Arm, Cambridge, UK

An Overview of Clang

Anastasia Stulova

Sven van Haastregt

LLVM Developers’ Meeting, 22 October 2019

Purpose of this Tutorial

Aimed at people with some basic compiler knowledge but no Clang background.

- Overview of the Clang architecture.
- Taking a simple C program through Clang’s components.
- Working on Clang and testing Clang.

The reality has been simplified in this presentation.

About us

- Working in the Arm Mali GPU OpenCL compiler team.
- Anastasia is the Code Owner of OpenCL in Clang.
- Working with the Clang codebase since 2014.

About us

- Working in the Arm Mali GPU OpenCL compiler team.
- Anastasia is the Code Owner of OpenCL in Clang.
- Working with the Clang codebase since 2014.

We still feel like beginners most of the time!

Outline

Outline

arm

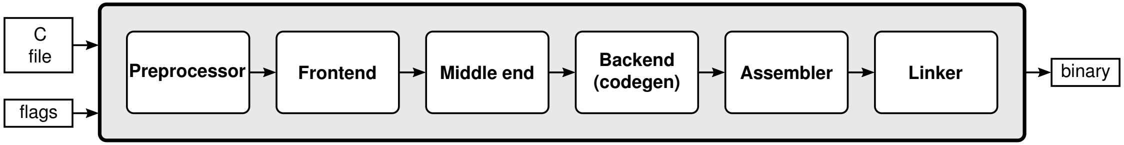
Clang Project

- Part of the LLVM monorepo: `github.com/llvm/llvm-project`
- 21k files (of which 18k are tests).
- Core consists of 830k lines of code plus 33k lines of TableGen definitions.
- Supporting C, C++, Objective C/C++, OpenCL, CUDA, RenderScript.

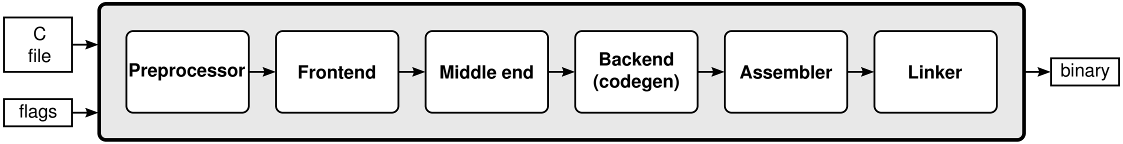
Clang vs Clang

- Clang is a compiler driver.
  - Clang often gets credit/blame for work actually done by LLVM.
    “Clang -O3 is/isn’t doing a great job on this file.”
  - Driving all phases of a compiler invocation, e.g. preprocessing, compiling, linking.
  - Setting flags for current build/installation (e.g. paths to include files).
- Clang is a C language family frontend.
  - Compiling C-like code to LLVM IR.
  - Also known as CFE, cc1, or clang_cc1.
  - The main topic of this tutorial.

Compiler driver phases



Compiler driver phases



```
> clang -ccc-print-phases factorial.c
0: input, "factorial.c", c
1: preprocessor, {0}, cpp-output
2: compiler, {1}, ir
3: backend, {2}, assembler
4: assembler, {3}, object
5: linker, {4}, image
```

Clang as compiler driver

- Phases combined into tool executions.
- Driver invokes the frontend (cc1), linker, ... with the appropriate flags.

```
> clang -### factorial.c
clang version 10.0.0
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /data/llvm/build/bin
"/data/llvm/build/bin/clang-10" "-cc1" "-triple" "x86_64-unknown-linux-gnu" "-emit-obj"
                                "-mrelax-all" "-disable-free" "-main-file-name" "factorial.c"
                                "-mrelocation-model" "static" "-mthread-model" "posix"
                                "-mframe-pointer=all" "-fmath-errno"
                                "-internal-isystem" "/data/llvm/build/lib/clang/10.0.0/include"
                                ...
                                "-x" "c" "factorial.c"
"/usr/bin/ld" "-z" "relro" "--hash-style=gnu" "--eh-frame-hdr" "-m" "elf_x86_64"
              "-dynamic-linker" "/lib64/ld-linux-x86-64.so.2" "-o" "a.out"
              ...
```

Clang as language frontend

Compiling C-like code to LLVM IR.

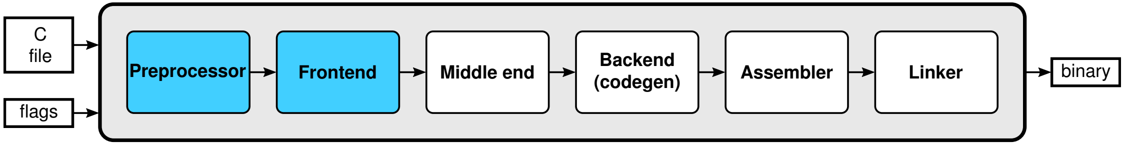
Clang as language frontend

Compiling C-like code to LLVM IR.

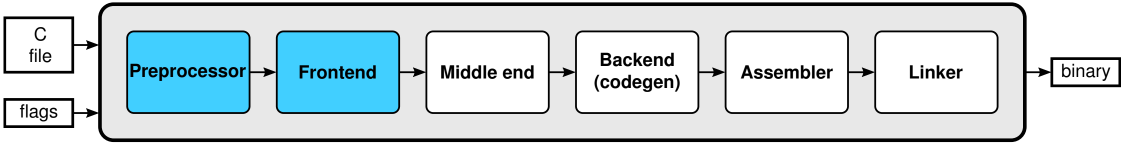
- …and emit helpful diagnostics.
- …and support various standards and dialects.
- …and record source locations for debug information.
- …and provide foundation for many other tools (syntax highlighting, code completion, code refactoring, static analysis, …).

