

C & C++

1.

--	--

Google C++ Style Guide C C++
 ,
 ;
 , C++ ;
 : shjborage@gmail.com

2.

 / C++

2.1.

--	--	--	--	--	--

☐ C++ Object-Oriented C++

2.1.1. struct vs class

```
[ ] struct [ ] ( /  
[ ], initialize(), reset(), validate()) [ ] class  
[ ]
```

Diagram illustrating memory layout for C++ and C:

- C++ side:** A single long memory block is shown, with 'struct' and 'class' labels pointing to it.
- C side:** A separate memory block is shown, with a 'struct' label pointing to it.
- A 'C' label points to a small memory block.

```
// 0000000000000000
struct Coordinate {
    int x;
    int y;
    int z;
};
```

```
// 00000000
class Cat {
public:
    void meow();
private:
    ...
};
```

2.1.2.

```
·[ ] [ ]  
[ ] DISALLOW_COPY_AND_ASSIGN [ ]  
[ ] ( ), [ ] init() [ ]
```

The diagram consists of four horizontal bars, each composed of small rectangular segments. The second bar from the top is the longest and is labeled "C++" at its right end. The other three bars are shorter and are not labeled.

```
class Person {
public:
    // 强制使用 explicit
    explicit Person(const std::string& name) : _name(name) {} // 强制使用 explicit
private:
    std::string _name;
};

DISALLOW_COPY_AND_ASSIGN(Person)

#define DISALLOW_COPY_AND_ASSIGN(TypeName) \
    TypeName(const TypeName&); \
    TypeName& operator=(const TypeName&)

class Foo {
public:
```



```

class String {
public:
    String() : // 空字符串
        _str(_S_EMPTY_C_STR) {} // 空_S_EMPTY_C_STR字符串

    ~String() {
        if (_str != _S_EMPTY_C_STR) {
            free(_str);
        }
    }

    const char* c_str() const {
        return _str;
    }

private:
    char* _str;
    static char _S_EMPTY_C_STR[1]; // = ""
};

```

2.1.4. 字符串

·[]

字符串

·[]

字符串

字符串

·

·

implicit constructor MyClass::MyClass(MyArg arg) 字符串 MyArg

MyClass 字符串

explicit constructor explicit MyClass::MyClass(MyArg arg) 字符串

· explicit 字符串

字符串

bug

字符串

·

Person(const std::string&) 字符串

explicit

string

字符串

Person 字符串

```
class Person {
public:
    explicit Person(const std::string& name) : _name(name) {}
private:
    std::string _name;
};
```

string  C 

```
class String {
public:
    String(const char* cs);           // 文字列の初期化

private:
    ...
};
```

--	--

“ More Effective C++, Item 05: Be wary of user-defined conversion functions

2.1.5.

--	--	--	--	--	--

[illegible]

11

--	--	--	--	--	--

copy constructor	MyClass::MyClass(const MyClass&)
------------------	----------------------------------




11

[illegible][illegible]

--	--

--	--	--	--	--	--	--	--

```
class Point {
public:
    Point(int x, int y) : _x(x), _y(y) {}
    Point(const Point& other) :    // 拷贝构造
        _x(other._x), _y(other._y) {}
};
```

```
private:
    int _x;
    int _y;
}
```

~~~~~ (RAII~~~~~)

```
class File {
public:
    explicit File(const char* file_name) :
        _fp(NULL) {
        ... // open file, and set _fp
    }
    ~File() {
        fclose(_fp);
    }
private:
    FILE *_fp;
    DISALLOW_COPY_AND_ASSIGN(File);
};
```

□□

“ Effective C++, item 06: Explicitly disallow the use of compiler-generated functions you do not want Effective C++, item 14: Think carefully about copying behavior in resource-managing classes

## 2.1.6. ~~~~~

·[□□ ] ~~~~~

□□

~~~~~

assignment operator=□□ MyClass& MyClass::operator=(const MyClass&) ~~~~~
~~~~~

□□

