How to Build Your First C++ Automated Refactoring Tool

KRISTEN SHAKER

Agenda

- Who Am I
- Refactoring Tooling Use Cases
- Clang Crash Course
- Writing a Clang Tidy Check Live
- Special Thanks & Questions

01

Who Am I

C++ Core Libraries

C++ Core Libraries

C++ Core Libraries

C++ Core Libraries

C++ Core Libraries

C++ Core Libraries

Large Scale Refactorings & Automated Refactoring Tools

02

Refactoring Tool Use Cases

Consistent Code Formatting

Migrations & Language Updates

C++ 98 -> C++14

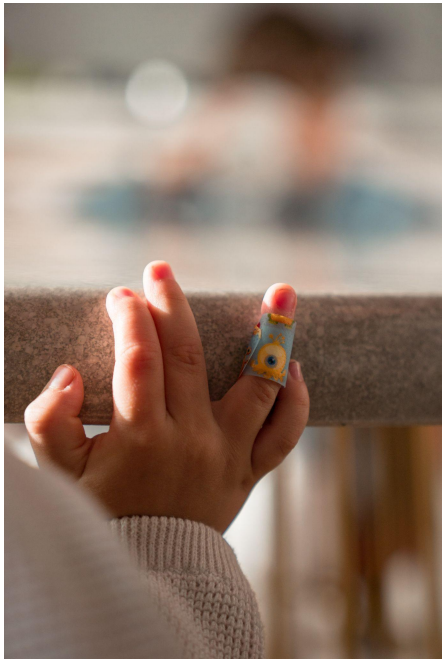
```
#include <memory>

int main() {
   int * c = new int(8);
   delete c;
}
```

```
#include <memory>

int main() {
   std::unique_ptr<int>c =
       std::make_unique<int>(8);
}
```

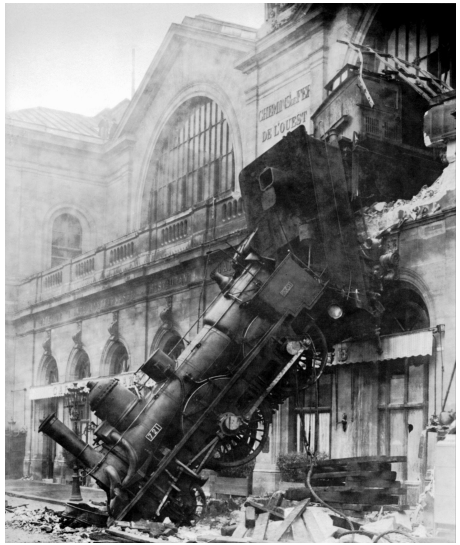
C++ 17 -> C++20

```
#include <string>
#include <string_view>

void foo(std::string_view a);

void foo(std::vector<const std::string> a);

int main() {
   foo({"hello", "world!"});
}
```

```
#include <string>
#include <string_view>

void foo(std::string_view a);

void foo(std::vector<const std::string> a);

int main() {
   foo(std::vector<const std::string>
                        {"hello", "world!"});
  }
```



