Coding rules for C++ programming language

J. Daniel Garcia (coordinator)
Computer Architecture
Computer Science and Engineering Department
University Carlos III of Madrid

2023

1 General code format

As a general rule, code must be well structured and organized, as well as properly documented.

To guarantee code readability and an homogeneous format, code will be formatted with **clang-format**. A common configuration will be used as the one that you will find below:

```
AccessModifierOffset: -2
AlignAfterOpenBracket: Align
AlignArrayOfStructures: Right
AlignConsecutiveAssignments:
  Enabled: true
  AcrossEmptyLines: false
  AcrossComments: false
  AlignCompound: true
  PadOperators: true
AlignConsecutiveBitFields:
  Enabled: true
  AcrossEmptyLines: false
  AcrossComments: false
  AlignCompound: true
  PadOperators: true
AlignConsecutiveDeclarations: None
AlignConsecutiveMacros: None
AlignEscapedNewlines: Left
AlignOperands: Align
AlignTrailingComments: Always
AllowAllArgumentsOnNextLine: true
AllowAllConstructorInitializersOnNextLine: true
AllowAllParametersOfDeclarationOnNextLine: false
AllowShortBlocksOnASingleLine: Always
AllowShortCaseLabelsOnASingleLine: false
AllowShortEnumsOnASingleLine: true
AllowShortFunctionsOnASingleLine: Inline
AllowShortIfStatementsOnASingleLine: AllIfsAndElse
AllowShortLambdasOnASingleLine: None
AllowShortLoopsOnASingleLine: true
AlwaysBreakAfterDefinitionReturnType: None
AlwaysBreakAfterReturnType: None
AlwaysBreakBeforeMultilineStrings: false
AlwaysBreakTemplateDeclarations: Yes
BinPackArguments: true
BinPackParameters: true
BitFieldColonSpacing: Both
BraceWrapping:
  AfterCaseLabel: false
  AfterClass: false
  AfterControlStatement: MultiLine
  AfterEnum: false
```

```
    AfterFunction: false
    AfterNamespace: false
    AfterStruct: false
    AfterUnion: false
    AfterExternBlock: false
    BeforeCatch: true
    BeforeElse: true
    BeforeLambdaBody: false
    BeforeWhile: false
    IndentBraces: false
    SplitEmptyFunction: false
    SplitEmptyRecord: false
    SplitEmptyNamespace: false
  BracedInitializerIndentWidth: 2
  BreakAfterAttributes: Never
  BreakBeforeBinaryOperators: None
  BreakBeforeBraces: Attach
  BreakBeforeConceptDeclarations: Always
  BreakBeforeInlineASMColon: Always
  BreakBeforeTernaryOperators: true
  BreakConstructorInitializers: BeforeColon
  BreakInheritanceList: AfterComma
  BreakStringLiterals: true
  ColumnLimit: 100
  CompactNamespaces: false
  ConstructorInitializerIndentWidth: 2
  ContinuationIndentWidth: 4
  Cpp11BracedListStyle: true
  EmptyLineAfterAccessModifier: Never
  EmptyLineBeforeAccessModifier: Always
  FixNamespaceComments: true
  IncludeBlocks: Regroup
  IndentAccessModifiers: true
  IndentCaseBlocks: true
  IndentCaseLabels: true
  IndentExternBlock: Indent
  IndentGotoLabels: true
  IndentPPDirectives: BeforeHash
  IndentRequiresClause: true
  IndentWidth: 2
  IndentWrappedFunctionNames: true
  InsertBraces: true
  InsertNewlineAtEOF: false
  KeepEmptyLinesAtTheStartOfBlocks: false
  LambdaBodyIndentation: Signature
  Language: Cpp
  MaxEmptyLinesToKeep: 1
  NamespaceIndentation: All
  PPIndentWidth: 2
  PackConstructorInitializers: BinPack
  PointerAlignment: Middle
  QualifierAlignment: Right
  ReferenceAlignment: Middle
  ReflowComments: true
  RequiresClausePosition: OwnLine
  RequiresExpressionIndentation: OuterScope
  SeparateDefinitionBlocks: Always
  ShortNamespaceLines: 0
  SortIncludes: CaseInsensitive
  SortUsingDeclarations: Never
  SpaceAfterCStyleCast: true
  SpaceAfterLogicalNot: false
  SpaceAfterTemplateKeyword: true
  SpaceAroundPointerQualifiers: Both
  SpaceBeforeAssignmentOperators: true
  SpaceBeforeCaseColon: false
  SpaceBeforeCpp11BracedList: false
  SpaceBeforeCtorInitializerColon: true
  SpaceBeforeInheritanceColon: true
```

```
SpaceBeforeParens: ControlStatementsExceptControlMacros
SpaceBeforeParensOptions:
  AfterControlStatements: true
  AfterFunctionDeclarationName: false
  AfterFunctionDefinitionName: false
  AfterForeachMacros: false
  AfterIfMacros: false
  AfterOverloadedOperator: false
  BeforeNonEmptyParentheses: false
SpaceBeforeRangeBasedForLoopColon: true
SpaceBeforeSquareBrackets: false
SpaceInEmptyBlock: true
SpacesBeforeTrailingComments: 2
SpacesInAngles: Never
SpacesInLineCommentPrefix:
  Minimum: 1
  Maximum: 1
SpacesInParentheses: false
SpacesInSquareBrackets: false
Standard: c++20
UseTab: Never
```

To format code you can place the file **.clang-format** in the root directory of your project. Most development environment will recognize the file and will automatically format your code.

