ClangFormat で C/C++コードを整形する

TN0001-00／2023 年 06 月 26 日／こがねさん(著)
https://www.kumikomist.com/

■目次

■文書内の記号

	取り扱いにおける禁止事項（してはいけないこと）を示しています。
	取扱における指示事項（必ずしなければいけないこと）を示しています。

	取り扱いにおける注記事項を示しています。
	取り扱いにおけるポイントを示しています。

1. はじめに

ClangFormat（クラン・フォーマット）とは、C/C++をターゲットとした強力なコード整形ツールです。eclipse や Visual Studio Code などで使用することができます。このようなツールを使用することで、人や環境が変わっても同じスタイルでコードを記述することができます。

本書では、Visual Studio Code での使用例を紹介します。

2. 必要なもの

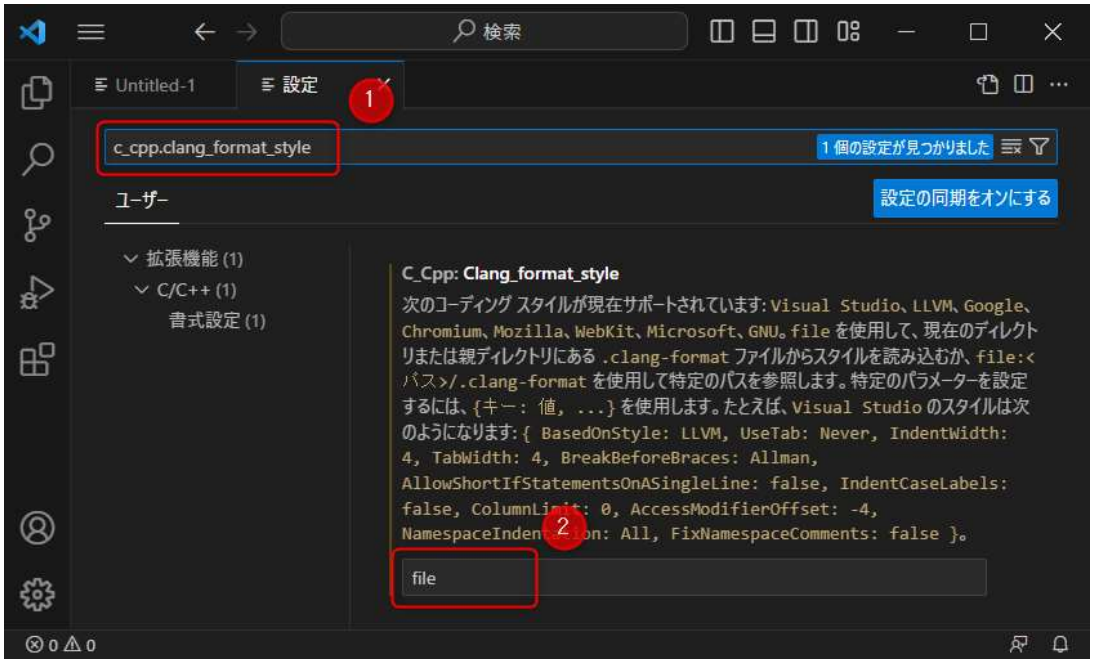
- Visual Studio Code
- C/C++拡張機能
  https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools

