Rtimeman CPP 编码规范

[1 文档基本信息 4](#_Toc18517211)

[1.1 文档状态 4](#_Toc18517212)

[1.2 修改记录 4](#_Toc18517213)

[2 简单介绍 5](#_Toc18517214)

[2.1 涵盖范围 5](#_Toc18517215)

[2.2 对谁可见 5](#_Toc18517216)

[2.3 项目信息 5](#_Toc18517217)

[2.4 缩写和简称 6](#_Toc18517218)

[2.5 相关文档 7](#_Toc18517219)

[3 前言 8](#_Toc18517220)

[4 通用约定 8](#_Toc18517221)

[4.1 接口函数（要求） 8](#_Toc18517222)

[4.2 文件后缀名（要求） 8](#_Toc18517223)

[4.3 日志文件组织方式及存储路径（待定） 8](#_Toc18517224)

[4.4 头文件保护（要求） 8](#_Toc18517225)

[4.5 命名空间（推荐） 9](#_Toc18517226)

[4.6 作用域（推荐） 9](#_Toc18517227)

[4.7 类成员的声明（推荐） 9](#_Toc18517228)

[4.8 复杂参数传递（推荐） 10](#_Toc18517229)

[4.9 输入参数检查（要求） 10](#_Toc18517230)

[4.10 函数重载与缺省参数（推荐） 10](#_Toc18517231)

[4.11 预处理宏（推荐） 10](#_Toc18517232)

[4.12 幻数（推荐） 11](#_Toc18517233)

[4.13 ++和—（要求） 11](#_Toc18517234)

[4.14 浮点型变量判断相等（要求） 11](#_Toc18517235)

[4.15 局部变量的指针和引用（要求） 12](#_Toc18517236)

[4.16 不确定的语句顺序（要求） 12](#_Toc18517237)

[5 命名规则 12](#_Toc18517238)

[5.1 通用命名规则（推荐） 12](#_Toc18517239)

[5.2 文件命名（要求 - 新代码执行） 13](#_Toc18517240)

[5.3 类型命名（要求） 14](#_Toc18517241)

[5.4 变量命名（要求） 14](#_Toc18517242)

[5.5 特殊变量（推荐） 14](#_Toc18517243)

[5.6 函数命名（要求） 14](#_Toc18517244)

[5.7 命名空间命名（要求） 15](#_Toc18517245)

[5.8 枚举命名（待定） 15](#_Toc18517246)

[5.9 宏命名（要求） 15](#_Toc18517247)

[6 代码注释 15](#_Toc18517248)

[6.1 注释风格（推荐） 15](#_Toc18517249)

[6.2 文件注释（推荐） 16](#_Toc18517250)

[6.3 类注释（推荐） 16](#_Toc18517251)

[6.4 函数注释（推荐） 17](#_Toc18517252)

[6.5 变量注释（推荐） 17](#_Toc18517253)

[6.6 代码中注释（推荐） 17](#_Toc18517254)

[7 代码格式 19](#_Toc18517255)

[7.1 行长度（要求） 19](#_Toc18517256)

[7.2 缩进（要求） 19](#_Toc18517257)

[7.3 函数声明与定义（要求） 19](#_Toc18517258)

[7.4 函数调用（要求） 20](#_Toc18517259)

[7.5 条件语句（要求） 20](#_Toc18517260)

[7.6 循环语句（要求） 21](#_Toc18517261)

[7.7 分支语句（要求） 21](#_Toc18517262)

[7.8 指针和引用表达式（要求） 22](#_Toc18517263)

[7.9 预处理指令（要求） 22](#_Toc18517264)

[7.10 类格式（推荐） 22](#_Toc18517265)

[7.11 命名空间（要求） 23](#_Toc18517266)

[7.12 水平留白（要求） 23](#_Toc18517267)

[7.13 垂直留白（推荐） 24](#_Toc18517268)

# 文档基本信息

## 文档状态

目前状态：*通过评审*.

