Chapter 9: Rules

Chapter 1:Style and Program Organization

Rule 1-1:

Organize programs for readability, just as you would expect an author to organize a book.

Rule 1-2:

Divide each module up into a public part (what's needed to use the module) and a private part (what's needed to get the job done). The public part goes into a .h file while the private part goes into a .c file.

Rule 1-3:

Use white space to break a function into paragraphs.

Rule 1-4:

Put each statement on a line by itself

Rule 1-5:

Avoid very long statements. Use multiple shorter statements instead.

Chapter 2:File Basics, Comments, and Program Headings

Rule 2-1:

Keep programs files to no longer than about 2,000 to 3,000 lines.

Rule 2-2:

Keep all lines in your program files down to 72 characters or fewer.

Rule 2-3:

Use 8-character tab stops.

Rule 2-4:

Use only the 95 standard ASCII characters in your programs. Avoid exotic characters. (Foreign characters may be used if you are writing comments in a foreign language.)

Rule 2-5:

Include a heading comment at the beginning of each file that explains the file.

Rule 2-6:

Leave out unnecessary comments if they require maintenance and if you are unlikely to maintain them.

Rule 2-7:

Comment your code as you write it.

Chapter 3:Variable Names

Rule 3-1:

Use simple, descriptive variable names.

Rule 3-2:

Good variable names are created by using one word or by putting two or three words together, separated by "_". For example:

Rule 3-3:

Never use `I` *(lowercase* `L`*) or* `O` *(uppercase* `O`*) as variable or constant names.*

Rule 3-4:

Don't use the names of existing C library functions or constants.

Rule 3-5:

Don't use variable names that differ by only one or two characters. Make every name obviously different from every other name.

Rule 3-6:

Use similar names for variables that perform similar functions.

Rule 3-7:

When creating a two word variable name where the words can be put in any order, always put the more important word first.

Rule 3-8:

Standard prefixes and suffixes are `_ptr`*,* `_p`*,* `_file`*,* `_fd`*, and* `n_`*.*

Rule 3-9:

Short names such as `x`*,* `y`*, and* `i` *are acceptable when their meaning is clear and when a longer name would not add information or clarity.*

Rule 3-10:

Use `argc` *for the number of command line arguments and* `argv` *for the argument list. Do not use these names for anything else.*

Rule 3-11:

Follow every variable declaration with a comment that defines it.

Rule 3-12:

Whenever possible, include the units of measure in the description of a variable.

Rule 3-13:

Name and comment each field in a structure or union like a variable.

Rule 3-14:

Begin each structure or union definition with a multi-line comment describing it.

Rule 3-15:

Put at least one blank line before and after a structure or union definition.

Rule 3-16:

When you can't put a descriptive comment at the end of a variable declaration, put it on a separate line above. Use blank lines to separate the declaration/comment pair from the rest of the code.

Rule 3-17:

Group similar variables together. When possible, use the same structure for each group.

Rule 3-18:

Don't use hidden variables.

Rule 3-19:

Use the names INT16, INT32, UINT16, and UINT32 for portable application

Rule 3-20:

Floating-point numbers must have at least one digit on either side f the decimal point.

Rule 3-21:

The exponent in a floating-point number must be a lowercase `e`*. This is always followed by a sign.*

Rule 3-22:

Start hexadecimal numbers with `0x`. *(Lowercase* `x` *only.)*

Rule 3-23:

Use uppercase A through F when constructing hexadecimal constants.

Rule 3-24:

Long constants should end with an uppercase `L`.

Chapter 4:Statement Formatting

Rule 4-1:

Write one statement per line.

Rule 4-2:

Put spaces before and after each arithmetic operator, just like you put spaces between words when you write.

Rule 4-3:

Change a long, complex statement into several smaller, simpler statements.

Rule 4-4:

In a statement that consists of two or more lines, every line except the first must be indented an extra level to indicate that it is a continuation of the first line.

Rule 4-5:

When writing multi-line statements, put the arithmetic and logical operators at the end of each line.

Rule 4-6:

When breaking up a line, the preferred split point is where the parenthetic nesting is lowest.

Rule 4-7:

Align like level parentheses vertically.

Rule 4-8:

Split long **for** *statements along statement boundaries.*

Rule 4-9:

Always split a **for** *statement into three lines.*

Rule 4-10:

Write **switch** *statements on a single line.*

Rule 4-11:

Keep conditionals on a single line if possible.

Rule 4-12:

*When splitting a conditional clause (***? :***), write it on three lines: the condition line, the true-value line, and the false-value line. Indent the second and third line an extra level.*

Rule 4-13:

Avoid side effects.

Rule 4-14:

Put the operator ++ and -- on lines by themselves. Do not use ++ and -- inside other statements.