3. Visual Studio Code での使用例

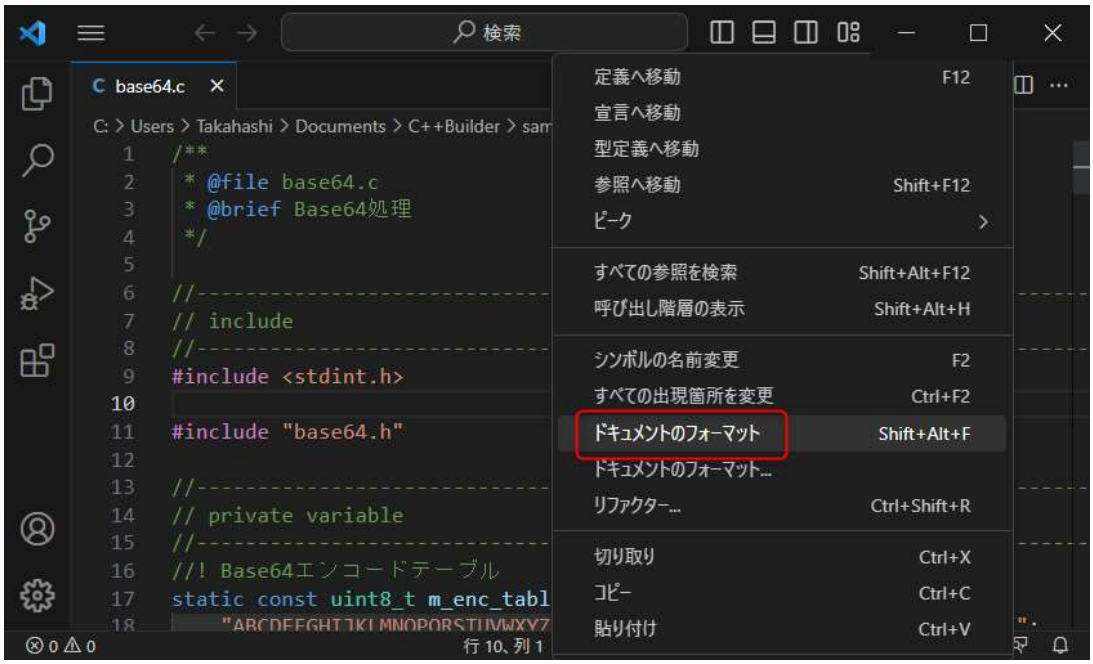
3.1. 設定

(1) 下図①②の順に操作し、設定を開きます。

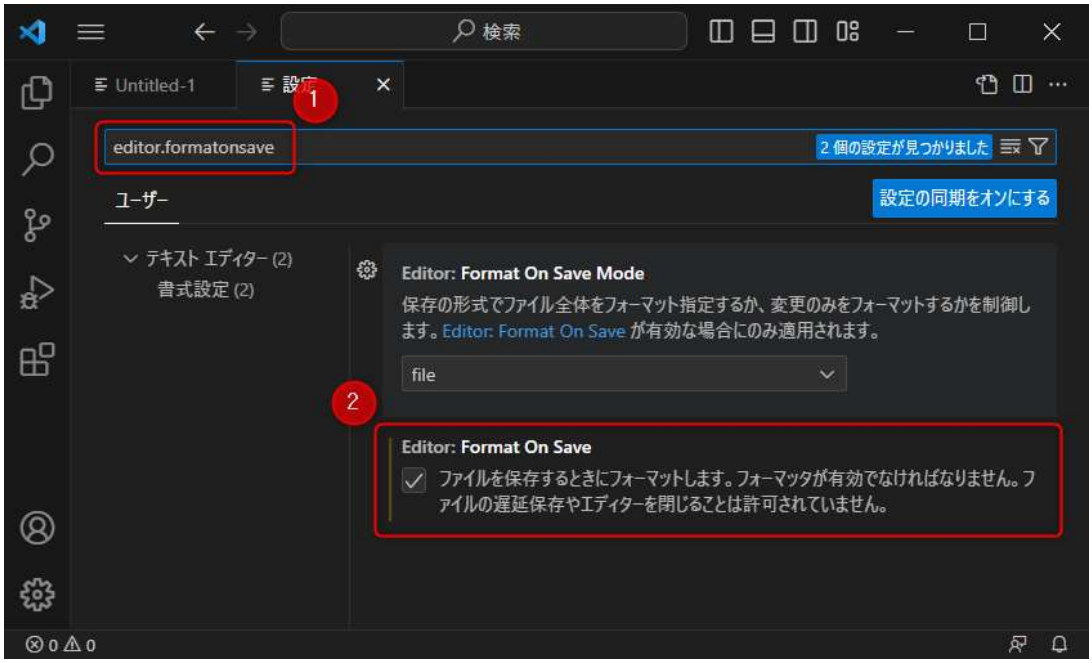
(2) 下図①に「c_cpp.clang_format_style」を入力した後、現れた②に「file」を入力します。



(3) フォーマットを実行するには、右クリックして「ドキュメントのフォーマット（上側)」をクリックします。



(4) 手動でフォーマットを実行するのが面倒な場合、ファイルを保存する際に自動でフォーマットを実行させることもできます。下図①に「editor.formatonsave」を入力した後、現れた②にチェックを付けます。