|  |  |
| --- | --- |
| 状态 | 描述 |
|  | |
| *不可用* | 文档在制订中，无法作为参考 |
| *完成* | 文档完成 |
| *通过评审* | 文档已经通过评审 |

## 修改记录

|  |  |  |  |
| --- | --- | --- | --- |
| ***文档修改记录*** | | | |
| ***版本号*** | ***日期***  ***yyyy-mm-dd*** | ***作者*** | ***修改描述*** |
| 1.0 | 2017-1-16 | 冯赟 | 初稿 |
|  |  |  |  |

# 简单介绍

参考网络上的编码规范，综合当前项目特点，制定本编码指导规范。

## 涵盖范围

cpp编码规范。

## 对谁可见

所有软件工程师。

## 项目信息

|  |  |  |  |
| --- | --- | --- | --- |
| # | 条目 | 信息 | 描述 |
| 1 |  |  |  |
| 2 |  |  |  |

## 缩写和简称

|  |  |
| --- | --- |
| 缩写和简称 | 描述 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 相关文档

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | 文档名称 | 版本 | 文档存储地址 | 描述 |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# 前言

每个人的代码都或多或少会带有个人的风格，而这种独特的风格很可能会给其他人造成阅读上的障碍，为了让我们的代码容易让其他人看懂，就需要在写代码时遵守一定的规范，尤其在同一个项目组内的成员，共同遵守统一的代码规范显得尤为重要。本规范参考Google C++编码规范以及MoveIt源代码编写，分为通用约定、命名规则、代码注释和代码格式四个部分。

规范中的条目分为三个执行等级：推荐、要求和待定；属于推荐等级的内容在允许的情况下应尽量遵守，属于要求等级的内容无论何时都应遵守，属于待定等级的内容还没有最终确认，不用遵守。

C++代码规范发布后，所有新增代码均受到代码规范的约束，对于规范发布之前的已有代码，在平衡工作量的基础上尽量进行标准化。

标记（要求）的，为编码必须遵守的规范。

标记（推荐）的，为编码时可以考虑的内容。

# 通用约定

## 接口函数（要求）

接口函数是指代码中供其他代码模块调用的函数，接口函数一旦发布后，不能私下修改，如确有必要修改的，需经所有关联人员会议通过后再行修改；

## 文件后缀名（要求）

C源代码文件以.c结尾，C++源代码文件以.cpp结尾，头文件以.h结尾，配置文件以.cfg结尾，其他需要在程序中插入的文件以.inc结尾，日志文件以.log结尾；

## 日志文件组织方式及存储路径（待定）

告警日志、运行日志等

## 头文件保护（要求）

所有头文件都应使用#define进行保护以防止被多重包含，为保证唯一性，其命名格式为：<PACKAGE>\_<FILE>\_H\_，例如在tp\_interface中的头文件timer.h应被定义为：

#ifndef TP\_INTERFACE\_TIMER\_H\_

#define TP\_INTERFACE\_TIMER\_H\_

…

…

#endif // ifndef TP\_INTERFACE\_TIMER\_H\_

## 命名空间（推荐）

使用命名空间将作用域进行细分，以最大限度防止命名冲突；在.cpp文件中可以使用不具名的命名空间将全部内容封装起来，而在.h文件中只能使用具名的命名空间；不推荐使用using namespace xxx指令，它将导出命名空间中的全部命名，增加命名冲突的可能，推荐使用using指令仅导出使用的命名；例如：

using namespace std; // 不推荐使用，污染命名空间

using std::vector; // 推荐使用，仅引入用到的命名

## 作用域（推荐）

函数中变量的定义位置应尽可能靠近其第一次使用的位置，且变量第一次定义的同时应对其进行初始化；变量应尽量置于最小作用域内，但如果变量是一个对象，每次进入、退出作用域都需要调用构造、析构函数，对象的声明位置需要酌情考虑；例如：

int result;

result = doSomething(); // 不推荐使用，声明和初始化分离

int result = doSomething(); // 推荐使用，声明变量的同时对其进行初始化

int cnt;

for (cnt = 0; cnt<1000; ++cnt) { // 不推荐使用，cnt的作用域被扩大

doSomething(i);