Outline

Core components of Clang



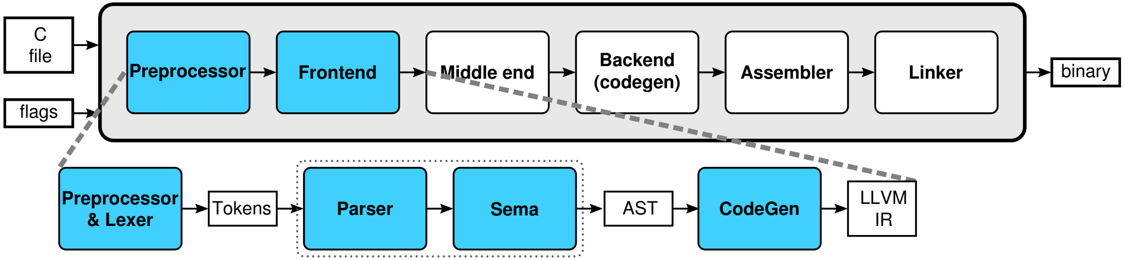
Core components of Clang



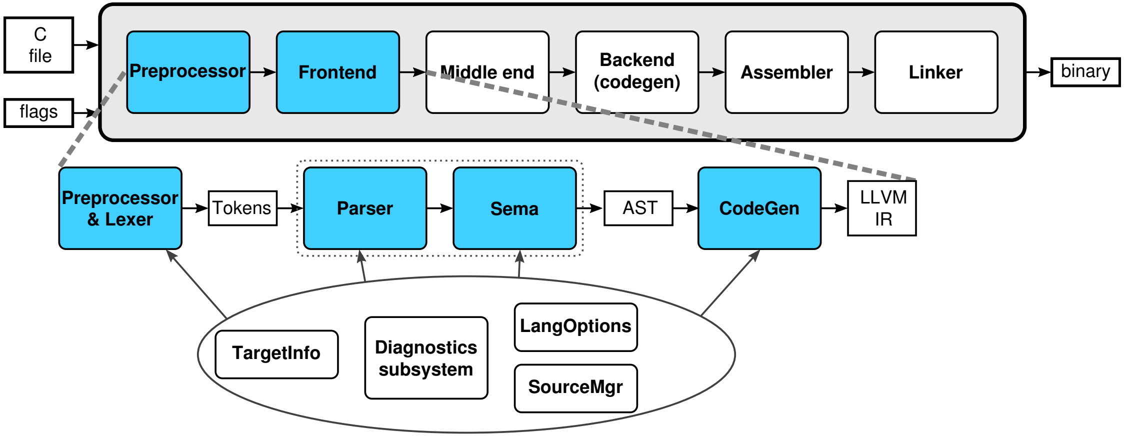
```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
> clang -c factorial.c
```

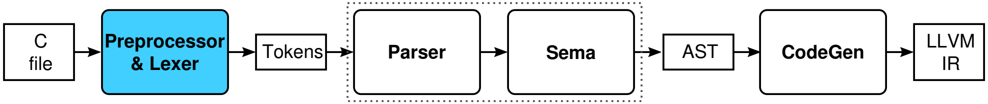
Core components of Clang



Core components of Clang



Lexer



- Converts input program into sequence of *tokens*.
- Performance-critical.
  - Also handles preprocessing.
  - Various "fast paths" for e.g. skipping through `#if 0` blocks, `MultipleIncludeOpt`, …
- Supports tentative parsing.

Lexer Example

```c
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
> clang -c -Xclang -dump-tokens factorial.c
int                'int'           [StartOfLine]                 Loc=<factorial.c:1:1>
identifier         'factorial'     [LeadingSpace]                Loc=<factorial.c:1:5>
l_paren            '('                                           Loc=<factorial.c:1:14>
int                'int'                                         Loc=<factorial.c:1:15>
identifier         'n'             [LeadingSpace]                Loc=<factorial.c:1:19>
r_paren            ')'                                           Loc=<factorial.c:1:20>
l_brace            '{'             [LeadingSpace]                Loc=<factorial.c:1:22>
if                 'if'            [StartOfLine] [LeadingSpace]  Loc=<factorial.c:2:3>
l_paren            '('             [LeadingSpace]                Loc=<factorial.c:2:6>
identifier         'n'                                           Loc=<factorial.c:2:7>
lessequal          '<='            [LeadingSpace]                Loc=<factorial.c:2:9>
numeric_constant   '1'             [LeadingSpace]                Loc=<factorial.c:2:12>
r_paren            ')'                                           Loc=<factorial.c:2:13>
...
```

Lexer Example

```c
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
> clang -c -Xclang -dump-tokens factorial.c
int              'int'        [StartOfLine]                Loc=<factorial.c:1:1>
identifier       'factorial'  [LeadingSpace]               Loc=<factorial.c:1:5>
l_paren          '('                                       Loc=<factorial.c:1:14>
int              'int'                                     Loc=<factorial.c:1:15>
identifier       'n'          [LeadingSpace]               Loc=<factorial.c:1:19>
r_paren          ')'                                       Loc=<factorial.c:1:20>
l_brace          '{'          [LeadingSpace]               Loc=<factorial.c:1:22>
if               'if'         [StartOfLine] [LeadingSpace] Loc=<factorial.c:2:3>
l_paren          '('          [LeadingSpace]               Loc=<factorial.c:2:6>
identifier       'n'                                       Loc=<factorial.c:2:7>
lessequal        '<='         [LeadingSpace]               Loc=<factorial.c:2:9>
numeric_constant '1'          [LeadingSpace]               Loc=<factorial.c:2:12>
r_paren          ')'                                       Loc=<factorial.c:2:13>
...
```

Lexer Example

```c
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
> clang -c -Xclang -dump-tokens factorial.c
int                'int'         [StartOfLine]                Loc=<factorial.c:1:1>
identifier         'factorial'   [LeadingSpace]               Loc=<factorial.c:1:5>
l_paren            '('                                        Loc=<factorial.c:1:14>
int                'int'                                      Loc=<factorial.c:1:15>
identifier         'n'           [LeadingSpace]               Loc=<factorial.c:1:19>
r_paren            ')'                                        Loc=<factorial.c:1:20>
l_brace            '{'           [LeadingSpace]               Loc=<factorial.c:1:22>
if                 'if'          [StartOfLine] [LeadingSpace] Loc=<factorial.c:2:3>
l_paren            '('           [LeadingSpace]               Loc=<factorial.c:2:6>
identifier         'n'                                        Loc=<factorial.c:2:7>
lessequal          '<='          [LeadingSpace]               Loc=<factorial.c:2:9>
numeric_constant   '1'           [LeadingSpace]               Loc=<factorial.c:2:12>
r_paren            ')'                                        Loc=<factorial.c:2:13>
...
```

Lexer Example

```c
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
> clang -c -Xclang -dump-tokens factorial.c
int                  'int'          [StartOfLine]                 Loc=<factorial.c:1:1>
identifier           'factorial'    [LeadingSpace]                Loc=<factorial.c:1:5>
l_paren              '('                                          Loc=<factorial.c:1:14>
int                  'int'                                        Loc=<factorial.c:1:15>
identifier           'n'            [LeadingSpace]                Loc=<factorial.c:1:19>
r_paren              ')'                                          Loc=<factorial.c:1:20>
l_brace              '{'            [LeadingSpace]                Loc=<factorial.c:1:22>
if                   'if'           [StartOfLine] [LeadingSpace]  Loc=<factorial.c:2:3>
l_paren              '('            [LeadingSpace]                Loc=<factorial.c:2:6>
identifier           'n'                                          Loc=<factorial.c:2:7>
lessequal            '<='           [LeadingSpace]                Loc=<factorial.c:2:9>
numeric_constant     '1'            [LeadingSpace]                Loc=<factorial.c:2:12>
r_paren              ')'                                          Loc=<factorial.c:2:13>
...
```