~~~~~

~~~~~

[illegible]

11

2.1.7. 

|  |  |  |  |
|--|--|--|--|
|  |  |  |  |
|--|--|--|--|

 $\cdot [\square \square]$ 

·[ ] RULE010

• [ ]

• [ ]

~~~~~

❏ destructor❏❏ MyClass::~MyClass() ❏❏

❏ ~~~~~ delete pBase;
~~~~~

~~~~~

❏

~~~~~

❏ ~~~~~ RAII~~~~~

```
class Connection {  
public:  
    void close();    // 连接池"连接"池"连接->连接"  
    ~Connection() {  
        if (_connection_state == CONNECTED) {  
            close(); // 连接池"连接"池"连接->连接"  
        }  
    }  
};
```

❏

“Effective C++, item 07: Declare destructors virtual in polymorphic base classes  
Effective C++, item 08: Prevent exceptions from leaving destructors More  
Effective C++, item 11: Prevent exceptions from leaving destructors

## 2.1.8. ❏

·[❏ ] ~~~~~ “❏ ”~~~~~ ·[❏ ]  
~~~~~ ·[❏ ] [RULE014] ~~~~~

❏

~~~~~

~~~~~

~~~~~ bug ❏❏ “❏ ”

~~~~~

❏ ❏❏ “❏ ”~~~~~

```
class Chinese {
public:
    explicit Chinese(const std::string& name) : _name(name) {}

    // 析构函数
    virtual ~Chinese() {}

    // 说话函数
    void say(const std::string& word);

    // 打招呼函数
    virtual void greet() { say("你好"); }

private:
    std::string _name;
};

// 实例化"小明"对象

class Cantonese : public Chinese {
public:
    explicit Cantonese(const std::string& name) : Chinese(name) {}

    virtual ~Cantonese() {}

    // 打招呼函数
    virtual void greet() { say("你好?"); }
};
```

Effective C++, Item 36: Never redefine an inherited non-virtual function

Effective C++, Item 32: Make sure public inheritance models “is-a”

Effective C++, Item 38: Model “has-a” or “is-implemented-in-terms-of” through composition

Effective C++, Item 39: Use private inheritance judiciously

C++ Coding Standards, § 37

C++ Coding Standards, § 34


```

// 资源池
typedef void(*DestructorFunc)(void * p_object); // 析构函数
// 资源池的构造函数/析构函数
ResourcePool();
~ResourcePool() {
    // 资源池的析构函数inline
    reset();
}

// 资源池
void attach(void * p_object, DestructorFunc destructor); // 添加资源池buffer
... // attach

template <typename T>
T& create(); // 创建
... // create
... // 析构

create_raw(),clone_cstring(),reset()
private:
    // private
    // 资源池
    struct ObjectInfo; // 资源池信息
    ...
    // 资源池
    void push_info(); // 资源池push_info()pop_info()
    void pop_info(); // 资源池private
    ...
    // 资源池
    ObjectInfo *_p_object_info_list;
    ...
    DISALLOW_COPY_AND_ASSIGN(ResourcePool); // 禁止拷贝
};

```

2.1.12. 字典的迭代器

```

// 字典的迭代器
// 字典的迭代器

```

```

// 字典的迭代器 DictIterator
Dict& )

```

```
operator ==( const Dict&, const
```

--	--

--	--


```

// a.h
#include <string>

using namespace std; // 命名空间using namespace

class A {
public:
    string name() const;
}

// a.cpp
#include "bsl/bsl_string.h"
#include "a.h"

using namespace bsl; // 在.cpp文件中using namespace

int main() {
    string str; // 使用命名空间bsl::string和std::string

    A a;
    cout << a.name() << endl; // 使用std命名空间
}

```

编译

```

// a.h
#include <string>

class A {
public:
    std::string name() const; // 使用std::string和string命名空间
};

// a.cpp
#include "bsl/bsl_string.h"
#include "a.h"

using std::cout; // 使用using class命名空间
using std::endl;

int main() {

```

```
bsl::string str; // 空字符串

A a;

cout << a.name() << endl;

}
```

2.2.2.

