GSA C++编码规范

1 头文件

通常，每个 C++源文件都有一个对应的头文件，正确使用头文件可以提高代码的可读性、减少文件大小，同时提高编译性能。

1.1 #define保护

使用 #define 防止头文件被多次引入，命名格式为：_PROJECT_PATH_FILE_H_。为保证唯一性，头文件的命名应基于其所在项目的路径，如 foo/bar/baz.h，可定义如下：

```
#ifndef _FOO_BAR_BAZ_H_
#define _FOO_BAR_BAZ_H_

#endif //_FOO_BAR_BAZ_H_
```

1.2 内联函数 - inline

基本原则是函数执行体代码行数少（小于 10 行）时将其定义为内联函数（如存取函数或短小的关键执行函数）。

注意：类的构造函数和析构函数要小心对待，因为编译器会在背后扩展添加其他代码。

注意：内联函数中最好不要有循环语句或分支代码。

1.3 -inl.h头文件

对于复杂的内联函数定义，或函数模板、类模板成员函数定义，可以单独置于-inl.h 头文件中，既可以在需要时引入，也方便进行代码管理。

1.4 函数声明

声明函数时，其参数顺序为：先输入参数，后输出参数（输入/输出参数）。

输入参数：按值传递（内置类型）或常量引用（const&）。

输出参数（输入/输出参数）：非常量指针（由于引用在语法上是值，但拥有指针的语义，为避免误用，尽量不使用非常量引用）。

1.5 头文件的包含

将头文件按照一定的次序包含可以增强可读性、减少隐性依赖：C 库 -> C++库 -> 其他库的头文件 -> 本项目的头文件。头文件引用不要使用绝对路径。

2 作用域

2.1 using声明

头文件：不在全局范围内使用 using 声明，如 using namespace std; 这将导致 std 命名空间里所有定义的标识符对包含此头文件的所有源码文件可见，仅在头文件的函数、方法或类中使用 using 声明。

实现文件：可以使用 using 声明。

2.2 局部变量

在函数中使用的局部变量尽可能置于最小作用域内，声明的同时完成初始化。对于仅用于 for, if, while 语句内的变量，在其循环或条件的构建语句中定义。

2.3 全局变量

由于全局变量在构造、初始化、析构的调用顺序上存在不确定性，所以不要使用 class 类型的全局变量（需要时用单件模式替代 class 类型的全局变量），可以使用内置类型的全局变量。

对于字符串常量，使用 C 字符串，不要使用 STL 中的字符串。

注意：静态成员变量视作全局变量，所以也不能是 class 类型。

3 C++类

3.1 构造函数

构造函数的职责只负责成员数据的初始化，且确保异常安全。对单参数的构造函数使用 explicit 关键字，防止意外的类型转换。

明确类型的拷贝构造函数和拷贝赋值操作符语义，即使使用编译器缺省定义，也使用 default 关键字明确定义：

```
Foo(Foo const&) = default;
Foo& operator = (Foo const&) = default;
```

如果类型禁止通过已有对象创建或赋值，使用 delete 关键字明确禁止：

```
Foo(Foo const&) = delete;
```

```
Foo& operator = (Foo const&) = delete;
```

3.2 继承

作为子对象，成员对象通常比基类对象更适合，所以优先使用组合，而非继承。

严格区分实现继承和接口继承的语义差别：纯虚拟方法（接口继承）、虚拟方法（接口和实现继承，子类可变）、非虚拟方法（实现继承，子类不可变）。

严格限制使用多重继承，仅在最多只有一个实现基类，其余都是接口基类时使用。

3.3 定义

数据成员定义为 private（struct 定义的数据类例外），仅在子类中访问的成员方法为 protected。

用于基类（且通过基类指针或引用访问）或含有虚拟方法的类，其析构函数也为虚拟。子类覆写虚拟函数时添加 override 关键字，确保函数规格的一致性。

4 语言特性

4.1 类型转换

static_cast<>：替换所有()的类型转换，使操作语义更加明确。

const_cast<>：仅用于移除 const 限定。

reinterpret_cast<>：只在你了解其后果的情形下使用。

dynamic_cast<>：仅用于测试代码，在实际代码中使用说明设计有缺陷。

4.2 整型

C++没有规定 int 类型的位数，使用<stdint.h>中的精确位数的整型，如 int16_t, uint64_t，降低代码的编译器或平台依赖性。还可以使用标准类型，如 size_t, ptrdiff_t。

