□□□□

- □□□□□□□□
- C & C++ □□□□

□□□□□□□

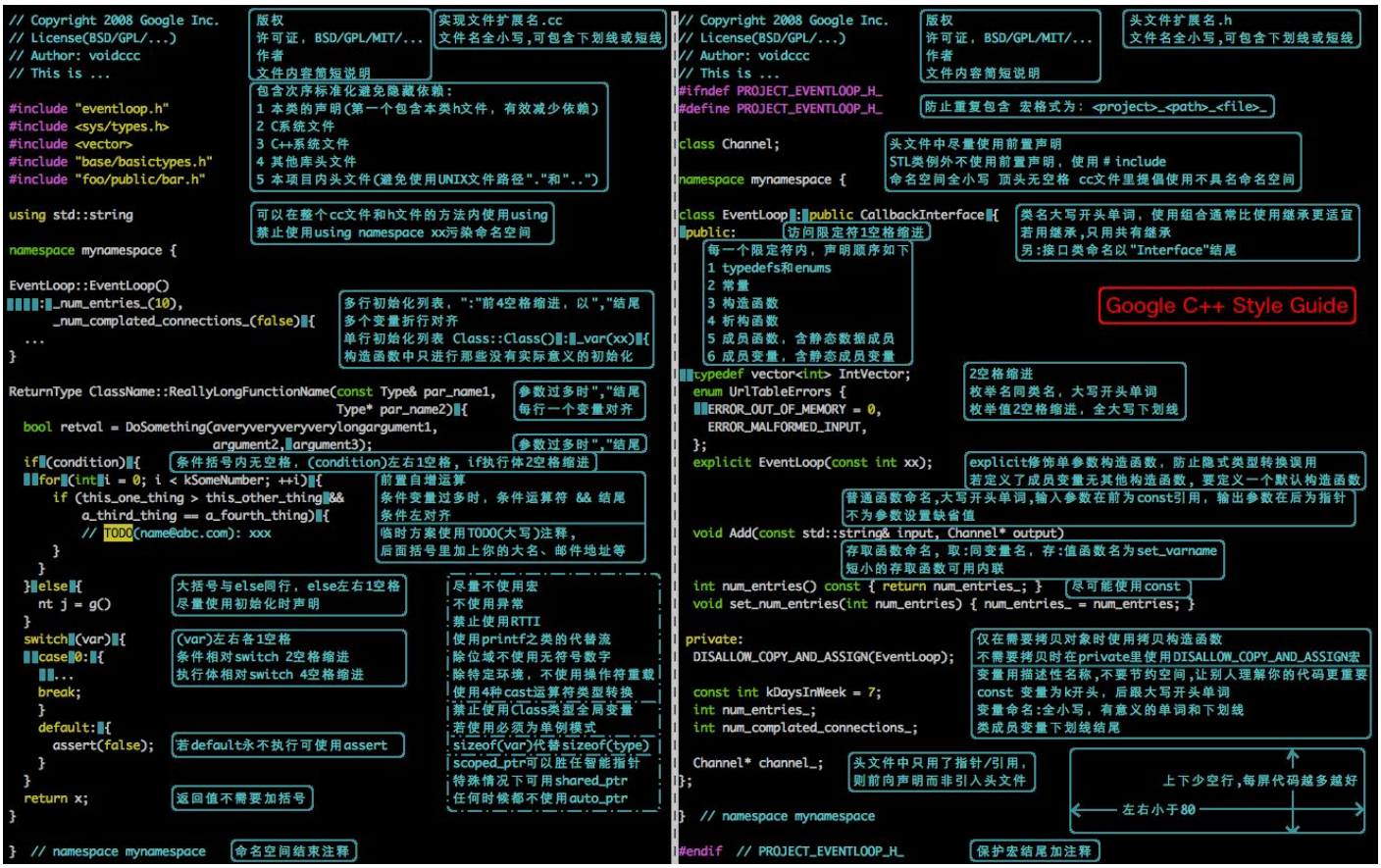
vscode □□□

```
□{
□// Place your snippets for cpp here. Each snippet is defined under a snippet name and has a prefix, body and
□// description. The prefix is what is used to trigger the snippet and the body will be expanded and inserted.
Possible variables are:
□// $1, $2 for tab stops, $0 for the final cursor position, and ${1:label}, ${2:another} for placeholders.
Placeholders with the
□// same ids are connected.
□// Example:
□"copyright info":{
        "prefix": "//file",
        "body": [
            "//-----------------------------------------------------------------------------",
            "//! Copyright(C) 2020-2022, netflt.com",
            "//! All rights reserved.",
            "// ",
            "// name: $TM_FILENAME",
            "// author: Danny.Zhu"
            "// date: $CURRENT_YEAR-$CURRENT_MONTH-$CURRENT_DATE
$CURRENT_HOUR:$CURRENT_MINUTE:$CURRENT_SECOND",
            "//-----------------------------------------------------------------------------"
        ],
        "description": "add copyright information"
    },
    "simple comment":{
        "prefix": "//",
        "body": [
            "//<! ${1}"
        ],
        "description": "add simple comment"
    },
```

```
    "block comment":{
        "prefix": "///",
        "body": [
            "/***",
            "//<! ${1}",
            "***/"
        ],
        "description": "add block comments"
    },
□"TODO": {
□□"prefix": "//td",
□□"body": [
□□□"//<! TODO (Danny.Zhu): $1"
□□],
□□"description": "Log output to console"
□}

}
```