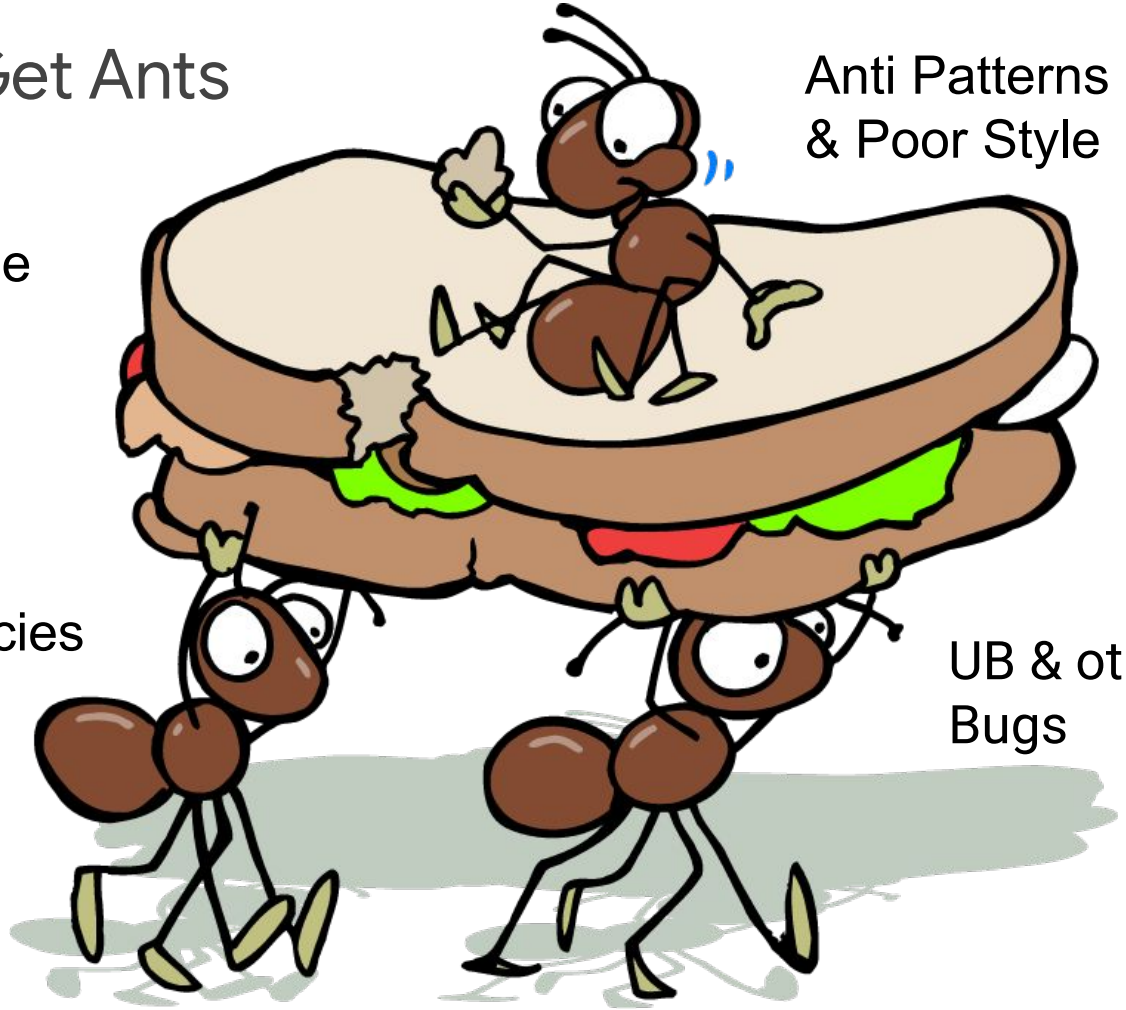
Saving You From Yourself



That's How You Get Ants

Anti Patterns
& Poor Style

Your Code Base



Inefficiencies

UB & other
Bugs

03

Clang Tidies!

What is a Clang Tidy

Existing Clang Tidies

Existing Clang Tidies

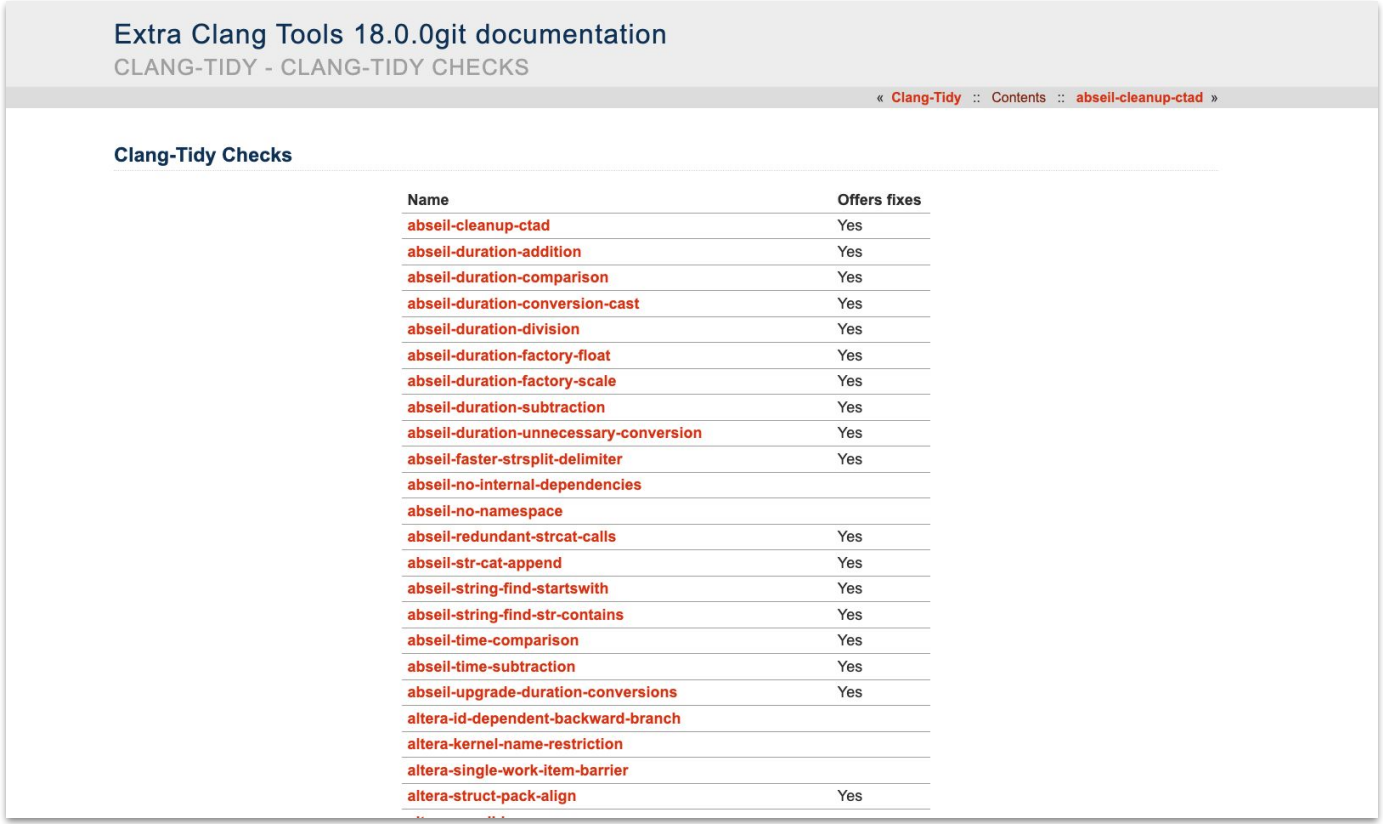
Existing Clang Tidies

Existing Clang Tidies

Existing Clang Tidies

Existing Clang Tidies

Existing Clang Tidies



Extra Clang Tools 18.0.0git documentation

CLANG-TIDY - CLANG-TIDY CHECKS

« Clang-Tidy :: Contents :: abseil-cleanup-ctad »

