Clang-Format Tutorial

CS255 Systems Programming Lab
Spring 2021

Computer Science Department
University of Crete

What is Clang-Format

- Clang-Format is a popular code formatting tool that helps maintain common code style across team members and IDEs.
- It provides an option to store formatting settings in special YAML files named **.clang-format.**
- That way a person, a group of developers or an organization can establish a homogeneous style of format across all the code sources.

How to Run

- Basic command:
  clang-format -i -style="**code style**" "**source code suffixes to be affected**"

  E.g: clang-format -i -style=**llvm *.c *.h**

Basic Configuration options

- Clang-Format supports two options to provide custom style options
- **1st option**: specify style configuration in the **-style="style-option"** command line option
- **2st option**: use **-style=file** and put style configuration in the .clang-format or _clang-format file in the project directory.

Configuring Styles

- Clang-Format supports multiple ready to use style options

Possible values:

- `LLVM` A style complying with the **LLVM coding standards**
- `Google` A style complying with **Google's C++ style guide**
- `Chromium` A style complying with **Chromium's style guide**
- `Mozilla` A style complying with **Mozilla's style guide**
- `WebKit` A style complying with **WebKit's style guide**
- `Microsoft` A style complying with **Microsoft's style guide**
- `GNU` A style complying with the **GNU coding standards**

Custom Configuring Style Options

- AlignConsecutiveAssignments

  Possible values: (true, false)

```
int a            = 1;
int somelongname = 2;
double c         = 3;
```

Custom Configuring Style Options

- AlignConsecutiveDeclarations

  Possible values: (true, false)

```
int         aaaa = 12;
float       b = 23;
std::string ccc = 23;
```

Custom Configuring Style Options

- AlignConsecutiveMacros

  Possible values: (true, false)

```
#define SHORT_NAME       42
#define LONGER_NAME      0x007f
#define EVEN_LONGER_NAME (2)
#define foo(x)           (x * x)
#define bar(y, z)        (y + z)
```

Custom Configuring Style Options

- IndentCaseBlocks
  Possible values: (true, false)

```
false:                                  vs.     true:
switch (fool) {                                 switch (fool) {
case 1: {                                       case 1:
  bar();                                          {
} break;                                            bar();
default: {                                      }
  plop();                                       break;
}                                               default:
}                                               {
                                                    plop();
                                                }
                                            }
```

Custom Configuring Style Options

- IndentWidth (unsigned)

```
IndentWidth: 3

void f() {
   someFunction();
   if (true, false) {
      f();
   }
}
```

Formatting on a Piece of Code

```
int formatted_code;
// clang-format off
    void    unformatted_code  ;
// clang-format on
void formatted_code_again;
```

Configuration for multiple languages

```
---
# We'll use defaults from the LLVM style, but with 4 columns indentation.
BasedOnStyle: LLVM
IndentWidth: 4
---
Language: Cpp
# Force pointers to the type for C++.
DerivePointerAlignment: false
PointerAlignment: Left
---
Language: JavaScript
# Use 100 columns for JS.
ColumnLimit: 100
---
Language: Proto
# Don't format .proto files.
DisableFormat: true
---
Language: CSharp
# Use 100 columns for C#.
ColumnLimit: 100
...
```

More information

For more information about the Clang-Format tool please visit:

- https://clang.llvm.org/docs/ClangFormat.html

For more information about styling options please visit:

- https://releases.llvm.org/11.0.0/tools/clang/docs/ClangFormatStyleOptions.html