

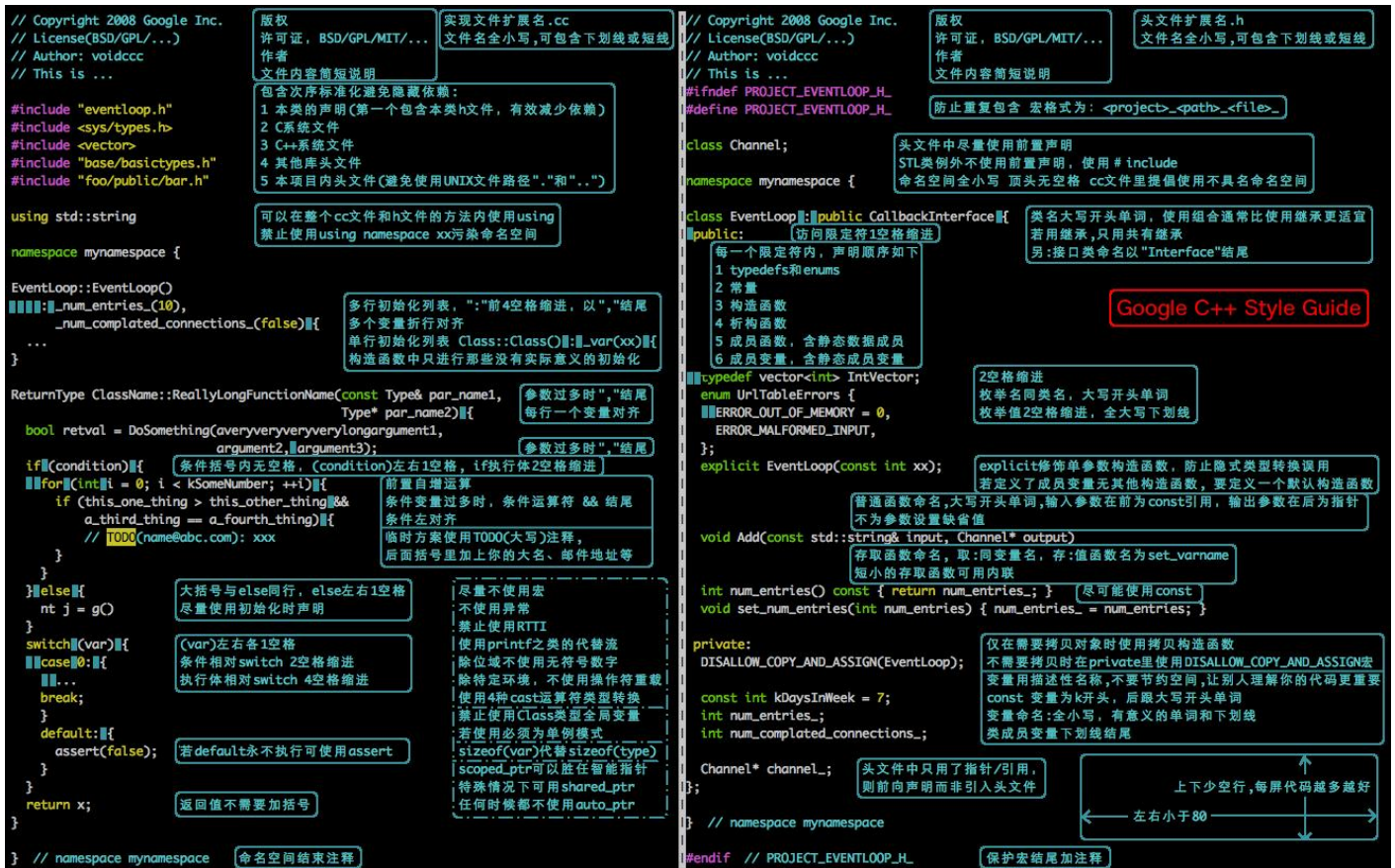


# vscode

```
{
// 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

IncludesMainRegex: '([-\_](test|unittest))?\$'

IncludesMainSourceRegex: ''

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 typenamees 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 }
```

Revision #3

Created 9 August 2024 13:41:20 by Danny

Updated 18 August 2024 12:42:49 by Danny