clang-format

Automatic formatting for C++

(Daniel Jasper - djasper@google.com)

Why?

- A consistent coding style is important
- Formatting is tedious
  - Clang's source files contain ~25% whitespace characters

```
Sema::NameClassification Sema::ClassifyName(Scope *S,
                                            CXXScopeSpec &SS,
                                            IdentifierInfo *&Name,
                                            SourceLocation NameLoc,
                                            const Token &NextToken,
                                            bool IsAddressOfOperand,
                                            CorrectionCandidateCallback *CCC) {
}
```

Why?

- A consistent coding style is important
- Formatting is tedious
  - Clang's source files contain ~25% whitespace characters

```
Sema::NameClassification Sema::ClassifySomeName(Scope *S,
                                                CXXScopeSpec &SS,
                                                IdentifierInfo *&Name,
                                                SourceLocation NameLoc,
                                                const Token &NextToken,
                                                bool IsAddressOfOperand,
                                                CorrectionCandidateCallback *CC) {

}
```

Why?

- Time wasted on style discussions, e.g. in code reviews
- From cfe-commits@:

> ...
>
> ...
>
> +   while(TemplateParameterDepth <= MemberTemplateDepth)

Space after "while", no spaces immediately inside parens.

...

...

Why?

- Source code becomes machine editable
  - Fully automated refactoring tools!
  - Example: `tools/extra/cpp11-migrate`

```
for (int i = 0; i < N; ++i) { sum += arr[i]; }
for (auto & elem : arr) { sum += elem; }
```

Why?

- Source code becomes machine editable
  - Fully automated refactoring tools!
  - Example: tools/extra/cpp11-migrate

```
for (int i = 0; i < N; ++i) { sum += arr[i]; }
for (auto & elem : arr) { sum += elem; }
```

Process

- Design document
- Feedback on cfe-dev@
- Key ideas / questions:
  - Indentation as well as line breaking
  - Editor integration and library for other tools
  - Only changing whitespaces
  - Parser vs. lexer
  - Style deduction

- Actual solutions might differ :-)

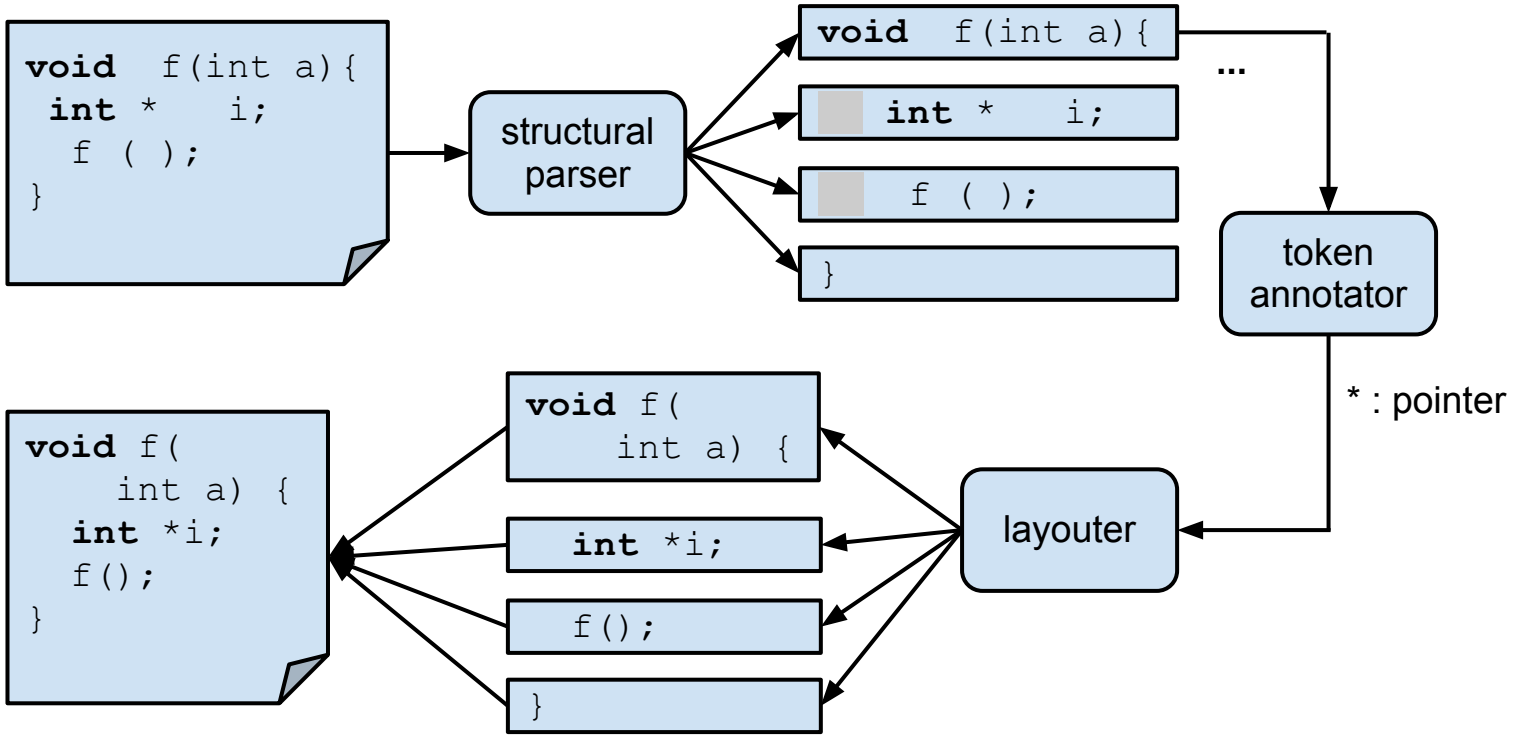
How?

