

The Clang AST

A Tutorial by Manuel Klimek

You'll learn:

1. The basic structure of the Clang AST
2. How to navigate the AST
3. Tools to understand the AST
4. Interfaces to code against the AST
(Tooling, AST matchers, etc)

The Structure of the Clang AST

- rich AST representation
- fully type resolved
- > 100k LOC

ASTContext

- Keeps information around the AST
 - Identifier Table
 - Source Manager
- Entry point into the AST
 - TranslationUnitDecl* getTranslationUnitDecl()

Core Classes

- Decl
- Stmt
- Type

Core Classes

- Decl
 - CXXRecordDecl
 - VarDecl
 - UnresolvedUsingTypenameDecl
- Stmt
- Type

Core Classes

- Decl
- Stmt
 - CompoundStmt
 - CXXTryStmt
 - BinaryOperator
- Type

Core Classes

- Decl
- Stmt
- Type
 - PointerType
 - ParenType
 - SubstTemplateTypeParmType

Glue Classes

- DeclContext
 - inherited by decls that contain other decls
- TemplateArgument
 - accessors for the template argument
- NestedNameSpecifier
- QualType

Glue Methods

- IfStmt: **getThen(), getElse(), getCond()**
- CXXRecordDecl:
getDescribedClassTemplate()
- Type: **getAsCXXRecordDecl()**

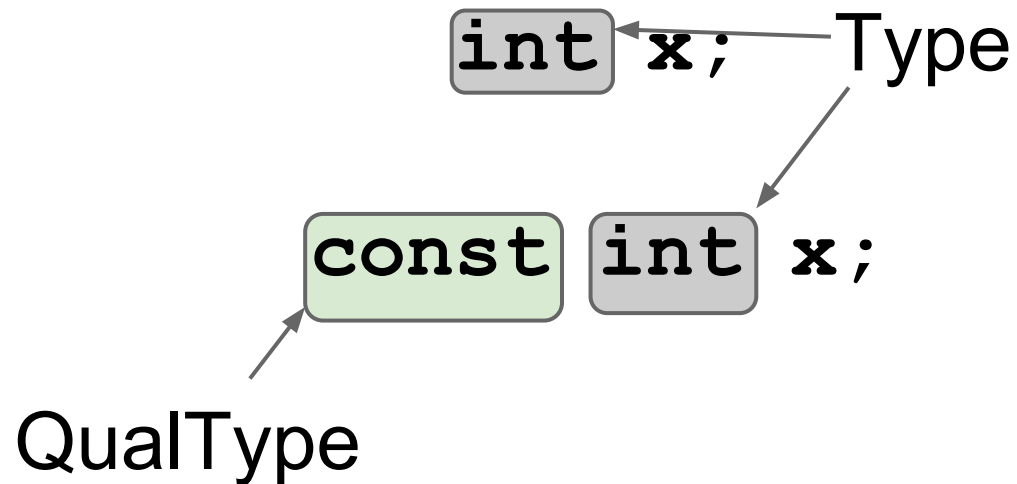
Types are complicated...

Types are complicated...

```
int x;
```

```
const int x;
```

Types are complicated...



Types are complicated...

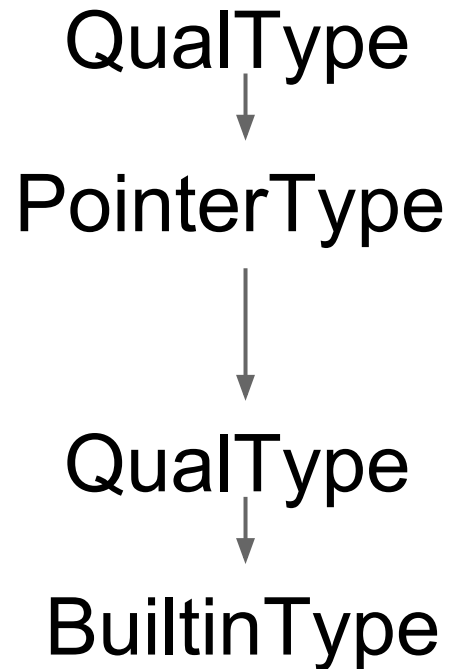
```
int * const * x;
```

Types are complicated...

```
class PointerType {  
    QualType getPointeeType() const;  
  
};
```