[] [RULE025] [] ·[] [] ([] 10 [])

☐ ☐

```
class Person {
public:
    std::string get_name() const { // 名前を返す
        return _name;
    }
    int get_age() const {
        return _age;
    }
private:
    std::string _name;
    int _age;
};

inline void greet(const Person& person) { // inline関数
    std::cout << "Hello, " << person.get_name() << "!" << std::endl;
}
```

Effective C++, item 30: understand the ins and outs of inlining

2.2.3.

• []

int arr[100];

int i; i++;

```
class MyContainer {
public:
    class iterator {
    public:
        iterator& operator++() {
            ...
            return *this;
        }
    };
};
```

2.2.4. int / int

int arr[100];
int arr[100];
int arr[100];

int arr[100];
int arr[100];

int i; i = 0; i = 0;

NULL

2.2.5. int

int i; i = 0; i = 0;

int i; i = 0; i = 0;

bug

int

Effective C++, Item 37: Never redefine a function's inherited default parameter value

2.2.6. int

RTTI

int i; i = 0; i = 0;

RTTI

int

int i; i = 0; i = 0;

dynamic_cast

2.2.7. 强制类型转换

·[] 强制类型转换 C++ 强制类型转换

C++ 强制类型转换 C++ 强制类型转换 static_cast dynamic_cast const_cast reinterpret_cast

强制类型转换 强制类型转换 C++
_cast

static_cast void *

```
void foo(const void* network_buf) {  
    const MyBuf * buf = static_cast<const MyBuf*>(network_buf); //  
}
```

const_cast

```
// void old_copy(void *dest, const void *src, size_t n);  
//  
void old_copy(void* dest, void* src, size_t n);  
  
//  
void new_copy(void* dest, const void* src, size_t n) {  
    old_copy(dest, const_cast<void *>(src), n); //  
}
```

“ Effective C++, item 27: Minimize casting More Effective C++, item 02: Prefer C++ style casts

2.2.8. ++/--

·[] , ().

++/--

2.3. C/C++ ☐☐☐☐

2.3.1.

--	--	--	--	--

[illegible][illegible]

11

```
// 初始化
if (is_valid) {
    ...
}

if (!is_finished) {
    ...
}

// 计算
if (my_value == 0) {
    ...
}

// 打印
if (p_value == NULL) {
    ...
}

// 检查边界条件
const double EPSILON = 1e-9; // 定义一个小的正数
if (fabs(my_value) < EPSILON) {
    ...
}
```

2.3.2. NULL, nullptr 0

·[] 0 0.0 NULL '\0' 0 ·[] C++11
 nullptr

·[]

·[]

```
int i = 0;
double i = 0.0;
void* p = NULL;
char ch = '\0';
```

2.3.3. sizeof

·[] sizeof() ·[] ·[] sizeof()
 sizeof() 32/64
 sizeof() sizeof()
 bug

·[]

```
int g_my_id;

void func() {
    size_t size_of_my_id = sizeof(g_my_id);    // OK
}

int g_my_id;

void func() {
    size_t size_of_my_id = sizeof(int);        // NO! g_my_id long long size_t
}
```

2.3.4. typedef

·[] type tag typedef ·[]
 struct typedef gcc

·[] typedef typedef

·[] YES:

```
class MyContainer {
public:
```

```

typedef char value_type;    // type tag

void my_method() {
    typedef std::vector<int>::iterator iterator_type;    //iterator
    ...
}
}

```

NO:

```

typedef tagMyStruct {
    ...
} MyStruct; // C++C

```

2.3.5. goto

·[] [[RULE037]] goto

goto

2.3.6.

