C Coding Conventions

# Consistency

As always, when extending existing code, **always** use the same conventions used by the rest of the code.

# Spaces

* Split your functions into code segments separated by a single empty line
* Leave an empty line between functions
* Operators are to be enclosed by one space on each side (apart from the negativity minus, pointer dereference and address of)
* A space will come after a comma and not before it

# Blocks

* Braces as block openers/closers (“{}”) should reside in separate lines (unlike JavaScript, for example)
* Braces are not indented, the lines within are
* Case labels (in switch statements) are not to be indented

# Header Files

* Use double-inclusion guards:

#ifndef EXAMPLE\_H

#define EXAMPLE\_H

... /\* body of example.h file \*/

#endif /\* EXAMPLE\_H \*/

* Header files needed by the source file and not by the header file should be included in the source file only

# Comments and Documentation

* Block comments should use C-style comments, where each line starts with \* and is aligned to the opening /\*. The opening and ending lines should not contain any text. Example:

/\*

* Here is a block comment.
* The comment text should be tabbed or spaced over uniformly.
* The opening slash-star and closing star-slash are alone on a line.

\*/

* Block comments inside a function are appropriate, and they should be tabbed over to the same tab setting as the code that they describe
* Each file, function and struct should begin with a block comment describing its purpose
* For functions, the comment must describe the arguments and return value
* For structs, an short inline comment describing each of the fields is recommended
* Prefer comments describing an entire code block over short comments describing each line
* Document *what* the code is doing, and not *how* it is doing it. If the *how* is non-trivial, consider linking to an external document describing the operations done by the code
* Do not nest comments (e.g. /\* External comment /\* internal comment \*/ back to external \*/) – it is not supported by the language and might lead to errors
* Do not use c++ comments (“//”), as older compilers might not support them

# Names

* Names should always be meaningful, except for names of very local variables (e.g, i is ok for loop iterators)
* Names of functions should be in PascalCasing (each word beginning with a capital letter)
* Names of variables should be in camelCasing (beginning with small letters, each new work beginning with a capital letter)
* Names of constants should be uppercase (e.g. GOLDEN\_RATIO)

# Variables

* The ‘‘pointer’’ qualifier, ‘\*’, should be with the variable name rather than with the type.

char \*s, \*t, \*u;

instead of

char\* s, t, u;

which is wrong, since ‘t’ and ‘u’ do not get declared as pointers.

* Always initialize your variables

# General

* Lines should not be longer than 80 chars (including indentation)
* **No** magic numbers! Use defines and enums.
* When writing conditions, place constant values on the left to avoid assignment (=) instead of comparison (==) errors:

if (NULL == data\_structure)

* Write const-correct code: Use the const keyword for arguments which are not modified by the function (not needed for arguments passed by value)
* Don’t wrap your return values with parentheses (e.g. return 0 and not return(0))
* There should only be one statement per line
* Always check return values of functions for errors