



Good Programming Practice 2.9

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



Common Programming Error 2.5

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.

White Space

Note the use of white space in Fig. 2.13. Recall that white-space characters, such as tabs, newlines and spaces, are normally ignored by the compiler. So, statements may be split over several lines and may be spaced according to your preferences. It's a syntax error to split identifiers, strings (such as "hello") and constants (such as the number 1000) over several lines.



Good Programming Practice 2.10

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.

Operator Precedence

Figure 2.14 shows the precedence and associativity of the operators introduced in this chapter. 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. Addition is left-associative, so an expression like `x + y + z` is evaluated as if it had been written `(x + y) + z`. The assignment operator `=` associates from *right to left*, so an expression such as `x = y = 0` is evaluated as if it had been written `x = (y = 0)`, which, as we'll soon see, first assigns 0 to `y`, then assigns the *result* of that assignment—0—to `x`.

Operators				Associativity	Type
()				[See caution in Fig. 2.10]	grouping 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.