quote. Used to print a double quote character.

Fig. 2.2 | Escape sequences.

2.2 First Program in C++: Printing a Line of Text

Common Programming Error 2.2

Omitting the semicolon at the end of a C++ statement is a syntax error. (Again, preprocessor directives do not end in a semicolon.) The syntax of a programming language specifies the rules for creating proper programs in that language. A syntax error occurs when the compiler encounters code that violates C++'s language rules (i.e., its syntax). The compiler normally issues an error message to help you locate and fix the incorrect code. Syntax errors are also called compiler errors, compile-time errors or compilation errors, because the compiler detects them during the compilation phase. You cannot execute your program until you correct all the syntax errors in it. As you'll see, some compilation errors are not syntax errors.

2.2 First Program in C++: Printing a Line of Text

Good Programming Practice 2.4

Set a convention for the size of indent you prefer, then apply it uniformly. The tab key may be used to create indents, but tab stops may vary. We recommend using either 1/4-inch tab stops or (preferably) three spaces to form a level of indent.

```
1  // Fig. 2.1: fig02_01.cpp
2  // Text-printing program.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome to C++!\n"; // display message
9
10    return 0; // indicate that program ended successfully
11 } // end function main
```

```
Welcome to C++!
```

Fig. 2.1 | Text-printing program.

2.3 Modifying Our First C++ Program

◆Welcome to C++! can be printed several ways

```
1  // Fig. 2.3: fig02_03.cpp
2  // Printing a line of text with multiple statements.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome ";
9     std::cout << "to C++!\n";
10 } // end function main
```

```
Welcome to C++!
```

Fig. 2.3 | Printing a line of text with multiple statements.

2.3 Modifying Our First C++ Program

- A single statement can print multiple lines by using newline characters.
- Each time the \n (newline) is encountered, the screen cursor is positioned to the beginning of the next line.
- Place two newline characters back to back → a blank line

```
1  // Fig. 2.4: fig02_04.cpp
2  // Printing multiple lines of text with a single statement.
3  #include <iostream> // allows program to output data to the screen
4
5  // function main begins program execution
6  int main()
7  {
8     std::cout << "Welcome\nto\n\nC++!\n";
9  } // end function main
```

```
Welcome
to

C++!
```

Fig. 2.4 | Printing multiple lines of text with a single statement.

2.4 Another C++ Program: Adding Integers

- The input stream object std::cin and the stream extraction operator, >>, can be used obtain data from the user at the keyboard.

2.4 Another C++ Program: Adding Integers

```
1  // Fig. 2.5: fig02_05.cpp
2  // Addition program that displays the sum of two integers.
3  #include <iostream> // allows program to perform input and output
4
5  // function main begins program execution
6  int main()
7  {
8     // variable declarations
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
12
13    std::cout << "Enter first integer: "; // prompt user for data
14    std::cin >> number1; // read first integer from user into number1
15
16    std::cout << "Enter second integer: "; // prompt user for data
17    std::cin >> number2; // read second integer from user into number2
18
19    sum = number1 + number2; // add the numbers; store result in sum
20
21    std::cout << "Sum is " << sum << std::endl; // display sum; end line
22 } // end function main
```

Fig. 2.5 | Addition program that displays the sum of two integers. (Part 1 of 2.)

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

Fig. 2.5 | Addition program that displays the sum of two integers. (Part 2 of 2.)

2.4 Another C++ Program: Adding Integers (cont.)

```
9     int number1; // first integer to add
10    int number2; // second integer to add
11    int sum; // sum of number1 and number2
```

- Declarations introduce the identifiers `number1`, `number2` and `sum` into programs; they are the names of variables.
- A variable is a location in the computer's memory where a value can be stored for use by a program.
- Variables `number1`, `number2` and `sum` are data of type **int**, meaning that these variables will hold integer values.
- All variables must be declared with a name and a data type before they can be used in a program.
- If more than one name is declared in a declaration, the names are separated by commas (,) → called comma-separated list.
  - `int number1, number2;`

Good Programming Practice 2.5
Place a space after each comma (,) to make programs more readable.

2.4 Another C++ Program: Adding Integers (cont.)

- Data type `double` is for specifying real numbers, and data type `char` for specifying character data.
  - Real numbers are numbers with decimal points, such as 3.4, 0.0 and –11.19.
- A `char` variable may hold only a single lowercase letter, a single uppercase letter, a single digit or a single special character (e.g., $ or *).
- Types such as `int`, `double` and `char` are called fundamental types.
- Fundamental-type names are keywords and therefore must appear in all lowercase letters.

2.4 Another C++ Program: Adding Integers (cont.)

- Appendix C contains the complete list of fundamental types.

Integral types	Floating-point types
bool	float
char	double
signed char	long double
unsigned char	
short int	
unsigned short int	
int	
unsigned int	
long int	
unsigned long int	
wchar_t	

2.4 Another C++ Program: Adding Integers (cont.)

- A variable name is any valid identifier that is not a keyword.
- An identifier is a series of characters consisting of letters, digits and underscores (_) that does not begin with a digit.
- C++ is case sensitive—uppercase and lowercase letters are different, so `a1` and `A1` are different identifiers.
- Declarations of variables can be placed almost anywhere in a program, but they must appear before their corresponding variables are used in the program.

2.4 Another C++ Program: Adding Integers (cont.)

Portability Tip 2.1
C++ allows identifiers of any length, but your C++ implementation may restrict identifier lengths. Use identifiers of 31 characters or fewer to ensure portability.