Rule 4-15:

Never put an assignment statement inside any other statement.

Rule 4-16:

If putting two or more statements on a single line improves program clarity, then do so.

Rule 4-17:

When using more than one statement per line, organize the statement into columns.

Rule 4-18:

Indent one level for each new level of logic.

Rule 4-19:

The best indentation size is four spaces.

Chapter 5:Statement Details

Rule 5-1:

Always put a comment in the null statement, even if it is only

Rule 5-2:

In C expressions, you can assume that `*`*,* `/`*, and % come before* `+` *and* `-`*. Put*

parentheses around everything else.

Rule 5-3:

Use ANSI style function declarations whenever possible.

Rule 5-4:

When using K&R parameters, declare a type for every parameter.

Rule 5-5:

When using K&R parameters, put the type declarations for the parameters in the same order as the occur in the function header.

Rule 5-6:

Always declare a function type

Rule 5-7:

Always declare functions that do not return a value as **void**.

Rule 5-8:

Allow no more that five parameters to a function.

Rule 5-9:

Avoid using global variables where function parameters will do.

Rule 5-10:

Avoid variable length parameter lists. They are difficult to program and can easily cause trouble.

Rule 5-11:

When an if affects more than one line, enclose the target in braces.

Rule 5-12:

In an **if** *chain, treat the words* **else** *if as one keyword.*

Rule 5-13:

Never use the comma operator when you can use braces instead.

Rule 5-14:

When looping forever, use `while (1)` *instead of* `for(;;)`.

Rule 5-15:

Avoid using **do/while**. *Use* **while** *and* **break** *instead.*

Rule 5-16:

Use the comma operator inside a for statement only to put together two statements. Never use it to combine three statements.

Rule 5-17:

Use one printf per line of output.

Rule 5-18:

Unless extreme efficiency is warranted, use printf instead of puts and putc.

Rule 5-19:

Start **goto** *labels in the first column.*

Rule 5-20:

End every case in a **switch** *with a* **break** *or the comment* `/* Fall Through */`

Rule 5-21:

Always put a break at the end of the last case in a switch statement.

Rule 5-22:

Always include a **default** *case in every* **switch**, *even if it consists of nothing but a null statement.*

Chapter 6:Preprocessor

Rule 6-1:

#define *constants are declared like variables. Always put a comment describes the constant after each declaration.*

Rule 6-2:

Constant names are all upper-case.

Rule 6-3:

If the value of a constant is anything other than a single number, enclose it in parentheses.

Rule 6-4:

The use of **const** *is preferred over* **#define** *for specifying constants.*

Rule 6-5:

When possible, use **typedef** *instead of* **#define**.

Rule 6-6:

Don't use **#define** *to define new language elements.*

Rule 6-7:

Never use **#define** *to redefine C keywords or standard functions.*

Rule 6-8:

Enclose parameterized macros in parentheses.

Rule 6-9:

Enclose each argument to a parameterized macro in parenthesis.

Rule 6-10:

Always enclose macros that define multiple C statements in braces.

Rule 6-11:

If a macro contains more than one statement, use a do/while structure to enclose the macro. (Don't forget to leave out the semicolon of the statement).

Rule 6-12:

When creating multi-line macros, align the backslash continuation characters (\) in a column.

Rule 6-13:

Always comment any parameterized macros that look like functions.

Rule 6-14:

#include *directives come just after the heading comments. Put system includes first, followed by local includes.*

Rule 6-15:

Do not use absolute paths in **#include** *directives. Let the -I compile opt*

Rule 6-16:

Comment **#else** *and* **#endif** *directives with the symbol used in the initial* **#ifdef** *or* **#endif** *directive.*

Rule 6-17:

Use conditional compilation sparingly. Don't let the conditionals obscure the code.

Rule 6-18:

Define (or undefine) conditional compilation control symbols in the code rather than using the `-D` *option to the compiler.*

Rule 6-19:

Put #define and #undef statements for compilation control symbols at the beginning of the program.

Rule 6-20:

Do not comment out code. Use conditional compilation (`#ifdef UNDEF`*) to get rid of unwanted code.*

Rule 6-21:

Use `#ifdef QQQ` *to temporarily eliminate code during debugging.*

Chapter 7:Directory Organization and Makefile Style

Rule 7-1:

Whenever possible, put all the files for one program or library in one directory.

Chapter 8:User-Friendly Programming

Rule 8-1:

Law of Least Astonishment: The program should act in a way that least astonishes the user.

Rule 8-2:

Begin each error message with Error:. Begin each warning message with Warning:.

Rule 8-3:

Don't let users do something stupid without warning them.

Chapter 9:Rules