**C++ 技术管理文档**

**一、强制规范（a）**

1. **代码风格**：所有C++代码必须遵循统一的命名规范和格式要求。
2. **命名规则**：类名使用PascalCase（如MyClass），变量名使用camelCase（如myVariable）。
3. **头文件保护**：每个头文件必须使用#ifndef、#define、#endif来防止重复包含。
4. **命名空间**：尽量避免使用using namespace，特别是在头文件中。
5. **类成员变量**：所有类的成员变量应以m\_为前缀（如m\_variable）。
6. **常量声明**：所有常量必须使用const或者constexpr进行声明。
7. **析构函数**：所有动态分配内存的类都必须提供虚析构函数。
8. **拷贝构造函数与赋值运算符**：需要实现深拷贝并遵循“拷贝并交换”惯例。
9. **智能指针**：优先使用std::unique\_ptr、std::shared\_ptr，避免使用裸指针管理资源。
10. **C++标准库**：优先使用C++标准库中的容器（如std::vector、std::map等），而不是手动管理数组或自定义容器。
11. **异常处理**：使用try-catch语句捕获并处理异常，不允许抛出未经处理的异常。
12. **内存管理**：使用new和delete时必须确保对称，优先使用RAII原则来管理资源。
13. **编译器警告**：编译器警告必须清除，且不得忽视任何编译器警告。
14. **避免使用goto**：除非极为必要，必须避免使用goto语句。
15. **函数的复杂度**：每个函数应尽量只做一件事，函数复杂度应尽量控制在合理范围内。
16. **接口设计**：类的接口必须明确，每个函数的功能必须清晰，并保持接口简洁。
17. **类大小**：每个类的成员变量不应过多，避免类过大导致难以维护。
18. **线程安全**：多线程程序中，必须保证对共享资源的访问是线程安全的。

**二、推荐规范（b）**

1. **空格与缩进**：使用4个空格进行缩进，避免使用制表符。
2. **行长度**：每行代码最大长度不超过120个字符。
3. **函数参数**：函数参数应该尽量使用引用传递，避免不必要的复制。
4. **命名空间**：在实现文件（.cpp）中，可以使用using namespace，但避免在头文件中使用。
5. **类型别名**：建议使用using替代typedef来创建类型别名。
6. **强类型枚举**：尽量使用enum class而不是传统的enum，以避免命名冲突。
7. **自动类型推导**：优先使用auto来声明变量，避免手动指定类型，增加代码可维护性。
8. **模板**：使用模板时，尽量避免复杂的模板元编程，保持模板代码简洁明了。
9. **函数返回类型**：函数返回类型建议显式地写明，避免使用auto作为返回类型。
10. **构造函数初始化列表**：构造函数的成员变量应通过初始化列表进行初始化，避免在函数体内进行赋值。
11. **常量引用**：对于传递较大对象的函数参数，建议使用const引用传递，避免不必要的拷贝。
12. **头文件组织**：每个头文件应保持简洁，只包含必要的声明，避免在头文件中实现功能。
13. **日志记录**：使用统一的日志记录库，避免随意打印调试信息。
14. **内存池**：在频繁分配和释放内存的情况下，建议使用内存池技术。
15. **智能指针循环引用**：在使用std::shared\_ptr时，应避免出现循环引用，导致内存泄漏。
16. **命名规范**：函数、变量和类的命名应符合项目的命名标准，保持一致性。
17. **设计模式**：合理使用设计模式，如单例模式、工厂模式等，来解决常见问题。
18. **优化顺序**：优先进行算法优化，再考虑代码的微观优化。

**三、允许规范（c）**

1. **宏定义**：可以使用宏来简化一些简单的常量定义，但应避免宏过度使用。
2. **inline函数**：对于小型函数，可以使用inline进行优化，但不建议滥用。
3. **局部变量**：局部变量的定义可以在需要的地方进行，而非在函数开头一次性定义。
4. **资源管理**：对于临时资源，使用原始指针也是可以的，但要小心内存泄漏。
5. **自定义容器**：如果C++标准库的容器不符合需求，可以适当实现自定义容器。
6. **未使用的变量**：可以暂时注释掉未使用的变量，但在提交代码时必须清理掉。
7. **使用C风格的函数**：如果项目中有遗留的C风格函数，可适当使用，但要逐步迁移到C++风格。
8. **条件编译**：可以使用#ifdef等预处理指令来支持不同平台的编译，但要尽量减少使用。
9. **字符串操作**：对于较小的字符串操作，C风格字符串函数可以使用，但应该逐步迁移到std::string。
10. **类型转换**：尽量避免使用C风格的类型转换，推荐使用static\_cast、dynamic\_cast等C++的类型转换方式。