C Style Guide

Based on Google C++ Style Guide

Background

C语言作为一种底层高效的系统编程语言，在嵌入式开发、驱动开发、系统内核、性能关键模块等领域中仍占据核心地位。然而，其自由度极高，缺乏内建的安全机制，容易引发维护困难、内存错误、可读性差等问题。随着项目规模扩大与团队协作增多，不一致的编码风格会迅速演化为难以维护、难以复用、难以测试的技术债。

因此，我们需要建立一套标准化、清晰、一致的编码规范，统一代码风格，增强可读性和安全性，提升代码质量，并使代码更容易进行静态分析、单元测试、审查和迁移。

本规范参考 Google C++ Style Guide风格及工业实践经验，制定符合现代软件工程的 C 编码管理体系，适用于中大型 C 项目及团队协作开发场景。

Goals of the Style Guide

本指南旨在实现以下目标：

- 一致性
  所有团队成员书写的 C 代码在风格和结构上保持一致，降低认知成本。
- 可读性
  提高代码的清晰度，使非原作者也能快速理解并修改代码。
- 可维护性
  降低因风格不一致、命名混乱和结构设计差带来的维护负担。
- 安全性
  通过避免危险函数、限制隐式类型和强制边界检查，降低程序运行时的风险。
- 易审查与易测试
  提高代码的可测试性和可审查性，便于团队代码评审与自动化测试。
- 兼容工具链
  与静态分析工具（如 clang-tidy, cppcheck）和 CI 工具集成良好，便于自动风格检查与质量保障。
- 鼓励最佳实践
  在不限制开发自由的前提下，倡导良好的编码习惯，提升工程质量。

Required

以下规则必须遵守，任何违反将视为编码错误，禁止提交到主分支。

文件结构

1. 所有源文件必须以 `.c` 结尾，头文件必须以 `.h` 结尾。
2. 每个头文件必须使用 `#pragma once` 或传统的 `#define` 宏防卫。
3. 源文件应首先包含自己的头文件（如有），再包含其他模块和标准库。

命名规范

4. 所有全局函数和变量必须使用 `模块名_功能名()` 命名方式（如：`net_init()`）。
5. 常量宏使用全大写字母加下划线，例如 `MAX_SIZE`。
6. 使用 `typedef` 定义的类型必须以 `_t` 结尾（如：`user_t`）。

类型与初始化

7. 所有变量在使用前必须声明并初始化。
8. 函数必须显式声明返回类型，禁止使用隐式 `int`（C99 已废弃）。
9. 所有通过 `malloc` / `calloc` 等申请的内存必须进行空指针检查。

控制结构

10. 所有 `if/else`、`for`、`while`、`do` 语句都必须使用花括号 `{}` 包围代码块，即使只有一行语句。
11. 所有 `switch` 语句必须包含 `default` 分支。
12. `switch` 中每个 `case` 分支必须使用 `break`，若需要 fall-through，必须写注释 `// fall through`。

安全性与头文件设计

13. 禁止使用 `gets()`、`strcpy()`、`sprintf()` 等不安全函数。
14. 禁止将 `void*` 转换为不兼容的指针类型。
15. 所有 `free()` 后必须将指针设为 `NULL`，防止悬挂指针。
16. 头文件中不允许定义变量或非 `static inline` 的函数。

Recommended

这些规则非强制，但遵循它们可极大提高代码质量、可维护性与协作效率。

编码风格

17. 使用 4 个空格缩进，禁止使用 Tab 字符。
18. 每行代码建议不超过 100 个字符。
19. 不同逻辑段之间使用空行分隔，提高可读性。

命名习惯

20. 局部变量和函数参数使用 `snake_case` 命名。
21. 指针变量建议加前缀 `p_` 或 `ptr_`（如 `ptr_buf`）。
22. 变量名应使用全词，如 `index`、`count`，避免 `idx`、`cnt` 等缩写。

函数设计

23. 函数长度建议不超过 50 行（不含空行和注释）。
24. 函数参数不建议超过 4 个，超过时应封装为结构体。
25. 函数形参若不修改，应使用 `const` 修饰。
26. 每个函数定义前应有注释，说明功能、参数和返回值。

内存管理

27. 使用 `malloc` 或 `calloc` 时，推荐写法为 `malloc(sizeof(*ptr))` 而非具体类型。
28. 所有堆内存申请必须在正常路径与异常路径均释放。
29. 若需零初始化，优先使用 `calloc` 而非 `malloc + memset`。

注释与文档

30. 所有对外公开的函数与结构体都应在头文件中添加文档注释。
31. 使用 `/* */` 做模块或块注释，使用 `//` 写简单行内注释。
32. 注释应说明“为什么”这么写，而不是“做了什么”。
33. TODO/FIXME/NOTE 标签需大写并附开发者名缩写（如：`// TODO(ly): 优化此函数`）。

Permitted

以下做法在特定场景下允许使用，但应谨慎评估其影响，避免滥用。

技术实践

34. 允许使用 `goto` 用于资源清理逻辑，但必须限制在函数内部，且必须配合注释。
35. 可使用函数指针表（如回调或状态机实现），需清楚文档说明其行为。
36. 在节省空间或硬件操作场景中允许使用 `union`，但需注明具体用途。

条件编译与平台兼容

37. 可使用 `#ifdef` 进行平台差异控制，但应保持作用范围最小。
38. 推荐将平台特定代码拆分到 `os_linux.c`、`os_windows.c` 等文件中管理。

日志与调试

39. `printf()` 可用于调试，但发布版本必须统一替换为日志系统或移除。
40. 日志输出应统一封装为宏，如 `LOG_INFO`、`LOG_ERR` 等，便于统一管理与禁用。