Types are complicated...

int * p;

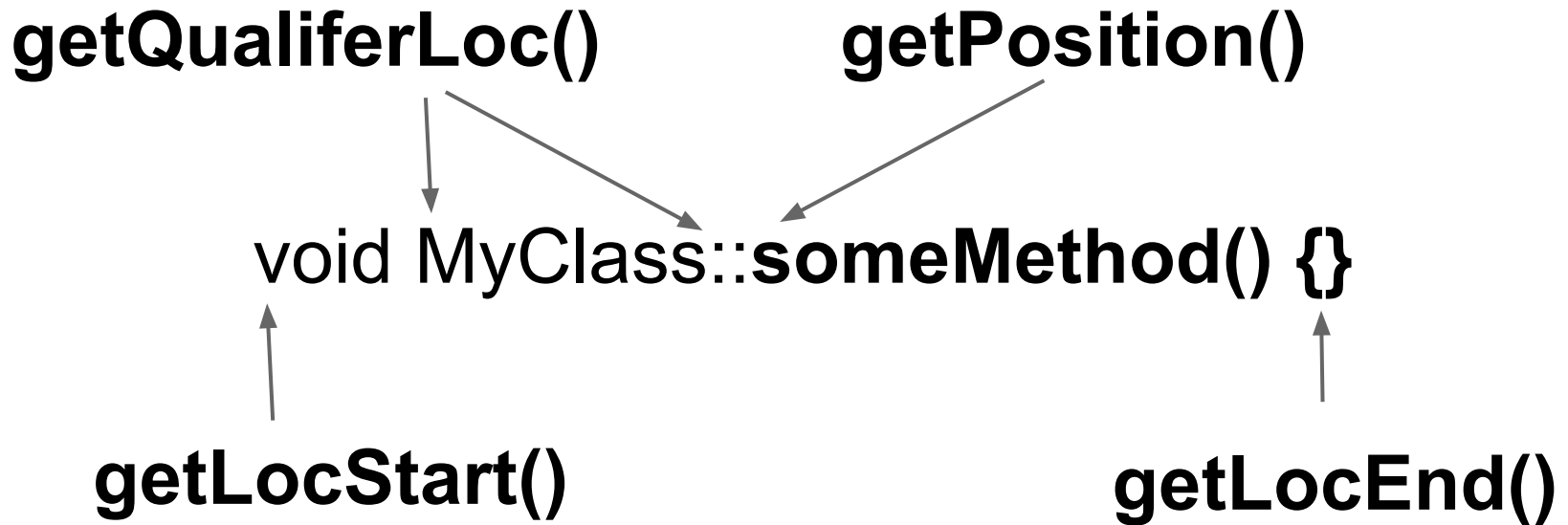


Location, Location, Location

```
class SourceLocation { unsigned ID; };
```

- points to **Tokens**
- managed by **SourceManager**

Navigating Source: Declarations



Navigating Source: Call Expressions

getCallee()

->getBase()

->getNameInfo()

->getLoc()

getCallee()

->getMemberNameInfo()

->getLoc()

Var.function()

```
graph TD; A["getCallee()<br/>->getBase()<br/>->getNameInfo()<br/>->getLoc()"] --> D["Var.function()"]; B["getCallee()<br/>->getMemberNameInfo()<br/>->getLoc()"] --> D; C["getLocStart()"] --> D; E["getLocEnd()"] --> D;
```

getLocStart()

getLocEnd()

Navigating Source: Types

MyClass c;

void f(MyClass c);

