# C++ 技术管理规范

2022141461008 江浩睿

说明：（a）强制；（b）推荐；（c）允许

## 一、头文件 (Header Files)

1. **(a) 所有头文件必须有 #define 保护宏**：宏名格式为 <PROJECT>_<RELATIVE_PATH>_<FILE>_H_ (全大写，下划线分隔)，确保唯一性。文件末尾添加 #endif // MACRO_NAME 注释。
2. **(a) 头文件应自包含 (Self-contained)**：必须包含其所有依赖声明的头文件，不能依赖前置声明或包含顺序。
3. **(b) 避免使用前置声明 (Forward Declarations)**：优先使用 #include。前置声明隐藏依赖、易破坏、对模板和嵌套类型不友好。
4. **(a) #include 顺序与分组**（顺序不可乱，组间空行分隔）：
    - 关联头文件 (.cc 对应的 .h)
    - C 系统头文件 (<cstdio>, <unistd.h>)
    - C++ 系统头文件 (<vector>, <string>)
    - 其他库头文件 (<gtest/gtest.h>, "third_party/bar.h")
    - 本项目头文件 ("project/foo.h")
5. **(b) #include 使用完整路径**：项目内头文件使用从项目根目录开始的相对路径，避免 ./ 和 ../。
6. **(b) 头文件中只存放声明**：变量/函数定义应放在 .cc 文件中 (内联函数和模板特例化除外)。

7. **(a) 禁止在头文件中定义非内联函数/非 constexpr 变量：**会导致链接错误（ODR 违反）。

## 二、作用域 (Scoping)

8. **(a) 全局变量/函数禁止使用：**极特殊情况需严格评审。使用命名空间封装或静态成员替代。
9. **(b) 在 .cc 文件中，优先使用匿名命名空间或 static：**将文件作用域实体限定为内部链接性。
10. **(a) 禁止在头文件中使用匿名命名空间或 static：**违反 ODR 原则。
11. **(b) 合理使用命名空间：**
    - 项目代码应置于项目专属的顶级命名空间内。
    - 命名空间名全小写，下划线分隔。
    - 命名空间结束处添加 // namespace name 注释。
    - 避免深层嵌套。
12. **(a) 禁止 using namespace（在头文件或全局作用域）：**导致命名空间污染。在 .cc 文件函数/实现作用域内谨慎使用 using 声明 (using std::vector;)。
13. **(c) 允许在函数或方法内部使用局部 using 声明或别名：**提高局部可读性。

## 三、类 (Classes)

14. **(a) 类访问控制顺序：**public: -> protected: -> private:。各部分内声明顺序：类型 (typedef/using)、常量、构造函数、工厂方法、普通方法、数据成员（public/protected 极少）。
15. **(a) 成员变量必须为 private**（static const 常量或极特殊案例除外）。

16. **(b) 避免暴露基类成员**：优先使用组合而非继承。若使用继承，使用 public 继承，基类析构函数为 virtual 或 protected。
17. **(a) 多态基类的析构函数必须为 virtual 或 protected**：防止通过基类指针删除派生类对象时资源泄漏。
18. **(b) 重写虚函数必须显式标记 override 或 final**：提高可读性，编译器可检查签名。
19. **(a) 单参数构造函数和转换运算符必须用 explicit 标记**：防止隐式转换导致的意外行为（拷贝/移动构造除外）。
20. **(b) 使用 = default 或 = delete 显式声明/删除特殊成员函数**：明确意图（即使编译器会自动生成）。
21. **(a) 禁止在构造函数中调用虚函数**：此时对象未完全构造，虚函数机制未按预期工作。
22. **(b) 类定义应短小精悍**：过大的类考虑拆分职责。