Clang-Tidy Checks

Name	Offers fixes
abseil-cleanup-ctad	Yes
abseil-duration-addition	Yes
abseil-duration-comparison	Yes
abseil-duration-conversion-cast	Yes
abseil-duration-division	Yes
abseil-duration-factory-float	Yes
abseil-duration-factory-scale	Yes
abseil-duration-subtraction	Yes
abseil-duration-unnecessary-conversion	Yes
abseil-faster-strsplit-delimiter	Yes
abseil-no-internal-dependencies	
abseil-no-namespace	
abseil-redundant-strcat-calls	Yes
abseil-str-cat-append	Yes
abseil-string-find-startswith	Yes
abseil-string-find-str-contains	Yes
abseil-time-comparison	Yes
abseil-time-subtraction	Yes
abseil-upgrade-duration-conversions	Yes
altera-id-dependent-backward-branch	
altera-kernel-name-restriction	
altera-single-work-item-barrier	
altera-struct-pack-align	Yes

I Want My Own!

Anatomy of a Clang Tidy

```
class AwesomeFunctionNamesCheck : public ClangTidyCheck {
public:
  AwesomeFunctionNamesCheck(StringRef Name, ClangTidyContext *Context)
      : ClangTidyCheck(Name, Context) {}
  void registerMatchers(ast_matchers::MatchFinder *Finder) override;
  void check(const ast_matchers::MatchFinder::MatchResult &Result) override;
};
```

Anatomy of a Clang Tidy

```
class AwesomeFunctionNamesCheck : public ClangTidyCheck {
public:
  AwesomeFunctionNamesCheck(StringRef Name, ClangTidyContext *Context)
      : ClangTidyCheck(Name, Context) {}
  void registerMatchers(ast_matchers::MatchFinder *Finder) override;
  void check(const ast_matchers::MatchFinder::MatchResult &Result) override;
};
```

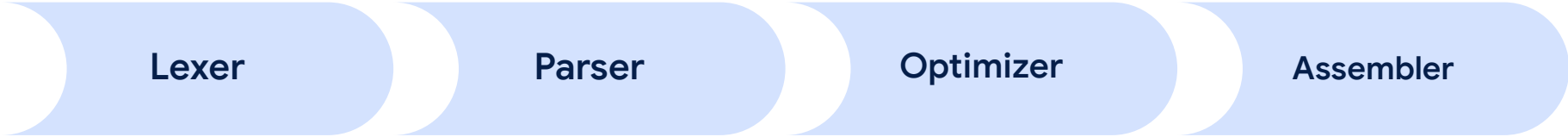
Anatomy of a Clang Tidy

```
class AwesomeFunctionNamesCheck : public ClangTidyCheck {
public:
  AwesomeFunctionNamesCheck(StringRef Name, ClangTidyContext *Context)
      : ClangTidyCheck(Name, Context) {}
  void registerMatchers(ast_matchers::MatchFinder *Finder) override;
  void check(const ast_matchers::MatchFinder::MatchResult &Result) override;
};
```

04

Compilation in C++

Compilation in C++



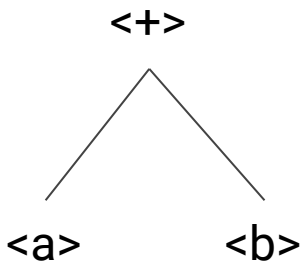
Front End

Back End

Code | Lexer | Parser

a + b

<a>, <+>, <b>



Let’s Just Use A Regular Expression

Go Ahead and Try It

Give Me Your Reg Ex for This

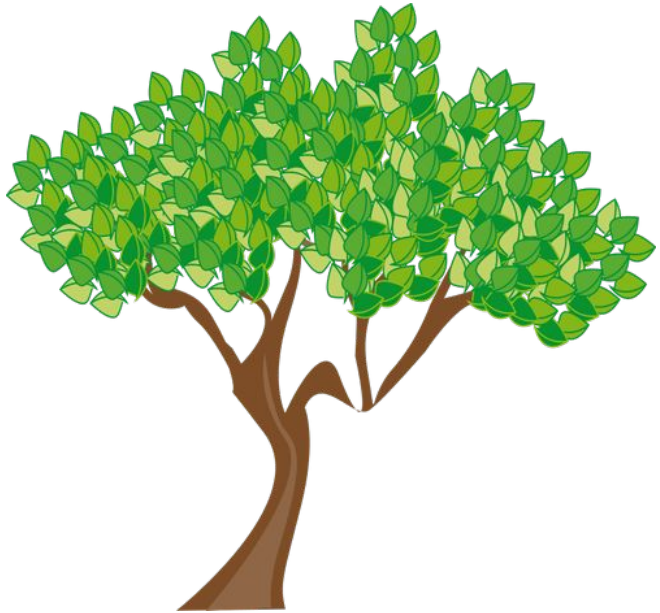
```
int main() {
    int x{};
    int z = {};
    int q = 0;
    auto w = int{0};
    auto u = int(0);
    auto f = 0 + 0;
    auto e = 0;
}
```

Or This

```
class A {
public:
    virtual void foo() = 0;
};

class B : public A {
public:
    virtual void foo() {}
}
```

Why Clang?