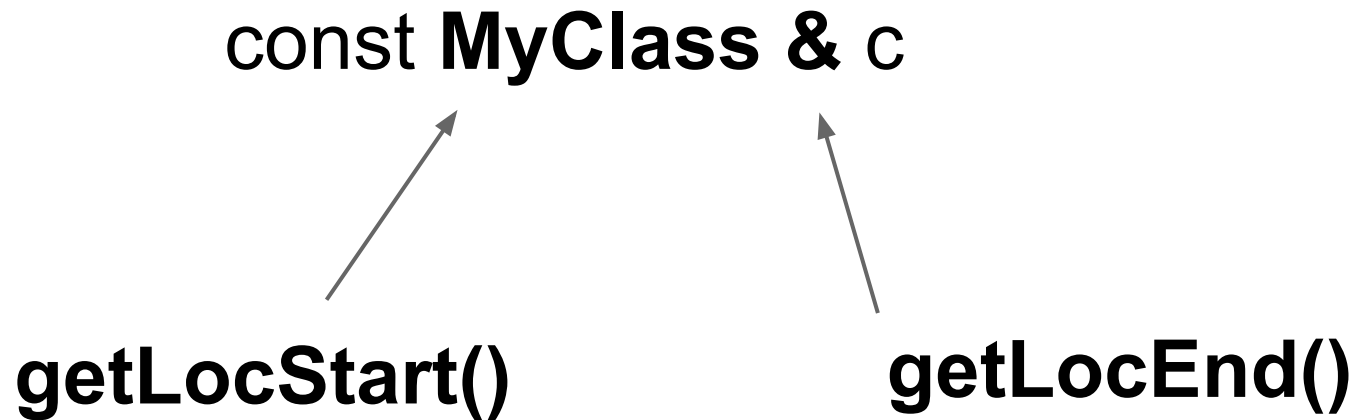
TypeLoc

TypeLoc

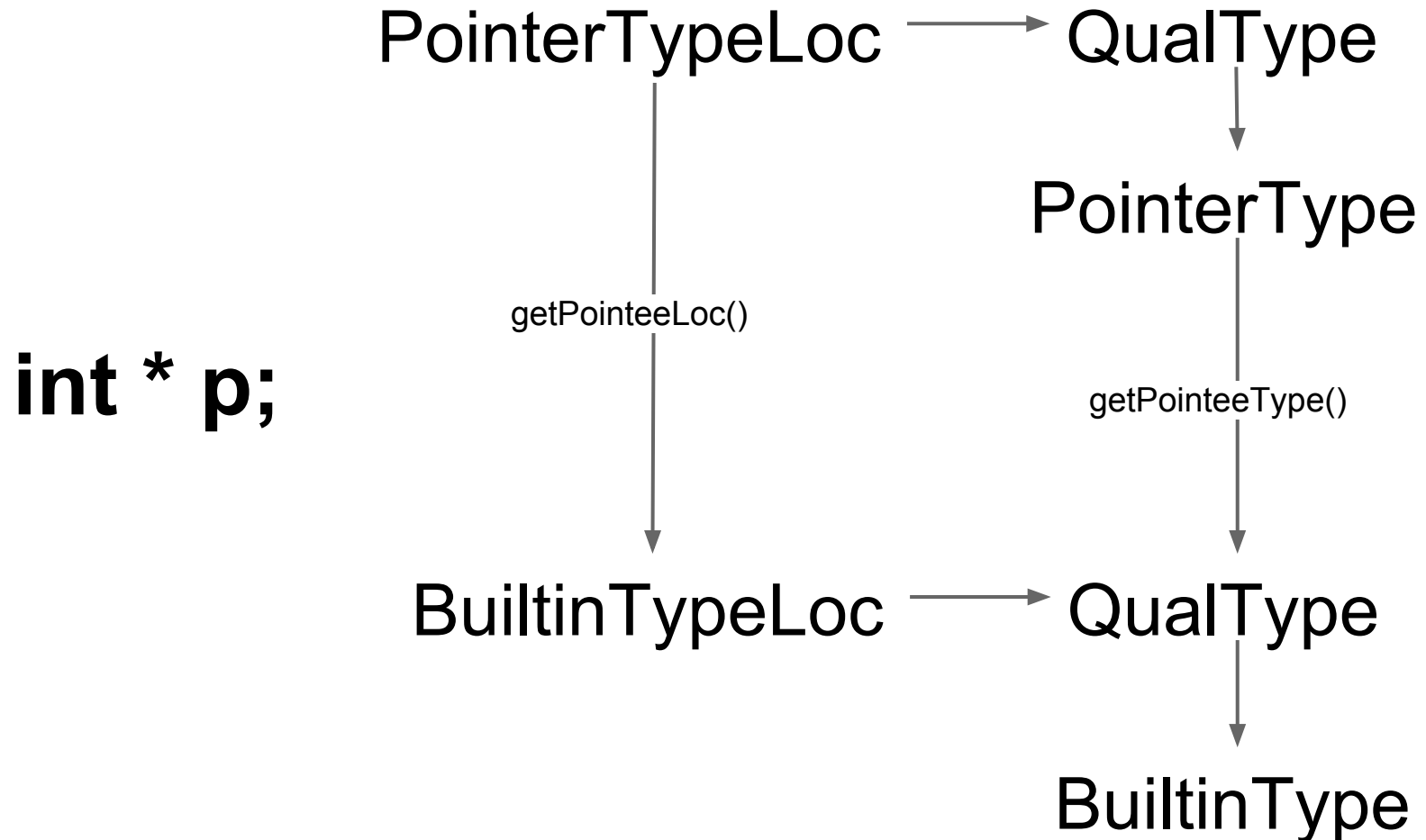
Type:
MyClass

```
graph TD; A["MyClass c;"] --> B["TypeLoc"]; C["void f(MyClass c);"] --> D["TypeLoc"]; B --> E["Type:  
MyClass"]; D --> E;
```