}

for (int cnt = 0; cnt<1000; ++cnt) { // 推荐使用，cnt作用域仅限于for循环中

doSomething(i);

}

for (int cnt = 0; cnt<10000000; ++cnt) {

MyClass object; // 不推荐使用，构造和析构函数被重复调用多次

object.doSomething(i);

}

MyClass object; //推荐使用，构造和析构函数仅调用1次

for (int cnt = 0; cnt<10000000; ++cnt) {

object.doSomething(i);

}

## 类成员的声明（推荐）

类中的成员应按照构造函数、析构函数、公有函数、私有函数和数据成员的顺序依次声明，且成员属性按public、protected、private的次序排列；cpp文件中的函数定义顺序应尽可能和声明次序一致，函数定义和函数声明中的形参均不能省略且命名必须一致；

## 复杂参数传递（推荐）

复杂的结构体或类做为参数传递时优先使用引用传递，且使用const修饰，除非该参数用于函数输出；使用引用做为函数输出时应在注释中进行说明；参数传递尽量避免用指针传递，除非必要，例如传递函数指针、NULL指针等；使用指针传递参数时应尽可能将其声明为const类型，使用指针应有理有据有必要，且在注释中进行说明；

## 输入参数检查（要求）

接口函数必须对传入的参数进行检查，尤其是指针类型的参数，以确保参数的合法性；对于模块内部使用的函数，在其调用者完全明确的情况下，可以不进行参数检查；在不影响性能的前提下，内部函数也可以适当加入参数检查环节以尽量减少将来可能出现的风险；

## 函数重载与缺省参数（推荐）

推荐使用函数重载，在用少量函数重载即可替换缺省参数函数的情况下，应尽量避免使用缺省参数函数；因为很多人会根据已有的代码来确定函数的调用方法，如果参数缺省，调用者可能会错误的理解该函数，在调用时增加了出现bug的可能；缺省参数会增加潜在的程序风险，且缺省参数会严重损害代码的可读性甚至出现二义性，例如；

void function(int argument1, int argument2 = 0);

void function(int argument1);

if (true) {

function(50); // 调用哪一个function？

}

## 预处理宏（推荐）

使用宏时要保持警惕，且尽量以内联函数、枚举和数值常量代替之；宏意味着我们和编译器看到的代码是不同的，由此可能导致程序异常行为，尤其是当宏存在于全局作用域中；并且宏缺乏类型检测机制，容易引发一些隐藏极深的错误；值得庆幸的是，在C++中，宏不像C中那么必要，例如宏存储常量可以使用const常量或枚举变量替换，宏定义缩写长变量名可以用引用替代，宏内联效率关键代码可以使用内联函数替代；但在头文件保护、条件编译等情况下可以合理使用宏，使用宏时必须同时给出详细的注释；

## 幻数（推荐）

幻数是指在代码中直接以数字形式出现的常数；因为在代码中的幻数通常只能靠上下文来揣测其意义，且当这个常数需要改变时，需要仔细甄别所有代码中的幻数是否是要改的那个常数，极其繁琐还有可能有遗漏；所以在编程时应尽量避免使用幻数，通常把幻数定义为枚举变量或const类型的常量；例如：

for (int axis=0; axis < 6; ++axis) { // 不推荐，这里的6是什么意义？

doSomething(axis);

}

const c\_robot\_joint\_number = 6; // 推荐，常量的意义明晰且维护方便

for (int axis = 0; axis < c\_robot\_joint\_number; ++axis) {

doSomething(axis);

}

## ++和—（要求）

不考虑返回值的话，前置自增（++i）或自减（--i）通常比后置的自增或自减效率高，因为后置自增或自减操作需要将对象进行一次拷贝操作；如果i是迭代器或其他非数值类型，拷贝所花费的时间代价通常很高；所以在使用前置和后置运算符都可以的场合，使用前置自增、自减运算符；