Good Programming Practice 2.6
Choosing meaningful identifiers makes a program self-documenting—a person can understand the program simply by reading it rather than having to refer to manuals or comments.

Good Programming Practice 2.9
Always place a blank line between a declaration and adjacent executable statements. This makes the declarations stand out in the program and contributes to program clarity.

2.4 Another C++ Program: Adding Integers (cont.)

```
13    std::cout << "Enter first integer: "; // prompt user for data
14    std::cin >> number1; // read first integer from user into number1
```

- A prompt directs the user to take a specific action.
- A `cin` statement uses the input stream object cin (of namespace `std`) and the stream extraction operator, `>>`, to obtain a value from the keyboard.
- Using the stream extraction operator with `std::cin` takes character input from the standard input stream, which is usually the keyboard.

Error-Prevention Tip 2.2

Programs should validate the correctness of all input values to prevent erroneous information from affecting a program's calculations.

2.4 Another C++ Program: Adding Integers (cont.)

```
14    std::cin >> number1; // read first integer from user into number1
```

- When the computer executes an input statement, it waits for the user to enter a value for variable `number1`.
- The user types the number (as characters) then press the *Enter* key (or the Return key) to send the characters to the computer.
- The computer converts the character representation of the number to an integer and put the value to the variable `number1`.
- Any subsequent references to `number1` in this program will use this same value.

2.4 Another C++ Program: Adding Integers (cont.)

```
19    sum = number1 + number2; // add the numbers; store result in sum
```

- In this program, an assignment statement adds the values of variables `number1` and `number2` and assigns the result to variable `sum` using the assignment operator `=`.
  - Most calculations are performed in assignment statements.
- The = operator and the + operator are called binary operators because each has two operands.

Good Programming Practice 2.10

Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.

2.4 Another C++ Program: Adding Integers (cont.)

```
21    std::cout << "Sum is " << sum << std::endl; // display sum; end line
```

- `std::endl` is a so-called stream manipulator.
  - The name `endl` is an abbreviation for "end line"
- `std::endl` outputs a newline and flushes the output buffer.
  - Some systems may accumulate outputs in the machine (output buffer) and display them simultaneously on the screen.
  - `std::endl` forces any accumulated outputs to be displayed.
- Using multiple stream insertion operators (<<) in a single statement is referred to as concatenating, chaining or cascading stream insertion operations.
- Calculations can also be performed in output statements.

2.5 Memory Concepts

- Variable names such as `number1`, `number2` and `sum` actually correspond to locations in the computer's memory.
- Every variable has a name, a type, a size and a value.
- When a value is placed in a memory location, it overwrites the previous value in that location → destructive
- When a value is read out of a memory location, the process is nondestructive.

number1	45
number2	72
sum	117

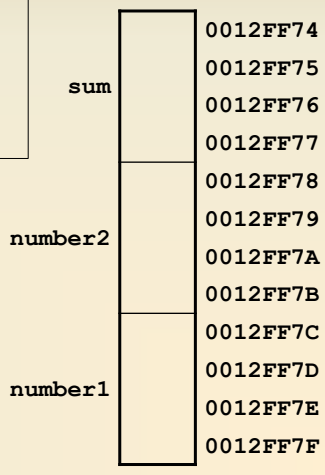
Fig. 2.8 | Memory locations after calculating and storing the `sum` of `number1` and `number2`.