3.2. 使用方法

(1) 下記の様な構成であるとします。

```
src
 ├ main.c
 ├ main.h
 ├ ...
 └ ...
```

(2) ソースファイルと同じフォルダーに、「.clang-format」ファイルを置きます。

```
src
 ├ main.c
 ├ main.h
 ├ ...
 └ .clang-format
```

(3) Visual Studio Code で src フォルダーを開きます。

(4) ソースファイルを保存、または右クリックの「ドキュメントのフォーマット」を実行すると、コードが整形されます。

4. ClangFormat ファイルの記述例

「.clang-format」ファイルは、下記の様に記述します。
先頭行の「---」は必要なものなので削除しないでください。

```
---
# アクセス修飾子のインデント
AccessModifierOffset: -2

# 引数は左揃え
AlignAfterOpenBracket: Align

# 初期化子リストを左揃え
AlignArrayOfStructures: Left

# 連続する行の位置を揃える
AlignConsecutiveAssignments: true   # "="
AlignConsecutiveDeclarations: true  # 宣言
AlignConsecutiveMacros: true        # マクロ
AlignTrailingComments: true         # コメント

# enum 定数を項目毎に改行する
AllowShortEnumsOnASingleLine: false

# 短い関数を単一行にしない
AllowShortFunctionsOnASingleLine: None

# "{ }"を独立行に配置する
# "BreakBeforeBraces: Custom" の場合に有効
BraceWrapping:
  AfterClass: true              # class の後
  AfterControlStatement: false  # 制御文の後
  AfterEnum: true               # enum の後
  AfterFunction: true           # 関数の後
  AfterStruct: true             # struct の後
  AfterUnion: true              # union の後
  AfterExternBlock: false       # extern の後
  BeforeCatch: false            # catch の前
  BeforeElse: false             # else の前

# "{ }"の位置をカスタム設定する
BreakBeforeBraces: Custom

# 1 行の文字数を制限する (0 で無制限)
ColumnLimit: 120

# include を名前順にソートする
IncludeBlocks: Preserve

# swicth 分の case にインデントを設定する
IndentCaseLabels: false

# インデント
IndentWidth: 2

# ポインターの"*"、"&"は変数宣言の左に記述する
PointerAlignment: Right

# コメントは改行しない
ReflowComments: false

# キャストの後ろにスペースを入れない
SpaceAfterCStyleCast: false

# 制御文の"("の前にスペースを入れる
SpaceBeforeParens: ControlStatements

# コメントの前にスペース入れる
SpacesBeforeTrailingComments: 2
```

5. ClangFormat のスタイルオプション

- ClangFormat にもバージョンがあります。
  使用環境が最新バージョンに対応していないこともあるので、動作確認しながら使用してください。
- Visual Studio Code で動作確認できた範囲のものだけを抜き出しています。最新の ClangFormat とは合わないところもあるかもしれませんが、参考にどうぞ。
- ClangFormat の最新バージョンの情報は下記を参照ください。
  https://clang.llvm.org/docs/ClangFormat.html

5.1. AccessModifierOffset

アクセス修飾子 public、protected、private のインデントを設定します。
IndentWidth を設定している場合は、その値が基準となります。

```
// AccessModifierOffset: -2
// IndentWidth: 2
class Sample {
private:
  int func1(void);

public:
  int func2(void);
}
```

5.2. AlignAfterOpenBracket

引数の整列方法を設定します。

```
// AlignAfterOpenBracket: Align
someLongFunction(argument1,
                 argument2);

// AlignAfterOpenBracket: DontAlign
someLongFunction(argument1,
  argument2);

// AlignAfterOpenBracket: AlwaysBreak
someLongFunction(
  argument1, argument2);

// AlignAfterOpenBracket: BlockIndent
someLongFunction(
  argument1, argument2
);
```

5.3. AlignArrayOfStructures

構造体の初期化子リストを整列します。

```
// AlignArrayOfStructures: None
// 整列しない

// AlignArrayOfStructures: Left
struct test demo[] =
{
  {56, 23, "hello"},
  {7,  5,  "abc"  }
};

// AlignArrayOfStructures: Right
struct test demo[] =
{
  {56, 23, "hello"},
  { 7,  5,   "abc"}
};
```