## 浮点型变量判断相等（要求）

由于浮点型变量表示方式的特殊性，浮点数存在不可避免的误差，因此用’==’来比较两个浮点型变量的结果是不能确定的；要判断两个浮点数是否相等，需要对这两个浮点数的差值进行判断，例如：

double temperature\_today = 32.18;

double temperature\_yesterday = 32.18;

// 因为double类型存在表示误差，判断的结果是不确定的

if (temperature\_today == temperature\_yesterday) {

doSomething();

}

// 判断结果正确

if (fabs(temperature\_today - temperature\_yesterday) < 0.0000001) {

doSomething();

}

## 局部变量的指针和引用（要求）

函数返回时禁止返回在函数内部定义的局部变量，不论返回的是局部变量的指针还是其引用；指向局部变量的指针在函数返回后将变成野指针，运气好的情况下这个错误将立即引发崩溃，而有些时候这个错误会潜伏一段时间后才爆发导致程序崩溃，这时想要debug将会极其困难；在函数返回后局部变量被销毁，局部变量的引用也同时失去意义，因此函数返回的引用不能绑定在局部变量上；

## 不确定的语句顺序（要求）

禁止使用执行顺序不明确的语句，尤其是将会引发未定义行为的赋值语句；例如在下列语句中，由于语句只包含1个Sequence Point，赋值的顺序将由编译器在优化阶段决定，如果代码只要求在特定平台或编译器运行，那只是可读性差的问题，但如果代码需要跨平台或者编译器运行，则会产生意想不到的错误，带来不必要的风险；

int num = 3;

int result = (++num) + (++num) + (num++); // result的结果是不确定的；

void swap(int &a, int &b) {

a ^= b ^= a ^= b; // 此处的运算顺序将完全由编译器决定

}

// 由于doSomething函数调用之前是Sequence Point，因此function1与

// function2将优先于doSomething被调用，但function1与function2的调用顺

// 序却是未定义的，应避免类似的应用；

doSomething(function1(++num), function2(++num));

// function1和function2的求值顺序将由编译器决定，若function1或function2

// 中包含对全局变量或静态变量的操作，可能会导致预期之外的结果；

// 除非function1和function2的返回结果完全不受执行顺序影响，否则禁止类似

// 的调用方式；

int result = function1() + function2();

# 命名规则

## 通用命名规则（推荐）

文件命名，函数命名，类型命名，变量命名等都都需要有描述性，让代码易于理解；文件名、类型名和变量名一般为名词，而函数名一般为动宾结构，例如：

double max\_velocity;

double planning\_time;

RobotModel robot\_model;

void waitForAction();

double getMaxVelocity();

不要使用随意的字符以及模糊的缩写命名，这会让除了作者之外的读者无法很快领会代码的意图，严重损害代码的可读性，甚至在一段时间后作者自己也弄不明白这样的命名代表了什么，例如下列名称就会让读者直挠头：

int xyz;

int cstmr\_id;

int wgc\_connections;

void fun(int a, double b, bool c);

命名中应该尽量少用缩写，除非是所有代码阅读者都熟知其意义的缩写，例如number->num，length->len，error->err，InverseKinematics->IK等：

int cnt;

int num;

ErrorCode err;

int computeIK(void);