clang-format □□



.clang-format

```
---
Language:        Cpp
# BasedOnStyle:  Chromium
SortIncludes:    true
AccessModifierOffset: -4
DerivePointerAlignment: false
PointerAlignment: Left
ConstructorInitializerIndentWidth: 4
AlignEscapedNewlines: Left
AlignAfterOpenBracket: Align
AlignConsecutiveMacros: Consecutive
AlignConsecutiveAssignments: None
AlignConsecutiveBitFields: Consecutive
AlignConsecutiveDeclarations: false
AlignOperands:   Align
AlignTrailingComments: true
AllowShortBlocksOnASingleLine: Never
AllowShortFunctionsOnASingleLine: Empty
AllowAllArgumentsOnNextLine: true
```

```
AllowAllConstructorInitializersOnNextLine: false
AllowAllParametersOfDeclarationOnNextLine: false
AllowShortEnumsOnASingleLine: false
AllowShortLambdasOnASingleLine: Empty
AllowShortIfStatementsOnASingleLine: Never
AllowShortCaseLabelsOnASingleLine: false
AllowShortLoopsOnASingleLine: false
AlwaysBreakTemplateDeclarations: true
BinPackArguments: false
AlwaysBreakBeforeMultilineStrings: false
BreakBeforeBinaryOperators: NonAssignment
BreakBeforeTernaryOperators: true
BreakConstructorInitializersBeforeComma: false
BinPackParameters: false
ColumnLimit:     96
ConstructorInitializerAllOnOneLineOrOnePerLine: false
DerivePointerBinding: false
ExperimentalAutoDetectBinPacking: false
IndentCaseLabels: false
MaxEmptyLinesToKeep: 1
NamespaceIndentation: None
ObjCSpaceBeforeProtocolList: true
PenaltyBreakBeforeFirstCallParameter: 1
PenaltyBreakComment: 300
PenaltyBreakString: 1000
PenaltyBreakFirstLessLess: 120
PenaltyExcessCharacter: 1000000
PenaltyReturnTypeOnItsOwnLine: 1000
AlwaysBreakAfterDefinitionReturnType: None
AlwaysBreakAfterReturnType: None
PointerBindsToType: true
Cpp11BracedListStyle: false
Standard:        Auto
IndentWidth:     4
TabWidth:        4
UseTab:          Never
BreakBeforeBraces: Custom
AttributeMacros:
  - __capability
```

```
BraceWrapping:
  AfterCaseLabel:  false
  AfterClass:      true
  AfterControlStatement: false
  AfterEnum:       false
  AfterFunction:   true
  AfterNamespace:  true
  AfterObjCDeclaration: true
  AfterStruct:     false
  AfterUnion:      false
  AfterExternBlock: true
  BeforeCatch:     true
  BeforeElse:      true
  BeforeLambdaBody: false
  BeforeWhile:     false
  IndentBraces:    false
  SplitEmptyFunction: false
  SplitEmptyRecord: false
  SplitEmptyNamespace: false
SpacesInParentheses: false
SpacesInAngles:  false
BreakBeforeConceptDeclarations: true
BreakBeforeInheritanceComma: false
BreakInheritanceList: BeforeComma
BreakConstructorInitializers: BeforeComma
BreakAfterJavaFieldAnnotations: false
BreakStringLiterals: true
CommentPragmas:  '^ IWYU pragma:'
CompactNamespaces: false
ContinuationIndentWidth: 4
DeriveLineEnding: true
DisableFormat:   false
EmptyLineBeforeAccessModifier: LogicalBlock