You can also consult other ways to apply the format in the tool documentation: `https://clang.llvm.org/docs/ClangFormat.html`.

2 Coding rules

As a general rule, you are recommended to follow **C++ Core Guidelines** (available at `http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines`). However, it is not mandatory to follow those rules.

The detailed description of the rules checked by **clang-tidy** may be found at `https://clang.llvm.org/extra/clang-tidy/checks/list.html`.

To automatically check the rules you may use a configuration as the following:

```
Checks: '-*,
bugprone-*,
cert-*,
cppcoreguidelines-*,
hicpp-*.
misc-*,
modernize-*,
performance-*,
readability-*,
-misc-include-cleaner,
-modernize-use-trailing-return-type,
-readability-redundant-access-specifiers'

CheckOptions:
  bugprone-easily-swappable-parameters.MinimumLength: 3
  bugprone-implicit-widening-of-multiplication-result.UseCXXStaticCastsInCppSources: true
  bugprone-implicit-widening-of-multiplication-result.UseCXXHeadersInCppSources: true
  bugprone-misplaced-widening-cast.CheckImplicitCasts: true
  cppcoreguidelines-pro-type-member-init.UseAssignment: false
  misc-non-private-member-variables-in-classes.IgnoreClassesWithAllMemberVariablesBeingPublic: true
  modernize-deprecated-headers.CheckHeaderFile: true
  readability-function-cognitive-complexity.Threshold: 50
  readability-function-cognitive-complexity.Threshold.DescribeBasicIncrements: true
  readability-function-size.LineThreshold: 25
  readability-function-size.ParameterThreshold: 4
  readability-identifier-length.MinimumParameterNameLength: 1

WarningsAsErrors: '*'
```

```
HeaderFileExtensions: [ '', 'hpp' ]

FormatStyle: file
```

To check the rules you may place the file **.clang-tidy** in the root directory of your project. Most development environments will recognize the file and will perform the checks.

You can also use other ways to apply the checks that can be found at `https://clang.llvm.org/extra/clang-tidy/index.html`.

Below you will find a set of rules that must be followed.

2.1 General rules

In general, the following principles must be considered:

- No global variables shall be used except constants.
- No array shall be passed to a function as a pointer parameter.
- No function or member function shall have more than 4 parameters.
- No function shall have more than 25 lines when properly formatted.
- All parameters shall be passed to functions by value, by reference or by const reference.
- Functions **malloc()** or **free** shall not be explicitly invoked.
- Operators **new** or **delete** shall not be explicitly invoked.
- No macros use is allowed, except for defining include guards.
- No cast shall be performed except **static_cast**, **dynamic_cast** or **const_cast**.
- **reinterpret_cast** shall be used only when passing parameters to functions form the standard library (e.g. **read()** or **write()**.

2.2 Rules to avoid common mistakes

All these rules are under category **cppcoreguidelines**.

For the rules in this category, the following options shall be considered:

- **bugprone-easily-swappable-parameters**: Functions taking 4 or more parameters from the same type shall be defined.
- **bugprone-implicit-widening-of-multiplication-result**: A **static_cast<>** shall be used to convert to a wider type. If needed file **<cstddef>** shall be included.
- **bugprone-misplaced-widening-cast**: When widening implicit conversions shall be considered.

2.3 Rules from CERT C++ Secure Coding

All these rules are under category **cert**.

2.4 Rules from C++ Core Guidelines

All these rules are under category **cppcoreguidelines**.

For the rules in this category, the following options shall be considered:

- **cppcoreguidelines-pro-type-member-init**: Literal initializer shall be used (**int i = 0**) instead of brace initializers (**int i{}**).

2.5 Rules from HICPP

All these rules are under category **hicpp**.

2.6 Misc rules

All these rules are under category **misc**.

EXCEPTION: The following rule shall not enabled:

- **misc-include-cleaner**.

For the rules in this category, the following options shall be considered:

- **misc-non-private-member-variables-in-classes**: Data members shall be private unless the type is a structure with all data members being public.

2.7 Rules for modernization

All these rules are under category **modernize**.

EXCEPTION: The following rule shall not enabled:

- **modernize-use-trailing-return-type**.

For the rules in this category, the following options shall be considered:

- **modernize-deprecated-headers**: Only C++ header files shall be used.

2.8 Rules oriented to performance

All these rules are under category **performance**.

2.9 Rules for readability

All these rules are under category **readability**.

EXCEPTION: The following rule shall not enabled:

- **readability-redundant-access-specifiers**.

For the rules in this category, the following options shall be considered:

- **readability-function-cognitive-complexity**: Maximum cognitive complexity will be **50**.
- **readability-function-size**: Maximum number of lines per function will be **25**.
- **readability-function-size**: Maximum number of parameters per function will be **4**.
- **readability-identifier-length**: Minimum identifier length for function parameters will be **1**.