5.4. AlignConsecutiveAssignments

連続する行の「=」の位置を揃えます。

```
// AlignConsecutiveAssignments: true
aaa  = 0;
bb   = 1;
cccc = 2;
```

5.5. AlignConsecutiveDeclarations

連続する行の宣言を揃えます。

```
// AlignConsecutiveDeclarations: true
char   sc;
int    si;
double d;
```

5.6. AlignConsecutiveMacros

連続する行のマクロを揃えます。

```
// AlignConsecutiveMacros: true
#define SHORT_NAME       42
#define EVEN_LONGER_NAME (2)
#define foo(x)           (x * x)
```

5.7. AlignEscapedNewlines

エスケープされた改行の、円マークの位置を揃えます。

```
// AlignEscapedNewlines: true  (左寄せで揃える)
#define TEST       ¥
  "long long test" ¥
  "string"

// AlignEscapedNewlines: false  (ColumnLimit 位置で揃える)
#define TEST                 ¥
  "long long test"           ¥
  "string"
```

5.8. AlignOperands

水平方向に二項演算子と三項演算子を揃えます。

```
// AlignOperands: true
int aaa = bbbbbbbbbbbbbb +
          ccccccccccccc;
```

5.9. AlignTrailingComments

連続する行のコメントを揃えます。

```
// AlignTrailingComments: true
int a;      // comment a
int b = 2;  // comment b
```

5.10. AllowShortEnumsOnASingleLine

enum 定数を項目毎に改行します。

なお false に設定すると、波括弧も独立した行に表示されます。この仕様は BraceWrapping の個別設定でも回避できません。

```
// AllowShortEnumsOnASingleLine: true
enum { A, B } myEnum;

// AllowShortEnumsOnASingleLine: false
enum
{
  A,
  B
} myEnum;
```

5.11. AllowShortFunctionsOnASingleLine

短い関数を単一行にします。

```
// AllowShortFunctionsOnASingleLine: None
// (単一行にしない)

// AllowShortFunctionsOnASingleLine: InlineOnly
// (class 内で定義された関数だけ単一行にする)
class Foo {
  void f() { foo(); }
};
void f() {
  foo();
}
void f() {
}

// AllowShortFunctionsOnASingleLine: Inline
// (class 内で定義された関数と空の関数だけ単一行にする)
class Foo {
  void f() { foo(); }
};
void f() {
  foo();
}
void f() {}

// AllowShortFunctionsOnASingleLine: Empty
// (空の関数だけ単一行にする)
void f() {}
void f2() {
  bar2();
}

// AllowShortFunctionsOnASingleLine: All
// (すべて単一行にする)
class Foo {
  void f() { foo(); }
};
void f() { bar(); }
```

5.12. BraceWrapping

{ }の位置を個々に設定します。true に設定すると、{ }を独立した行に記述します。
なお個々に設定するためには、BreakBeforeBraces を Custom に設定する必要があります。

```
// インデントはスペースで入れること
BraceWrapping:
  AfterClass: true             // class の後
  AfterControlStatement: true  // 制御文の後
  AfterEnum: true              // enum の後
  AfterFunction: true          // 関数の後
  AfterNamespace: true         // namespace の後
  AfterStruct: true            // struct の後
  AfterUnion: true             // union の後
  AfterExternBlock: true       // extern の後
  BeforeCatch: true            // catch の前
  BeforeElse: true             // else の前
```

5.13. BreakBeforeBraces

{ }の位置を設定します。
なお下記以外にも Linux、Mozilla、Stroustrup、GNU、Webkit がありますが省略します。

```
// BreakBeforeBraces: Attach  (常にコンテキストに付ける)
void function(int val) {
  if (val > MAX) {
    ...
  }
  ...
}

// BreakBeforeBraces: Allman  (常に独立した行に付ける)
void function(int val)
{
  if (val > MAX)
  {
    ...
  }
  ...
}

// BreakBeforeBraces: Custom  (BraceWrapping で個々に設定する)
```