FixNamespaceComments: true
ForEachMacros:
  - foreach
  - Q_FOREACH
  - BOOST_FOREACH
StatementAttributeLikeMacros:
```

```
  - Q_EMIT
IncludeBlocks:   Regroup
IncludeCategories:
  - Regex:           '^<ext/.*\.h>'
    Priority:        2
    SortPriority:    0
    CaseSensitive:   false
  - Regex:           '^<.*\.h>'
    Priority:        1
    SortPriority:    0
    CaseSensitive:   false
  - Regex:           '^<.*'
    Priority:        2
    SortPriority:    0
    CaseSensitive:   false
  - Regex:           '.*'
    Priority:        3
    SortPriority:    0
    CaseSensitive:   false
IncludeIsMainRegex: '([-_](test|unittest))?$'
IncludeIsMainSourceRegex: ''
IndentCaseBlocks: false
IndentGotoLabels: true
IndentPPDirectives: BeforeHash
IndentExternBlock: AfterExternBlock
IndentRequires:  false
IndentWrappedFunctionNames: true
InsertTrailingCommas: None
JavaScriptQuotes: Leave
JavaScriptWrapImports: true
KeepEmptyLinesAtTheStartOfBlocks: true
MacroBlockBegin: ''
MacroBlockEnd:   ''
ObjCBinPackProtocolList: Auto
ObjCBlockIndentWidth: 2
ObjCBreakBeforeNestedBlockParam: true
ObjCSpaceAfterProperty: false
PenaltyBreakAssignment: 2
PenaltyBreakTemplateDeclaration: 10
```

```
PenaltyIndentedWhitespace: 0
RawStringFormats:
  - Language:        Cpp
    Delimiters:
      - cc
      - CC
      - cpp
      - Cpp
      - CPP
      - 'c++'
      - 'C++'
    CanonicalDelimiter: ''
    BasedOnStyle:     google
  - Language:        TextProto
    Delimiters:
      - pb
      - PB
      - proto
      - PROTO
    EnclosingFunctions:
      - EqualsProto
      - EquivToProto
      - PARSE_PARTIAL_TEXT_PROTO
      - PARSE_TEST_PROTO
      - PARSE_TEXT_PROTO
      - ParseTextOrDie
      - ParseTextProtoOrDie
      - ParseTestProto
      - ParsePartialTestProto
    CanonicalDelimiter: ''
    BasedOnStyle:     google
ReflowComments:  true
SortJavaStaticImport: Before
SortUsingDeclarations: true
SpaceAfterCStyleCast: false
SpaceAfterLogicalNot: false
SpaceAfterTemplateKeyword: true
SpaceBeforeAssignmentOperators: true
SpaceBeforeCaseColon: false
```

```
SpaceBeforeCpp11BracedList: true
SpaceBeforeCtorInitializerColon: true
SpaceBeforeInheritanceColon: true
SpaceBeforeParens: ControlStatements
SpaceAroundPointerQualifiers: Default
SpaceBeforeRangeBasedForLoopColon: true
SpaceInEmptyBlock: false
SpaceInEmptyParentheses: false
SpacesBeforeTrailingComments: 1
SpacesInConditionalStatement: false
SpacesInContainerLiterals: false
SpacesInCStyleCastParentheses: false
SpacesInSquareBrackets: false
SpaceBeforeSquareBrackets: false
BitFieldColonSpacing: Both
StatementMacros:
  - Q_UNUSED
  - QT_REQUIRE_VERSION
UseCRLF:         false
WhitespaceSensitiveMacros:
  - STRINGIZE
  - PP_STRINGIZE
  - BOOST_PP_STRINGIZE
  - NS_SWIFT_NAME
  - CF_SWIFT_NAME
...
```

