deitel-good-programming-practice

Derek Harter

January 20, 2015

Contents

1 Chapter 1

- 1.1 Write your C++ programs in a simple and straightforward manner. This is sometimes referred to as KISS ("keep it simple stupid"). Do not "stretch" the language by trying bizarre usages.

2 Chapter 2

- 2.1 Every program should begin with a comment that describes the purpose of the program.
- 2.2 Use blank lines, space characters and tabs to enhance program readability.

- 2.3 Indent the entire body of each function one level within the braces that delimit the body of the function. This makes a program's functional structure stand out and makes the program easier to read.
- 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. (I require 2 spaces for all indentation).
- 2.5 Place a space after each comma (,) to make programs more readable.
- 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.
- 2.7 Avoid using abbreviations in identifiers. This promotes program readability.
- 2.8 Avoid identifiers that begin with underscores and double underscores, because C++ compilers may use names like that for their own purposes internally. This will prevent names you choose from being confused with names the compilers choose.
- 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.
- 2.10 Place spaces on either side of a binary operator. This makes the operator stand out and makes the program more readable.
- 2.11 Using redundant parentheses in complex arithmetic expressions can make the expressions clearer.
- 2.12 Indent the statement(s) in the body of an if statement to enhance readability.
- 2.13 For readability, there should be no more than one statement per line in a program.
- 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.

- 2.15 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 are 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.

3 Chapter 3

- 3.2 Choosing meaningful function names and meaningful parameter names makes programs more readable and helps avoid excessive use of comments.
- 3.3 Place 2 blank lines between member/function definitions to enhance program readability.

4 Chapter 4

- 4.1 Consistently applying reasonable indentation conventions throughout your programs greatly improves program readability. We suggest two blanks per indent. Some people prefer using tabs, but these can vary across editors, causing a program written on one editor to align differently when used with another.
- 4.2 Whatever indentation convention you choose should be applied consistently throughout your programs. It's difficult to read programs that do not obey uniform spacing conventions.
- 4.3 Indent both body statements of an if ... else statement.
- 4.4 If there are several levels of indentation, each level should be indented the same additional amount of space to promote readability and maintainability.

- 4.5 Always putting the braces in an if ... else statement (or any control statement) helps prevent their accidental omission, especially when adding statements to an if or else clause at a later time. To avoid omitting one or both of the braces, some programmers prefer to type the beginning and ending braces of blocks even before typing the individual statements within the braces.
- 4.6 Separate declarations from other statements in functions with a blank line for readability.
- 4.7 Declare each variable on a separate line with its own comment for readability.
- 4.9 Unlike binary operators, the unary increment and decrement operators should be placed next to their operands, with no intervening spaces.

5 Chapter 5

- 5.1 Put a blank line before and after each control statement to make it stand out in the program.
- 5.2 Too many levels of nesting can make a program difficult to understand. As a rule, try to avoid using more than three levels of indentation.
- 5.3 Vertical spacing above and below control statements and indentation of the bodies of control statements give programs a two-dimensional appearance that improves readability.
- 5.10 Provide a default case in switch statements. Cases not explicitly tested in a switch statement without a default case are ignored. Including a default case focuses you on the need to process exceptional conditions. There are situations in which no default processing is needed. Although the case clauses and the default case clause in a switch statement can occur in any order, it's common practice to place the default clause last.

6 Chapter 6

- 6.1 Capitalize the first letter of an identifier used as a user-defined type name. (e.g. a Class or Struct name, for example)

- 6.2 Use only uppercase letters in constant names. This makes these constants stand out in a program and reminds you that constants are not variables.

7 Chapter 7

- 7.2 Defining the size of an array as a constant variable instead of a literal constant makes programs clearer. This technique eliminates so-called magic numbers. For example, repeatedly mentioning the size 10 in array-processing code for a 10-element array gives the number 10 an artificial significance and can be confusing when the program includes other 10s that have nothing to do with the array size.