

C++ 编码风格

一、通用:

- 使用等于 4 个空格的 TAB
- 无论在何处，都应使用尾部大括号形式 (if,else,函数,结构体,typedef,class 等)。
- 表达式中括号间需要内嵌空格：

```
x = ( y * 0.5f );
```

```
x=(y*0.5f);
```

- 使用精确的浮点描述值:

```
float f = 0.5f;
```

```
float f = 0.5;
```

```
float f = 1.0f;
```

```
float f = 1.f;
```

- 函数名称以大写字母开头：

```
void Function( void );
```

- 多单词的函数，则每个单词首字母大写：

```
void ThisFunctionDoesSomething( void );
```

- 标准的函数头：

```
/**
```

```
    函数名称  函数功能描述
```

```
**/
```

- 变量命名遵循小写字母开始的 camelCase。

```
float someValue;
```

- 多单词的变量，首字母以小写字母开始，其余单词首字母大写：

如: float maxDistanceFromPlane;

- 全局变量, 以“g_”前缀的全小写形式:

```
extern GameCore*    g_gamecore;
```

- typedef 名称规则和类型命名类似, 前缀“A”:

```
typedef int AFileHandle;
```

- 结构体使用前缀大写 S, 表明为结构体:

```
struct SRenderEntity;
```

- enum 命名规则和变量类似, 以 E 开头, 每个值都以 k 开头:

如: enum EContact {

```
    kContact_Node,
```

```
    kContact_Edge,
```

```
    kContact_ModelVertex,
```

```
};
```

或者: c++11:

```
enum class EContact{
```

```
    kNode,
```

```
    kEdge,
```

```
    kModelVertex
```

```
};
```

- 递归函数名称以 “_r” 结尾: void WalkBSP_r(int node);
- 导出函数名称以 “_f” 结尾: void LuaWalkBSP_f(int node);
- 宏与常量名称统一大写, 多词则以下划线分开:

如: `define SIDE_FRONT 0`

- 尽可能地使用 `const`。

如: `const int * p; int * const p; const int * const p;` 不要使用: `int const * p;`

- 泛型类型使用“T”开头, 泛型函数使用“T”后缀。

`template<class T> TList;`

`template<class T> MinT`

- 控制台命令和参数使用 `snake_case` 形式。

二、类:

- 类的标准开头:

```
/*  
  
=====
```

类的作用以及用法示例。

```
=====
```

**/*

- 类名称: 首字母大写, 多单词类名则每个单词首字母大写, 并且前缀“F”:

如: `class FVec3;`

`class FSceneManager;`

- 接口类: 前缀“I”:

如: `struct IGameCore;`

`struct IFileSystem;`

- 类变量名称规则和变量名称规则相同:

如: `class Vec3 { float x; float y; float z; };`

- 类方法和函数的名称规则相同:

`class Vec3 { float Length(void) const; };`

- 将类的变量和方法缩进, 以保持垂直空间的美观。类型作为第一列, 标识符作为第二列:

如:

```
class Vec3 {  
  
    float x;  
  
    float y;  
  
    float z;  
  
    float      Length( void ) const;  
  
    const float* ToFloatPtr( void ) const;  
  
};
```

- ``*`号放在第一列以表示作为类型的一部分。

如: `int* data;` `int *data;`

- 类变量和方法的顺序:

1. friend 类
2. public 变量
3. public 方法
4. protected 变量
5. protected 方法
6. private 变量
7. private 方法

- 如果不改变类的成员变量，则尽可能将类的方法设为 const。
- 避免使用 const_cast，当需要改变对象时，以非 const 形式重载函数。
- 避免函数重载，避免隐式类型转换带来的错误：

如：

```
const Anim *      GetAnimByIndex( int index ) const;  
const Anim *      GetAnimByName( const char *name ) const;  
const Anim *      GetRandomAnim( float randomDiversity ) const;
```

而非：

```
const Anim *      GetAnim( int index ) const;  
const Anim *      GetAnim( const char *name ) const;  
const Anim *      GetAnim( float randomDiversity ) const;
```

如上所示，重载的 GetAnim 函数，将一个 short 值作为参数传入，会发生隐式类型转换，而发生调用出错。

- 谨慎使用操作符重载。操作符重载通常用来简化代码，但通常会产生词不达意的弊端。

如果没有必要，一定不要使用操作符重载

三、文件名：

- 如果一个类的实现文件太庞大，可以根据不同的功能分成多个源文件：

如： class FGameCore; 可以分为： GameCore.h 与

GameCore_frame.cc,GameCore_print.cc,GameCore_debug.cc 等

- 按功能划分头文件和源文件，而非一个类一个头文件及其源文件。

四、杂项：

- 复杂的判断，需要断为多行：

```
如： if (foo->next == NULL &&  
  
      totalCount < needed &&  
  
      needed <= MAX_ALLOT &&  
  
      server_active(currendInput)){  
  
      ....  
  
}
```

- 在循环中，尽量将中间值提取出来：

```
如： size_t strLength = strlen(str);  
  
     for ( size_t i = 0; i < strLength; ++i ) { }
```

而非：

```
for ( int i = 0; i < strlen(str); ++i ) { }
```

- 尽量克制地使用 template。template 会导致代码膨胀和编译时间增长。Template 应该只在底层库中使用，逻辑层不该滥用 template。
- 尽量克制地使用 C++ 标准库可其他 C++ 模板库，使用 C 标准库。使代码更加可控。
- 在跨平台时，不要假设系统的字长。如果不太明确时，使用 void* 和 void(*)(), 它们能保证有足够地空间容纳任意数据对象和任意函数。
- 优先复用已有代码，其次再实现。
- 尽可能将关键的值声明为 volatile。
- 对浮点值判断，永远不要使用精确比较（即 == !=），而应该使用范围比较。

如：

```
bool IsZero(float f) {
    if ( t + 0.001 >= 0 && t <= 0.001f) return true;
    return false;
}
```

替换：

```
bool IsZero(float f) {
    return f == 0.0f;
}
```

- using namespace xxx; 应出现在源文件而非头文件中。
- 可用匿名命名空间组织代码结构。
- 尽量使用前向声明，而非文件包含。
- 头文件包含顺序：
 1. 类的自身头文件
 2. C 库文件
 3. C++库文件
 4. 其他第三方库文件
 5. 本地项目库文件
- 谨慎使用 C++ 的最新特性。不要追赶 C++ 特性潮流，采用最熟悉、最稳妥的、保证可控的技巧。

游戏中，采用的技术是图形等方面的最新技术，而非关注编程语言的最新技术；同时，游戏开发关注的是 **协作** 与 **快速开发**，写出的代码应该****简洁****、****浅显易懂****、****明确可控****。并非所有的编译器都支持最新的 C++ 标准。

作为商业项目，应该首选已经成熟稳定的特性，而非危险、易变、难以捉摸的特性。