·[]
DISALLOW_COPY_AND_ASSIGN, CFATAL_LOG etc.)
·[]
·[]
return ·[] / /
do...while(0)

break, continue,
#

inline

inline

```

inline int abs(int i) {
    return (i >=0 ? i : -i);
}

// 
template <typename T>
inline T abs( T value ) {

```

```
return (value >= 0 ? value : -value);
}
```

[illegible]

```
enum Color {
    RED,
    BLUE,
    GREEN,
    COLOR_COUNT
};
```

```
const [ ]
```

```
const size_t BUFFER_SIZE = 1048576;
```

--	--

“ Effective C++, item 02: Prefer consts, enums, and inlines to #define

2.3.7. ☐ ☐ ☐ ☐

[illegible]

11

[illegible]

--	--	--	--	--

The image shows two tens rods and six ones units. The tens rods are represented by two vertical rectangles, each divided into ten horizontal segments. The ones units are represented by six small squares.

```
class Iterator; //[] [] [] []
class Container {
    Iterator* begin();
};
```

--	--

“ Effective C++, item 31: Minimize compilation dependencies between files

2.3.8. / /

```
·[ ] singleton·[ ] class
·[ ] ·[ ] main()
·[ ] /
·[ ]
·[ ]
·[ ]

    ,
C++ , ,
bug. , , ,
, C++
:
some_compiler_1-3-8 crash
/

```

2.3.9.

```
·[ ] [RULE050] /·[ ] [RULE051]
enum const define·[ ] enum
·[ ] ·[ ]
·[ ] ·[ ]
·[ ] XXX_COUNT(·[ ] )·[ ]
·[ ]

  define / define
·[ ]
·[ ]
·[ ] 8 strlen("filename")
·[ ] file_name
·[ ]

 
```

```
enum Color {
    RED,
    BLUE,
    GREEN,
    COLOR_COUNT
};
```

```
  const
```

```
const size_t BUFFER_SIZE = 1048576;
```

❏

“ Effective C++, item 02: Prefer consts, enums, and inlines to #define

2.3.10. const

·[] const const
const /
const_cast const ·[] const
const

const bug
const_cast C C const
const

```
class MyClass {  
public:  
    void my_const_method(const OtherClass& arg) const;  
};  
  
const MyClass my_obj;    // const  
const MyClass* p1;       //  
const MyClass* const p2 = &my_obj; //
```

❏

“ Effective C++, item 03: Use const whenever possible

2.3.11.

·[] ·[] 4KB

(variable-length array): , GCC

crash crash
crash

3. 头文件 / 源文件

3.1. 头文件

3.1.1. 头文件扩展名

头文件扩展名通常为 `.h` 或 `.hpp`。在 C 语言中，头文件扩展名为 `.h`；在 C++ 语言中，头文件扩展名为 `.hpp`。

头文件通常包含变量声明、函数声明、宏定义等。源文件通常包含变量定义、函数实现等。

头文件通常位于项目根目录下的 `include` 子目录中。源文件通常位于项目根目录下的 `src` 子目录中。

```
#include "foo/public/fooserver.h" // 头文件
#include <sys/types.h>
#include <unistd.h>
#include <hash_map>
#include <vector>
#include "base/basicTypes.h"
#include "base/commandlineflags.h"
#include "foo/public/bar.h"
```

3.1.2. `#include <>` 与 `#include ""`

`#include <>` 用于包含系统头文件，`#include ""` 用于包含用户头文件。

3.1.3. `#include` 的别名

头文件通常使用 `#include` 来包含其他头文件。

头文件通常使用 `#include` 来包含其他头文件。

头文件

```
#include "bsl/ResourcePool.h"
#include "bsl/var/IVar.h"
#include "mylib/MyClass.h"
```

3.1.4. 头文件 include guards

```
·[[] ] [RULE060] [] include guards ·[[] ] [RULE061] include guards[]  
<SVNPATH>_<FILE>_H[]HPP[] [] SVNPATH [] /[] : trunk, branch-  
XXX;
```

```
[] include guards [] include [] #ifndef, #define, #endif  
[]
```

```
[] [] include guards [] svn []  
include guards [] ; []
```

```
// [], svn [] https://svn.saick.net/ps/se/trunk/ac/strategy/  
StrategyQueue/default/time_open_cluster_data.h [] :
```

```
#ifndef PS_SE_AC_STRATEGY_STRATEGYQUEUE_DEFAULT_TIME_OPEN_CLUSTER_DATA_H  
#define PS_SE_AC_STRATEGY_STRATEGYQUEUE_DEFAULT_TIME_OPEN_CLUSTER_DATA_H  
...  
#endif // PS_SE_AC_STRATEGY_STRATEGYQUEUE_DEFAULT_TIME_OPEN_CLUSTER_DATA_H
```

