# C语言技术管理文档

## 1. 代码风格

### a. 强制

1. 所有代码必须使用4个空格缩进，禁止使用Tab键
2. 每个源文件必须包含文件头注释，注明版权、作者、创建日期和修改历史
3. 函数之间必须用空行分隔
4. 每行代码不得超过80个字符
5. 大括号必须遵循K&R风格：左大括号不换行

### b. 推荐

1. 变量命名使用小写加下划线风格，如max\_count
2. 宏定义和枚举常量使用全大写加下划线风格，如MAX\_BUFFER\_SIZE
3. 函数命名使用小写加下划线风格，如calculate\_average()
4. 指针声明时，星号应靠近变量名：char \*ptr
5. 运算符前后应添加空格，如a = b + c

### c. 允许

1. 简单的单行条件语句可以不使用大括号
2. 简单的getter/setter函数可以写在一行内
3. 短小的switch语句可以不使用缩进

## 

## 2. 头文件管理

### a. 强制

1. 头文件必须使用#ifndef/#define/#endif防止多重包含
2. 头文件必须包含其依赖的其他头文件，不能依赖间接包含
3. 头文件中的函数声明必须包含参数名
4. 禁止在头文件中定义变量（extern声明除外）

### b. 推荐

1. 头文件应按功能模块组织，避免过大
2. 头文件应包含清晰的注释说明模块功能
3. 头文件中的函数应按相关性分组

### c. 允许

1. 简单的内联函数可以定义在头文件中
2. 小型项目可以使用单个头文件组织所有声明

## 

## 3. 函数设计

### a. 强制

1. 函数长度不得超过50行
2. 函数参数不得超过5个
3. 函数必须明确指定返回值类型，禁止省略
4. 函数必须有清晰的注释说明其功能、参数和返回值
5. 禁止使用全局变量在函数间传递数据

### b. 推荐

1. 函数应尽量保持单一职责原则
2. 函数参数应按输入、输出、输入输出的顺序排列
3. 复杂函数应分解为多个小函数
4. 函数内部应使用局部变量而非静态变量

### c. 允许

1. 工具函数可以使用较短的命名
2. 回调函数可以使用较短的命名

## 

## 4. 内存管理

### a. 强制

1. 所有动态分配的内存必须有对应的释放操作
2. 使用malloc/calloc分配内存后必须检查返回值是否为NULL
3. 禁止对已释放的内存进行访问
4. 禁止对NULL指针进行解引用

### b. 推荐

1. 使用free()后应立即将指针置为NULL
2. 复杂的内存分配应封装成专用函数
3. 内存分配和释放应成对出现在同一抽象层次

### c. 允许

1. 小型临时缓冲区可以使用栈分配
2. 性能关键路径可以使用内存池技术

## 

## 5. 错误处理

### a. 强制

1. 所有可能失败的函数调用必须检查返回值
2. 错误码必须明确处理，不能忽略
3. 资源分配失败必须有恢复或清理机制

### b. 推荐

1. 使用一致的错误码定义方案
2. 错误信息应包含足够上下文以便调试
3. 错误处理代码应集中管理

### c. 允许

1. 简单的工具函数可以直接返回错误码
2. 致命错误可以直接终止程序

## 

## 6. 安全性

### a. 强制

1. 禁止使用不安全的字符串函数如strcpy，必须使用strncpy等安全版本
2. 所有数组访问必须进行边界检查
3. 格式化字符串必须使用常量或严格控制的变量

### b. 推荐

1. 敏感数据使用后应立即清除
2. 密码等敏感信息不应以明文形式存储
3. 关键操作应有日志记录

### c. 允许

1. 内部工具可以适当放宽安全限制
2. 性能关键路径可以适当放宽安全检查

## 

## 7. 文档

### a. 强制

1. 公共API必须有详细文档
2. 复杂算法必须有实现说明

### b. 推荐

1. 模块应有设计文档
2. 关键数据结构应有说明

### c. 允许

1. 内部工具可以简化文档
2. 自解释代码可以省略部分注释