## 文件命名（要求 - 新代码执行）

文件只能使用小写字符、数字加下划线的任意组合命名，且只能以小写字符起始，尽量使用有意义的单词组合做为文件名，且单词间以下滑线进行分割；为避免命名冲突，文件名的起始应冠以软件包名，例如：

ur10\_move\_group\_interface.h

fr701\_robot\_algorithm.cpp

fr701\_lib\_controller.cfg

一般情况下，应尽量让文件名更加明确，例如movej\_joint\_trajectory\_output.log就比output.log要好得多；定义类时文件名应与类名保持一致并且成对出现，例如在定义类ArmGroup时，对应的文件应为：

fr701\_arm\_group.h

fr701\_arm\_group.cpp

内联函数必须放在.h文件中，若内联函数比较短则直接放在相应.h文件中，若代码较长则放在以\_inl.h结尾的文件中；

## 类型命名（要求）

类型（类、结构体、类型定义typedef、枚举）以字符和数字的组合形式命名，命名尽量使用有意义的单词组合，且每个单词的首字母必须大写，例如：

class RobotModel {

…

};

struct JointValues {

…

};

enum ErrorCode {

…

};

typedef std::map<JointValues,string> JointValuesMap;

## 变量命名（要求）

变量名称全部使用小写字符、数字加下划线组成，命名尽量使用有意义的单词组合，且单词之间使用下划线进行分隔，例如：

int fifo\_length;

double max\_velocity;

JointValues joint\_current;

类的成员变量需以下划线结尾，例如：

class ArmGroup {

double cycle\_time\_;

JointValues latest\_ik\_reference\_;

…

};

## 特殊变量（推荐）

全局变量需冠以g\_做为前缀，静态变量需冠以s\_前缀，常量需冠以c\_前缀，例如：

int g\_buffer\_size = 1000;

static int s\_total\_connections;

const double c\_pi = 3.14159;

## 函数命名（要求）

函数命名可以使用大小写字符或数字的组合，要求采用有意义的单词组合而成，首单词使用小写字符其后每个单词的首字母使用大写字符，命名函数的单词组合应使用动宾结构，即第一个单词使用动词之后使用名词性单词，例如：

int getQueueLength(void);

bool setJointValueTarget(const JointValues &joint\_target);

bool isQueueLocked(void);

double computeFK(const JointValues &joint\_values, Pose &pose);

## 命名空间命名（要求）

命名空间的命名只能使用小写字符、数字和下划线的组合，且只能以字符起始，例如：

namespace fst\_controller

{

int getMaxVelocity(void);

}

## 枚举命名（待定）

由于枚举属于类型中的一种，所以枚举类型名参考3.3节类型命名，而枚举值名规则参考3.4的变量命名规则，且需要冠以e\_做为前缀，例如：

enum CommandType {

e\_type \_other = 0,

e\_type \_movel = 1,

e\_type \_movec = 2,

e\_type \_movej = 3,

};

## 宏命名（要求）

宏采用大写字符和下划线的组合来命名，宏命名中不能使用小写字符，例如：

#define \_DEBUG

#define PI 3.1415926

# 代码注释

## 注释风格（推荐）

对于文件注释、类注释、函数注释以及行尾注释，使用//做为注释的起始，行尾注释符//之后有1个空格，例如：

//-----------------------------------------------------------------------------

// Function: setSomeParameters

// Summary: To set some parameters in the class

// In: parameter1 -> input parameter 1

// parameter2 -> input parameter 2

// Out: output -> output

// Return: return1 -> return state 1

// return2 -> return state 2

//-----------------------------------------------------------------------------

ReturnType setSomeParameters(MyType parameter1,

MyType parameter2) {

if (conditions) {

setMaxVelocity(velocity\*scaling\_factor); // set MoveL velocity to xxxx

}

}

对于一段暂时不需要参加编译的代码，而将来又可能需要用到不能将之删除的情况，使用/\*\*/将其注释掉，例如：

if (conditions) {

/\*

setMaxVelocity(xxxx); // set cartesian space velocity to xxxx

setMaxAcceleration(xxxx); // set cartesian space acceleration to xxxx

\*/

doSometing();

}

对于注释中所使用的语言，约定使用英文。

## 文件注释（推荐）

一个完整的文件注释至少应该包含版权声明、文件名、版本号，创建日期、作者信息以及内容描述；如果该文件内容发生过修改，还应该包含修改日期，修改人信息和修改内容描述；文件每经过一次修改后都应在原文件注释的末尾新增加修改日期、修改人和修改内容描述三项并更新版本号；一个完成的文件注释格式类似于：

//-----------------------------------------------------------------------------

// Copyright(c) 2015-2020 Foresight-Robotics

// All rights reserved.

