C++ Programming
Chapter 2 Introduction to C++ Programming

Yih-Peng Chiou

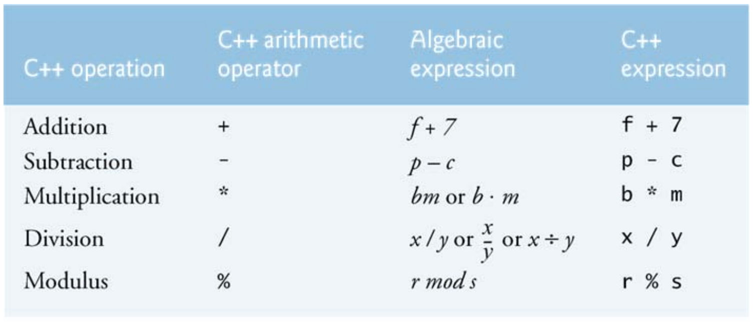
Room 617, BL Building

(02) 3366-3603

ypchiou@cc.ee.ntu.edu.tw

Photonic Modeling and Design Lab.
Graduate Institute of Photonics and Optoelectronics &
Department of Electrical Engineering
National Taiwan University

2.6 Arithmetic



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.

- Integer division yields an integer quotient. Any fractional part in integer division is discarded (i.e., truncated) — no rounding occurs.

Common Programming Error 2.3
Attempting to use the modulus operator (`%`) with noninteger operands is a compilation error.

2.6 Arithmetic

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.

Common Programming Error 2.3
Attempting to use the modulus operator (%) with noninteger operands is a compilation error.

2.6 Arithmetic

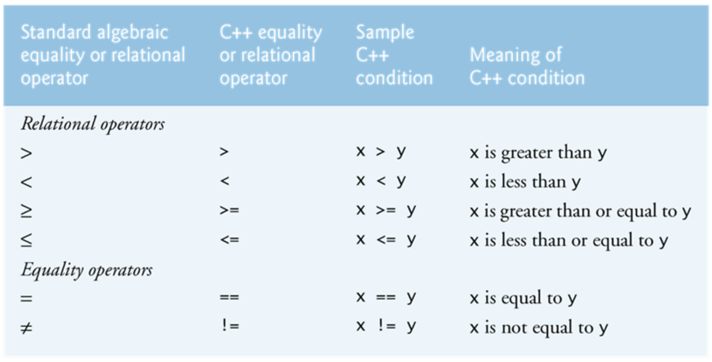
Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they're evaluated left to right.
*, /, %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

Fig. 2.10 | Precedence of arithmetic operators.

- It's acceptable to place unnecessary parentheses in an expression to make the expression clearer. These are called redundant parentheses.

2.7 Decision Making: Equality and Relational Operators

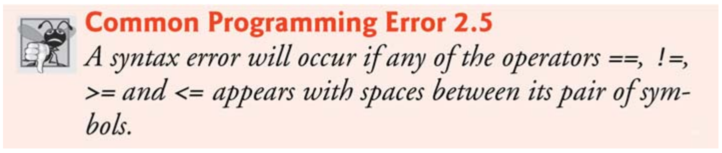
- if statement:
  - True => statement in the body of the if statement is executed
  - False => the body statement is not executed
  - The relational operators have higher level of precedence over the equality operators. (All rel. op. have the same level. All eq. op. have the same level.)



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

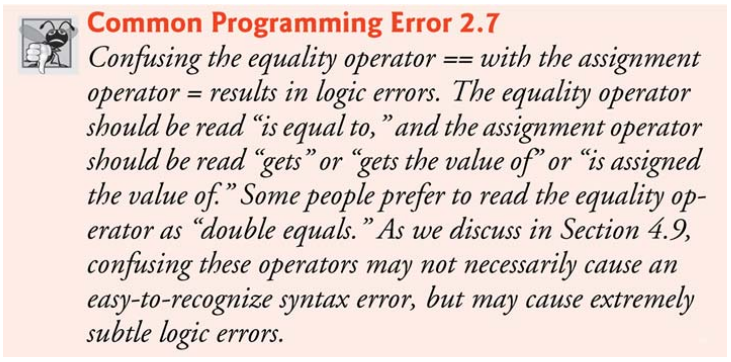
Fig. 2.12 | Equality and relational operators.

