C++代码规范：

**一、排版**

1.程序块要 采用缩进 风格编 写,缩 进的空 格数为 4个

2.相对独立 的程序块 之间、 变量说 明之后 必须加 空行

3.较长的语 句(>80字符)要分 成多行 书写, 长表达 式要在低 优先级 操作符 处划分 新行, 操作符放 在新行 之首,划 分出的 新行要 进行适 当的缩 进,使排 版整齐 ,语句 可读

4.循环、判 断等语 句中若 有较长 的表达式 或语句 ,则要 进行适 应的划 分,长 表达式 要在低 优先 级操作符 处划分 新行,操 作符放 在新行 之首

5.若函数或过程中的参数较长,则要进行适当的划分

6.不允许把多个短语句写在一行中,即一行只写一条语句

7.if、for、do、while、case、switch、default等语句自占一行,且if、for、do、while等语句的执行语句部分无论多少都要加括号{}

8.对齐只使用空格键,不使用TAB键

9.函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格,case语句下的情况处理语句也要遵从语句缩进要求

10.程序块的分界符(如C/C++语言的大括号‘{’和‘}’)应各独占一行并且位于同一列,同时与引用它们的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义以及if、for、do、while、switch、case语句中的程序都要采用如上的缩进方式

11.程序块的分界符(如C/C++语言的大括号‘{’和‘}’)应各独占一行并且位于同一列,同时与引用它们的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义以及if、for、do、while、switch、case语句中的程序都要采用如上的缩进方式

12.在两个以上的关键字、变量、常量进行对等操作时,它们之间的操作符之前、之后或者前后要加空格;进行非对等操作时,如果是关系密切的立即操作符(如->),后不应加空格。说明:采用这种松散方式编写代码的目的是使代码更加清晰

**二、注释**

1. 一般情况下,源程序有效注释量必须在20%以上
2. 说明性文件(如头文件.h文件、.inc文件、.def文件、编译说明文件.cfg等)头部应进行注释,注释必须列出:版权说明、版本号、生成日期、作者、内容、功能、与其它文件的关系、修改日志等,头文件的注释中还应有函数功能简要说明
3. 函数头部应进行注释,列出:函数的目的/功能、输入参数、输出参数、返回值

4.边写代码边注释,修改代码同时修改相应的注释,以保证注释与代码的一致性。不再有用的注释要删除

5.注释的内容要清楚、明了,含义准确,防止注释二义性

6.避免在注释中使用缩写，在使用缩写时或之前,应对缩写进行必要的说明。

7.注释应与其描述的代码相近,对代码的注释应放在其上方或右方(对单条语句的注释)相邻位置,不可放在下面,如放于上方则需与其上面的代码用空行隔开

8.对于所有有物理含义的变量、常量,如果其命名不是充分自注释的,在声明时都必须加以注释,说明其物理含义。变量、常量、宏的注释应放在其上方相邻位置或右方

9数据结构声明(包括数组、结构、类、枚举等),如果其命名不是充分自注释的,必须加以注释。对数据结构的注释应放在其上方相邻位置,不可放在下面;对结构中的每个域的注释放在此域的右方

10.全局变量要有较详细的注释,包括对其功能、取值范围、哪些函数或过程存取它以及存取时注意事项等的说明

11.注释与所描述内容进行同样的缩排

12.对变量的定义和分支语句(条件分支、循环语句等)必须编写注释

13.对于switch语句下的case语句,如果因为特殊情况需要处理完一个case后进入下一个case 处理,必须在该case语句处理完、下一个case语句前加上明确的注释