clang-tidy

.clang-tidy

```
---
# Configure clang-tidy for this project.

# Here is an explanation for why some of the checks are disabled:
#
#  -google-readability-namespace-comments: the *_CLIENT_NS is a macro, and
#      clang-tidy fails to match it against the initial value.
#
#  -modernize-use-trailing-return-type: clang-tidy recommends using
#      `auto Foo() -> std::string { return ...; }`, we think the code is less
#      readable in this form.
#
#  -modernize-return-braced-init-list: We think removing typenames and using
#      only braced-init can hurt readability.
#
#  -modernize-avoid-c-arrays: We only use C arrays when they seem to be the
#      right tool for the job, such as `char foo[] = "hello"`. In these cases,
#      avoiding C arrays often makes the code less readable, and std::array is
#      not a drop-in replacement because it doesn't deduce the size.
#
#  -performance-move-const-arg: This warning requires the developer to
#      know/care more about the implementation details of types/functions than
#      should be necessary. For example, `A a; F(std::move(a));` will trigger a
#      warning IFF `A` is a trivial type (and therefore the move is
#      meaningless). It would also warn if `F` accepts by `const&`, which is
#      another detail that the caller need not care about.
#
#  -readability-redundant-declaration: A friend declaration inside a class
#      counts as a declaration, so if we also declare that friend outside the
#      class in order to document it as part of the public API, that will
#      trigger a redundant declaration warning from this check.
#
#  -readability-function-cognitive-complexity: too many false positives with
#      clang-tidy-12. We need to disable this check in macros, and that setting
#      only appears in clang-tidy-13.
#
#  -bugprone-narrowing-conversions: too many false positives around
#      `std::size_t`  vs. `*::difference_type`.
#
#  -bugprone-easily-swappable-parameters: too many false positives.
```

```
#
#  -bugprone-implicit-widening-of-multiplication-result: too many false positives.
#     Almost any expression of the form `2 * variable` or `long x = a_int * b_int;`
#     generates an error.
#
#  -bugprone-unchecked-optional-access: too many false positives in tests.
#     Despite what the documentation says, this warning appears after
#     `ASSERT_TRUE(variable)` or `ASSERT_TRUE(variable.has_value())`.
#
Checks: >
  -*,
  abseil-*,
  bugprone-*,
  google-*,
  misc-*,
  modernize-*,
  performance-*,
  portability-*,
  readability-*,
  -google-readability-braces-around-statements,
  -google-readability-namespace-comments,
  -google-runtime-references,
  -misc-non-private-member-variables-in-classes,
  -misc-const-correctness,
  -misc-use-anonymous-namespace,
  -modernize-return-braced-init-list,
  -modernize-use-trailing-return-type,
  -modernize-use-nodiscard,
  -modernize-avoid-c-arrays,
  -performance-move-const-arg,
  -readability-braces-around-statements,
  -readability-identifier-length,
  -readability-magic-numbers,
  -readability-named-parameter,
  -readability-redundant-declaration,
  -readability-function-cognitive-complexity,
  -readability-convert-member-functions-to-static,
  -readability-qualified-auto,
  -readability-implicit-bool-conversion,
  -bugprone-narrowing-conversions,
```

```
  -bugprone-easily-swappable-parameters,
  -bugprone-implicit-widening-of-multiplication-result,
  -bugprone-unchecked-optional-access,
  -bugprone-branch-clone

# Turn all the warnings from the checks above into errors.
WarningsAsErrors: "*"

CheckOptions:
  - { key: readability-identifier-naming.ClassCase,              value: CamelCase  }
  - { key: readability-identifier-naming.StructCase,             value: CamelCase  }
  - { key: readability-identifier-naming.TemplateParameterCase,  value: CamelCase  }
  - { key: readability-identifier-naming.FunctionCase,           value: camel_back  }
  - { key: readability-identifier-naming.PrivateMemberPrefix,    value: _          }
  - { key: readability-identifier-naming.ProtectedMemberPrefix,  value: _          }
  - { key: readability-identifier-naming.EnumConstantCase,         value: CamelCase }
  - { key: readability-identifier-naming.EnumConstantPrefix,       value: k         }
  - { key: readability-identifier-naming.ConstexprVariableCase,    value: CamelCase }
  - { key: readability-identifier-naming.ConstexprVariablePrefix,  value: k         }
  - { key: readability-identifier-naming.GlobalConstantCase,       value: CamelCase }
  - { key: readability-identifier-naming.GlobalConstantPrefix,     value: k         }
  - { key: readability-identifier-naming.MemberConstantCase,       value: CamelCase }
  - { key: readability-identifier-naming.MemberConstantPrefix,     value: k         }
  - { key: readability-identifier-naming.StaticConstantCase,       value: CamelCase }
  - { key: readability-identifier-naming.StaticConstantPrefix,     value: k         }
  - { key: readability-implicit-bool-conversion.AllowIntegerConditions,  value: 1   }
  - { key: readability-implicit-bool-conversion.AllowPointerConditions,  value: 1   }
  - { key: readability-function-cognitive-complexity.IgnoreMacros,  value: 1   }
```

