**Programming Practices**

1. Good Programming Practice 2.1

Every program should begin with a comment that describes the purpose of the program.

1. Common Programming Error 2.1

Forgetting to include the <iostream> header in a program that inputs data from the keyboard

or outputs data to the screen causes the compiler to issue an error message.

1. Good Programming Practice 2.2

Indent the body of each function one level within the braces that delimit the function’s body. This makes a program’s functional structure stand out, making the program easier to read.

1. Common Programming Error 2.2

Omitting the semicolon at the end of a C++ statement is a syntax error. 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.

1. Good Programming Practice 2.3

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 prefer three spaces per level of indent.

1. Error-Prevention Tip 2.1

Although it’s not always necessary to initialize every variable explicitly, doing so will help you avoid many kinds of problems.

1. Good Programming Practice 2.4

Declare only one variable in each declaration and provide a comment that explains the variable’s purpose in the program.

1. 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 (and readability).

1. Good Programming Practice 2.5

Choosing meaningful identifiers helps make a program self-documenting—a person can understand the program simply by reading it rather than having to refer to program comments

or documentation.

1. Good Programming Practice 2.6

Avoid using abbreviations in identifiers. This improves program readability.

1. Good Programming Practice 2.7

Do not use identifiers that begin with underscores and double underscores, because C++ compilers use names like that for their own purposes internally.

1. Common Programming Error 2.3

Reversing the order of the pair of symbols in the operators !=, >= and <= (by writing them as =!, => and =<, respectively) is normally a syntax error. In some cases, writing != as =! will not be a syntax error, but almost certainly will be a logic error that has an effect at execution time. You’ll understand why when you learn about logical operators in Chapter 5. A fatal logic error causes a program to fail and terminate prematurely. A nonfatal logic error allows a program to continue executing, but usually produces incorrect results.

1. Common Programming Error 2.4

Confusing the equality operator == with the assignment operator = results in logic errors. We like to read the equality operator as “is equal to” or “double equals,” and the assignment operator as “gets” or “gets the value of” or “is assigned the value of.” As you’ll see in Section 5.12, confusing these operators may not necessarily cause an easy-to-recognize syntax error, but may cause subtle logic errors.

1. Good Programming Practice 2.9

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

1. Error-Prevention Tip 2.2

You don’t need to use braces, { }, around single-statement bodies, but you must include the braces around multiple-statement bodies. You’ll see later that forgetting to enclose multiple-statement bodies in braces leads to errors. To avoid errors, as a rule, always enclose an if statement’s body statement(s) in braces.

1. 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.

1. Good Programming Practice 2.10

A lengthy statement may be spread over several lines. If a 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.

1. Good Programming Practice 2.11

Refer to the operator precedence and associativity chart (Appendix A) 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.