2.5 Memory Concepts

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

Output

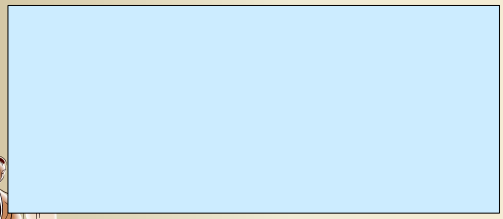
Memory



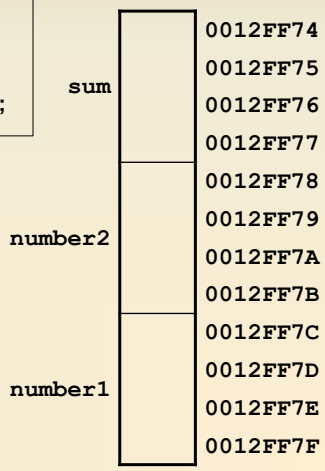
2.5 Memory Concepts

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

Output



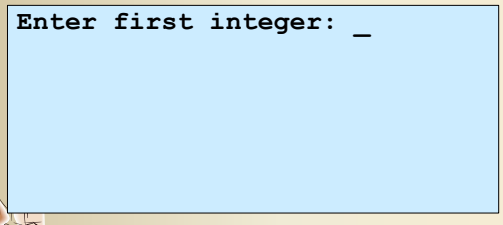
Memory



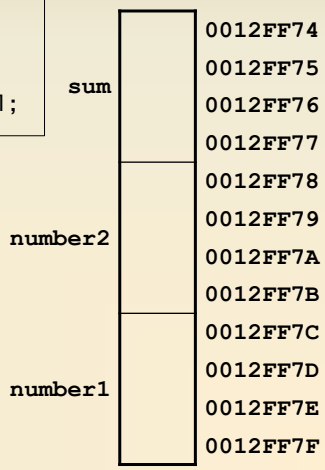
2.5 Memory Concepts

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

Output



Memory



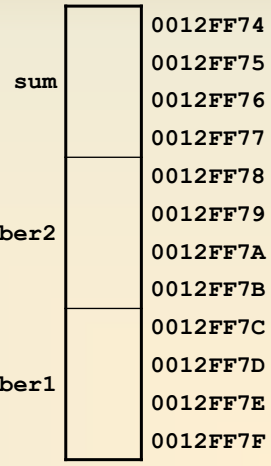
2.5 Memory Concepts

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

Output

```
Enter first integer: 45_
```

Memory



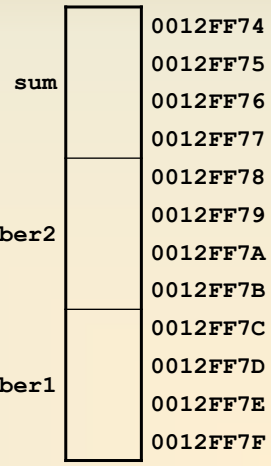
2.5 Memory Concepts

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

Output

```
Enter first integer: 45
_
```

Memory



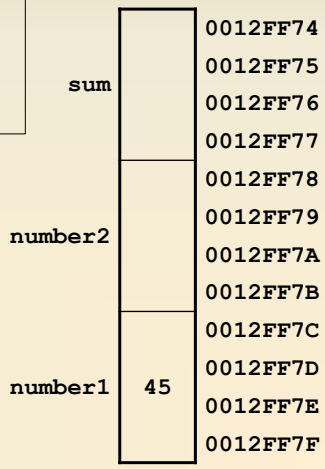
2.5 Memory Concepts

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

Output

```
Enter first integer: 45
_
```

Memory



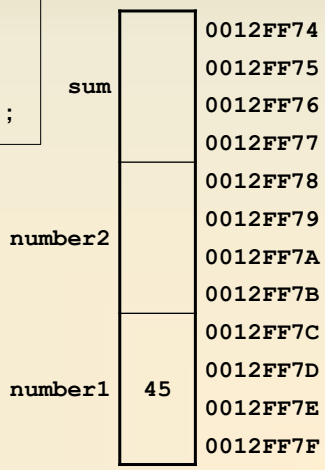
2.5 Memory Concepts

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

Output

```
Enter first integer: 45
Enter second integer: _
```

Memory



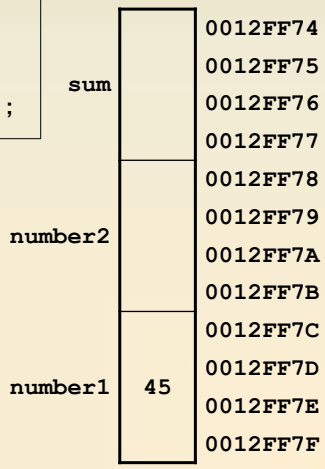
2.5 Memory Concepts

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

Output

```
Enter first integer: 45
Enter second integer: 72_
```

Memory



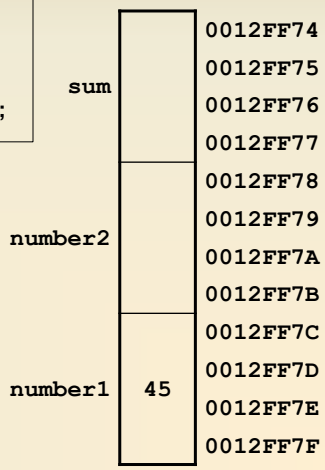
2.5 Memory Concepts

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

Output

```
Enter first integer: 45
Enter second integer: 72
_
```

Memory



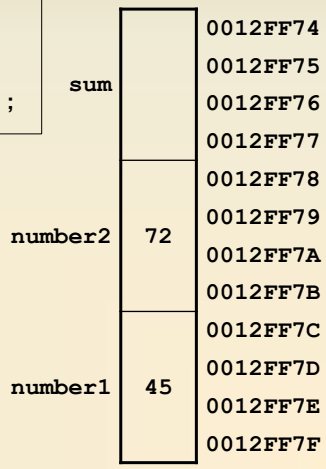
2.5 Memory Concepts

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

Output

```
Enter first integer: 45
Enter second integer: 72
_
```

Memory



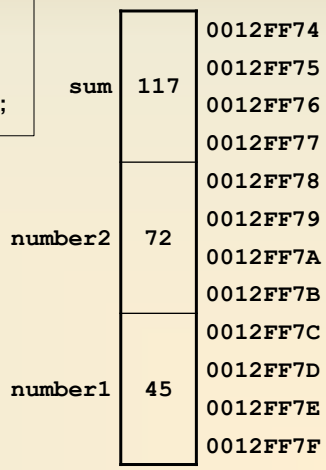
2.5 Memory Concepts

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

Output

```
Enter first integer: 45
Enter second integer: 72
_
```

Memory



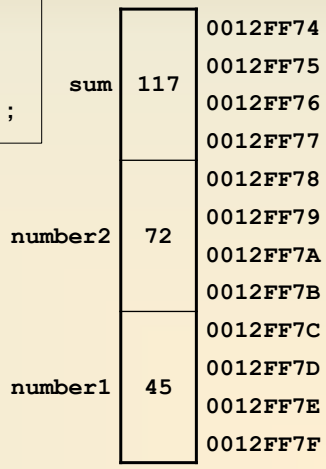
2.5 Memory Concepts

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

Output

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

Memory



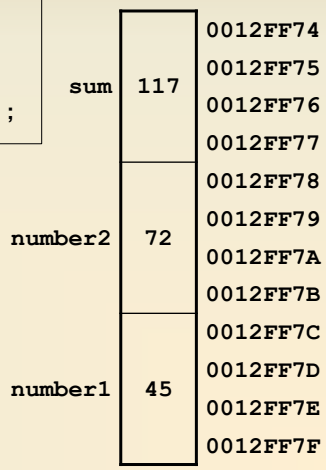
2.5 Memory Concepts

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

Output

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
Press any key to continue_
```

Memory



2.5 Memory Concepts

binary representation	hexadecimal	decimal
00000000 00000000 00000000 00000000	00000000	0
00000000 00000000 00000000 00000001	00000001	1
00000000 00000000 00000000 00000010	00000002	2
00000000 00000000 00000000 00000011	00000003	3
00000000 00000000 00000000 00000100	00000004	4
00000000 00000000 00000000 00000101	00000005	5
00000000 00000000 00000000 00000110	00000006	6
00000000 00000000 00000000 00000111	00000007	7
00000000 00000000 00000000 00001000	00000008	8
00000000 00000000 00000000 00001001	00000009	9
00000000 00000000 00000000 00001010	0000000A	10
00000000 00000000 00000000 00001011	0000000B	11
00000000 00000000 00000000 00001100	0000000C	12
00000000 00000000 00000000 00001101	0000000D	13
00000000 00000000 00000000 00001110	0000000E	14
00000000 00000000 00000000 00001111	0000000F	15

2.5 Memory Concepts

Name	Radix	Digits
Binary	**2**	**0,1**
Octal	**8**	**0,1,2,3,4,5,6,7**
Decimal	**10**	**0,1,2,3,4,5,6,7,8,9**
Hexadecimal	**16**	**0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F**

- The six letters (in addition to the 10 integers) in hexadecimal represent: 10 (A), 11 (B), 12 (C), 13 (D), 14 (E), and 15 (F), respectively.

2.5 Memory Concepts

binary representation	hexadecimal	decimal
00000000 00000000 00000000 00010000	00000010	16
00000000 00000000 00000000 00010001	00000011	17
00000000 00000000 00000000 00010010	00000012	18
00000000 00000000 00000000 00010011	00000013	19
00000000 00000000 00000000 00010100	00000014	20
00000000 00000000 00000000 00010101	00000015	21
00000000 00000000 00000000 00010110	00000016	22
00000000 00000000 00000000 00010111	00000017	23
00000000 00000000 00000000 00011000	00000018	24
00000000 00000000 00000000 00011001	00000019	25
00000000 00000000 00000000 00011010	0000001A	26
00000000 00000000 00000000 00011011	0000001B	27