//

// File: xxxx.h

// Version: x.x.xx

//

// Create: Aug.16 2016

// Author: Ying, Zhang

// Summary: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

// xxxxxxxxxxxxxxxxxxxxxxx.

//-----------------------------------------------------------------------------

## 类注释（推荐）

每个类在定义的同时应该附加描述该类功能和用法的注释，例如：

//-----------------------------------------------------------------------------

// Brief client class for the MoveGroup action. This class includes many

// default settings to make things easy to use.

// Sample usage:

// MoveGroup \*new\_group = new MoveGroup();

// std::string group\_name = new\_group->getName();

// delete new\_group;

//-----------------------------------------------------------------------------

class MoveGroup {

…

};

## 函数注释（推荐）

函数声明以及函数定义处均要添加注释，一般在函数声明出描述函数的功能及用法，而在函数的定义处描述函数的功能和实现要点；函数注释应包含方法名、输入参数、输出参数、返回值和简要描述，例如：

//-----------------------------------------------------------------------------

// Function: setMaxVelocityScalingFactor

// Summary: Set a scaling factor for optionally reducing the maximum joint

// velocity. Allowed values are in (0,1]. The maximum joint velocity

// specified in the robot model is multiplied by the factor. If outside valid

// range (importantly, this includes it being set to 0.0), the factor is set to

// a default value of 1.0 internally (i.e. maximum joint velocity).

// In: max\_velocity\_scaling\_factor

// Out: None

// Return: true -> set successfully

// false -> set unsuccessfully

//-----------------------------------------------------------------------------

bool setMaxVelocityScalingFactor(double max\_velocity\_scaling\_factor);

## 变量注释（推荐）

通常来说变量名就是最好的注释，特定情况下，需要额外的注释说明；类的成员变量应添加注释说明用途，如果变量可以接受NULL或-1等警戒值，必须注释说明，例如：

// Keeps track of the total number of entries in the table.

// Used to ensure we do not go over the limit. -1 means that

// we don't yet know how many entries the table has.

int num\_total\_entries\_;

和类的成员变量相似，所有全局变量、静态变量和符号常量均需要添加注释说明其含义及用途，例如：

// The total number of tests cases that we run through in this regression test.

const int c\_num\_test\_cases = 6;

## 代码中注释（推荐）

对于在代码中巧妙的、晦涩的、重要的、有趣的地方加以注释；对于代码块的注释应位于代码块的上方，且与代码块同缩进；对于代码行注释应位于该行代码之后，且注释与代码之间添加若干空格，对于前后几行代码都有注释的情况下，应当适当调整空格的数量使这些注释处于同一起始位置，使之可读性更好；例如：

// Divide result by two, taking into account that x

// contains the carry from the add.

for (int i = 0; i < result->size(); i++) {

x = (x << 8) + (\*result)[i];

(\*result)[i] = x >> 1;

x &= 1;

}

// If we have enough memory, mmap the data portion too.

mmap\_budget = max<int64>(0, mmap\_budget - index\_->length());

if (mmap\_budget >= data\_size\_ && !MmapData(mmap\_chunk\_bytes, mlock))

return; // Error already logged.

DoSomething(); // Comment here so the comments line up.

DoSomethingElseThatIsLonger(); // Comment here so there are two spaces

// between the code and the comment.

在函数调用时若存在传递的参数为数值、布尔值或空指针的情况，推荐使用注释对参数的意义进行说明，例如：

int result = calculateSomething(interesting\_value,

10,

false,

NULL);

加入注释后代码变得更加友善：

int result = caculateSomething(interesting\_value,

10, // Default base value

false, // Not the first time we’re calling this

NULL); // No callback.