C & C++ □□□□

1. □□

□□□□□□□□ Google C++ Style Guide□□□□ C□ C++ □□□□□□□□□

2.1.2. □□□□□□

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

2.1.3. □□□□□□

·[□□] □□□□□□

```
};
```

2.1.4. □□□□□□

·[□□] □□□□□□

□□
□□□□□□
copy constructor[`MyClass::MyClass(const MyClass&)` □□□□□
□□□□□□□□□□/□□□□□□□□□□□□□□□□□□□□□□□□□□

□□
□□□□□□□□□□□□□□□□□□□

> behavior in resource-managing classes

2.1.6. □□□□□□□

·[□□] □□□□□□□

□□

> 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

2.1.7. □□□□

·[□□] □□□□

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

    // □□□□□□□□□□□□□□□□
    virtual ~Chinese() {}

    // □□□□□□□□□□□□□□□
    void say(const std::string& word);

    // □□□□□□□□□
    virtual void greet() { say("□□□□"); }

private:
    std::string _name;
};

// □□□□□"□□□"□□□

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

    // □□□□□□□□□□□□□□□□
    virtual void greet() { say("□□□□?"); }
};
```

□□

> 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□□□□□□□□□

2.1.9. □□□□

·[□□] □□□□□□□□□□□□□□□□□□□□

□□ □□□□

```
    ...                                         // attach□□□□□□□□□□□□□□□□

    template <typename T>
    T& create();                        // □□□□□□□□
    ...                                 // create□□□□□□□□□□□□□□□□
    ...                                 // □□□□□□□□□□□□□create_raw(),clone_cstring(),reset()□
private:
    // private□□□□□□□□□□□□□□□□□□□□□□□□
    // □□□□
    struct ObjectInfo;                  // □□□□□□□□□□□□
    ...
    // □□□□
    void push_info();                   // □□□□□□□push_info()□pop_info()□□□□□□□□□□□□□□□
    void pop_info();                    // □□□□□□private□□□□□□□□□□□□□□
    ...
    // □□□□□□□
    ObjectInfo *_p_object_info_list;
    ...
    DISALLOW_COPY_AND_ASSIGN(ResourcePool);  // □□□□□□
};
```

2.1.12. □□ ·[□□] □□□□□□□□□□□□□□□□□□

□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□

□□

> Effective C++, Item 23, Prefer non-member non-friend functions to member functions

2.1.13. □□□□

·[□□] `catch` □□□□□□□□□□□□/□□□□□□□□□□ □□□□□□

□□

□□: □□□□□□□□□□□□□□□□□□□□□□□□ “□□□□□□” □□, □□□□□□□□□ `initialize()` □□□□□□□□□□□□□□□□□□□□

```
}  // namespace saick
```

□□□□□□□□BadCase□□□□□□□:

```
// a.h
#include <string>
using namespace std;  // □□□□□□□□□using namespace

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

// a.cpp
#include “bsl/bsl_string.h”
#include “a.h”

using namespace bsl;  // □□□.cpp□□□□□using namespace

int main() {
    string str; // □□□□□□□□□□□□bsl::string□□std::string

    A a;
    cout << a.name() << endl;  // □□□□□std□□□□□□□□□
}
```

□□□□□

```
// a.h
#include <string>

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

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

```
using std::cout;  // □□□using class□□□□□
using std::endl;

int main() {
    bsl::string str;  // □□□□□

    A a;
    cout << a.name() << endl;
}
```

2.2.2. □□□□

·[□□] [RULE025] □□□□□□□ ·[□□] □□□□□□□□□□□□□□□□(□□10□□□)

□□ □□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□

·[□□] □□□□

·[□□] □□□□□□□□□□□ ·[□□] □□□□C++□□□□□□□

□□ C++ □□□□□`static_cast` `dynamic_cast` `const_cast` `reinterpret_cast` □□□

□□ □□□□□`_cast` □□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□C++ □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□`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);   // □□const□□□□□□□□□□const□□□□□□□□□□□□□□□□□□□□
}
```

□□

> 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. □□□□□

·[□□] □□□□□□□□□□□□□□□□□□□□□□□□□□□□ ·[□□] □□□□□□□□□□□□□ ·[□□] □□□□□□□□□□□□□□□□□□□□□□□□□==□□□□□□□ ·[□□] [RULE033] □□□□□□□

□□ [`if`]□□ `==` □□ `=` □□□□ `==` □□ `!=` □□□□□□□□□□□□□□□□□□□□