Lexer Internals

Tokens declared in `include/clang/Basic/TokenKinds.def`

```
...
KEYWORD(if                        , KEYALL)
KEYWORD(inline                    , KEYC99|KEYCXX|KEYGNU)
KEYWORD(int                       , KEYALL)
...
```

Lexer Internals

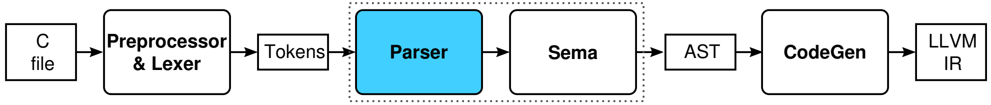
Tokens declared in `include/clang/Basic/TokenKinds.def`

```
...
KEYWORD(if                                  , KEYALL)
KEYWORD(inline                              , KEYC99|KEYCXX|KEYGNU)
KEYWORD(int                                 , KEYALL)
...
```

Token is consumed by `include/clang/Parse/Parser.h`

```
SourceLocation ConsumeToken() {
  ...
  PP.Lex(Tok);
  ...
}
bool TryConsumeToken(tok::TokenKind Expected) {
  if (Tok.isNot(Expected))
    return false;
  PP.Lex(Tok);
  ...
```

Parser



- Handwritten recursive-descent parser.
- Tentative parsing by looking at the tokens ahead.
- Tries to recover from errors to parse as much as possible (and suggest fix-it hints).

Parser Example

```
Call stack:
clang::Parser::ParseRHSOfBinaryExpression
clang::Parser::ParseAssignmentExpression
clang::Parser::ParseExpression
clang::Parser::ParseParenExprOrCondition
clang::Parser::ParseIfStatement
...
clang::Parser::ParseStatementOrDeclaration
clang::Parser::ParseCompoundStatementBody
...
clang::Parser::ParseFunctionDefinition
...
clang::Parser::ParseTopLevelDecl
clang::Parser::ParseFirstTopLevelDecl
clang::ParseAST
...
clang::FrontendAction::Execute
clang::CompilerInstance::ExecuteAction
clang::ExecuteCompilerInvocation
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

Parser Example

```
Call stack:
clang::Parser::ParseRHSOfBinaryExpression
clang::Parser::ParseAssignmentExpression
clang::Parser::ParseExpression
clang::Parser::ParseParenExprOrCondition
clang::Parser::ParseIfStatement
...
clang::Parser::ParseStatementOrDeclaration
clang::Parser::ParseCompoundStatementBody
...
clang::Parser::ParseFunctionDefinition
...
clang::Parser::ParseTopLevelDecl
clang::Parser::ParseFirstTopLevelDecl
clang::ParseAST
...
clang::FrontendAction::Execute
clang::CompilerInstance::ExecuteAction
clang::ExecuteCompilerInvocation
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

Parser Example

```
Call stack:
clang::Parser::ParseRHSOfBinaryExpression
clang::Parser::ParseAssignmentExpression
clang::Parser::ParseExpression
clang::Parser::ParseParenExprOrCondition
clang::Parser::ParseIfStatement
...
clang::Parser::ParseStatementOrDeclaration
clang::Parser::ParseCompoundStatementBody
...
clang::Parser::ParseFunctionDefinition
...
clang::Parser::ParseTopLevelDecl
clang::Parser::ParseFirstTopLevelDecl
clang::ParseAST
...
clang::FrontendAction::Execute
clang::CompilerInstance::ExecuteAction
clang::ExecuteCompilerInvocation
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
function-definition: [C99 6.9.1]
    decl-specs
    declarator
    declaration-list[opt]
    compound-statement
```

Parser Example

```
Call stack:
clang::Parser::ParseRHSOfBinaryExpression
clang::Parser::ParseAssignmentExpression
clang::Parser::ParseExpression
clang::Parser::ParseParenExprOrCondition
clang::Parser::ParseIfStatement
...
clang::Parser::ParseStatementOrDeclaration
clang::Parser::ParseCompoundStatementBody
...
clang::Parser::ParseFunctionDefinition
...
clang::Parser::ParseTopLevelDecl
clang::Parser::ParseFirstTopLevelDecl
clang::ParseAST
...
clang::FrontendAction::Execute
clang::CompilerInstance::ExecuteAction
clang::ExecuteCompilerInvocation
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
compound-statement: [c99 6.8.2]
    '{'
    block-item-list[opt]
    '}'
```

Parser Example

```
Call stack:
clang::Parser::ParseRHSOfBinaryExpression
clang::Parser::ParseAssignmentExpression
clang::Parser::ParseExpression
clang::Parser::ParseParenExprOrCondition
clang::Parser::ParseIfStatement
...
clang::Parser::ParseStatementOrDeclaration
clang::Parser::ParseCompoundStatementBody
...
clang::Parser::ParseFunctionDefinition
...
clang::Parser::ParseTopLevelDecl
clang::Parser::ParseFirstTopLevelDecl
clang::ParseAST
...
clang::FrontendAction::Execute
clang::CompilerInstance::ExecuteAction
clang::ExecuteCompilerInvocation
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
block-item-list:
    block-item |
    block-item-list block-item

block-item:
    declaration | statement
```

Parser Example

```
Call stack:
clang::Parser::ParseRHSOfBinaryExpression
clang::Parser::ParseAssignmentExpression
clang::Parser::ParseExpression
clang::Parser::ParseParenExprOrCondition
clang::Parser::ParseIfStatement
...
clang::Parser::ParseStatementOrDeclaration
clang::Parser::ParseCompoundStatementBody
...
clang::Parser::ParseFunctionDefinition
...
clang::Parser::ParseTopLevelDecl
clang::Parser::ParseFirstTopLevelDecl
clang::ParseAST
...
clang::FrontendAction::Execute
clang::CompilerInstance::ExecuteAction
clang::ExecuteCompilerInvocation
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
if-statement: [C99 6.8.4.1]
    'if' '(' expression ')' statement |
    'if' '(' expression ')' statement
    'else' statement
```

Parser Example

```
Call stack:
clang::Parser::ParseRHSOfBinaryExpression
clang::Parser::ParseAssignmentExpression
clang::Parser::ParseExpression
clang::Parser::ParseParenExprOrCondition
clang::Parser::ParseIfStatement
...
clang::Parser::ParseStatementOrDeclaration
clang::Parser::ParseCompoundStatementBody
...
clang::Parser::ParseFunctionDefinition
...
clang::Parser::ParseTopLevelDecl
clang::Parser::ParseFirstTopLevelDecl
clang::ParseAST
...
clang::FrontendAction::Execute
clang::CompilerInstance::ExecuteAction
clang::ExecuteCompilerInvocation
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