Navigating Source: Types



Navigating Source: Types



Getting the Text

Use the **Lexer!**

- makeFileCharRange
- measureTokenLength

Template tree transformations

- Full AST of **template definition** available
- Full AST for **all instantiations** available
- Nodes are **shared**

RecursiveASTVisitor

- Trigger on **Types** you care about
- Knows all the connections
- Does not give you context information

AST Matchers

- Trigger on **Expressions**
- Bind **Context**
- Get all context inside a callback

Tools!

- clang
 - -ast-dump -ast-dump-filter
 - -ast-list
- clang-check
 - clang + tooling integration

Example 1: The Real World

```
bool TGParser::AddValue(Record *CurRec, SMLoc Loc,
                        const RecordVal &RV) {
    if (CurRec == 0)
        CurRec = &CurMultiClass->Rec;

    if (RecordVal *ERV = CurRec->getValue(RV.getNameInit())) {
        // The value already exists in the class, treat this as a set.
        if (ERV->setValue(RV.getValue()))
            return Error(Loc, "New definition of '" + RV.getName() + "' of type '"
+
                            RV.getType()->getAsString() + "' is incompatible with " +
                            "previous definition of type '" +
                            ERV->getType()->getAsString() + "'");
    } else {
        CurRec->addValue(RV);
    }
    return false;
}
```

From: [llvm/lib/TableGen/TGParser.cpp](#)

Example 1: The Real World

Get the AST for **AddValue**

```
$ clang-check -ast-list lib/TableGen/TGParser.cpp \  
    |grep AddValue  
llvm::TGParser::AddValue  
llvm::TGParser::AddValue
```

Example 1: Dump!

```
$ clang-check -ast-dump \
               -ast-dump-filter=llvm::TGParser::AddValue \
               lib/TableGen/TGParser.cpp
```

[illegible]

Example 1: Dump Details

```
<...>
|-ReturnStmt 0x7f9047a23c28 <line:70:7, line:73:55>
`-ExprWithCleanups 0x7f9047a23c10 <line:70:14, line:73:55> '_Bool'
  -CXXMemberCallExpr 0x7f9047a23ad8 <line:70:14, line:73:55> '_Bool'
    -MemberExpr 0x7f9047a21ff0 <line:70:14> '<bound member function type>' ->Error 0x7f9047b18410
      -ImplicitCastExpr 0x7f9047a23b10 <col:14> 'const class llvm::TGParser *' <NoOp>
        -CXXThisExpr 0x7f9047a21fd8 <col:14> 'class llvm::TGParser *' this
      -CXXConstructExpr 0x7f9047a23b40 <col:20> 'class llvm::SMLoc' 'void (const class llvm::SMLoc &) throw()'
        -ImplicitCastExpr 0x7f9047a23b28 <col:20> 'const class llvm::SMLoc' lvalue <NoOp>
          -DeclRefExpr 0x7f9047a22020 <col:20> 'class llvm::SMLoc' lvalue ParmVar 0x7f9047a218e0 'Loc' 'class llvm::SMLoc'
        -MaterializeTemporaryExpr 0x7f9047a23bf8 <col:25, line:73:52> 'const class llvm::Twine' lvalue
          -ImplicitCastExpr 0x7f9047a23be0 <line:70:25, line:73:52> 'const class llvm::Twine' <ConstructorConversion>
            -CXXConstructExpr 0x7f9047a23ba8 <line:70:25, line:73:52> 'const class llvm::Twine' 'void (const std::string &)'
<...>
```