- Build upon Clang component
  - Lexer: C++ token stream
  - Parser: Syntax tree

```
#define TYPE(Class, Parent)                                          \
  case Type::Class: {                                                \
    const Class##Type *ty = cast<Class##Type>(split.Ty);             \
    if (!ty->isSugared())                                            \
      goto done;                                                     \
    next = ty->desugar();                                            \
    break;                                                           \
  }
```

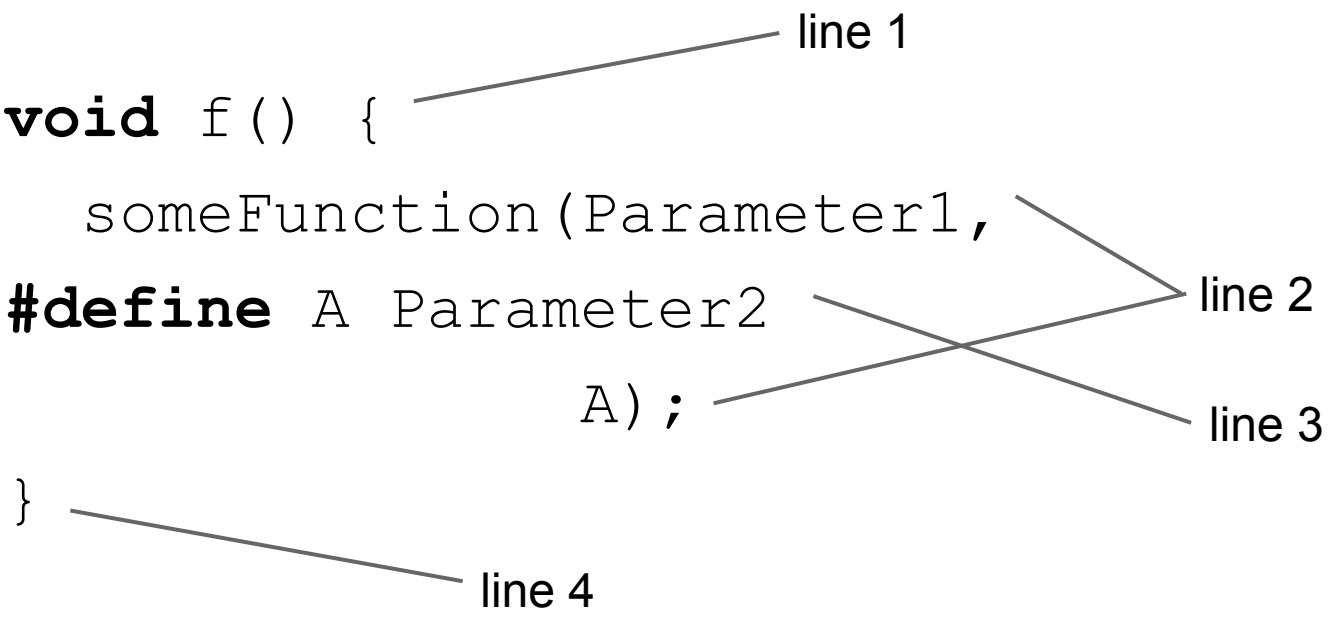
Architecture

- Structural parser: Unwrapped lines
- Layouter: Arrange tokens



Unwrapped lines

- Everything we'd like to put on a single line
- One unwrapped line does not influence other unwrapped lines