The Clang AST



The Clang AST

```
int main() {
  int a;
  int b;
  int c = a + b;
}
```

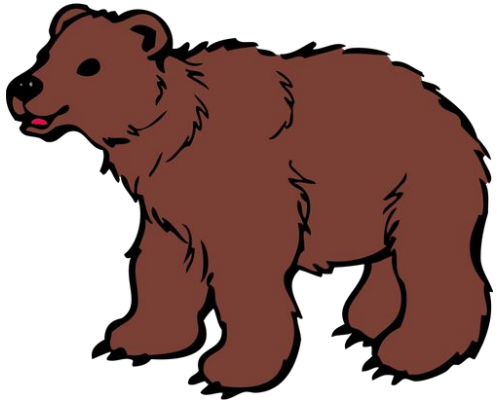
Field Trip!

https://bit.ly/cpp-con-a-plus-b

Oh My!

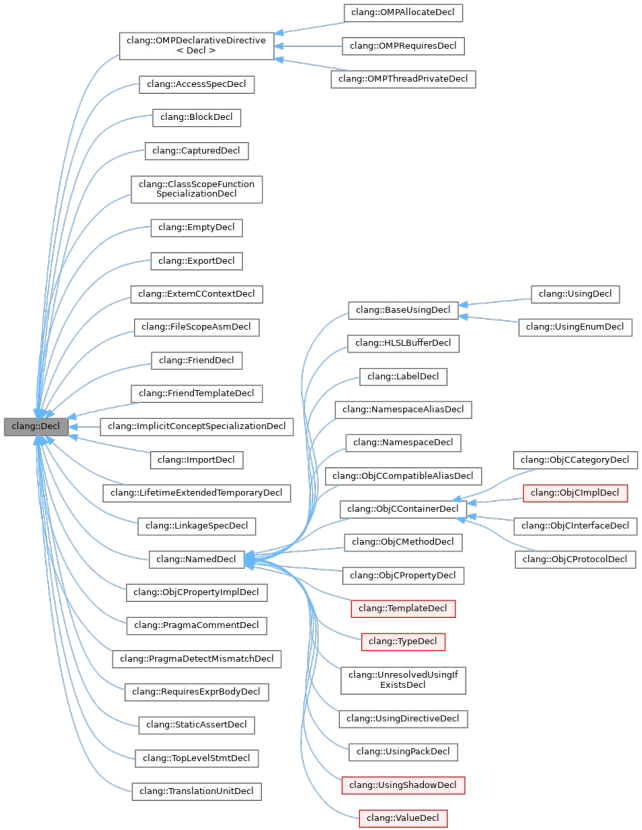
Decls

Stmts

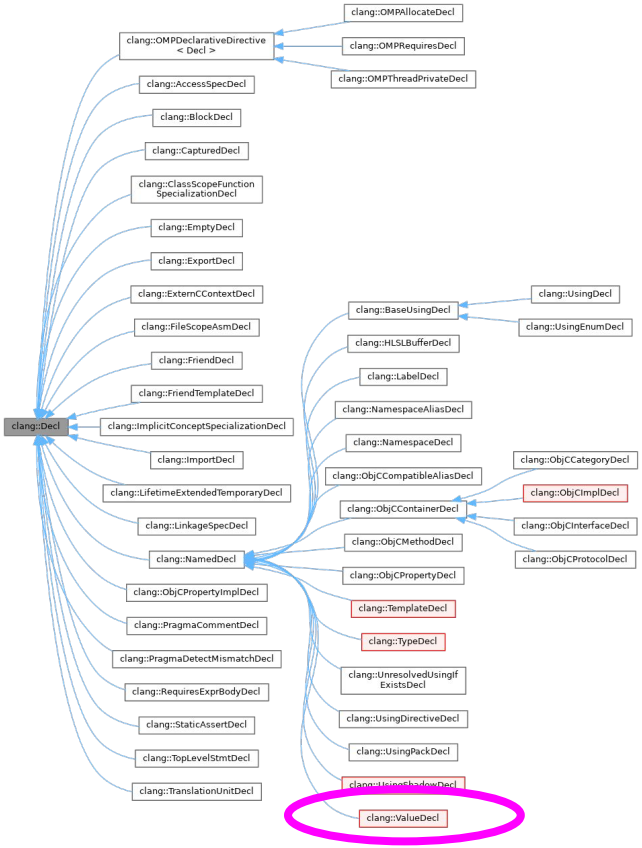


Types

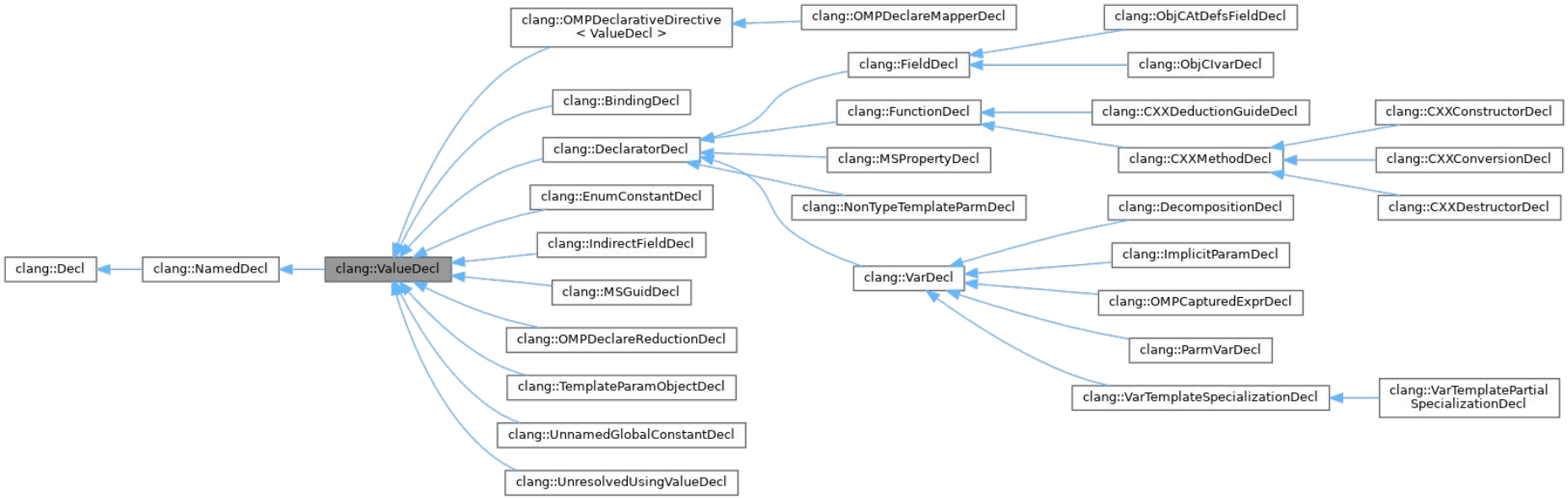
Decls


clang::Decl
clang::OMPDeclarativeDirective < Decl >
clang::OMPAllocateDecl
clang::OMPRequiresDecl
clang::OMPThreadPrivateDecl
clang::AccessSpecDecl
clang::BlockDecl
clang::CapturedDecl
clang::ClassScopeFunction SpecializationDecl
clang::EmptyDecl
clang::ExportDecl
clang::ExternCContextDecl
clang::FileScopeAsmDecl
clang::FriendDecl
clang::FriendTemplateDecl
clang::ImplicitConceptSpecializationDecl
clang::ImportDecl
clang::LifetimeExtendedTemporaryDecl
clang::LinkageSpecDecl
clang::NamedDecl
clang::ObjCPropertyImplDecl
clang::PragmaCommentDecl
clang::PragmaDetectMismatchDecl
clang::RequiresExprBodyDecl
clang::StaticAssertDecl
clang::TopLevelStmtDecl
clang::TranslationUnitDecl
clang::BaseUsingDecl
clang::UsingDecl
clang::UsingEnumDecl
clang::HLSLBufferDecl
clang::LabelDecl
clang::NamespaceAliasDecl
clang::NamespaceDecl
clang::ObjCCompatibleAliasDecl
clang::ObjCContainerDecl
clang::ObjCCategoryDecl
clang::ObjCImplDecl
clang::ObjCInterfaceDecl
clang::ObjCProtocolDecl
clang::ObjCMethodDecl
clang::ObjCPropertyDecl