```
expression: [C99 6.5.17]
    assignment-expression ...[opt] |
    expression ','
    assignment-expression ...[opt]

assignment-expression: [C99 6.5.16]
    conditional-expression |
    unary-expression assignment-operator
    assignment-expression
```

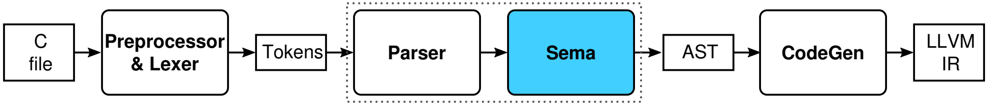
Parser Example

```
Call stack:
clang::Parser::ParseRHSOfBinaryExpression
clang::Parser::ParseAssignmentExpression
clang::Parser::ParseExpression
clang::Parser::ParseParenExprOrCondition
clang::Parser::ParseIfStatement
...
clang::Parser::ParseStatementOrDeclaration
clang::Parser::ParseCompoundStatementBody
...
clang::Parser::ParseFunctionDefinition
...
clang::Parser::ParseTopLevelDecl
clang::Parser::ParseFirstTopLevelDecl
clang::ParseAST
...
clang::FrontendAction::Execute
clang::CompilerInstance::ExecuteAction
clang::ExecuteCompilerInvocation
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

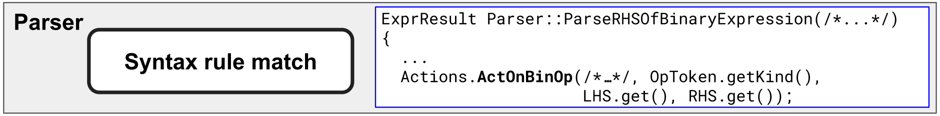
```
primary-expression:   [C99 6.5.1]
    identifier |
    id-expression |
    constant |
    ...
...

relational-expression: [C99 6.5.8]
    shift-expression |
    relational-expression '<' shift-expression |
    relational-expression '>' shift-expression |
    relational-expression '<=' shift-expression |
    relational-expression '>=' shift-expression
```

Sema



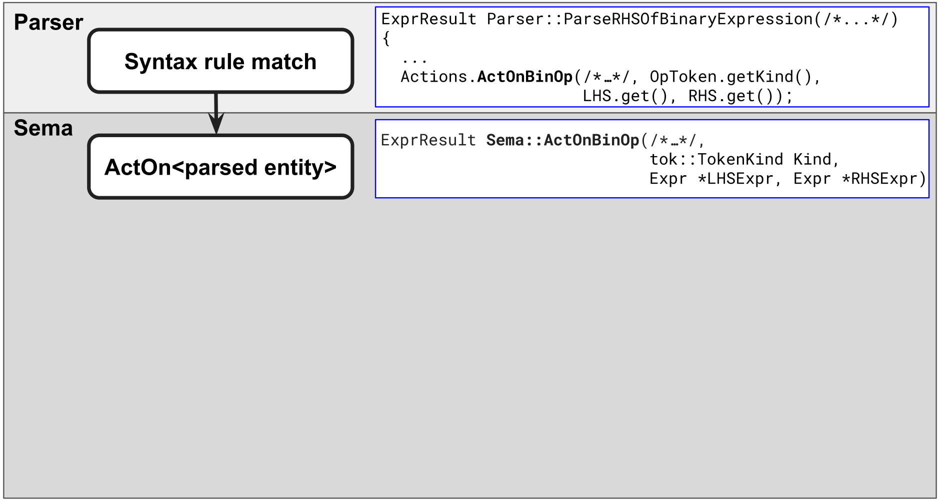
- Tight coupling with parser.
- Biggest client of the Diagnostics subsystem.

Sema Example



```
ExprResult Parser::ParseRHSOfBinaryExpression(/*...*/)
{
  ...
  Actions.ActOnBinOp(/*…*/, OpToken.getKind(),
                           LHS.get(), RHS.get());
```

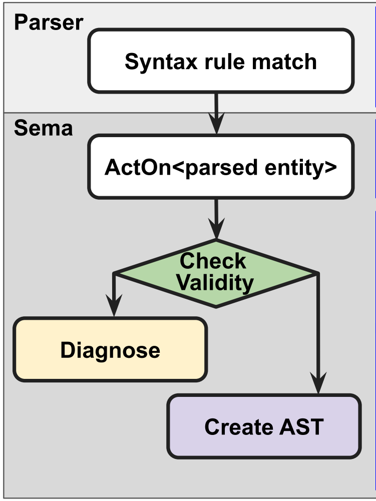
Sema Example



```
ExprResult Parser::ParseRHSOfBinaryExpression(/*...*/)
{
  ...
  Actions.ActOnBinOp(/*…*/, OpToken.getKind(),
                     LHS.get(), RHS.get());
```

```
ExprResult Sema::ActOnBinOp(/*…*/,
                            tok::TokenKind Kind,
                            Expr *LHSExpr, Expr *RHSExpr)
```

Sema Example



```
ExprResult Parser::ParseRHSOfBinaryExpression(/*...*/)
{
  ...
  Actions.ActOnBinOp(/*…*/, OpToken.getKind(),
                               LHS.get(), RHS.get());
```

```
ExprResult Sema::ActOnBinOp(/*…*/,
                            tok::TokenKind Kind,
                            Expr *LHSExpr, Expr *RHSExpr)
```

```
// Called by functions in ActOnBinOp
QualType Sema::CheckMultiplyDivideOperands(/*…*/) {
  ...
  if (compType.isNull() || !compType->isArithmeticType())
    return InvalidOperands(Loc, LHS, RHS);
```

```
QualType Sema::InvalidOperands(/*…*/) {
  ...
  Diag(Loc, diag::err_typecheck_invalid_operands)
      << OrigLHS.getType() << OrigRHS.getType()
      << LHS.get()->getSourceRange() << /*...*/;
```

```
// Called by functions in ActOnBinOp
return new (Context) BinaryOperator(LHS.get(), RHS.get(),
```

Diagnostics subsystem

- Purpose: communicate with human through *diagnostics*:
  - Severity, e.g. note, warning, or error.
  - A source location, e.g. `factorial.c:2:1`.
  - A message, e.g. “unknown type name ’intt’; did you mean ’int’?”
- Defined in `Diagnostic*Kinds.td` TableGen files.
- Emitted through helper function `Diag()`.