Layouter

- Every line break has a certain penalty

```
aaaaaaaa(aaaaaaaaaaaa, aaaaaaaaaaaaaa(aaaaaaaaaaaaaaaaaaaaaaaaaaaa(  Penalty: 100
                           aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa)),  Penalty:  41
         aaaaaaaa(aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa(                    Penalty: 100
             aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa)));  Total:   241
```

- Factors
  - Nesting level
  - Token types
  - Operator precedence
  - ...
- Best formatting: Formatting with lowest penalty

Layouter

- Try "all" the **combinations**
- Clang-format can split or not split at each token

```
int x = a + b + c + d + e + f + g;
    ^   ^   ^   ^   ^   ^   ^   ^
```

- $2^8 = 256$ combinations
- **Memoization** using an "indent state"
  - Consumed **n** Tokens
  - Currently in column **m**
  - ...
- Find cheapest state-path with **Dijkstra**'s algorithm

More important problems

`int *a;` or `int* a;`

- Clang-format has an adaptive mode:
  - Count cases in input
  - Take majority vote

Example: `for`-loops (`Sema.cpp`)

```
for (OverloadExpr::decls_iterator It = Overloads.begin(),
       DeclsEnd = Overloads.end(); It != DeclsEnd; ++It) {}
for (SmallVectorImpl<sema::PossiblyUnreachableDiag>::iterator
     i = Scope->PossiblyUnreachableDiags.begin(),
     e = Scope->PossiblyUnreachableDiags.end();
     i != e; ++i) {}
for (TentativeDefinitionsType::iterator
          T = TentativeDefinitions.begin(ExternalSource),
       TEnd = TentativeDefinitions.end();
     T != TEnd; ++T) {}
for (Module::submodule_iterator Sub = Mod->submodule_begin(),
                                SubEnd = Mod->submodule_end();
     Sub != SubEnd; ++Sub) {}
```

Example: Expression indentation

```
bool value = ((aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa +
               bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb +
               ccccccccccccccccccccccccccccccccccccc) ==
              ((ddddddddddddddddddddddddddddddddddddddddddddd *
                eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee) +
               fffffffffffffffffffffffffffffffff)) &&
             ((ggggggggggggggggggggggggggggggggggggggggggggg *
               hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh) >
              iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii);
```

Example: Expression indentation

```
bool value = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa +
             bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb +
             ccccccccccccccccccccccccccccccccccccccc ==
            ddddddddddddddddddddddddddddddddddddddddddd *
             eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee +
             fffffffffffffffffffffffffffffffffffffff &&
           ggggggggggggggggggggggggggggggggggggggggggggg *
             hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh >
            iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii;
```

Example: Expression indentation

```
bool value = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa +
                     bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb +
                     ccccccccccccccccccccccccccccccccccc ==
                 ddddddddddddddddddddddddddddddddddddddddd *
                         eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee +
                     fffffffffffffffffffffffffffffffffff &&
               ggggggggggggggggggggggggggggggggggggggggggg *
                     hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh >
                  iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii;
```

Example: Expression indentation

```
bool value = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa +
                 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb +
                 ccccccccccccccccccccccccccccccccccccc ==
                ddddddddddddddddddddddddddddddddddddddd *
                      eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee +
                 fffffffffffffffffffffffffffffffffffff &&
              ggggggggggggggggggggggggggggggggggggggggg *
                    hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh >
                iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii;
```

Demo time

How can you use clang-format?

Integration into editors / workflows available:

- vim: `clang-format.py`
- emacs: `clang-format.el`
- diff: `clang-format-diff.py`

All in: `clang/tools/clang-format/`

More to come: Eclipse, TextMate, ...

How can you use clang-format?

As a library (`include/clang/Format/Format.h`):

```
tooling::Replacements reformat(const FormatStyle &Style, Lexer &Lex,
                               SourceManager &SourceMgr,
                               std::vector<CharSourceRange> Ranges,
                               DiagnosticConsumer *DiagClient = 0);
```

- E.g. as postprocessing for refactoring tools
- Interface can be extended

Where are we now?

- Clang-format understands most C++ / ObjC constructs
- Three style guides supported
  - LLVM / Clang
  - Google
  - Chromium
- Clang-format can format its own source code

What next?

- Bugs and formatting improvements
- Configuration (files, command-line, ...)
- More coding styles
  - Coding styles using tabs?
  - Coding styles without column limit?
- C++ 11 features (lambdas, trailing return types, ...)
- clang-tidy
  - Based on Clang's AST
  - Find and fix stuff like:

    "Don’t evaluate `end()` every time through a loop"

Thank you!

clang.llvm.org/docs/ClangFormat.html