clang::TemplateDecl
clang::TypeDecl
clang::UnresolvedUsingIf ExistsDecl
clang::UsingDirectiveDecl
clang::UsingPackDecl
clang::UsingShadowDecl
clang::ValueDecl

Decls


clang::OMPAllocateDecl
clang::OMPDeclarativeDirective < Decl >
clang::OMPRequiresDecl
clang::OMPThreadPrivateDecl
clang::AccessSpecDecl
clang::BlockDecl
clang::CapturedDecl
clang::ClassScopeFunction SpecializationDecl
clang::EmptyDecl
clang::ExportDecl
clang::ExternCContextDecl
clang::FileScopeAsmDecl
clang::FriendDecl
clang::FriendTemplateDecl
clang::Decl
clang::ImplicitConceptSpecializationDecl
clang::ImportDecl
clang::LifetimeExtendedTemporaryDecl
clang::LinkageSpecDecl
clang::NamedDecl
clang::ObjCPropertyImplDecl
clang::PragmaCommentDecl
clang::PragmaDetectMismatchDecl
clang::RequiresExprBodyDecl
clang::StaticAssertDecl
clang::TopLevelStmtDecl
clang::TranslationUnitDecl
clang::BaseUsingDecl
clang::UsingDecl
clang::UsingEnumDecl
clang::HLSLBufferDecl
clang::LabelDecl
clang::NamespaceAliasDecl
clang::NamespaceDecl
clang::ObjCCompatibleAliasDecl
clang::ObjCCategoryDecl
clang::ObjCContainerDecl
clang::ObjCImplDecl
clang::ObjCInterfaceDecl
clang::ObjCMethodDecl
clang::ObjCProtocolDecl
clang::ObjCPropertyDecl
clang::TemplateDecl
clang::TypeDecl
clang::UnresolvedUsingIf ExistsDecl
clang::UsingDirectiveDecl
clang::UsingPackDecl
clang::UsingShadowDecl
clang::ValueDecl