仅在需要表示一个位组而非一个数值时使用无符号整型，如果需要确定数值不能为负时，使用断言。

Bug1：for (unsigned i = foo.Length() – 1; i >= 0; --i) 导致无限循环。

Bug2：关系运算符两边操作数为有符号变量和无符号变量时，C 语言内置的类型提升操作会造成出人意料的行为。

4.3 sizeof

尽可能使用 sizeof(varname)，而非 sizeof(type)，防止变量类型改变时无意引入的 bug。

5 命名约定

命名对提高代码的自描述性至关重要，通过一致性的命名风格可以明确命名实体：类型、变量、函数、常量、宏等等。

5.1 通用规则

优先考虑其描述性，不要过度缩写。

好的命名：

```
int num_errors;
```

```
int num_completed_connections;
```

不好的命名：

```
int nerr;
```

```
int n_comp_conns;
```

类型和变量名一般为名词：FileOpener, num_errors

函数名多为动词：OpenFile(), set_num_errors()

5.2 文件命名

文件名全部小写，可以按照不同项目的约定，中间用下划线_或短线-分隔：myusefulclass.h, my_useful_class.h, my-useful-class.h。

头文件以.h 结尾，C 源码文件以.c 结尾，C++源码文件以.cpp 结尾。若采用仅头文件实现，没有对应的源码文件，则以.hpp 结尾。

5.3 类型命名

类型命名（类、结构体、typedef、enum）每个单词以大写字母开头：MyExcitingClass。对于常用的简写或缩略语，第一个字母大写，其余小写：class UrlTable;

5.4 变量命名

变量名一律小写，单词间以下划线_分隔，类的成员变量以下划线_结尾：my_exciting_local_variable, my_exciting_member_variable_

结构体的数据成员命名同普通变量一样。

全局变量以 g_作为前缀（尽量少用全局变量）。

静态成员变量以 s_作为前缀。

5.5 常量命名

在名称前面加 k：const　int　kDaysInAWeek = 7;

5.6 函数命名

接口函数：大写字母开头，每个单词首字母大写：AddTableEntry()

get|set 函数：与变量名匹配：int age(); void set_age(int v);

对于短小的内联函数，小写字母，单词间以下划线_分隔。

私有函数：同接口函数，以下划线_结尾。

5.7 命名空间

命名空间的名称全部小写字母，单词间以下划线_分隔，命名基于项目名称和目录结构：myproject_basic_config

5.8 枚举命名

枚举类型命名参照类型命名，而枚举值则全部大写，单词间以下划线分隔：

```
enum UrlTableErrors
{
    ERROR_OUT_OF_MEMORY,
};
```

5.9 宏命名

参照枚举值的命名方式：

```
#define ROUND(X) ...

#define PI_ROUNDED 3.0
```

5.10 命名规则例外

如果命名的实体是对已有的 C/C++/STL 标准库的封装，请使用被封装实体相同的命名方式。

6 代码注释

代码本身就是最好的注释，选用描述性更好的类型和变量名要远远好于用注释来描述。

通常，.h 文件是对所声明的类功能和使用方法的简要说明，.cpp 则包含了更多的实现细节或算法描述。

为了便于自动生成文档，采用 Doxygen 注释语法：

单行注释：///

多行注释：/**…*/

6.1 文件注释

通常放在整个文件开头。

```
/**
 * @file
 * @brief
 * @details
 * @author
 * @email
 * @version
 * @date
 */
```

6.2 类注释

使用@brief 填写类的简要描述，换行填写类的详细信息。

```
/**
 * @brief 类的简单描述
 * 类的详细概述
 */
```

6.3 常量/变量的注释

包括全局常量变量、宏、类成员变量、枚举成员等。注释可以采用代码前注释：

```
/// 缓存大小
#define BUF_SIZE 1024*4
```

或代码后注释：

```
#define BUF_SIZE 1024*4 ///< 缓存大小
```

6.4 函数注释

```
/**
 * @brief main()
 * @details 程序入口
 *
 * @param argc 命令参数数量
 * @param argv 命令参数指针数组
 * @return 程序执行成功与否
 *      @retval 0 程序执行成功
 *      @retval 1 程序执行失败
 * @note 简单示例
 */
int main(int argc, char* argv[])
```

7 格式

统一的代码格式便于阅读和理解代码。

7.1 行长度

每一行代码字符数不超过 80。

7.2 函数声明与定义

返回类型和函数名置于同一行，如有可能，参数也放在同一行。

```
ReturnType ClassName::FunctionName(Type par_name1, Type par_name2)
{
}
```

如果文本一行无法放下：

```
ReturnType LongClassName::ReallyReallyReallyLongFunctionName(
    Type par_name1,
    Type par_name2,
    Type par_name3)
{
}
```