Braces { }

- Braces shall always surround blocks of code, following `if`, `else`, `switch`, `while`, `do` and `for` statements. Single and empty statements shall also be surrounded by braces.
- The left brace shall appear by itself on the line below the start of the block of code. The right brace shall appear in the same position later in the file.

Example:

```
if ((a < b) && (b < c))
{
   result = (3 * a) + b;
}
return result;
```

Parentheses ( )

Do not rely on C operator precedence rules. To add clarity, use parentheses. See example above.

Keywords to avoid

The `goto` and `continue` keywords shall not be used. The `break` keyword shall not be used outside of a `switch` statement. These keywords lead to spaghetti code.

Keywords to use

Use the `volatile` keyword to declare a global variable accessible by an interrupt service routine.

Comments

- Do not use nested comments
- Do not use C-style comments `/*.....*/` to disable a ***block*** of code.
- Use conditional compilation to disable a block of code: `#if (0) …#endif`.
- The most useful comments generally precede a block of code. The comments should be at the same indentation level as the block of code. A blank line shall follow the code block.
- Avoid explaining the obvious.

White Space

- `If`, `else`, `while`, `for`, `switch` and `return` shall be followed by one space.
- Assignment operators `=`, `+=`, `-=`, `/=`, `%=`, `&=`, `|=`, `^=`, `~=`, and `!=` shall be preceded and followed by one white space.
- Binary operators `+`, `-`, `*`, `/`, `%`, `<`, `<=`, `>`, `>=`, `==`, `!=`, `<<`, `>>`, `&`, `&&`, and `||` shall be preceded and followed by one white space.
- Unary operators `+`, `-`, `++`, `--`, `!`, and `~`, shall be written without a space on the operand side and with one space on the other side.
- The pointer operators `*` and `&` shall be written with a space on each side within declarations but otherwise without a space on the operand side.
- The left and right brackets of the array subscript operator `[` and `]` shall be without surrounding spaces.
- There shall be no white spaces adjacent to the left and right parenthesis characters and an expression within the parentheses.

- The left and right parentheses of the function call operator shall be without surrounding spaces, except the function declaration which shall have one space between the function name and left parenthesis to allow it to be easily located.
- Each comma separating function parameters shall be followed by one space.
- Each semicolon separating the elements of a `for` statement shall be followed by one space.
- Each semicolon shall follow the statement it terminates without a preceding space.

Blank Lines

- No line of code shall contain more than one statement.
- There shall be a blank line before and after each natural block of code. Examples of natural blocks of code are loops, `if-else` and `switch` statements and consecutive declarations.

Indentation

- Each indentation level within a module should consist of 3 spaces.
- Do not use tabs for indentation except if the editor allows you to setup the tab-key to use spaces. (True for the Renesas editor.)
- Within a `switch` statement, each `case` statement should be indented and the contents of the case block should be indented once more.
- Whenever a line of code is too long to fit within the maximum line width, indent the second and subsequent lines in the most readable manner possible.
- Hint: If the source file is published using Microsoft Word ensure that a fixed width font is used (examples: `Courier New` or `MS Mincho`).

Example:

```
sys_error_handler(int error)
{
   switch (err)
   {
      case ERR_ONE:
         ……;
         ……;
      break;

      default:
         ……;
         ……;
      break;
   }

   if ((very_long_comparison_here
         && second very_long_comparison_here)
         || third very_very_long_comparison_here)
   {
      ……;
      ……;
   }
}
```

Tabs

Tabs shall not be used. (Tabs vary by editor and programmer preference.)

Modules

- Module names shall consist of lowercase letters, numbers and underscores.

- All module names shall be unique in their first eight characters, with `.h` and `.c` used for the suffixes.
- No module name shall share the name of a standard library header file. For example “stdio” or “math”.
- Any module containing a `main()` function shall have the word “main” in its filename.

Header Files

- There shall be one header file for each source file and they shall have the same root name.
- The header file shall identify only the procedures, constants and data types about which it is strictly necessary for other modules to know about.
- No storage for variables shall be declared in a header file.
- No header file shall contain a `#include` statement.

Source Files

- Each source file shall comprise some or all of the following sections, in the order listed: Comment block, include statements, data type and constant definitions, static data declarations, private function prototypes, public function bodies, private function bodies.
- Each source file shall `#include` the header file of the same name.
- Source files shall be free of unused include files.
- No source file shall include another source file.

Procedures and Functions

- No procedure shall have a name that is a keyword of C. Examples: “interrupt”, “inline”, “true”.
- No function name shall contain any uppercase letters.
- Underscores shall be used to separate words in procedure names.
- The procedure’s name shall be descriptive of its purpose: Use noun-verbs (example: `adc_read()`) or alternatively name the procedure according to the question answered (example: `input_is_high()`).
- Public functions shall be prefixed with the module name.
- Examples: `display_lcd_initialise()`, `display_led_is_on()`
- Effort shall be taken to limit the length of functions to 100 lines.
- All functions shall have one exit point at the bottom of the function. That is, the `return` keyword shall appear only once.
- Parameters shall be explicitly declared and meaningfully named.

Variables and constants

- No variable shall have a name that is a keyword of C.
- No variable shall contain uppercase letters.
- Underscores shall be used to separate words in variable names.
- Each variable name shall be descriptive of its purpose.
- Global variable names shall begin with the letter ‘g’. For example `g_zero_offset`.
- Pointer variable names shall begin with the letter ‘p’. For example `*p_led_reg`.
- Boolean variables shall begin with the letter ‘b’. For example a global boolean variable: `gb_is_buffer_full`
- Use `#define` rather than magic numbers. Use capitals for constants.
- Note: To make defined values available in more than one file, place the definitions in a header file and include the header file in any .c file that uses the defined value.

Example:

```
#define MAX_LOOPS 10
```

```
int loop;
for (loop = 0; loop < MAX_LOOPS; loop++)
{
   ……;
}
```

Variable declarations

Do not put pointers and other variables on the same line. Example:

```
char * x, y;    // No
char * x;       // Yes
char y;         // Yes
```

If-Else statements

- Preferably place the shortest of the `if` and `else-if` clauses first.
- Nested `if-else` clauses should not be deeper than two levels.
- Any `if` statement with an `else-if` clause shall end with an `else` clause.
- When evaluating a variable and a constant, place the constant on the left side of the comparison.

Example:

```
if (ON == power_supply)    // "if (ON = power_supply)" will result in an error
                           // message, but not "if (power_supply = ON)".
{
   ……;                     // Placing the shorter clause here and the longer
                           // clause last will make the logic easier to follow.
}
else if (OFF == power_led)
{
   ……;
}
else                       //similar to "default " in a switch statement
{
   ……;
   ……;
   ……;
   ……;
}
```

Switch statements

- The `break` for each `case` shall align with the associated `case`.
- All switch statements shall contain a `default` block.

Example:

```
switch (err)
{
   case ERR_A:
      ……;
   break;

   case ERR_B:
      ……;
   break;

   default:
      ……;
   break;

}
```

Loops

- Use constants rather than magic numbers to terminate loops.

Example:

```
for (int loop = 0; loop < MAX_LOOPS; loop++)   //constants should be in capitals
{
   ……;
}
```

- Use
  ```
  while(1);
  ```
  or
  ```
  for(;;);
  ```
  for infinite loops.

Most of the recommendations listed in this document are from: *Netrino Embedded C Coding Standard*

Reference