Decls


clang::Decl
clang::NamedDecl
clang::ValueDecl
clang::OMPDeclarativeDirective < ValueDecl >
clang::BindingDecl
clang::DeclaratorDecl
clang::EnumConstantDecl
clang::IndirectFieldDecl
clang::MSGuidDecl
clang::OMPDeclareReductionDecl
clang::TemplateParamObjectDecl
clang::UnnamedGlobalConstantDecl
clang::UnresolvedUsingValueDecl
clang::OMPDeclareMapperDecl
clang::FieldDecl
clang::FunctionDecl
clang::MSPropertyDecl
clang::NonTypeTemplateParmDecl
clang::VarDecl
clang::ObjCAtDefsFieldDecl
clang::ObjCIvarDecl
clang::CXXDeductionGuideDecl
clang::CXXMethodDecl
clang::DecompositionDecl
clang::ImplicitParamDecl
clang::OMPCapturedExprDecl
clang::ParmVarDecl
clang::VarTemplateSpecializationDecl
clang::CXXConstructorDecl
clang::CXXConversionDecl
clang::CXXDestructorDecl
clang::VarTemplatePartial SpecializationDecl

05

AST Matchers

AST Matchers - Three Flavors

AST Matchers - Node

clang::Decl ⇒ decl()

clang::NamedDecl ⇒ namedDecl()

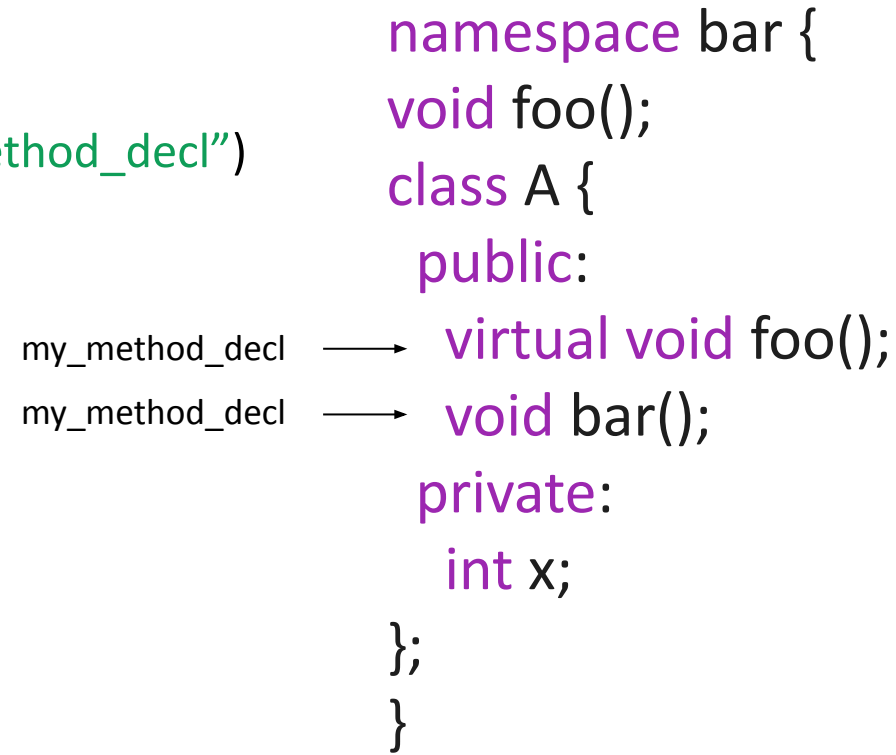
clang::FunctionDecl ⇒ functionDecl()

clang::CXXMemberDecl ⇒ cxxMemberDecl()

```
namespace bar {
void foo();
class A {
 public:
  virtual void foo();
  void bar();
 private:
  int x;
};
}
```

AST Matchers - Node

cxxMethodDecl().bind("my_method_decl")



AST Matchers - Narrowing

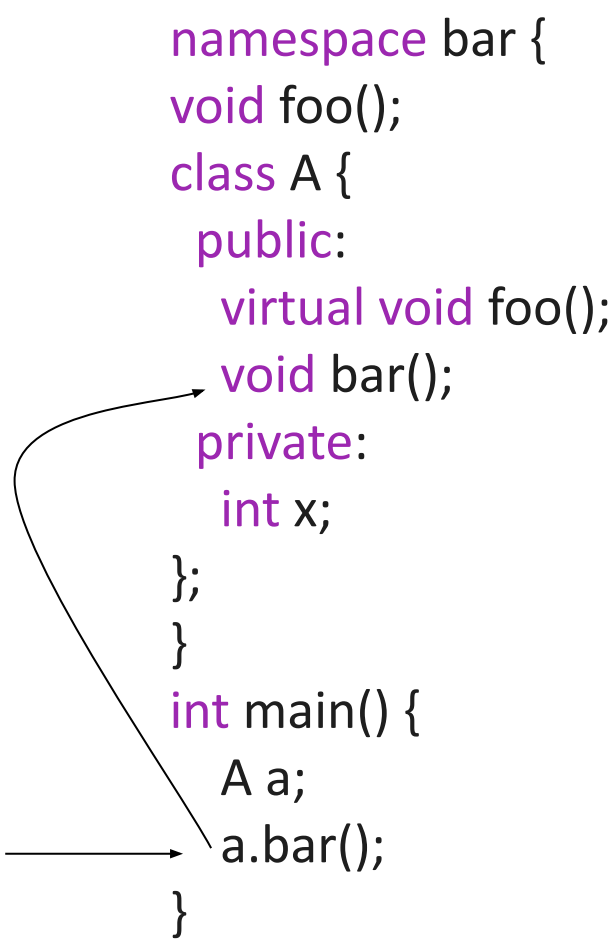
```
cxxMethodDecl(isVirtual()).
      bind("my_virtual_method_decl")
```

my_virtual_method_decl →

```
namespace bar {
void foo();
class A {
 public:
  virtual void foo();
   void bar();
 private:
   int x;
};
}
```