00000000 00000000 00000000 00011100	0000001C	28
00000000 00000000 00000000 00011101	0000001D	29
00000000 00000000 00000000 00011110	0000001E	30
00000000 00000000 00000000 00011111	0000001F	31

binary representation	hexadecimal	decimal
00000000 00000000 00000000 00100000	00000020	32
00000000 00000000 00000000 00100001	00000021	33
00000000 00000000 00000000 00100010	00000022	34
00000000 00000000 00000000 00100011	00000023	35
.....	.....	.....
00000000 00010010 11111111 01110100	0012FF74	1245044
00000000 00010010 11111111 01111000	0012FF78	1245048
00000000 00010010 11111111 01111100	0012FF7C	1245052
00000000 00010010 11111111 10000000	0012FF80	1245056
.....	.....	.....
11111111 11111111 11111111 11111010	FFFFFFFA	4294967290
11111111 11111111 11111111 11111011	FFFFFFFB	4294967291
11111111 11111111 11111111 11111100	FFFFFFFC	4294967292
11111111 11111111 11111111 11111101	FFFFFFFD	4294967293
11111111 11111111 11111111 11111110	FFFFFFFE	4294967294
11111111 11111111 11111111 11111111	FFFFFFFF	4294967295

2.5 Memory Concepts

Data types

long double (8 bytes)	絕對值範圍大約是 $2.2 \cdot 10^{-308}$ ~ $1.8 \cdot 10^{308}$
double (8 bytes)	絕對值範圍大約是 $2.2 \cdot 10^{-308}$ ~ $1.8 \cdot 10^{308}$
float (4 bytes)	絕對值範圍大約是 $1.2 \cdot 10^{-38}$ ~ $3.4 \cdot 10^{38}$
unsigned long long int (unsigned long long) (8 bytes)	0 ~ $2^{64}-1$
long long int (long long) (8 bytes)	-2^{63} ~ $2^{63}-1$
unsigned long int (unsigned long) (4 bytes)	0 ~ $2^{32}-1$
long int (long) (4 bytes)	-2^{31} ~ $2^{31}-1$ (2147483647)
unsigned int (unsigned) (4 bytes)	0 ~ $2^{32}-1$ (4294967295)
int (4 bytes)	-2^{31} ~ $2^{31}-1$ (2147483647)
unsigned short int (unsigned short) (2)	0 ~ $2^{16}-1$ (65535)
short int (short) (2 bytes)	-2^{15} ~ $2^{15}-1$ (32767)
unsigned char (1 byte)	0 ~ $2^{8}-1$ (0 ~ 255)
char (1 byte)	-2^{7} ~ $2^{7}-1$ (-128 ~ 127)
bool (1 byte)	(false becomes 0, true becomes 1)

Promotion hierarchy for fundamental data types

2.6 Arithmetic

- The arithmetic operators in Fig. 2.9 are all binary operators.
- Integer division yields an integer quotient.
  - Any fractional part in integer division is discarded (i.e., truncated) without rounding.
- The percent sign (%) is the modulus operator.
  - The modulus operator provides the remainder after integer division.
  - The modulus operator can be used only with integer operands.

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	f + 7
Subtraction	-	$p - c$	p - c
Multiplication	*	bm or $b \cdot m$	b * m
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	x / y
Modulus	%	$r \bmod s$	r % s

Fig. 2.9 | Arithmetic operators.

2.6 Arithmetic (cont.)

- Arithmetic expressions in C++ must be entered into the computer in straight-line form.
- Expressions such as "a divided by b" must be written as `a / b`, so that all constants, variables and operators appear in a straight line.
  - $\frac{a}{b}$ is not acceptable
- Parentheses are used in C++ expressions in the same manner as in algebraic expressions.
- For example, to multiply `a` times the quantity `b + c`, we write `a * ( b + c )`.

2.6 Arithmetic (cont.)

- C++ applies the operators in arithmetic expressions in a precise sequence determined by the following rules of operator precedence, which are generally the same as those followed in algebra.

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
*, /, %	Multiplication, Division, Modulus	Evaluated second. If there are several, they're evaluated left to right.
+	Addition	Evaluated last. If there are several, they're evaluated left to right.
-	Subtraction	

Fig. 2.10 | Precedence of arithmetic operators.

2.6 Arithmetic (cont.)

- There is no arithmetic operator for exponentiation in C++, so x^2 is represented as `x * x`.
- Figure 2.11(next page) illustrates the order in which the operators in a second-degree polynomial are applied.
- As in algebra, it's acceptable to add unnecessary parentheses to make the expression clearer.
- These are called redundant parentheses.

Good Programming Practice 2.11

Using redundant parentheses in complex arithmetic expressions can make the expressions clearer.

2.6 Arithmetic (cont.)

- $y = ax^2 + bx + c$
  - `y=(a*x*x)+(b*x)+c;`
  - a=2, b=3, c=7, x=5