3.2. []

3.2.1. []

·[[]] []

1. []
2. include guards
3. []
4. [], (,), /·[[]] CPP[]
5. []
6. []
7. [] ·[[]]
[]

[] []

[] []

```
// Copyright 2017 Saick net. All Rights Reserved.  
// Author: LastName FirstName (eric@saick.net)  
//  
// <[]>  
[]Author[]  
// Copyright 2017 Saick net. All Rights Reserved.
```

```
// Author: LastName FirstName (eric@saick.net)
//      LastName2 FirstName2 (x@saick.net)
//
// <XXXXXXXXXXXXXXXXXXXX>
```

3.2.2. XXXXX

```
·[XX ][RULE064]XXXXXXXXXXXX·[XX ]XXXXXXXXXXXX·[XX ]
[RULE066]XXXXXX·[XX ]XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XX : XX XXXXXXXXXXXXXXX; XX : XXXXXXX ifXX , XXXXXXX else/else if
XX , XX else/else ifXXXXX.
```

```
XX XXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
XX
```

```
// mylib/AwesomeClass.h
namespace mylib {
class AwesomeClass {
    void foo() { //XXXX
        ...
    }
    void bar();
}; // MyClass

} //namespace mylib

// mylib/AwesomeClass.cpp
namespace mylib {
void AwesomeClass::bar() {
    while (...) {
        if (...) {
            ...
        } else {
            ...
        }
    } // while loop
}

} // namespace mylib
```

3.2.3.

·[] [RULE068] ·[] 5 ·[] [RULE069] 4

```
  terminal
  4
```

3.2.3.1.

·[] [RULE070] namespace (100 ,); namespace

```
namespace bsl {
namespace var {
class IVar {
};

} // namespace var
} // namespace bsl
```

3.2.3.2.

·[] [RULE071]

3.2.3.3.

·[] [RULE072] public/protected/private

```
class MyClass {
public:
    void foo();
protected:
    void bar();
private:
    void meow();
```

```
int _haha;

};
```

3.2.3.4. 初始化器

在 C++ 中，初始化器用于在类成员函数之外初始化成员变量。初始化器通常用于初始化静态成员变量、常量成员变量以及需要在类成员函数之外初始化的非静态成员变量。初始化器通常以 `static` 关键字开头，后跟成员变量的名称和初始化值。例如，以下代码展示了如何初始化静态成员变量 `_some_var` 和 `_some_other_var`。

```
// 初始化器
MyClass::MyClass(int var) : _some_var(var), _some_other_var(var + 1) {}

// 初始化器 8 个空格 初始化器
// the first initializer line:
MyClass::MyClass(int var) :
    _some_var(var),           // 8 space indent
    _some_other_var(var + 1) { // lined up
    do_something();
    ...
}
```

3.2.3.5. 初始化器

在 C++ 中，初始化器用于在类成员函数之外初始化成员变量。初始化器通常用于初始化静态成员变量、常量成员变量以及需要在类成员函数之外初始化的非静态成员变量。初始化器通常以 `static` 关键字开头，后跟成员变量的名称和初始化值。例如，以下代码展示了如何初始化静态成员变量 `_some_var` 和 `_some_other_var`。

```
void foo();
void this_is_a_very_long_function_name(
    VeryLongLongDogType *p_my_dog,
    const std::map<std::string, int>& a_wierd_dictionary);
```

3.2.3.6. 初始化器

在 C++ 中，初始化器用于在类成员函数之外初始化成员变量。初始化器通常用于初始化静态成员变量、常量成员变量以及需要在类成员函数之外初始化的非静态成员变量。初始化器通常以 `static` 关键字开头，后跟成员变量的名称和初始化值。例如，以下代码展示了如何初始化静态成员变量 `_some_var` 和 `_some_other_var`。

```
grey_wolf.eat(white_sheep);
hungry_lion.eat(
    big_grey_wolf,
```

```

        little_white_sheep,
        quick_brown_fox,
        lazy_dog);