2.7 Decision Making: 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.



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 4.9, 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

```
// Fig. 2.13: fig02_13.cpp
// Comparing integers using if statements, relational operators
// and equality operators.
#include <iostream> // allows program to perform input and output

using std::cout; // program uses cout
using std::cin; // program uses cin
using std::endl; // program uses endl

// function main begins program execution
int main()
{
   int number1; // first integer to compare
   int number2; // second integer to compare

   cout << "Enter two integers to compare: "; // prompt user for data
   cin >> number1 >> number2; // read two integers from user

   if ( number1 == number2 )
      cout << number1 << " == " << number2 << endl;

```

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

```
   if ( number1 != number2 )
      cout << number1 << " != " << number2 << endl;

   if ( number1 < number2 )
      cout << number1 << " < " << number2 << endl;

   if ( number1 > number2 )
      cout << number1 << " > " << number2 << endl;

   if ( number1 <= number2 )
      cout << number1 << " <= " << number2 << endl;

   if ( number1 >= number2 )
      cout << number1 << " >= " << number2 << endl;
} // 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.)

- `using` declarations
  - eliminate the need to repeat the `std::` prefix
  - `cout` instead of `std::cout`, `cin` instead of `std::cin` and `endl` instead of `std::endl`
- using namespace std`;`
  - enables a program to use all the names in any standard C++ header file (such as `<iostream>`) that a program might include
- if statement
  - has a single statement in its body and each body statement is indented
  - 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.*

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

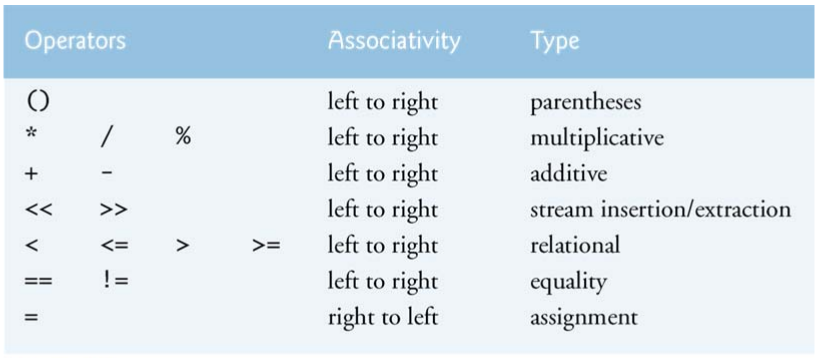
Good Programming Practice 2.14
A lengthy statement may be spread over several lines. If a single statement must be split across lines, choose meaningful breaking points, such as after a comma in a comma-separated list, or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines and left-align the group of indented lines.

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.*

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`*).*

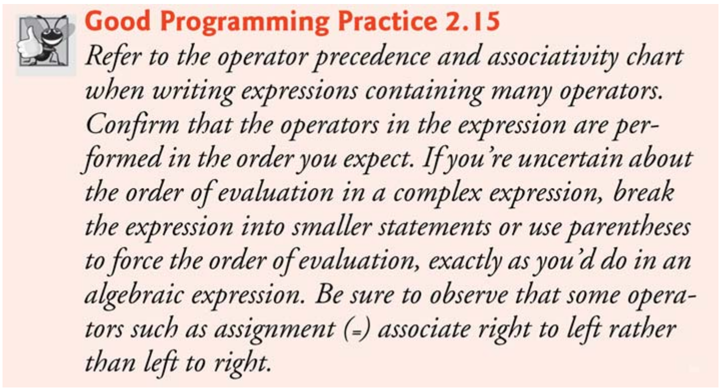
2.7 Decision Making: Equality and Relational Operators



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.