AST Matchers - Traverse

```
callExpr(callee(
      cxxMethodDecl(hasName("bar"))))
```



AST Matchers - Traverse

```
callExpr(callee(
      cxxMethodDecl(hasName("bar"))))
```

```
namespace bar {
void foo();
class A {
 public:
   virtual void foo();
   void bar();
 private:
   int x;
};
}
int main() {
   A a;
   a.bar();
}
```

AST Matchers - Traversal Methods

```
fieldDecl(
  hasType(asString("int")))
```

```
struct S {
  int m_i;
};

void func() {
  int a = 0;
  int c = 0;

  auto l = [a, b = c](int d) { int e = d; };
  l(43);
}
```

AST Matchers - Traversal Methods

```
fieldDecl(
  hasType(asString("int")))
```

```
struct S {
  safe_int m_i;
};

void func() {
  int a = 0;
  int c = 0;

  auto l = [safe_a, safe_b = c](int d) { int e = d; };
  l(43);
}
```

AST Matchers - Traversal Methods

```
struct S {
  int m_i;
};

void func() {
  int a = 0;
  int c = 0;

  auto l = [a, b = c](int d) { int e = d; };
  l(43);
}
```

```
traverse(
    TK_IgnoreUnlessSpelledInSource,
    fieldDecl(
        hasType(asString("int"))))
```

AST Matcher Reference

AST Matcher Reference

This document shows all currently implemented matchers. The matchers are grouped by category and node type they match. You can click on matcher names to show the matcher's source documentation.

There are three different basic categories of matchers:

- Node Matchers: Matchers that match a specific type of AST node.
- Narrowing Matchers: Matchers that match attributes on AST nodes.
- Traversal Matchers: Matchers that allow traversal between AST nodes.

Within each category the matchers are ordered by node type they match on. Note that if a matcher can match multiple node types, it will appear multiple times. This means that by searching for Matcher<Stmt> you can find all matchers that can be used to match on Stmt nodes.

The exception to that rule are matchers that can match on any node. Those are marked with a * and are listed in the beginning of each category.

Note that the categorization of matchers is a great help when you combine them into matcher expressions. You will usually want to form matcher expressions that read like english sentences by alternating between node matchers and narrowing or traversal matchers, like this:

```
recordDecl(hasDescendant(
    ifStmt(hasTrueExpression(
        expr(hasDescendant(
            ifStmt())))))
```

Traverse Mode

The default mode of operation of AST Matchers visits all nodes in the AST, even if they are not spelled in the source. This is `AsIs` mode. This mode requires writing AST matchers that explicitly traverse or ignore implicit nodes, such as parentheses surrounding an expression or expressions with cleanups. These implicit nodes are not always obvious from the syntax of the source code, and so this mode requires careful consideration and testing to get the desired behavior from an AST matcher.

In addition, because template instantiations are matched in the default mode, transformations can be accidentally made to template declarations. Finally, because implicit nodes are matched by default, transformations can be made on entirely incorrect places in the code.

For these reasons, it is possible to ignore AST nodes which are not spelled in the source using the `IgnoreUnlessSpelledInSource` mode. This is likely to be far less error-prone for users who are not already very familiar with where implicit nodes appear in the AST. It is also likely to be less error-prone for experienced AST users, as difficult cases do not need to be encountered and matcher expressions adjusted for these cases.

In clang-query, the mode can be changed with

```
set traversal IgnoreUnlessSpelledInSource
```

AST Matchers - Matcher Expressions

Looking for calls to a member function on a class named A and whose called function is named foo and whose first parameter is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
            hasName("foo"),
            hasParameter(0, hasType(isInteger())))))
```

AST Matchers - Matcher Expressions

Looking for **calls to a member function** on a class named A and whose called function is named foo and whose first parameter is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
            hasName("foo"),
            hasParameter(0, hasType(isInteger())))))
```

AST Matchers - Matcher Expressions

Looking for calls to a member function **on** a class named A and whose called function is named foo and whose first parameter is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
            hasName("foo"),
            hasParameter(0, hasType(isInteger())))))
```

AST Matchers - Matcher Expressions

Looking for calls to a member function on **a class** named A and whose called function is named foo and whose first parameter is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
        hasName("foo"),
        hasParameter(0, hasType(isInteger())))))
```

AST Matchers - Matcher Expressions

Looking for calls to a member function on a class **named A** and whose called function is named foo and whose first parameter is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
            hasName("foo"),
            hasParameter(0, hasType(isInteger())))))
```

AST Matchers - Matcher Expressions

Looking for calls to a member function on a class named A **and** whose called function is named foo and whose first parameter is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
        hasName("foo"),
        hasParameter(0, hasType(isInteger())))))
```