Diagnostics example

```
factorial.c:2:1: error: unknown type name 'i'
i factorial(int n) {
^
```

Diagnostics example

```
factorial.c:2:1: error: unknown type name 'i'
i factorial(int n) {
^
```

Defined in `include/clang/Basic/DiagnosticSemaKinds.td`:

```
def err_unknown_typename : Error<
  "unknown type name %0">;
```

Diagnostics example

```
factorial.c:2:1: error: unknown type name 'i'
i factorial(int n) {
^
```

Defined in `include/clang/Basic/DiagnosticSemaKinds.td`:

```
def err_unknown_typename : Error<
  "unknown type name %0">;
```

Triggered in `lib/Sema/SemaDecl.cpp`:

```
void Sema::DiagnoseUnknownTypeName(IdentifierInfo *&II,
                                   SourceLocation IILoc,
  ...
  if (!SS || (!SS->isSet() && !SS->isInvalid()))
    Diag(IILoc, IsTemplateName ? diag::err_no_template
                               : diag::err_unknown_typename)
        << II;
```

Diagnostics example

```
factorial.c:2:1: error: unknown type name 'i'
i factorial(int n) {
^
```

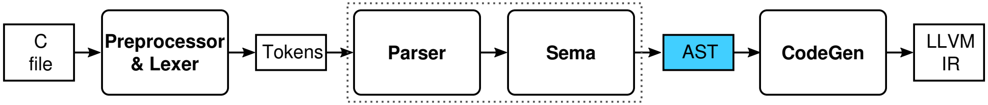
Defined in `include/clang/Basic/DiagnosticSemaKinds.td`:

```
def err_unknown_typename : Error<
  "unknown type name %0">;
```

Triggered in `lib/Sema/SemaDecl.cpp`:

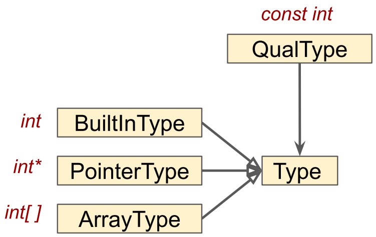
```
void Sema::DiagnoseUnknownTypeName(IdentifierInfo *&II,
                                   SourceLocation IILoc,
  ...
  if (!SS || (!SS->isSet() && !SS->isInvalid()))
    Diag(IILoc, IsTemplateName ? diag::err_no_template
                               : diag::err_unknown_typename)
        << II;
```

Abstract Syntax Tree (AST)



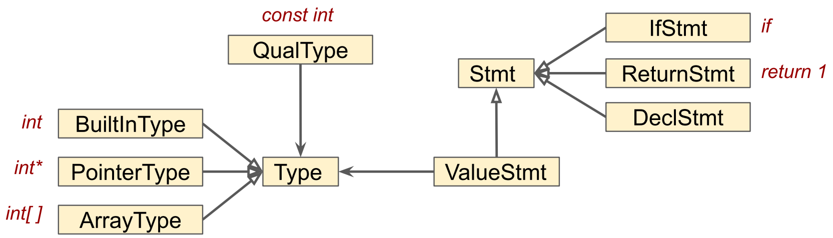
- Representing the original source in a “faithful” way.
- Mostly immutable.

AST Nodes



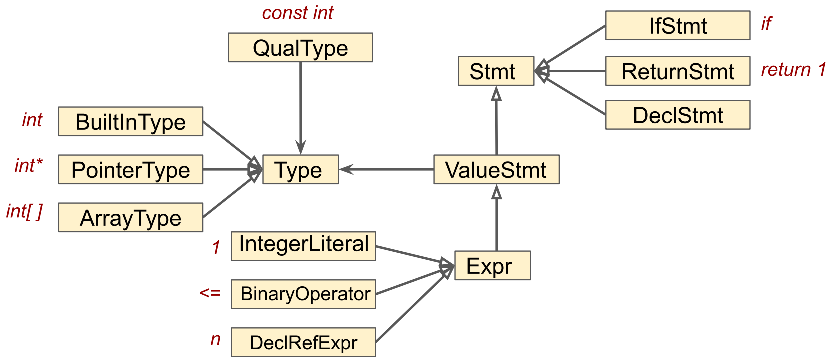
See full diagram: `https://clang.llvm.org/doxygen/inherits.html`

AST Nodes



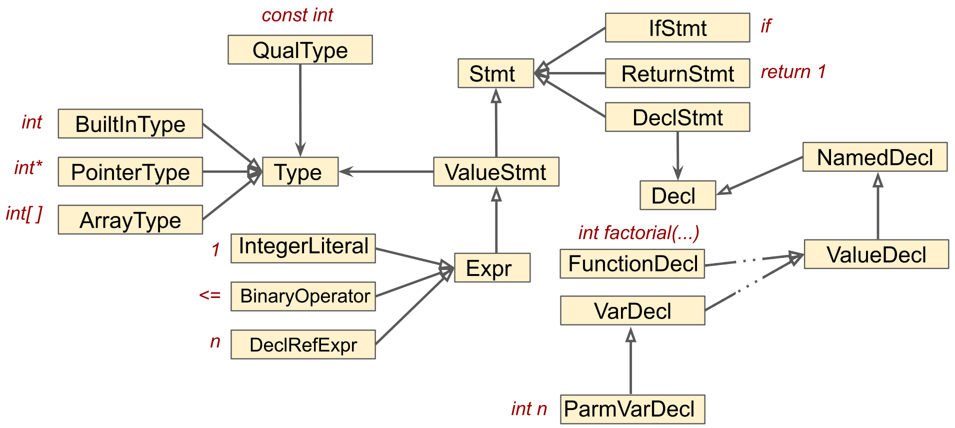
See full diagram: `https://clang.llvm.org/doxygen/inherits.html`

AST Nodes



See full diagram: `https://clang.llvm.org/doxygen/inherits.html`

AST Nodes



See full diagram: `https://clang.llvm.org/doxygen/inherits.html`

AST Example

```
> clang -c -Xclang -ast-dump factorial.c
FunctionDecl <factorial.c:2:1, line:6:1> line:2:5 referenced factorial 'int (int)'
|-ParmVarDecl <col:15, col:19> col:19 used n 'int'
`-CompoundStmt <col:22, line:6:1>
  |-IfStmt <line:3:3, line:4:12>
  | |-BinaryOperator <line:3:7, col:12> 'int' '<='
  | | |-ImplicitCastExpr <col:7> 'int' <LValueToRValue>
  | | | `-DeclRefExpr <col:7> 'int' lvalue ParmVar 'n' 'int'
  | | `-IntegerLiteral <col:12> 'int' 1
  | `-ReturnStmt <line:4:5, col:12>
  |   `-IntegerLiteral <col:12> 'int' 1
  `-ReturnStmt <line:5:3, col:29>
    `-...
```