14.避免在一行代码或表达式的中间插入注释

15.注释应考虑程序易读及外观排版的因素,使用的语言若是中、英兼有的,建议多使用中文,除非能用非常流利准确的英文表达

**三、标识符命名**

1 标识符的命名要清晰、明了,有明确含义,同时使用完整的单词或大家基本可以理解的缩写,避免使人产生误解；

2命名中若使用特殊约定或缩写,则要有注释说明

3.自己特有的命名风格,要自始至终保持一致,不可来回变化

4对于变量命名,禁止取单个字符(如i、j、k...),建议除了要有具体含义外,还能表明其变量类型、数据类型等,但i、j、k作局部循环变量是允许的

5.用正确的反义词组命名具有互斥意义的变量或相反动作的函数等

6.除了编译开关/头文件等特殊应用,应避免使用\_EXAMPLE\_TEST\_之类以下划线开始和结尾的定义

1. **可读性**

1.注意运算符的优先级,并用括号明确表达式的操作顺序,避免使用默认优先级

2.避免使用不易理解的数字,用有意义的标识来替代。涉及物理状态或者含有物理意义的常量,不应直接使用数字,必须用有意义的枚举或宏来代替

3.源程序中关系较为紧密的代码应尽可能相邻

4.不要使用难懂的技巧性很高的语句,除非很有必要时

5.注意运算符的优先级,并用括号明确表达式的操作顺序,避免使用默认优先级

1. **变量与结构**
2. 去掉没必要的公共变量
3. 仔细定义并明确公共变量的含义、作用、取值范围及公共变量间的关系
4. 明确公共变量与操作此公共变量的函数或过程的关系,如访问、修改及创建等
5. 当向公共变量传递数据时,要十分小心,防止赋与不合理的值或越界等现象发生
6. 防止局部变量与公共变量同名
7. 严禁使用未经初始化的变量作为右值
8. 不要设计面面俱到、非常灵活的数据结构
9. 不同结构间的关系不要过于复杂
10. .结构中元素的个数应适中。若结构中元素个数过多可考虑依据某种原则把元素组成不同的子结构,以减少原结构中元素的个数
11. 仔细设计结构中元素的布局与排列顺序,使结构容易理解、节省占用空间,并减少引起误用现象
12. 结构的设计要尽量考虑向前兼容和以后的版本升级,并为某些未来可能的应用保留余地(如预留一些空间等)
13. 留心具体语言及编译器处理不同数据类型的原则及有关细节
14. 编程时,要注意数据类型的强制转换
15. .对编译系统默认的数据类型转换,也要有充分的认识
16. 尽量减少没有必要的数据类型默认转换与强制转换
17. 合理地设计数据并使用自定义数据类型,避免数据间进行不必要的类型转换
18. 对自定义数据类型进行恰当命名,使它成为自描述性的,以提高代码可读性。注意其命名方式在同一产品中的统一
19. 当声明用于分布式环境或不同CPU间通信环境的数据结构时,必须考虑机器的字节顺序、使用的位域及字节对齐等问题
20. **函数与过程**

1.对所调用函数的错误返回码要仔细、全面地处理

2.明确函数功能,精确(而不是近似)地实现函数设计

3.编写可重入函数时,应注意局部变量的使用(如编写C/C++语言的可重入函数时,应使用auto 即缺省态局部变量或寄存器变量)