Example 2: std::string Arguments

```
#include <string>

void f(const std::string& s);

void StdStringArgumentCall(
    const std::string& s) {
    f(s.c_str());
}
```

Example 2: Dump!

```
$ clang-check StdStringArgs.cc -ast-dump -ast-dump-filter=StdStringA --
```

Dumping StdStringArgumentCall:

FunctionDecl

| -ParmVarDecl

`-CompoundStmt

`-ExprWithCleanups

`-CallExpr

| -ImplicitCastExpr <**FunctionToPointerDecay**>

| ` -DeclRefExpr '**f**' 'void (const std::string &)'

`-MaterializeTemporaryExpr

`-CXXBindTemporaryExpr

`-CXXConstructExpr 'void (const char *, const class std::allocator<char> &)'

| -CXXMemberCallExpr 'const char *'

| ` -MemberExpr .c_str

| ` -DeclRefExpr '**s**' 'const std::string &'

`-CXXDefaultArgExpr 'const class std::allocator<char>'

Example 2: Dump!

```
$ clang-check StdStringArgs.cc -ast-dump -ast-dump-filter=StdStringA --
```

Dumping StdStringArgumentCall:

FunctionDecl

| -ParmVarDecl

`-CompoundStmt

`-ExprWithCleanups

`-CallExpr

| -ImplicitCastExpr <FunctionToPointerDecay>

| ` -DeclRefExpr 'f' 'void (const std::string &)'

`-MaterializeTemporaryExpr

`-CXXBindTemporaryExpr

`-CXXConstructExpr 'void (const char *, const class std::allocator<char> &)'

| -CXXMemberCallExpr 'const char *'

| ` -MemberExpr .c_str

| ` -DeclRefExpr 's' 'const std::string &'

`-CXXDefaultArgExpr 'const class std::allocator<char>'

s.c_str()

Example 2: Dump!

```
$ clang-check StdStringArgs.cc -ast-dump -ast-dump-filter=StdStringA --
```

Dumping StdStringArgumentCall:

FunctionDecl

|-ParmVarDecl

`-CompoundStmt

`-ExprWithCleanups

`-CallExpr

| -ImplicitCastExpr <FunctionToPointerDecay>

| `DeclRefExpr 'f' 'void (const std::string &)'

-MaterializeTemporaryExpr

`-CXXBindTemporaryExpr

`-CXXConstructExpr 'void (const char *, const class std::allocator<char> &)'

| -CXXMemberCallExpr 'const char *'

| `MemberExpr .c_str

| `DeclRefExpr 's' 'const std::string &'

-CXXDefaultArgExpr 'const class std::allocator<char>'

string(s.c_str())

Example 2: Dump!

```
$ clang-check StdStringArgs.cc -ast-dump -ast-dump-filter=StdStringA --
```

Dumping StdStringArgumentCall:

FunctionDecl

| -ParmVarDecl

`-CompoundStmt

`-ExprWithCleanups

`-CallExpr

| -ImplicitCastExpr <FunctionToPointerDecay>

|`-DeclRefExpr 'f' 'void (const std::string &)'

`-MaterializeTemporaryExpr

`-CXXBindTemporaryExpr

`-CXXConstructExpr 'void (const char *, const class std::allocator<char> &)'

| -CXXMemberCallExpr 'const char *'

|`-MemberExpr .c_str

|`-DeclRefExpr 's' 'const std::string &'

`-CXXDefaultArgExpr 'const class std::allocator<char>'

f(s.c_str())

Example 2: std::string Arguments

```
#include <string>

void f(const std::string& s);

void StdStringArgumentCall(
    const std::string& s) {
    f(s.c_str());
}
```

Example 2: std::string Arguments

```
#include <string>

void f(const std::string& s);

void StdStringArgumentCall(
    const std::string& s) {
    f(std::string(s.c_str()));
}
```

Example 2: Dump!