```
Step 1.  y = 2 * 5 * 5 + 3 * 5 + 7;   (Leftmost multiplication)
             2 * 5 is 10
Step 2.  y = 10 * 5 + 3 * 5 + 7;      (Leftmost multiplication)
             10 * 5 is 50
Step 3.  y = 50 + 3 * 5 + 7;          (Multiplication before addition)
                 3 * 5 is 15
Step 4.  y = 50 + 15 + 7;             (Leftmost addition)
             50 + 15 is 65
Step 5.  y = 65 + 7;                  (Last addition)
             65 + 7 is 72
Step 6.  y = 72                       (Last operation—place 72 in y)
```

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

2.7 Decision Making: Equality and Relational Operators

- The if statement allows a program to take alternative action based on whether a condition is true or false.
- If the condition is true, the statement in the body of the `if` statement is executed.
  - Otherwise, the body statement is not executed.
- Conditions can be formed by using the equality operators and relational operators summarized in Fig. 2.12 (next page).
- The relational operators all have the same precedence level and associate left to right.
- The precedence level of equality operators is lower than that of the relational operators. (association is still left to right)

2.7 Decision Making: Equality and Relational Operators

- Operations

Standard algebraic equality or relational operator	C++ equality or relational operator	Sample C++ condition	Meaning of C++ condition
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Fig. 2.12 | Equality and relational operators.

Common Programming Error 2.5
A syntax error will occur if any of the operators ==, !=, >= and <= appears with spaces between its pair of symbols.

2.7 Decision Making: Equality and Relational Operators

Common Programming Error 2.7

Confusing the equality operator == with the assignment operator = results in logic errors. The equality operator should be read "is equal to," and the assignment operator should be read "gets" or "gets the value of" or "is assigned the value of." Some people prefer to read the equality operator as "double equals." As we discuss in Section 5.9, confusing these operators may not necessarily cause an easy-to-recognize syntax error, but may cause extremely subtle logic errors.

2.7 Decision Making: Equality and Relational Operators

```
1  // Fig. 2.13: fig02_13.cpp
2  // Comparing integers using if statements, relational operators
3  // and equality operators.
4  #include <iostream> // allows program to perform input and output
5
6  using std::cout; // program uses cout
7  using std::cin; // program uses cin
8  using std::endl; // program uses endl
9
10 // function main begins program execution
11 int main()
12 {
13    int number1; // first integer to compare
14    int number2; // second integer to compare
15
16    cout << "Enter two integers to compare: "; // prompt user for data
17    cin >> number1 >> number2; // read two integers from user
18
19    if ( number1 == number2 )
20       cout << number1 << " == " << number2 << endl;
21
```

Fig. 2.13 | Comparing integers using `if` statements, relational operators and equality operators. (Part 1 of 3.)

2.7 Decision Making: Equality and Relational Operators

```
22     if ( number1 != number2 )
23        cout << number1 << " != " << number2 << endl;
24
25     if ( number1 < number2 )
26        cout << number1 << " < " << number2 << endl;
27
28     if ( number1 > number2 )
29        cout << number1 << " > " << number2 << endl;
30
31     if ( number1 <= number2 )
32        cout << number1 << " <= " << number2 << endl;
33
34     if ( number1 >= number2 )
35        cout << number1 << " >= " << number2 << endl;
36  } // end function main
```

```
Enter two integers to compare: 3 7
3 != 7
3 < 7
3 <= 7
```

Fig. 2.13 | Comparing integers using `if` statements, relational operators and equality operators. (Part 2 of 3.)

2.7 Decision Making: Equality and Relational Operators