AST Example

```
> clang -c -Xclang -ast-dump factorial.c
FunctionDecl <factorial.c:2:1, line:6:1> line:2:5 referenced factorial 'int (int)'
|-ParmVarDecl <col:15, col:19> col:19 used n 'int'
`-CompoundStmt <col:22, line:6:1>
  |-IfStmt <line:3:3, line:4:12>
  | |-BinaryOperator <line:3:7, col:12> 'int' '<='
  | | |-ImplicitCastExpr <col:7> 'int' <LValueToRValue>
  | | | `-DeclRefExpr <col:7> 'int' lvalue ParmVar 'n' 'int'
  | | `-IntegerLiteral <col:12> 'int' 1
  | `-ReturnStmt <line:4:5, col:12>
  |   `-IntegerLiteral <col:12> 'int' 1
  `-ReturnStmt <line:5:3, col:29>
    `-...
```

AST Example

```
> clang -c -Xclang -ast-dump factorial.c
FunctionDecl <factorial.c:2:1, line:6:1> line:2:5 referenced factorial 'int (int)'
|-ParmVarDecl <col:15, col:19> col:19 used n 'int'
`-CompoundStmt <col:22, line:6:1>
  |-IfStmt <line:3:3, line:4:12>
  | |-BinaryOperator <line:3:7, col:12> 'int' '<='
  | | |-ImplicitCastExpr <col:7> 'int' <LValueToRValue>
  | | | `-DeclRefExpr <col:7> 'int' lvalue ParmVar 'n' 'int'
  | | `-IntegerLiteral <col:12> 'int' 1
  | `-ReturnStmt <line:4:5, col:12>
  |   `-IntegerLiteral <col:12> 'int' 1
  `-ReturnStmt <line:5:3, col:29>
    `-...
```

AST Example

```
> clang -c -Xclang -ast-dump factorial.c
FunctionDecl <factorial.c:2:1, line:6:1> line:2:5 referenced factorial 'int (int)'
|-ParmVarDecl <col:15, col:19> col:19 used n 'int'
`-CompoundStmt <col:22, line:6:1>
  |-IfStmt <line:3:3, line:4:12>
  | |-BinaryOperator <line:3:7, col:12> 'int' '<='
  | | |-ImplicitCastExpr <col:7> 'int' <LValueToRValue>
  | | | `-DeclRefExpr <col:7> 'int' lvalue ParmVar 'n' 'int'
  | | `-IntegerLiteral <col:12> 'int' 1
  | `-ReturnStmt <line:4:5, col:12>
  |   `-IntegerLiteral <col:12> 'int' 1
  `-ReturnStmt <line:5:3, col:29>
    `-...
```

AST Example

```
> clang -c -Xclang -ast-dump factorial.c
FunctionDecl <factorial.c:2:1, line:6:1> line:2:5 referenced factorial 'int (int)'
|-ParmVarDecl <col:15, col:19> col:19 used n 'int'
`-CompoundStmt <col:22, line:6:1>
  |-IfStmt <line:3:3, line:4:12>
  | |-BinaryOperator <line:3:7, col:12> 'int' '<='
  | | |-ImplicitCastExpr <col:7> 'int' <LValueToRValue>
  | | | `-DeclRefExpr <col:7> 'int' lvalue ParmVar 'n' 'int'
  | | `-IntegerLiteral <col:12> 'int' 1
  | `-ReturnStmt <line:4:5, col:12>
  |   `-IntegerLiteral <col:12> 'int' 1
  `-ReturnStmt <line:5:3, col:29>
    `-...
```

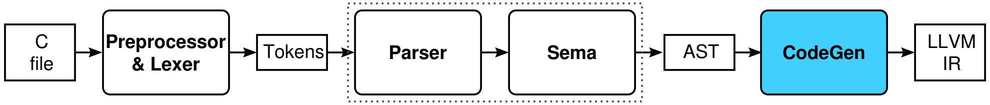
AST Example

```
> clang -c -Xclang -ast-dump factorial.c
FunctionDecl <factorial.c:2:1, line:6:1> line:2:5 referenced factorial 'int (int)'
|-ParmVarDecl <col:15, col:19> col:19 used n 'int'
`-CompoundStmt <col:22, line:6:1>
  |-IfStmt <line:3:3, line:4:12>
  | |-BinaryOperator <line:3:7, col:12> 'int' '<='
  | | |-ImplicitCastExpr <col:7> 'int' <LValueToRValue>
  | | | `-DeclRefExpr <col:7> 'int' lvalue ParmVar 'n' 'int'
  | | `-IntegerLiteral <col:12> 'int' 1
  | `-ReturnStmt <line:4:5, col:12>
  |   `-IntegerLiteral <col:12> 'int' 1
  `-ReturnStmt <line:5:3, col:29>
    `-...
```

AST Visitors

- `RecursiveASTVisitor` for visiting the full AST.
- `StmtVisitor` for visiting `Stmt` and `Expr`.
- `TypeVisitor` for visiting `Type` hierarchy.

CodeGen



- Not to be confused with LLVM CodeGen! (which generates machine code)
- Uses AST visitors, `IRBuilder`, and `TargetInfo`.
- `CodeGenModule` class keeps global state, e.g. LLVM type cache.
  Emits global and some shared entities.
- `CodeGenFunction` class keeps per function state.
  Emits LLVM IR for function body statements.

CodeGen Example

```
Call stack:
(anonymous namespace)::ScalarExprEmitter::VisitIntegerLiteral
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
(anonymous namespace)::ScalarExprEmitter::EmitBinOps
(anonymous namespace)::ScalarExprEmitter::EmitCompare
(anonymous namespace)::ScalarExprEmitter::VisitBinLE
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
clang::CodeGen::CodeGenFunction::EmitScalarExpr
...
clang::CodeGen::CodeGenFunction::EmitBranchOnBoolExpr
clang::CodeGen::CodeGenFunction::EmitIfStmt
clang::CodeGen::CodeGenFunction::EmitStmt
clang::CodeGen::CodeGenFunction::EmitCompoundStmtWithoutScope
clang::CodeGen::CodeGenFunction::EmitFunctionBody
clang::CodeGen::CodeGenFunction::GenerateCode
clang::CodeGen::CodeGenModule::EmitGlobalFunctionDefinition
...
clang::CodeGen::CodeGenModule::EmitTopLevelDecl
...
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

CodeGen Example