所有独占一行的右大括号，应在其后添加注释，表明它与它所对应的左大括号所服务的语句；除非左右大括号相距很近，且对应情况一目了然时可以不添加注释；例如：

if (condition1) {

for (int cnt=0; cnt < 10; ++cnt) {

switch (cnt) {

case 0:

doSomethine1();

break;

case 1:

doSomething2()

break;

} // switch (cnt)

} // for (int cnt=0; cnt < 10; ++cnt)

} // if (condition1)

if (condition2) {

doSomething3();

} // 此处的大括号对应情况很清楚，可以不使用注释

# 代码格式

## 行长度（要求）

出于方便阅读者的考虑，原则上每一行代码的长度不应该超过120个字符，如果某行长度超出限制，应采取合适的方式进行分行，以下几种情况可不遵守此规则：

1. 注释中包含了超过120个字符的命令或URL，出于复制粘贴的方便可以无视该原则；
2. 包含长路径的文件可以超出该限制，但是应尽量避免包含长路径文件；
3. 头文件保护可以无视该原则；

## 缩进（要求）

代码缩进只能使用空格，且每个层次缩进4个空格；不能使用Tabs缩进，设定编辑器将Tabs转为空格；

## 函数声明与定义（要求）

函数返回值、函数名和左圆括号总是在同一行，且左圆括号和函数名之间没有空格；

函数形参应尽量与函数名位于同一行，且形参与左圆括号间没有空格，若放不下则应对形参进行分行，同一行中的形参之间总是有1个空格相隔，多行形参的起始位置应对齐，独立封装的参数保持2个缩进层次；

右圆括号应与最后一个参数位于同一行，且右圆括号和参数之间没有空格；右大括号位于函数体的最后，独立成行；

（待定）左大括号与右圆括号同处一行，且右圆括号与左大括号之间有1个空格相隔；

例如：

ReturnType ClassName::functionName(Type name1, Type name2) {

DoSomething(); // 4 空格缩进

}

ReturnType ClassName::longFunctionName(Type name1, Type name2,

Type name3) { //与上一行对齐

DoSomething(); // 4 空格缩进

}

ReturnType LongClassName::reallyReallyReallyLongFunctionName(

Type par\_name1, // 8 空格缩进

Type par\_name2,

Type par\_name3) {

DoSomething(); // 4 空格缩进

}

## 函数调用（要求）

调用函数时要么一行写完函数调用，要么对参数进行分行，要么参数另起一行且缩进8格；可以适当将多个参数放在同一行中，也可以每行只放一个参数；圆括号与参数间没有空格，同一行的参数间有1个空格相隔；例如：

bool value = doSomething(argument1, argument2, argument3);

bool value = doSomething(very\_very\_very\_long\_argument1,

argument2, argument3); //与上一行参数对齐

bool value = doSomethingThatRequiresALongFunctionName (

argument1, argument2, // 8 空格缩进

argument3, argument4);

bool value = doSomethingThatRequiresALongFunctionName(

very\_long\_argument1, // 8 空格缩进

argument2, // 与上一行对齐

argument3 \* 5 + argument4);

## 条件语句（要求）

if与左圆括号间有1个空格，左圆括号之后与右圆括号之前没有空格；if中只有1条语句时可以不使用大括号，且语句另起一行放置，if中包含1条以上语句时必须使用大括号，右大括号独立成行；else需要另起一行，其他格式参考if；

左大括号或者与右大括号同处一行且相隔1个空格，或者独立成行；例如：

if (condition)

doSomething(); // 只有一条语句可以不使用{}

if (condition1) { // 1条以上语句需使用大括号，且左大括号与右圆括号同行

doSomthing();

doOtherthing();

} // if (condition1)

else if (condition2) { // else另起一行，其他格式参考if语句

getSomething();

setSomething();

} // else if (condition2)

else { // 如果if中有分支使用了{}，则其他分支必须使用{}

continue;

} // else

if (condition1 && condition2) {

doSomething();