AST Matchers - Matcher Expressions

Looking for calls to a member function on a class named A and whose **called function** is named foo and whose first parameter is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
        hasName("foo"),
        hasParameter(0, hasType(isInteger())))))
```

AST Matchers - Matcher Expressions

Looking for calls to a member function on a class named A and whose called function is **named foo** and whose first parameter is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
        hasName("foo"),
        hasParameter(0, hasType(isInteger())))))
```

AST Matchers - Matcher Expressions

Looking for calls to a member function on a class named A and whose called function is named foo **and** whose first parameter is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
            hasName("foo"),
            hasParameter(0, hasType(isInteger()))))
```

AST Matchers - Matcher Expressions

Looking for calls to a member function on a class named A and whose called function is named foo and whose **first parameter** is an integer.

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
        hasName("foo"),
        hasParameter(0, hasType(isInteger())))))
```

AST Matchers - Matcher Expressions

Looking for calls to a member function on a class named A and whose called function is named foo and whose first parameter **is an integer.**

```
cxxMemberCallExpr(
    on(hasType(cxxRecordDecl(hasName("A")))),
    callee(functionDecl(
        hasName("foo"),
        hasParameter(0, hasType(isInteger()))))
```

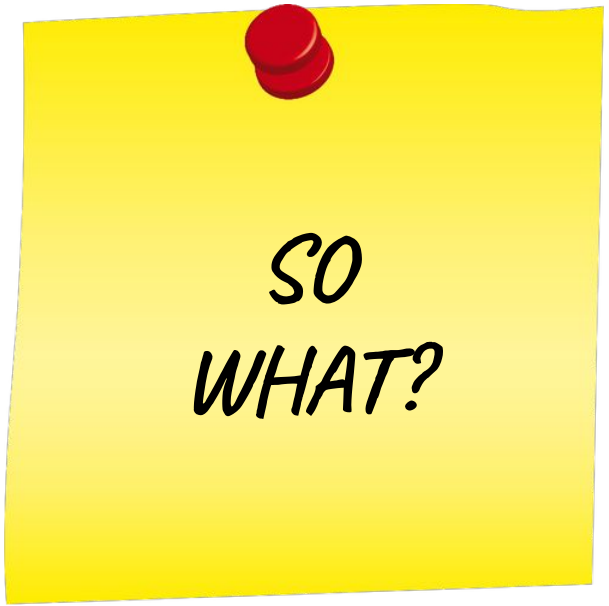
Field Trip!

bit.ly/cpp-con-ast-matchers

AST Matchers - RegisterMatchers

```
void registerMatchers(ast_matchers::MatchFinder *Finder) {
    finder->AddMatcher(
        traverse(TK_IgnoreUnlessSpelledInSource,
        cxxMemberCallExpr(
            on(hasType(cxxRecordDecl(hasName("A")))),
            callee(functionDecl(
                hasName("foo"),
                hasParameter(0, hasType(isInteger()))))), this)
}
```

So you have a node



So you have a node

06

Refactoring

The Bound Nodes

```
cxxMemberCallExpr(stmt().bind(“call_expr”),
        on(hasType(cxxRecordDecl(decl().bind(“class_a”), hasName(“A”)))),
        callee(functionDecl(decl().bind(“foo_decl”),
                        hasName(“foo”),
                        hasParameter(0, hasType(isInteger())))))
```

```
void check(const ast_matchers::MatchFinder::MatchResult &Result) {
      const auto * call_expr = Result.Nodes.getNodeAs<clang::CallExpr>("call_expr");
      const auto * class_decl = Result.Nodes.getNodeAs<clang::CXXRecordDecl>("class_a");
      const auto * function_decl = Result.Nodes.GetNodeAs<clang::FunctionDecl>("foo_decl");
…
}
```

Clang Diagnostics & Fix-It Hints

```
diag(function_decl->getLocation(), "Your function is insufficiently awesome")
      << FixItHint::CreateInsertion(function_decl->getLocation(), "awesome_");
```

07

Writing A New Clang Tidy Check

Writing Our Own Tidy Check

```
struct Expensive {
        int array[128];
};

int main() {

        Expensive arr[42] = {};
        for (const auto i: arr){
                // analyze i
        }
}
```

Writing Our Own Tidy Check

```
struct Expensive {
        int array[128];
};

int main() {

        Expensive arr[42] = {};
        for (const auto i: arr){
                // analyze i
        }
}
```

Writing Our Own Tidy Check

```
struct Expensive {
        int array[128];
};

int main() {

        Expensive arr[42] = {};
        for (const auto& i: arr){
                // analyze i
        }
}
```

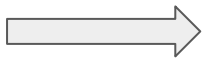
Really Cool Stuff You Can Build

```
ABSL_DEPRECATE_AND_INLINE()
void OldAPI(int v, int m, int d){
   NewAPI(m, v);
}

void NewAPI(int m, int v);
```

```
-    OldAPI(vel, mass, dist);
+    NewAPI(mass, vel);
```

Really Cool Stuff You Can Build

```
if (absl::GetFlag(FLAGS_Example_for_cpp_con)) {
   foo();
}
else {
   bar();
}
```



```
foo();
```

07

In Conclusion

Special Thanks

People Who Taught Me About Clang

Samuel Benzaquen
Matthew Kulukundis
Andy Soffer
Eric Li
Yitzhak Mandelbaum
Samira Bazuzi

People Who Told Me To Give This Talk Repeatedly Over and Over Again Until I Finally Wrote It

Peter Muldoon
Daisy Hollman
David Sankel
Bret Brown
Margot Leibold

07

Questions