# 四、函数 (Functions)

23. **(b) 函数应简短**（推荐 <40 行）：功能单一，易于理解和测试。
24. **(b) 函数参数顺序**：输入参数（值/const 引用）在前，输出参数（指针/非 const 引用）在后。
25. **(b) 优先以值或 const 引用传递输入参数**：输出参数使用指针（优先）或非 const 引用。可选输入参数考虑 std::optional。
26. **(b) 避免函数参数过多（建议 ≤4）**：过多考虑使用结构体封装。
27. **(a) 禁止函数宏 (Function-like Macros)**：用内联函数、模板或常量替代。宏调试困难且易出错。
28. **(b) 内联函数必须非常小（建议 ≤10 行）**：避免包含循环/switch。编译器决定权最终大于标记。

29. **(a) 虚函数禁止使用默认参数**：默认参数绑定于静态类型，与虚函数动态绑定冲突。
30. **(b) 错误处理**：使用返回值、异常（项目统一规范）、abort() 或错误码对象。禁止忽略错误。

# 五、命名约定 (Naming Conventions)

31. **(a) 文件命名**：全小写，下划线 _ 分隔 (.h, .cc/.cpp)。
32. **(a) 类型命名**：类、结构体、枚举（类）、类型别名、模板参数 - **帕斯卡命名法** (MyClass, UrlTableError)。
33. **(a) 变量命名**（包括函数参数）：
    - 普通变量：全小写，下划线分隔 (my_variable)。
    - 类成员变量：全小写，下划线分隔，**末尾加下划线** (my_member_)。
    - 结构体成员：同普通变量（无尾下划线）。
34. **(a) 常量命名**：程序生命周期内不变的 const[expr] 变量 - **k 开头 + 帕斯卡** (kDaysInWeek, kMaxBufferSize)。
35. **(a) 函数命名**：
    - 常规函数：**帕斯卡命名法** (CalculateTotal(), OpenFile())。
    - 访问器/修改器 (getter/setter)：与变量名匹配 (count(), set_count(int count))。
36. **(a) 枚举命名**：
    - 枚举类名：帕斯卡命名法 (enum class UrlTableError）。
    - 枚举值：同常量 (kOk, kNotFound) 或同宏（不推荐新项目使用）。

37. **(a) 命名空间命名：**全小写，下划线分隔。顶级命名空间基于项目名。

# 六、格式 (Formatting)

38. **(a) 行长度 ≤ 120 字符：**在可读性前提下尽量遵守。
39. **(a) 缩进使用空格，宽度统一为 2 或 4 个空格**（项目内统一）。**禁止使用 Tab**（设置编辑器转换 Tab 为空格）。
40. **(b) 指针/引用声明符位置：** * 和 & **紧跟变量名** (char* ptr;, const string& str;)。
41. **(b) 操作符空格：**
    - 二元操作符 (= + - * / % < > == != <= >= && ||) 前后加空格。
    - 一元操作符 (! ~ ++ -- & *) 紧跟操作数，无空格。
    - . -> :: 前后无空格。
    - 逗号 ,、分号 ; 后加空格。
42. **(b) 控制结构 (if/for/while/switch) 格式：**
    - 关键字后加空格，左括号紧跟关键字。
    - 右括号 ) 与左大括号 { 间加空格。
    - 单行语句体也**强制使用 {}**。
    - else 另起一行。
43. **(b) 函数声明/调用格式：**
    - 函数名与左括号 ( 间无空格。
    - 参数列表过长可分行，参数缩进对齐或 4 空格缩进。
    - 左大括号 { 位于函数声明末尾同一行。

## 七、现代 C++ 特性（Modern C++）

44. **(b) 优先使用 constexpr 和 const**：代替宏定义常量，支持编译期计算。
45. **(b) 优先使用 nullptr**：代替 NULL 或 0，类型安全。
46. **(b) 优先使用基于范围的 for 循环**：更简洁安全。
47. **(b) 资源管理优先使用智能指针（unique_ptr, shared_ptr）和 RAII**：避免手动 new/delete，减少泄漏。
48. **(b) 优先使用 auto**：在类型明显或冗长时提高可读性（如迭代器、闭包、模板表达式结果）。避免滥用导致类型信息丢失。
49. **(c) 允许谨慎使用 Lambda 表达式**：尤其在 STL 算法和回调中。
50. **(b) 使用 enum class 代替 enum**：提供强作用域和类型安全。