Chad Standard

Draft 1, 2024, July 18th

1 Preface

A standard is meant to define a set of rules and guidelines, simply put, this document does that in a shorthand fashion. For any self-respecting project, the rulings here should lead to a decently readable codebase with a strong consistency. Above the standard there is only one rule: the Rule Of Consistency, prefer being self-consistence to any strict compliance.

Authors Written by *Emil Williams*, and was designed by an Anonymous contributor and *Ognjen M. Robovic*. The authors of this text democratically came to these outcomes over rational discussion with full agreement on each point.

This Document Being designed with only the C Programming Language in mind. Rulings should not apply to other languages. This document was invented by the means of LaTeX and by no other means could it possibly have been made.

2 Standard

1. **Line Width** 72 or higher (ex. 80, 120, 240, etc.)
2. **NULL Terminator In String Literal** Always use: $\backslash 00 \rightarrow \backslash 0$
3. **Hexadecimal Literals In String Literal** 2 digit hexadecimal values: $\backslash xDE \backslash xAD$
4. **String Literal NULL Terminator Breaks**
   "text$\backslash 0$" "more text$\backslash 0$"
   NULLs should be grouped: "$\backslash 0 \backslash 0$"
   Given that your creating an explicit String Literal containing many cstrings, individual NULLs may be given their own group.
5. **C99 Leading Commas** In enums or compound literals, leave a leading comma. If you're writing for C89, then you should never use Leading Commas.
6. **Do Not Use** $union$ **or** $struct$ **In Declarations** A $union$ or $struct$ may be stated such that variable a of struct type b would be declared as $struct\ b\ a$ rather than $b\ a$.
7. **C99 Compound Literals** If you explicitly set some item in the structure or union, you must do this for all other items in the structure.

```
typedef struct {
int a, b;
} type;

type a = {
.a = 1,
.b = 2,
};
```

8. **No Relative Header Paths** Do not use relative pathing in headers, instead use compiler flags or full paths to enable your compilation.
9. **Define At Beginning Header Guard**
   Use this kind of header guard:

```
#if HEADER_H
#define HEADER_H
...
#endif // HEADER_H
```

10. **No Double Blank Line** Shortens code and removes blank redundancy, use Single Blank Lines as much you want.
11. **Namespacing For Externally Exposed Headers & Variables** Add a namespace suffix for headers & variables meant to be used outside your project.
12. **Acceptable Naming** Names should be of one format and only one format internally. The endorsed naming style is_snake_case, snake_case_for_all.
13. **Use Tabs And Spaces** Effective use of both tabs and spaces aids in code density and universal styling regardless of tabwidth.
14. **Parenthesize All Macro Definitions** This is to insure precedent is maintained such that macros will be computed to one value.

15. **Include Padding After Commas** Do, This, In a list, so you can, separate things better, visually. rather,than,this,pile,of,nonsense.

16. $extern$ **All Function Declarations** Visually signifies your declarations.

    ```
    extern int func (int a, int b);
    ```

17. **Padding For All** Functions, Declarations, and Math should all have padding after their respective incantation, and before if and when applicable.

    ```
    int func (int a, int b) {
    int c = a * b + 1;
    return a + b + c;
    }

    func (2, 5);
    ```

18. **Center Pointer Declaration**

    ```
    void * pointer;
    ```

    even for typecasting

    ```
    (int *) malloc (sizeof (int));
    ```

19. **Sparse Pointers Types**

    ```
    void * *   double_pointer;
    void * * * triple_pointer;
    void *     single_pointer;
    ```

20. **Always Specify** $malloc$ **This Way**

    ```
    type a variable = (type a) malloc (sizeof (type b) * number of items);
    ```

21. **Single Line Parameters** Function parameters should remain on the same line. If a function has way too many parameters (line length greater than limit), format them into a block such that they are placed after the initial open parentheses.

22. **Alignment Of Variable Names** Align adjacent declarations horizontally.

23. **Alignment Of Assignments** Align adjacent assignments horizontally.

24. **always** $typedef$ $enum$**,** $struct$**, and** $union$ **constructs**

25. **Anonymous Structures** You may use these if you wish to.

26. **Always Bracket** With the only exception being $else\ if$. This includes switch statement case bodies.

27. **Full body** $switch$ **Format** Bodies may be either Multi-line or single line, but the body must be put inside braces.

    ```
    switch (...) {
    case ...: {
    break;
    }
    }
    ```

    Cases do not need to have break statements, and do not have any restriction or suggestion on control flow.

28. **Left Handed if Format**

```
if (!a
&&   b) {
...
}
```

29. **Reduce if Format** Avoid Pointless Comparisons

```
if (a)
```

instead of:

```
if (a != 0)
```

30. **Preferred Iterators i, j, k,** ...
31. **Preferred Iterator $while(1)$** Decide on whether you should obey this based on your compiler, if this is not equivalent to $for(;;)$ then use that instead.
32. **Prefer Prefix Decrement $--x$ Increment $++x$** If you must use the de/increment operator, then use Prefix Increment unless you can justify use of Suffix Increment.
33. **Use Preprocessor Comments To Comment Out Code**
They're actually recursive:
34. **C99 Style Declarations For Loop Iterators** Including the declaration is acceptable if there's no other use of the iterator, otherwise the declaration should be in the scope above.
35. **Parentheses For Absolutely Everything**

```
return (1);
```

```
#if 0
...
#endif // 0
```

3 Project Advice

These are just some suggestions that are per-project and are not to be followed word to word. Pick your poison and deception.

1. **Provide Makefile** It's generally universal in the Unix world.
2. **Provide @BAKE compilation header** For small projects :) https://github.com/emilwilliams/bake
3. **Cure, Sanitize, Analyze**
   Make sure to clear compiler warnings, such as those generated by gcc(1) and $clang$(1) under $-Wall$ $-Wextra$ $-Wpedantic$; runtime issues found by sanitation tools such as $-fsanitize = address, undefined, bounds$ or $valgrind$(1); and static analysis tool such as $split$(1)

   If you fail to preform these acts, the man of two dashes will be displeased, and you will lose 88% of your maximum health for 2d144 turns.
4. **You Don't Need More Than A Triple Pointer** It's rather excessive.
5. **You Don't Need More Than Three Iterators** It's rather excessive, unless your implementing an algorithm or larger system that needs it, even then it may be more reasonable to isolate these systems and then $static\ inline$ them.
6. **You Don't Need More Than 4 Levels Of Indentation** You should put that inside a function!
7. **Don't You Dare Use C Style Comments, Emil!** C Style Comments are not justified unless you absolutely need them, C++ Style Comments in all normal cases are useful for logical separation and denser comments. C Style Comments are for larger explanations, NOT for commenting out blocks of code!
8. **Use C Style Comments Always And Vanquish C++ Demons From Your Code, Anon!** Given that you hate C++ (obviously), you can vanish them by using C Style Comments always because you don't know how to make Emacs not do that by default, and you don't feel like fixing it or learning to type //.
9. **Use POSIX Reserved $\_t$ Suffix For New Types**