```

3.2.3.7. if/while

```

·[ ] [RULE076] if
8
·[ ]
if ... else if ...
else
//pass
//do nothing

```

```

if
if ... else if ...
else

```

```

if (you.come_from(EARTH)) {
    return "hello"; // 
} else if (you.come_from(MARS)) {
    return "@%!%#@!~&$%$"; // 
} else {
    throw YouCheatMeException() << BSL_EARG << "your name:" << you.name(); // 
}

// Demo 
if (you.com_from(EARTH)
    || you.com_from(MARS)
    || you.com_from(VENUS)) {
    return "you come from solar system";
}

```

“ 1TBS

3.2.3.8. switch

```

·[ ] switch
case
break
return
·[ ] switch
case

```

```
default [ ] //pass [ ] //do nothing [ ] switch [ ] case  
[ ] switch [ ] [ ]  
[ ] case [ ] break [ ] return [ ] case [ ] switch  
[ ] default [ ]  
[ ]
```

```
switch (cur_char) {
case '"':
case '\\': {
    return "it's a string!";
}
default:
    printf("not found yet");
    break;
}
```

```
switch (cur_char) {
case '\\':
    printf("wow!"); //break return
case '\\':
    return "it's a string!";
} // default
```

3.2.3.9.

--	--	--	--

[illegible]

The diagram illustrates the execution of a loop. A long horizontal bar represents memory. A 'while' loop is shown with a condition bar and a body bar. A 'do{...}' loop is shown with a body bar and a condition bar. A 'do-' loop is shown with a condition bar and a body bar.

```
while (iter.next()) {
    // pass
}
```

--	--	--	--	--	--	--

--	--

3.2.4.

--	--	--	--	--

$$a.b, a \rightarrow b, a.*b, a \rightarrow *b, a::b \quad \cdot [\top] \text{ [RULE092]}$$

--	--

```

call_some_func(arg1, arg2, arg3);    // []
call_some_func (arg1, arg2, arg3); // [], []
call_some_func( arg1, arg2, arg3 ); // [], []
call_some_func(arg1,arg2,arg3);      // [], []
call_some_func(arg1 , arg2 , arg3); // [], []

template <typename T>                // []
template<typename T>                  // [], template<[]>

class Derived : public Base // []
class Derived:public Base    // [], []

MyClass::MyClass() : _member_var(0) // []
MyClass::MyClass():_member_var(0)   // [], []

++i; !i; ~i; *i; &i;                // []
++ i; ! i; ~ i; * i; & i;           // []

a + b; a ~ b; a || b; a << b; a = b; a %= b; // []
a+b; a~b; a||b; a<<b; a=b; a%=b;           // [], []

a ? b : c    // []
a?b:c        // [], []

a.b; a->b; a.*b; a->*b; a::b;           // []
a . b; a -> b; a .* b; a ->* b; a :: b; // [], []

xx_comparator(a, b); xx_map["key"];    // []
xx_comparator (a, b); xx_map ["key"];  // []
xx_comparator( a, b ); xx_map[ "key" ]; // []

```

3.2.5. []

·[[]][RULE093] [] 100
 []

[] []
 []
 [] 12[] laptop[]
 [],


```

// 常量成员函数 (常量成员函数只能访问静态成员变量), 常量成员函数 ``const`` 的
// 声明, 常量成员函数: 静态, 常量成员函数. 静态/常量成员函数 (静态/常量成员函数) 只能访问静态成员变量, 常量成员函数.
*/

// 常量成员函数const
void foo(const std::string& input1, const MyClass& input2);

// 常量成员函数
void foo(int input1, float input2);

// 常量成员函数const
void foo(const MyClass* input1);

// 常量成员函数
void foo(int* output1, MyClass* output2);

// 常量成员函数const
void foo(std::string* out);

// 常量成员函数
void foo(const InClass& input1, OutClass* out1);

// 常量成员函数
void foo(const InClass& input1, // 常量const
        int input2, // 常量POD
        OutClass* output1); // 常量const

// 常量成员函数
void foo(const InClass& input1,
        int* inout, // 常量const
        OutClass* output1)

// 常量成员函数:
// Always have named parameters in interfaces.
class Shape {
public:
    virtual void rotate(double radians) = 0;
}

// Always have named parameters in the declaration.
class Circle : public Shape {