4.编写可重入函数时,若使用全局变量,则应通过关中断、信号量(即P、V操作)等手段对其加以保护

5.在同一项目组应明确规定对接口函数参数的合法性检查应由函数的调用者负责还是由接口函数本身负责,缺省是由函数调用者负责

6.防止将函数的参数作为工作变量

7..函数的规模尽量限制在200行以内

8.一个函数仅完成一件功能

9.为简单功能编写函数

10.不要设计多用途面面俱到的函数

11.函数的功能应该是可以预测的,也就是只要输入数据相同就应产生同样的输出

12尽量不要编写依赖于其他函数内部实现的函数

1. .避免设计多参数函数,不使用的参数从接口中去掉
2. 非调度函数应减少或防止控制参数,尽量只使用数据参数
3. 检查函数所有参数输入的有效性
4. 检查函数所有非参数输入的有效性,如数据文件、公共变量等
5. 函数名应准确描述函数的功能
6. 用动宾词组为执行某操作的函数命名。如果是OOP方法,可以只有动词(名词是对象本身)示例:参照如下方式命名函数。
7. 避免函数中不必要语句,防止程序中的垃圾代码
8. .防止把没有关联的语句放到一个函数中
9. 如果多段代码重复做同一件事情,那么在函数的划分上可能存在问题
10. 功能不明确较小的函数,特别是仅有一个上级函数调用它时,应考虑把它合并到上级函数中,而不必单独存在
11. 设计高扇入、合理扇出(小于7)的函数
12. 减少函数本身或函数间的递归调用
13. 仔细分析模块的功能及性能需求,并进一步细分,同时若有必要画出有关数据流图,据此来进行模块的函数划分与组织
14. 改进模块中函数的结构,降低函数间的耦合度,并提高函数的独立性以及代码可读性、效率和可维护性
15. 在多任务操作系统的环境下编程,要注意函数可重入性的构造
16. 避免使用BOOL参数
17. 对于提供了返回值的函数,在引用时最好使用其返回值
18. 当一个过程(函数)中对较长变量(一般是结构的成员)有较多引用时,可以用一个意义相当的宏代替
19. **可测性**
20. 在同一项目组或产品组内,调测打印出的信息串的格式要有统一的形式。信息串中至少要有所在模块名(或源文件名)及行号
21. .使用断言来发现软件问题,提高代码可测性
22. 仔细分析有关算法,并进行优化
23. 仔细考查、分析系统及模块处理输入(如事务、消息等)的方式,并加以改进
24. 对模块中函数的划分及组织方式进行分析、优化,改进模块中函数的组织结构,提高程序效率
25. .编程时,要随时留心代码效率;优化代码时,要考虑周全
26. 不应花过多的时间拼命地提高调用不很频繁的函数代码效率
27. 要仔细地构造或直接用汇编编写调用频繁或性能要求极高的函数
28. 在保证程序质量的前提下,通过压缩代码量、去掉不必要代码以及减少不必要的局部和全局变量,来提高空间效率
29. 在多重循环中,应将最忙的循环放在最内层
30. 尽量减少循环嵌套层次
31. 避免循环体内含判断语句,应将循环语句置于判断语句的代码块之中
32. 尽量用乘法或其它方法代替除法,特别是浮点运算中的除法
33. **质量保证**
34. 在软件设计过程中构筑软件质量
35. 代码质量保证优先原则
36. 防止引用已经释放的内存空间
37. 过程/函数中分配的内存,在过程/函数退出之前要释放
38. 过程/函数中申请的(为打开文件而使用的)文件句柄,在过程/函数退出之前要关闭
39. .防止内存操作越界
40. 认真处理程序所能遇到的各种出错情况
41. 系统运行之初,要初始化有关变量及运行环境,防止未经初始化的变量被引用
42. 系统运行之初,要对加载到系统中的数据进行一致性检查
43. 严禁随意更改其它模块或系统的有关设置和配置
44. 能随意改变与其它模块的接口
45. 充分了解系统的接口之后,再使用系统提供的功能
46. 编程时,要防止差1错误
47. 要时刻注意易混淆的操作符。当编完程序后,应从头至尾检查一遍这些操作符,以防止拼写错误
48. 有可能的话,if语句尽量加上else分支,对没有else分支的语句要小心对待;switch语句必须有default分支
49. 不要滥用goto语句。
50. 不使用与硬件或操作系统关系很大的语句,而使用建议的标准语句,以提高软件的可移植性和可重用性
51. .精心地构造、划分子模块,并按“接口”部分及“内核”部分合理地组织子模块,以提高“内核”部分的可移植性和可重用性
52. 精心构造算法,并对其性能、效率进行测试
53. 时刻注意表达式是否会上溢、下溢
54. 使用变量时要注意其边界值的情况
55. 留心程序机器码大小(如指令空间大小、数据空间大小、堆栈空间大小等)是否超出系统有关限制
56. 为用户提供良好的接口界面,使用户能较充分地了解系统内部运行状态及有关系统出错情况
57. 系统应具有一定的容错能力,对一些错误事件(如用户误操作等)能进行自动补救
58. 对一些具有危险性的操作代码(如写硬盘、删数据等)要仔细考虑,防止对数据、硬件等的安全构成危害,以提高系统的安全性
59. 使用第三方提供的软件开发工具包或控件时,要注意以下几点:
60. **代码编辑、编译、审查**
61. 在产品软件(项目组)中,要统一编译开关选项
62. 通过代码走读及审查方式对代码进行检查
63. 编写代码时要注意随时保存,并定期备份,防止由于断电、硬盘损坏等原因造成代码丢失
64. 要小心地使用编辑器提供的块拷贝功能编程
65. 合理地设计软件系统目录,方便开发人员使用
66. 某些语句经编译后产生告警,但如果你认为它是正确的,那么应通过某种手段去掉告警信息说明:在Borland C/C++中,可用“#pragma warn”来关掉或打开某些告警。
67. **代码测试、维护**
68. 单元测试要求至少达到语句覆盖
69. 单元测试开始要跟踪每一条语句,并观察数据流及变量的变化
70. 清理、整理或优化后的代码要经过审查及测试
71. 代码版本升级要经过严格测试
72. 使用工具软件对代码版本进行维护
73. 正式版本上软件的任何修改都应有详细的文档记录
74. 发现错误立即修改,并且要记录下来
75. 仔细设计并分析测试用例,使测试用例覆盖尽可能多的情况,以提高测试用例的效率
76. 尽可能模拟出程序的各种出错情况,对出错处理代码进行充分的测试
77. 仔细测试代码处理数据、变量的边界情况
78. 保留测试信息,以便分析、总结经验及进行更充分的测试
79. 不应通过“试”来解决问题,应寻找问题的根本原因
80. 对自动消失的错误进行分析,搞清楚错误是如何消失的
81. 修改错误不仅要治表,更要治本
82. 测试时应设法使很少发生的事件经常发生
83. .坚持在编码阶段就对代码进行彻底的单元测试,不要等以后的测试工作来发现问题
84. 去除代码运行的随机性(如去掉无用的数据、代码及尽可能防止并注意函数中的“内部寄存器”等),让函数运行的结果可预测,并使出现的错误可再现
85. **宏**
86. .用宏定义表达式时,要使用完备的括号
87. 将宏所定义的多条表达式放在大括号中
88. 使用宏时,不允许参数发生变化

4.常量与宏应该使用大写，多组次之间使用\_分隔各单词，并加注释说明。