```
Call stack:
(anonymous namespace)::ScalarExprEmitter::VisitIntegerLiteral
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
(anonymous namespace)::ScalarExprEmitter::EmitBinOps
(anonymous namespace)::ScalarExprEmitter::EmitCompare
(anonymous namespace)::ScalarExprEmitter::VisitBinLE
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
clang::CodeGen::CodeGenFunction::EmitScalarExpr
...
clang::CodeGen::CodeGenFunction::EmitBranchOnBoolExpr
clang::CodeGen::CodeGenFunction::EmitIfStmt
clang::CodeGen::CodeGenFunction::EmitStmt
clang::CodeGen::CodeGenFunction::EmitCompoundStmtWithoutScope
clang::CodeGen::CodeGenFunction::EmitFunctionBody
clang::CodeGen::CodeGenFunction::GenerateCode
clang::CodeGen::CodeGenModule::EmitGlobalFunctionDefinition
...
clang::CodeGen::CodeGenModule::EmitTopLevelDecl
...
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

CodeGen Example

```
Call stack:
(anonymous namespace)::ScalarExprEmitter::VisitIntegerLiteral
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
(anonymous namespace)::ScalarExprEmitter::EmitBinOps
(anonymous namespace)::ScalarExprEmitter::EmitCompare
(anonymous namespace)::ScalarExprEmitter::VisitBinLE
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
clang::CodeGen::CodeGenFunction::EmitScalarExpr
...
clang::CodeGen::CodeGenFunction::EmitBranchOnBoolExpr
clang::CodeGen::CodeGenFunction::EmitIfStmt
clang::CodeGen::CodeGenFunction::EmitStmt
clang::CodeGen::CodeGenFunction::EmitCompoundStmtWithoutScope
clang::CodeGen::CodeGenFunction::EmitFunctionBody
clang::CodeGen::CodeGenFunction::GenerateCode
clang::CodeGen::CodeGenModule::EmitGlobalFunctionDefinition
...
clang::CodeGen::CodeGenModule::EmitTopLevelDecl
...
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

CodeGen Example

```
Call stack:
(anonymous namespace)::ScalarExprEmitter::VisitIntegerLiteral
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
(anonymous namespace)::ScalarExprEmitter::EmitBinOps
(anonymous namespace)::ScalarExprEmitter::EmitCompare
(anonymous namespace)::ScalarExprEmitter::VisitBinLE
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
clang::CodeGen::CodeGenFunction::EmitScalarExpr
...
clang::CodeGen::CodeGenFunction::EmitBranchOnBoolExpr
clang::CodeGen::CodeGenFunction::EmitIfStmt
clang::CodeGen::CodeGenFunction::EmitStmt
clang::CodeGen::CodeGenFunction::EmitCompoundStmtWithoutScope
clang::CodeGen::CodeGenFunction::EmitFunctionBody
clang::CodeGen::CodeGenFunction::GenerateCode
clang::CodeGen::CodeGenModule::EmitGlobalFunctionDefinition
...
clang::CodeGen::CodeGenModule::EmitTopLevelDecl
...
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

CodeGen Example

```
Call stack:
(anonymous namespace)::ScalarExprEmitter::VisitIntegerLiteral
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
(anonymous namespace)::ScalarExprEmitter::EmitBinOps
(anonymous namespace)::ScalarExprEmitter::EmitCompare
(anonymous namespace)::ScalarExprEmitter::VisitBinLE
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
clang::CodeGen::CodeGenFunction::EmitScalarExpr
...
clang::CodeGen::CodeGenFunction::EmitBranchOnBoolExpr
clang::CodeGen::CodeGenFunction::EmitIfStmt
clang::CodeGen::CodeGenFunction::EmitStmt
clang::CodeGen::CodeGenFunction::EmitCompoundStmtWithoutScope
clang::CodeGen::CodeGenFunction::EmitFunctionBody
clang::CodeGen::CodeGenFunction::GenerateCode
clang::CodeGen::CodeGenModule::EmitGlobalFunctionDefinition
...
clang::CodeGen::CodeGenModule::EmitTopLevelDecl
...
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

CodeGen Example

```
Call stack:
(anonymous namespace)::ScalarExprEmitter::VisitIntegerLiteral
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
(anonymous namespace)::ScalarExprEmitter::EmitBinOps
(anonymous namespace)::ScalarExprEmitter::EmitCompare
(anonymous namespace)::ScalarExprEmitter::VisitBinLE
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
clang::CodeGen::CodeGenFunction::EmitScalarExpr
...
clang::CodeGen::CodeGenFunction::EmitBranchOnBoolExpr
clang::CodeGen::CodeGenFunction::EmitIfStmt
clang::CodeGen::CodeGenFunction::EmitStmt
clang::CodeGen::CodeGenFunction::EmitCompoundStmtWithoutScope
clang::CodeGen::CodeGenFunction::EmitFunctionBody
clang::CodeGen::CodeGenFunction::GenerateCode
clang::CodeGen::CodeGenModule::EmitGlobalFunctionDefinition
...
clang::CodeGen::CodeGenModule::EmitTopLevelDecl
...
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

lib/CodeGen/CGExprScalar.cpp:

```
BinOpInfo
ScalarExprEmitter::EmitBinOps(
    const BinaryOperator *E) {
  BinOpInfo Result;
  Result.LHS = Visit(E->getLHS());
  Result.RHS = Visit(E->getRHS());
  ...
}
```

CodeGen Example

```
Call stack:
(anonymous namespace)::ScalarExprEmitter::VisitIntegerLiteral
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
(anonymous namespace)::ScalarExprEmitter::EmitBinOps
(anonymous namespace)::ScalarExprEmitter::EmitCompare
(anonymous namespace)::ScalarExprEmitter::VisitBinLE
clang::StmtVisitorBase<...ScalarExprEmitter, llvm::Value*>::Visit
(anonymous namespace)::ScalarExprEmitter::Visit
clang::CodeGen::CodeGenFunction::EmitScalarExpr
...
clang::CodeGen::CodeGenFunction::EmitBranchOnBoolExpr
clang::CodeGen::CodeGenFunction::EmitIfStmt
clang::CodeGen::CodeGenFunction::EmitStmt
clang::CodeGen::CodeGenFunction::EmitCompoundStmtWithoutScope
clang::CodeGen::CodeGenFunction::EmitFunctionBody
clang::CodeGen::CodeGenFunction::GenerateCode
clang::CodeGen::CodeGenModule::EmitGlobalFunctionDefinition
...
clang::CodeGen::CodeGenModule::EmitTopLevelDecl
...
cc1_main
```

```
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

lib/CodeGen/CGExprScalar.cpp:

```
Value *VisitIntegerLiteral(
    const IntegerLiteral *E) {
  return Builder.getInt(E->getValue());
}
```

CodeGen Output