```

```
public:
    virtual void rotate(double radians);
}

// Comment out unused named parameters in definitions.
void Circle::rotate(double /*radians*/) {}
```

3.3. $\mathbb{Z}/\mathbb{Z}$

3.3.1.

--	--	--	--	--

```
[...] [ ]
[ ]
[ ]
[ ]
[ ]
[ ]
[ ]
[ ]

[ ] [ ]
[ ]
[ ]
    vse [ ]
_evaluator [ ] [FALL BACK] [ ] extern "C" [ ] /
[ ]
[ ]
```

```
// 000000000000 00000000000 000000000000000000;

// 00000000

int num_errors;           // Good.

int num_completed_connections; // Good.


// 00000000

int n;                    // Bad - 000

int nerr;                 // Bad - 0000000

int n_comp_conns;        // Bad - ambiguous abbreviation.


// 00: 000000000000;

// 0000:

int num_dns_connections; // 00000 "DNS"00000

int price_count_reader;   // OK, price count. Makes sense.
```

```
// 连接
int wgc_connections;      // wgc 连接??
int pc_reader;            // pc连接数!

// 错误计数
int error_count;          // Good!
int error_cnt;            // Bad! 连接数, 错误数;
```

3.3.2. 命名空间

·[命名空间] 命名空间 .h 或 .hpp 文件 C++ 命名空间 .cpp 文件 C 命名空间 .c 文件 ·[命名空间] [RULE100] 命名空间, 命名空间; 命名空间: this_is_my_awesome_file.cpp ·[命名空间] 命名空间 <命名空间>_test.h(.cpp) 命名空间 ·[命名空间] 命名空间 .h 命名空间 .hpp 命名空间 ·[命名空间] 命名空间 lib命名空间 <命名空间>_<命名空间>.h 命名空间 api 命名空间 include 命名空间 api 命名空间 mylib.h, mylib_utils.h, mylib_api.h

命名空间 命名空间

3.3.3. 命名空间

·[命名空间] [RULE102] 命名空间 ·[命名空间] [RULE103] 命名空间 ·[命名空间] 命名空间

命名空间 using namespace 命名空间 命名空间

3.3.4. 命名空间

·[命名空间] [RULE103] 命名空间 ·[命名空间] 命名空间 ·[命名空间] 命名空间 accessor 命名空间 : getter: my_member_variable(), setter: set_my_member_variable(), mutable: mutable_my_member_variable()

命名空间 accessor 命名空间 Google protobuf 命名空间

命名空间

```
class Person {
public:
    ...
    const IdCard& id_card() const { return _id_card; } // getter
    IdCard* mutable_id_card() { return &_amp_id_card; }
```

```
void set_id_card(const IdCard& id_card) { _id_card = id_card; } // setter

private:
    IdCard _id_card;
    int _age;
    int get_age() const { return _age; }
};
```

3.3.5.

--	--	--	--	--	--	--	--	--

·[] [RULE105] []

[illegible]

Interface ☐ ☐ ☐ ☐ ☐ ☐

·[] [RULE106]

$$\cdot \left[\begin{array}{|c|} \hline \square \\ \hline \end{array} \right] \begin{array}{|c|c|c|c|c|c|} \hline \square & \square & \square & \square & \square & \square \\ \hline \end{array}$$

enum struct class typedef ☐ ☐ ☐ ☐ ☐

template 

--	--

```
enum Color {
    RED,
    BLUE,
    GREEN,
    COLOR_COUNT
};

struct Point {
    int x;
    int y;
};
```

3.3.6.