5.14. ColumnLimit

1 行の文字数を設定します。

```
ColumnLimit: 120
```

5.15. IncludeBlocks

include を整列（ソート）します。

```
// IncludeBlocks: Preserve  (連続する行を整列する)
#include <stdio.h>
#include <stdint.h>

#include "b.h"
#include "a.h"
↓
#include <stdint.h>
#include <stdio.h>

#include "a.h"
#include "b.h"

// IncludeBlocks: Merge  (離れた行も引っ付けて整列する)
#include <stdio.h>
#include <stdint.h>

#include "b.h"
#include "a.h"
↓
#include "a.h"
#include "b.h"
#include <stdint.h>
#include <stdio.h>

// IncludeBlocks: Regroup  (IncludeCategories の順に整列する)
```

5.16. IncludeCategories

「IncludeBlocks: Regroup」の整列順を設定します。

```
IncludeCategories:
  - Regex:           '^<.*¥.h>'
    Priority:        1
  - Regex:           '^<.*'
    Priority:        2
  - Regex:           '.*'
    Priority:        3
```

5.17. IndentCaseLabels

case にインデントを設定します。

```
// IndentCaseLabels: true
switch (val) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  default:
    ...
}

// IndentCaseLabels: false
switch (val) {
case 1:
  ...
  break;
case 2:
  ...
  break;
default:
  ...
}
```

5.18. IndentWidth

インデント数を設定します。

```
IndentWidth: 2
```

5.19. PointerAlignment

ポインターの「*」や「&」の位置を揃えます。

```
// PointerAlignment: Left
int* p;

// PointerAlignment: Middle
int * p;

// PointerAlignment: Right
int *p;
```

5.20. ReflowComments

ColumnLimit を超えるコメントの、途中での改行を試みます。

```
// ReflowComments: true  (試みる)

// ReflowComments: false  (試みない)
```

5.21. SpaceAfterCStyleCast

キャストの後にスペースを入れるかを設定します。

```
// SpaceAfterCStyleCast: true
(int) i;

// SpaceAfterCStyleCast: false
(int)i;
```

5.22. SpaceBeforeAssignmentOperators

演算子の前にスペースを入れるかを設定します。

```
// SpaceBeforeAssignmentOperators: true
i = i + 1;

// SpaceBeforeAssignmentOperators: false
i= i + 1;
```

5.23. SpaceBeforeParens

開始括弧「(」の前にスペースを入れるかを設定します。

```
// SpaceBeforeParens: Never  (スペースを入れない)
void function(int val) {
  if(val > MAX) {
    ...
  }
}

// SpaceBeforeParens: ControlStatements  (制御文の前にはスペースを入れる)
void function(int val) {
  if (val > MAX) {
    ...
  }
}

// SpaceBeforeParens: Always  (すべてにスペースを入れる)
void function (int val) {
  if (val > MAX) {
    ...
  }
}
```

5.24. SpaceInEmptyParentheses

空の括弧( )内にスペースを入れるかを設定します。

```
// SpaceInEmptyParentheses: true
func( );

// SpaceInEmptyParentheses: false
func();
```

5.25. SpacesBeforeTrailingComments

コメントの前に入れるスペースの数を設定します。

```
// SpacesBeforeTrailingComments: 2
int name;  // 2 スペース
```

5.26. SpacesInAngles

<>内にスペースを入れるかを設定します。

```
// SpacesInAngles: true
static_cast< int >(arg);

// SpacesInAngles: false
static_cast<int>(arg);
```

5.27. SpacesInCStyleCastParentheses

キャストの( )内にスペースを入れるかを設定します。

```
// SpacesInCStyleCastParentheses: true
( int )i;

// SpacesInCStyleCastParentheses: false
(int)i;
```

5.28. SpacesInParentheses

( )内にスペースを入れるかを設定します。

```
// SpacesInParentheses: true
void function( int val ) {
  if ( val > MAX ) {
    ...
  }
  ...
}

// SpacesInParentheses: false
void function(int val) {
  if (val > MAX) {
    ...
  }
  ...
}
```

5.29. SpacesInSquareBrackets

[ ]内にスペースを入れるかを設定します。

```
// SpacesInSquareBrackets: true
int a[ 5 ];

// SpacesInSquareBrackets: false
int a[5];
```

6. その他

6.1. 部分的に整形の対象外にする

初期化子リストなど、視認性をあげるため意図的に位置揃えしたいこともあります。この様な場合は「clang-format off」と「clang-format on」で囲むことで部分的に整形の対象外にすることができます。

```
// clang-format off
...
// clang-format on
```