```
void* p = NULL;
char ch = '\0';
```

2.3.3. sizeof □□□

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

□□

```
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 □□□□□□□□□□□□□□□

□□ □□ typedef □□□□□□□□□□□□□□□□□□□□ typedef□

□□ YES:

```
class MyContainer {
public:
    typedef char value_type;    // type tag

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

NO:

```
typedef tagMyStruct {
    ...
```

```
} MyStruct; // □C++□□□□□□□□□□C□□□□□□□
```

2.3.5. goto

·[□□] [[RULE037]] □□ goto □□□

□□ goto □□□□□□□□□□□□□□□□

2.3.6. □

·[□ `DISALLOW_COPY_AND_ASSIGN` ,□ `CFATAL_LOG` etc.)□□□□□□□. ·[□□] □□□□□□□□ c `##` w `#` e □□□ `break` ,□ `continue` ,□ `return` □□ ·[□□] □□□□□□□/□□□□□□□/□□□□□□□□□□ ·[□□] □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ □□□□□□□□□□□□□□□□□□□□

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

2.3.7. [illegible]

·[illegible] [illegible]

[illegible] [illegible]

[illegible]

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

[illegible]

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

2.3.8. [illegible]/[illegible]/[illegible]

·[illegible] [illegible]single `main()` [illegible] ·[illegible] [illegible] class [illegible] ·[illegible] [illegible] ·[illegible] [illegible]

[illegible] [illegible], [illegible] C++ [illegible], [illegible], [

2.3.9. [illegible]

·[illegible] [RULE050] [illegible]/[illegible] ·[illegible] [RULE051] [illegible] enum [illegible] const [illegible] define ·[illegible] [illegible]

[illegible] define [illegible] `strlen("filename")` [illegible] `file_name` [illegible] [illegible]

[illegible]

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

[illegible]const[illegible]

```
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 □□□□□□□□

□□ □□□□□□ const □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ bug const_cast □□□□□□□□□□□C□□□□□C□□□□□□□□□□□□□□□□□□□□

```
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 □□□□□□□□□□□□□□□□□□□□□□

3. □□/□□

3.1. □□□□□

3.1.1. □□□□□□□

·[□□] □□□□□□□□□□□□□.cpp□□□□□□(□□□), C(□□)□□C++(□□)□□□□□□□□□□□.h□□

□□ □□□□□□□□□□□□ □□cpp□□□□□□□□□□□□□□□□□□□□ □□□□□□ □□□□□□□□□□; □□□ □□□□□□□□□□ □□□□□□□□□□;

□□ □□□□, some-project/foo/internal/fooserver.cc □□□□□□□□:

```
#include "foo/public/fooserver.h" // □□□□
#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<>□#include”“

·[□□] □□□□□#include <>□□□□□□#include ""

3.1.3. #include□□□□□□□□□□□□□□□□

·[□□] □□include□□□□□□□□□□□□□□□□□□

[`#include` □□□□□□□□□□□□□□□□□□□□

```
#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)
//
// <□□□□□□□□□□□□□□□□□□□□>
```

3.2.2. □□□□□

·[□□] [RULE064] □□□□□□□□□□□□□□ ·[□□] □□□□□□□□□□□□□□□□□□ ·[□□] [RULE066] □□□□□□□□□ ·[□□] □□□□□□□□□□□□□□□□□□□□

```
// mylib/AwesomeClass.h
namespace mylib {
class AwesomeClass {
    void foo() { //□□□□
        ...
    }
    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 □□□□□□□□□□□□□

```
};

}  // 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. □□□□□□□□

·[□□] □□□□□□□□□□□□□□□□□□□□□□□□□□□□, □□□□□□, □: □□□□□. □□ □□□□□□□□□□□□□□4□□□□□□□□□□□□□□□□□□□□□□□□□□□□8□□□ □□

```
// □□□□□□□□□
MyClass::MyClass(int var) : _some_var(var), _some_other_var(var + 1) {}

// □□□□□□□□ □□□□8□□□□ □□□□□□□□□□
// 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. □□□□

·[□□] □□□□