```
Enter two integers to compare: 22 12
22 != 12
22 > 12
22 >= 12
```

```
Enter two integers to compare: 7 7
7 == 7
7 <= 7
7 >= 7
```

Fig. 2.13 | Comparing integers using `if` statements, relational operators and equality operators. (Part 3 of 3.)

2.7 Decision Making: Equality and Relational Operators

```
1  // Fig. 2.13: fig02_13.cpp
2  // Comparing integers using if statements, relational operators
3  // and equality operators.
4  #include <iostream> // allows program to perform input and output
5
6  using std::cout; // program uses cout
7  using std::cin; // program uses cin
8  using std::endl; // program uses endl
```

- After inserting the `using` declarations, we can write `cout` instead of `std::cout` in the remainder of the program.
  - `std::cin` → `cin`, `std::endl` → `endl`
- Many programmers prefer to use the declaration

  `using namespace std;`

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

2.7 Decision Making: Equality and Relational Operators

```
13    int number1; // first integer to compare
14    int number2; // second integer to compare
15
16    cout << "Enter two integers to compare: "; // prompt user for data
17    cin >> number1 >> number2; // read two integers from user
```

- This program uses cascaded stream extraction operations to input two integers.

2.7 Decision Making: Equality and Relational Operators

```
19    if ( number1 == number2 )
20       cout << number1 << " == " << number2 << endl;
21
22    if ( number1 != number2 )
23       cout << number1 << " != " << number2 << endl;
24
25    if ( number1 < number2 )
26       cout << number1 << " < " << number2 << endl;
27
28    if ( number1 > number2 )
29       cout << number1 << " > " << number2 << endl;
30
31    if ( number1 <= number2 )
32       cout << number1 << " <= " << number2 << endl;
33
34    if ( number1 >= number2 )
```

- Each `if` statement in Fig. 2.13 has a single statement in its body and each body statement is indented.
- In Chapter 3 we show how to specify `if` statements with multiple-statement bodies (by enclosing the body statements in a pair of braces, `{ }`, creating what's called a compound statement or a block).

2.7 Decision Making: Equality and Relational Operators

Good Programming Practice 2.12
Indent the statement(s) in the body of an `if` statement to enhance readability.

```
22    if ( number1 != number2 )
23       cout << number1 << " != " << number2 << endl;
24
25    if ( number1 < number2 )
26       cout << number1 << " < " << number2 << endl;
27
28    if ( number1 > number2 )
29       cout << number1 << " > " << number2 << endl;
30
31    if ( number1 <= number2 )
32       cout << number1 << " <= " << number2 << endl;
33
34    if ( number1 >= number2 )
35       cout << number1 << " >= " << number2 << endl;
36 } // end function main
```

Good Programming Practice 2.13
For readability, there should be no more than one statement per line in a program.

2.7 Decision Making: Equality and Relational Operators

Common Programming Error 2.8

Placing a semicolon immediately after the right parenthesis after the condition in an `if` statement is often a logic error (although not a syntax error). The semicolon causes the body of the `if` statement to be empty, so the `if` statement performs no action, regardless of whether or not its condition is true. Worse yet, the original body statement of the `if` statement now becomes a statement in sequence with the `if` statement and always executes, often causing the program to produce incorrect results.

```
if (number1 < number 2)
   cout << number1 << " < " << number2;
```

```
if (number1 < number 2) ;
   cout << number1 << " < " << number2;
```

2.7 Decision Making: Equality and Relational Operators

- Statements may be split over several lines and may be spaced according to your preferences.