--	--	--	--	--	--

```

.[[  ] [RULE107] [

```

·[] [RULE108] []

[illegible]3.3.7.

--	--	--	--	--	--

```
.[[] ] [RULE109] [] [] [] [] [] [] [] [] [] []
```

[illegible]

--	--

[illegible]

☐ i,j,k ☐

[illegible]

• []

 i, j, k

```
cur_user_index
```

3.3.8.

```
·[ ] [RULE110]  s_  ·[ ] [RULE111]  _s_
;  _s_
static,  ;  _s_
```

3.3.9. /

```
·[ ] [RULE112]  ,
MyClass::_my_attr ·[ ] [RULE113]
,  MyStruct::my_attr
```

3.3.10.

```
·[ ] const
```

3.3.11. /

```
·[ ]
MY_LIB_MY_MACRO_THAT_SCARES_SMALL_CHILDREN
```

3.3.12.

```
·[ ]  int  ·[ ]  <stdint.h>
int64_t, int32_t
int
unsigned long long
```

3.4.

3.4.1.

```
·[ ] : , UTF-8  ·[ ]
·[ ]
·[ ]
·[ ]
·[ ]
·[ ]
```

3.4.2.

```

.[  ] [ ]
[ ]
[ ]
[ ]
[ ]
[ ] ; [ ]
[ ]
[ ]
[ ]
[ ]

```

```

    .[[] ] [] TODO
; .[[] ] TODO [] : [] [] ,
* [] .[[] ] [] DEPRECATED
DEPRECATED [] .[[] ]
.[[] ]
.[[] ]

```

3.4.3.

·[] , : 1) ; 2) () 3) 4)

/ []

·[]

·[]

3.5.

--	--	--	--

3.5.1.

```
·[ ] [RULE122] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
```

3.5.2.

--	--	--	--	--	--

. []

• []

[RULE123] []

--	--

3.6. `main`

```

·[ ] [RULE124] { } auto_ptr ·[ ] [RULE125] { }
strcpy()/strcat()/strdup()/sprintf() { } { }
snprintf()/strncat()/strndup()/snprintf() { } ·[ ] [RULE126] { } strcpy
{ } { } ullib { } ul_strncpy { }

{ } auto_ptr { } { } auto_ptr
{ } { } strcpy() { } \0
{ }
{ } snprintf { }

```

4. `main`

4.1. `main` / `main`

```

·[ ] { } , { } ·[ ]
{ } , { }

{ } { } . { } , { }
-- gtest. { } , { } ,
{ } .

{ } :

```

```

class FooEnvironment : public testing::Environment {
public:
    virtual void SetUp() {
        std::cout << "Foo FooEnvironment SetUP" << std::endl;
    }
    virtual void TearDown() {
        std::cout << "Foo FooEnvironment TearDown" << std::endl;
    }
};

```

// Example:

```
// template
class FooTest : public testing::Test {
public:

    ...

    typedef std::list List;
    static T shared_;
    T value_;
};
```

4.2.

```
·[ ] 
;
;
```

4.3. Windows

Windows C++ C++

4.3.1.

```
·[ ] [WCPP001] [WCPP002] [WCPP003],,·[ ]
[WCPP004],m_,·[ ] C
·[ ] C++
m_xxxx g_xxx ·[ ] I
: , EatApple :
, eatApple
```

4.3.2.

4.3.2.1.

```
·[ ] [WCPP006] { } ·[ ]
[WCPP005]


```

```
if (i == j)
{
    //
```

```
//for do while switch case
i++;
}
```

·[] ()

, int x = 3 & (4<<32);

4.3.3. WINAPI

VS IDE

·[] WINAPI

DWORD HANDLE ·[] C++

include guard #pragma once

```
#ifndef ABC_H
#define ABC_H
#endif
```

4.4.

" " "

·[]

" " "

Revision #1

Created 2026-10-06 08:49:38 UTC by Admin

Updated 2026-10-06 08:50:13 UTC by Admin