```
    return "you come from solar system";
}
```

□□

> 1TBS□□

3.2.3.8. switch□□

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

□ `case` □ `break` □ `return` □ `case` □□□□□□□□ □ `default` □□□□□□□□□□□

□□ □□□

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

□□□□□□□

```
for (size_t i = 0; i < N; ++i) {
    sum += arr[i];
}
// do-while
do {
    ...
} while (++it != end);  // □□
do {
    ...
}
while (++it != end);    // □□□□□□□□□□□□□□□□□□□□□□□□
```

□□

> 1TBS□□

3.2.4. □□□□□

·[□□] [RULE085] if/switch/while/for/catch □□□□□□□□□□□□□□□□ for □□□□□□□□□□□□□□□□□□□□

```
for (int i = 0 ; i < 10 ; ++i)    // □□, □□□□□□

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□□□□□□□□□□□□□□□□□□□

```
UB_LOG_WARNING(
    "An exception[%s] was thrown from[%s:%d:%s] "     // □□□□
```

```
// □□□□□□□const□□□□□□□□□□□□□□□□
void foo(const MyClass* input1);

// □□□□□□□□□□□□□□□□□□□□□
void foo(int* output1, MyClass* output2);

// □□□□□□□□□□□□□□□□□std::string*□□□□□char*□□□□□char*□□□□□□□buffer□□□□□□□□□□□□□
void foo(std::string* out);

// □□□□□□□□□□□□□
void foo(const InClass& input1, OutClass* out1);

// □□□□□□□□□
void foo(const InClass& input1,  // □□□□const□□
        int input2,  // □□□□POD□□□□
        OutClass* outpu1);  // □□□□□□□□

// □□□□□□□□□□□□□□□□□□□□
```

3.3.1. [illegible]

·[[illegible]] `creat` [, `usr` [illegible]) [illegible] ·[[illegible]] [illegible] ·[[illegible]] [illegible]

[illegible] `VScriptEvaluator` [illegible] `vse` [illegible] `_evaluator` [illegible] `extern “C”` [illegible]

[illegible]

```
// [illegible]
// [illegible]
int num_errors;                  // Good.
int num_completed_connections;   // Good.

// [illegible]
int n;              // Bad - [illegible]
int nerr;           // Bad - [illegible]
int n_comp_conns;   // Bad - ambiguous abbreviation.

// [illegible]:
// [illegible]:
int num_dns_connections;  // [illegible]"DNS"[illegible]
int price_count_reader;   // OK, price count. Makes sense.

// [illegible]
int wgc_connections;      // wgc [illegible]??
int pc_reader;            // pc[illegible]!

// [illegible]
int error_count;          // Good!
int error_cnt;            // Bad! [illegible], [illegible];
```

3.3.2. [illegible]

·[[illegible]] [illegible] .h [illegible] .hpp [illegible]C++ [illegible] .cpp [illegible] C [illegible] .c[illegible] ·[[illegible]] [RULE100] [illegible] `<[illegible]>_test.h(.cpp)` [illegible] `<[illegible]>_<[illegible]>.h` [illegible] `mylib.h,mylib_utils.h,mylib_api.h` [illegible] lib[illegible]

[illegible]

3.3.3. [illegible]

·[[illegible]] [RULE102] [illegible] ·[[illegible]] [RULE103] [illegible] ·[[illegible]] [illegible]

□□ □□□□

3.3.6. □□□□□□

·[□□] [RULE107] □□□□□□□□□□□□□□□□□□□□□g□□□□□□□□□□□□□□□□□□ ·[□□] [RULE108] □□□□□□□□□□□□□□□□□□□□□□□□

3.3.7. □□□□□□

·[□□] [RULE109] □□□□□□□□□□□□□□□□□□□□□□ ·[□□] □□□□□□□□□□□□□□□□□□□□□□

□□ □□□□□□□ `cur_user_index` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□i,j,k □□□□□□□□□□□□□□□□□□□□

·[□□] □□ `TODO` □□□□□ `TODO` □□□□□□□□□□□ `DEPRECATED` □□□□ `DEPRECATED` □*□□□□·[□□□□□□□□□□

□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□. □□, □□□□□□□□□□□□□□□□ -- 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. □□□□□□□□

·[□□□] □□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□

4.3.2.1. □□

·[□□] [WCPP006 `{` `}` □□□□