Dumping StdStringArgumentCall:

FunctionDecl

|-ParmVarDecl

`-CompoundStmt

 `-ExprWithCleanups

 `-CallExpr

 |-ImplicitCastExpr <FunctionToPointerDecay>

 | `-DeclRefExpr 'f' 'void (const std::string &)'

 `-MaterializeTemporaryExpr

 `-CXXBindTemporaryExpr

 `-CXXConstructExpr

 | -CXXMemberCallExpr 'const char *'

 | `-MemberExpr .c_str

 | `-DeclRefExpr 's' 'const std::string &'

 `-CXXDefaultArgExpr 'const class std::allocator<char>'

Example 2: Dump!

Dumping StdStringArgumentCall:

FunctionDecl

| **-ParmVarDecl**

`- **CompoundStmt**

 `- **ExprWithCleanups**

 `- **CallExpr**

 | **-ImplicitCastExpr** <FunctionToPointerDecay>

 | `- **DeclRefExpr** 'f' 'void (const std::string &)'

 `- **MaterializeTemporaryExpr**

 `- **ImplicitCastExpr** <NoOp>

 `- **CXXFunctionalCastExpr** to std::string <ConstructorConversion>

 `- **CXXBindTemporaryExpr**

 `- **CXXConstructExpr**

 | **-CXXMemberCallExpr** 'const char *'

 | `- **MemberExpr** .c_str

 | `- **DeclRefExpr** 's' 'const std::string &'

 `- **CXXDefaultArgExpr** 'const class std::allocator<char>'

Example 2: Dump!

Dumping StdStringArgumentCall:

FunctionDecl

| **-ParmVarDecl**

`- **CompoundStmt**

 `- **ExprWithCleanups**

 `- **CallExpr**

 | **-ImplicitCastExpr** <FunctionToPointerDecay>

 | `- **DeclRefExpr** 'f' 'void (const std::string &)'

 `- **MaterializeTemporaryExpr**

 `- **ImplicitCastExpr** <NoOp>

 `- **CXXFunctionalCastExpr** to std::string <ConstructorConversion>

 | **-CXXBindTemporaryExpr**

 | `- **CXXConstructExpr**

 | | **-CXXMemberCallExpr** 'const char *'

 | | `- **MemberExpr** .c_str

 | | `- **DeclRefExpr** 's' 'const std::string &'

 | `- **CXXDefaultArgExpr** 'const class std::allocator<char>'

Getting Real

```
#include "clang/ASTMatchers/ASTMatchers.h"
#include "clang/ASTMatchers/ASTMatchFinder.h"
#include "clang/Tooling/Tooling.h"
#include "gtest/gtest.h"

using namespace llvm;
using namespace clang;
using namespace clang::tooling;
using namespace clang::ast_matchers;

class DumpCallback : public MatchFinder::MatchCallback {
    virtual void run(const MatchFinder::MatchResult &Result) {
        llvm::errs() << "---\n";
        Result.Nodes.getNodeAs<CXXRecordDecl>("x")->dump();
    }
};

TEST(DumpCodeSample, Dumps) {
    DumpCallback Callback;
    MatchFinder Finder;
    Finder.addMatcher(recordDecl().bind("x"), &Callback);
    OwningPtr<FrontendActionFactory> Factory(newFrontendActionFactory(&Finder));
    EXPECT_TRUE(clang::tooling::runToolOnCode(Factory->create(), "class X {};"));
}
```

Getting Real

```
class DumpCallback : public MatchFinder::MatchCallback {
    virtual void run(const MatchFinder::MatchResult &Result) {
        llvm::errs() << "---\n";

        const CXXRecordDecl *D = Result.Nodes.getNodeAs<CXXRecordDecl>("x");
        if (const clang::ClassTemplateSpecializationDecl *TS =
            dyn_cast<clang::ClassTemplateSpecializationDecl>(D)) {
            TS->getLocation().dump(*Result.SourceManager);
            llvm::errs() << "\n";
        }
    }
};

"template <typename T> class X {}; X<int> y;"
```

Links

<http://clang.llvm.org/docs/Tooling.html>

<http://clang.llvm.org/docs/IntroductionToTheClangAST.html>

<http://clang.llvm.org/docs/RAVFrontendAction.html>

<http://clang.llvm.org/docs/LibTooling.html>

<http://clang.llvm.org/docs/LibASTMatchers.html>