```
> clang -S -emit-llvm -o - factorial.c
define dso_local i32 @factorial(i32 %n) #0 {
entry:
  %retval = alloca i32, align 4
  %n.addr = alloca i32, align 4
  store i32 %n, i32* %n.addr, align 4
  %0 = load i32, i32* %n.addr, align 4
  %cmp = icmp sle i32 %0, 1
  br i1 %cmp, label %if.then, label %if.end
if.then:                                          ; preds = %entry
  store i32 1, i32* %retval, align 4
  br label %return
if.end:                                           ; preds = %entry
  %1 = load i32, i32* %n.addr, align 4
  %2 = load i32, i32* %n.addr, align 4
  %sub = sub nsw i32 %2, 1
  %call = call i32 @factorial(i32 %sub)
  %mul = mul nsw i32 %1, %call
  store i32 %mul, i32* %retval, align 4
  br label %return
return:                                           ; preds = %if.end, %if.then
  %3 = load i32, i32* %retval, align 4
  ret i32 %3
```

Outline

Introduction

Overview

Components

Working on Clang

Summary/Questions

Repository Layout (simplified)

https://github.com/llvm/llvm-project/tree/master/clang

```
|-cmake/
|-docs/
|-examples/
|-include/
|  |-clang/Basic/Diagnostic*Kinds.td
|-lib/
|  |-AST/
|  |-Basic/
|  |-CodeGen/
|  |-Driver/
|  |-Lex/
|  |-Parse/
|  `-Sema/
|-test/
|  |-AST/
|  |-CodeGen/
|  |-Driver/
|  |-Lexer/
|  |-Parser/
|  `-Sema/
`-utils/
   `-TableGen/
```

Building Clang

Typically built as part of LLVM, see https://clang.llvm.org/get_started.html

From a developer's perspective:

```
cmake ... -DLLVM_ENABLE_PROJECTS='clang' ...
make
```

Building Clang

Typically built as part of LLVM, see `https://clang.llvm.org/get_started.html`

From a developer's perspective:

```
cmake ... -DLLVM_ENABLE_PROJECTS='clang' ...
make
```

Under the hood:

1. Builds **`clang-tblgen`**.
2. Runs **`clang-tblgen`** to get .inc files from .td files.
3. Builds rest of Clang.

Clang TableGen

Generate C^{++} code from concise TableGen descriptions.

- `Attr.td` Attributes.
- `Diagnostic*Kind.td` Diagnostics.
- `*Options.td` Command line options.
- `arm_neon.td`, `OpenCLBuiltins.td` Builtin functions.

Testing Clang

- `make check-clang` to run Clang tests.
- `clang/unittests` contains unit tests.
- `clang/test` contains many small C/C++ programs for llvm-lit to test that Clang…
  - …does not crash on certain inputs.
  - …parses certain constructs and generates corresponding AST.
  - …generates certain LLVM IR.
  - …emits diagnostics.

Testing Clang - Parser

```
// RUN: %clang_cc1 -ast-dump %s | FileCheck %s
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
// CHECK: FunctionDecl{{.*}}factorial
// CHECK-NEXT: ParmVarDecl
// CHECK-NEXT: CompoundStmt
// CHECK-NEXT: IfStmt
// CHECK: ReturnStmt
// CHECK: ReturnStmt
// CHECK: CallExpr
```

Testing Clang - CodeGen

```
// RUN: %clang -target aarch64-linux-gnu -S -emit-llvm -o - -O0 | FileCheck %s
int factorial(int n) {
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}

// CHECK: i32 @factorial(i32 %n)
// CHECK: icmp sle i32 {{.*}}, 1
// CHECK: [[sub:%.*]] = sub
// CHECK: [[call:%.*]] = call i32 @factorial(i32 [[sub]])
// CHECK: mul .*, [[call]]
// CHECK: ret
```

Testing Clang - Diagnostics

Put expected notes/warnings/errors in source comments:

```
// RUN: %clang_cc1 -verify %s
intt factorial(int n) {
  if (n <= 1) // expected-error{{cannot parse comparisons on Tuesdays}}
    return 1;
  return n * factorial(n - 1);
}
```

Testing Clang - Diagnostics

Put expected notes/warnings/errors in source comments:

```
// RUN: %clang_cc1 -verify %s
intt factorial(int n) {
  if (n <= 1) // expected-error{{cannot parse comparisons on Tuesdays}}
    return 1;
  return n * factorial(n - 1);
}
```

Run Clang with `-verify` to test diagnostics:

```
> clang -cc1 -verify factorial.c
error: 'error' diagnostics expected but not seen:
File factorial.c Line 3: cannot parse comparisons on Tuesdays
error: 'error' diagnostics seen but not expected:
File factorial.c Line 2: unknown type name 'intt'; did you mean 'int'?
```

Testing Clang - Diagnostics

Put expected notes/warnings/errors in source comments:

```
// RUN: %clang_cc1 -verify %s
intt factorial(int n) {
  if (n <= 1) // expected-error{{cannot parse comparisons on Tuesdays}}
    return 1;
  return n * factorial(n - 1);
}
```

Run Clang with `-verify` to test diagnostics:

```
> clang -cc1 -verify factorial.c
error: 'error' diagnostics expected but not seen:
File factorial.c Line 3: cannot parse comparisons on Tuesdays
error: 'error' diagnostics seen but not expected:
File factorial.c Line 2: unknown type name 'intt'; did you mean 'int'?
```

Testing Clang - Diagnostics

Put expected notes/warnings/errors in source comments:

```
// RUN: %clang_cc1 -verify %s
intt factorial(int n) { // expected-error{{unknown type name 'intt'; did you mean 'int'?}}
  if (n <= 1)
    return 1;
  return n * factorial(n - 1);
}
```

Run Clang with `-verify` to test diagnostics:

```
> clang -cc1 -verify factorial.c
(pass)
```

Outline

Introduction

Overview

Components

Working on Clang

Summary/Questions

More Information

- Getting started: `https://clang.llvm.org/get_started.html`
- Hacking on Clang: `https://clang.llvm.org/hacking.html`
- Clang Frontend Internals: `https://clang.llvm.org/docs/InternalsManual.html`
- Clang Driver Internals: `https://clang.llvm.org/docs/DriverInternals.html`
- AST Introduction: `https://clang.llvm.org/docs/IntroductionToTheClangAST.html`
- FileCheck: `https://www.llvm.org/docs/CommandGuide/FileCheck.html`
- We need your help to make Clang even better!
  - Clang bugs: `https://bugs.llvm.org/describecomponents.cgi?product=clang`
  - Clang beginner bugs: `https://bugs.llvm.org/buglist.cgi?product=clang&keywords=beginner`
  - Experts: please tag “easy” beginner bugs.

Questions