```
cout << number1 << " < " << number2;
```

```
cout << number1
     << " < " << number2;
```

- It's a syntax error to split identifiers, strings (such as `"hello"`) and constants (such as the number 1000) over several lines.

Common Programming Error 2.9

It's a syntax error to split an identifier by inserting white-space characters (e.g., writing `main` as `ma in`).

2.7 Decision Making: Equality and Relational Operators

- The operators are shown top to bottom in decreasing order of precedence.
- All these operators, with the exception of the assignment operator =, associate from left to right.

Operators				Associativity	Type
()				left to right	parentheses
*	/	%		left to right	multiplicative
+	-			left to right	additive
<<	>>			left to right	stream insertion/extraction
<	<=	>	>=	left to right	relational
==	!=			left to right	equality
=				right to left	assignment

Fig. 2.14 | Precedence and associativity of the operators discussed so far.

2.7 Decision Making: Equality and Relational Operators

Good Programming Practice 2.15

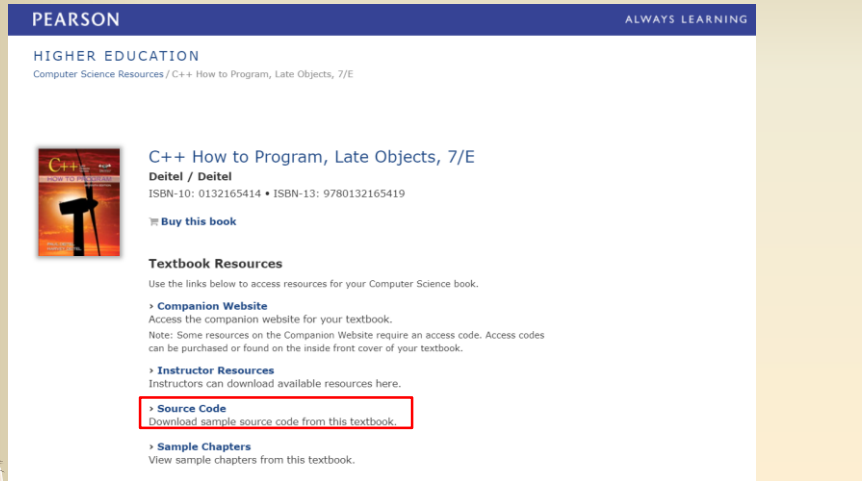
Refer to the operator precedence and associativity chart when writing expressions containing many operators. Confirm that the operators in the expression are performed in the order you expect. If you're uncertain