} // if (condition1 && condition2)

if (condition1 && condition2 || // 条件很长时可以分行放置

very\_very\_long\_condition3 ||

condition4) {

doSomething();

} // if (very\_very\_vert\_long\_condition1 &&

## 循环语句（要求）

for与左圆括号间有1个空格，左圆括号之后与右圆括号之前没有空格；for语句中的三条子语句之间有1个空格；空循环体必须使用{}或continue表示，不能仅使用分号；for语句的循环体只有1条语句时可以不使用大括号，且另起一行放置，循环体中包含超过1条语句时需使用大括号；

左大括号或者与右圆括号同处一行且相隔1个空格，或者独立成行，右大括号独立成行；while、do-while语句的格式参照for语句格式，例如：

while (true)

continue;

for (int cnt = 0; cnt < 10; ++cnt)

printf(“Something\n”);

while (cnt > 0) {

printf(“Doing something\n”);

doSomthing();

}

do { // do与{间有一个空格

printf(“Doing something\n”);

doSomething();

} while(cnt > 0); // while与右大括号间有1个空格

## 分支语句（要求）

switch与左圆括号间有1个空格，左圆括号之后与右圆括号之前没有空格，左大括号或者与右圆括号同处一行且相隔1个空格，或者独立成行，右大括号独立成行；case语句独立成行，case之中的语句不使用大括号且保持4个空格的缩进；每个switch语句必须包含一个default分支，例如：

switch (value) {

case 0:

…

break;

case 1:

…

break;

case 2:

…

break;

default:

…

} // switch (value)

## 指针和引用表达式（要求）

表示取成员的句点(.)和箭头(->)前后没有空格，操作符\*和&之后没有空格，在定义和声明参数、变量时，\*和&与变量名之间没有空格，例如：

Student wang\_xin;

Student \*zhao\_si = new Student;

wang\_xin.age = 19;

zhao\_si->age = 20;

int &age = wang\_xin.age;

char \*a;

const string &str;

## 预处理指令（要求）

预处理指令不遵从缩进约定，所有预处理指令均从行首起始，例如：

if (condition\_1) {

if (condition\_2) {

#ifdef DEBUG

printSomething();

#endif

} // if (condition\_2)

} // if (condition\_1)

## 类格式（推荐）

类应以public、protected、private的次序声明，除第一个public之外，其他的关键词之前空出一行；成员的注释应在成员声明之前且相互紧邻，而下一个成员的注释前应空出一行；例如：

class MyCLass : OtherClass {

public: // 缩进2个空格

//-----------------------------------------------------------------------------

// Function: MyClass

// Summary: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

// In: None

// Out: None

// Return: None

//-----------------------------------------------------------------------------

MyClass();

//-----------------------------------------------------------------------------

// Function: ~MyClass

// Summary: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

// In: None

// Out: None

// Return: None

//-----------------------------------------------------------------------------

~MyClass();

//-----------------------------------------------------------------------------

// Function: doSomething

// Summary: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

// In: None

// Out: None

// Return: None

//-----------------------------------------------------------------------------

void doSomthing();

private:

int parameter1\_;

char parameter2\_;

};

## 命名空间（要求）

命名空间不增加额外的缩进层次，例如：

namespace {

//-----------------------------------------------------------------------------

// Function: doSomething

// …

//-----------------------------------------------------------------------------

void doSomething() {

…

}

}

## 水平留白（要求）

赋值符号、数学运算符、逻辑运算符前后应加入1个空格，逗号之后加入1个空格，其他情况下应酌情加入空格以使代码更加清晰，提高可读性，例如：

int sum = 0;

for(int cnt = 0; cnt < 100; ++cnt) {

sum += cnt;

num = sum \* 5 + (cnt + 17) \* cnt;

}

## 垂直留白（推荐）

为了增加屏幕上同时展示的代码行数，应尽量减少垂直留白；具有强逻辑相关性的代码块之间可以加入空行，以增加可读性；