about the order of evaluation in a complex expression, break the expression into smaller statements or use parentheses to force the order of evaluation, exactly as you'd do in an algebraic expression. Be sure to observe that some operators such as assignment (=) associate right to left rather than left to right.

Summary

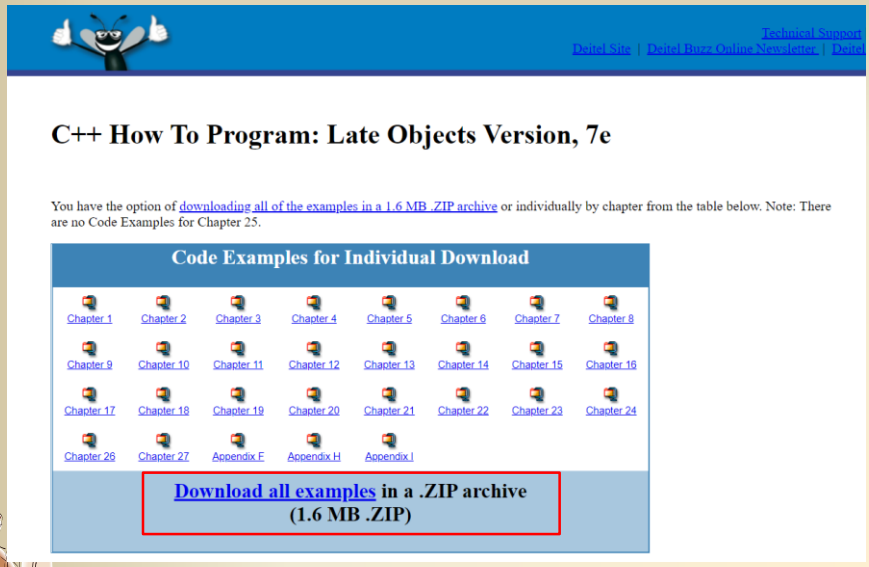
- The structure of a C++ program
- `std::cout` and `std::cin`
- Variable declarations
- Memory Concepts
- Arithmetic
- Decision making: `if` statement

How to Download Example Codes

- https://media.pearsoncmg.com/bc/abp/cs-resources/products/product.html#product,isbn=0132165414

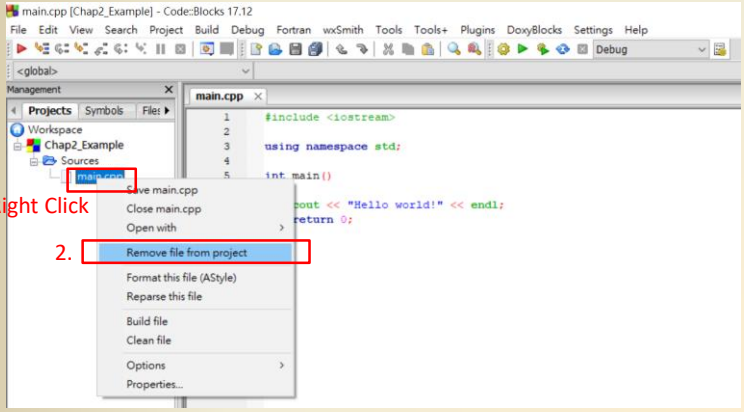


How to Download Example Codes



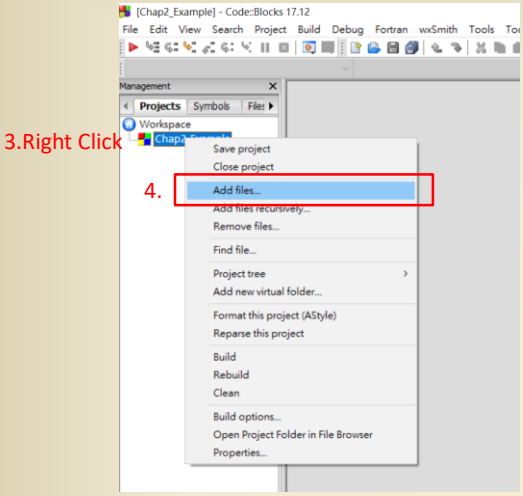
How to Execute Example Codes

- Create a new project in Code::Blocks
- Remove main.cpp

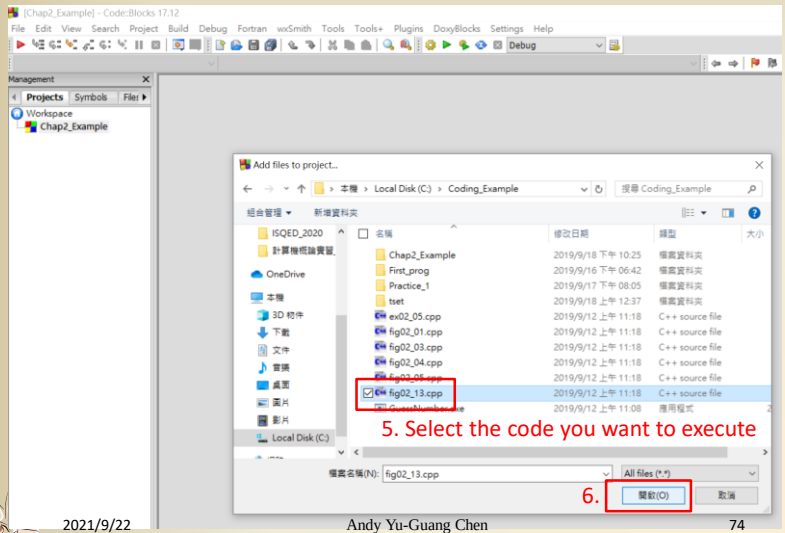


How to Execute Example Codes

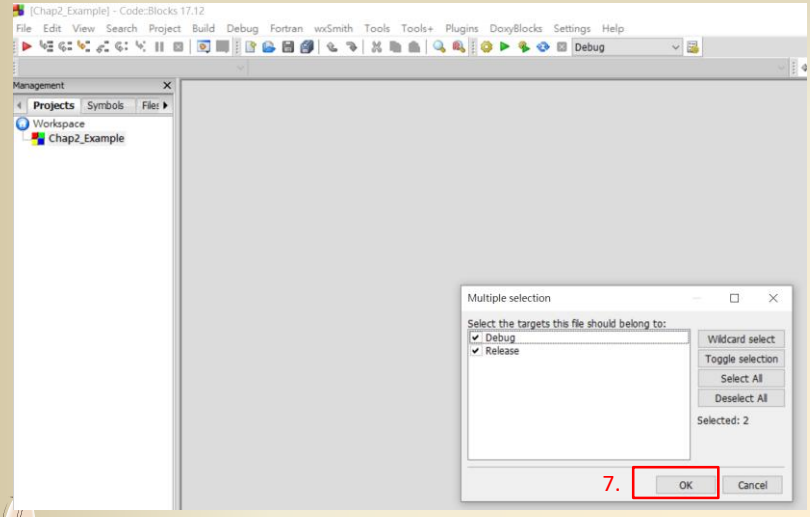
◆Add an example code



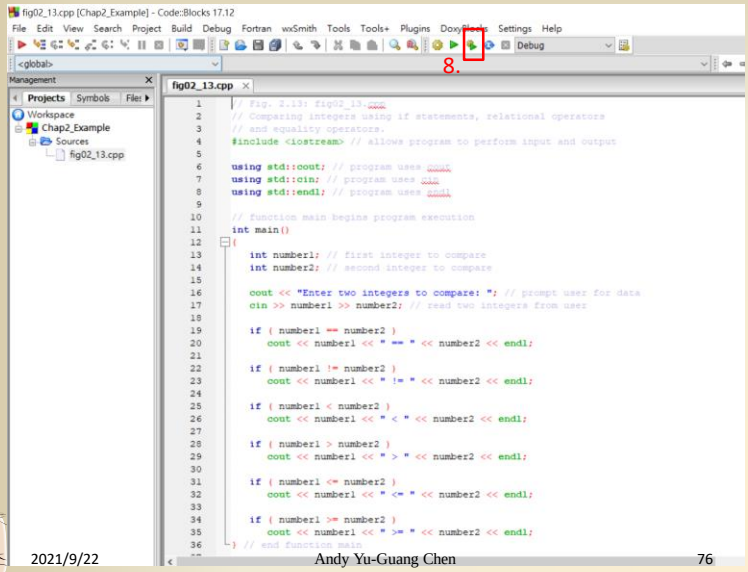
How to Execute Example Codes



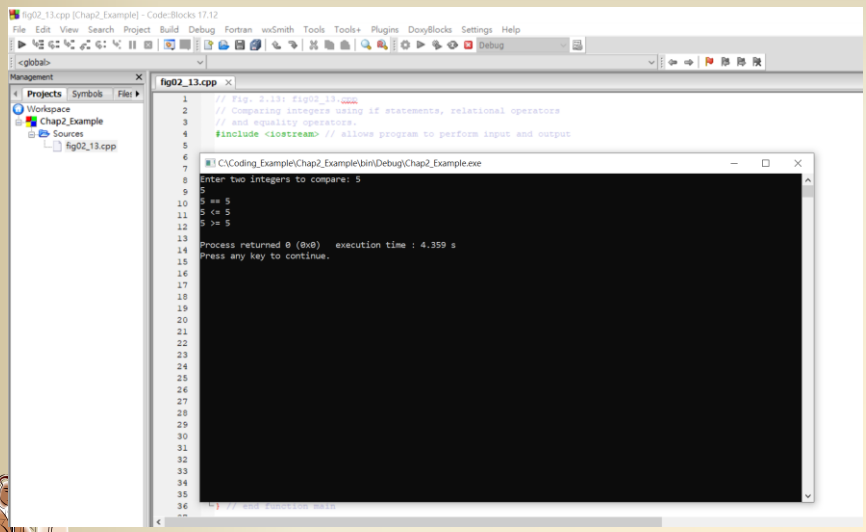
How to Execute Example Codes



How to Execute Example Codes



How to Execute Example Codes



Andy, Yu-Guang Chen

Assistant Professor, Department of EE, NCU

Email: andyygchen@ee.ncu.edu.tw