

Compilers 101

Compiler frontend.

Working with AST and static analysis

Previously...

Preprocessing

Lexical analysis

Syntax analysis

Semantic analysis

IR Generation

IR Optimization

Optimization

Code generation

Frontend

Middle-end

Backend

Today

- Static analysis and its use cases
- Control-flow graphs
- Practical usage of Clang framework

What is static analysis?

- Static analysis is the analysis of software without executing any program code

Why static analysis?

- Reduce number of bugs
 - Some bugs are hard to find, e.g., = vs ==
- Legal obligations
 - Safety critical code must pass certifications
- Code Quality Improvement, efficiency and automation
- Education
- Basically, any rule enforcement

Warnings

- The most basic kind of static analysis
 - Missing return statement, missing case in switch, use of deprecated functions
- Built into compiler
- BKM: always enable `-Wall` (`-Wextra`) `-Werror` flags (or analogs) for your project

Linters

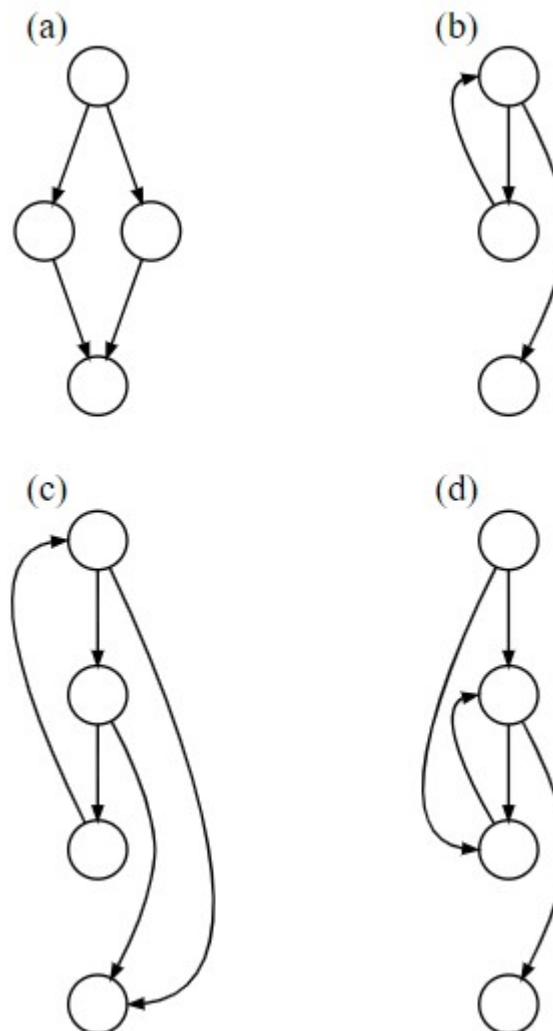
- Checks that your code follows some rules
 - Google Code Style rules
 - C++ core guidelines
- Those checks are not necessarily related to program functionality
 - Often highly opinionated

Static analyzers

- The most advanced static analyzers can perform data flow analysis and provide correctness checks
- Examples:
 - clang static analyzer
 - cppcheck
 - PVS Studio

Control-flow graphs

- A control-flow graph is a representation of all paths that might be traversed through a program during execution
- Source: https://en.wikipedia.org/wiki/Control-flow_graph



CFG demo

```
#include <stdio.h>

int main() {
    int x = 0;
    scanf("%d", &x);

    if (x > 0) {
        printf("x is positive\n");
    } else {
        printf("x is non-positive\n");
    }

    for (int i = 0; i < x; i++) {
        printf("%d ", i);
    }

    return 0;
}
```



clang -Xclang -analyze -Xclang -analyzer-checker=debug.ViewCFG -Xclang -analyzer-output=text cfg.cpp

Dominators

- A block M dominates block N if every path from the entry that reaches block N must pass through block M
- Block M postdominates block N if every path from N to exit must pass through M

Dominator trees

- Data structure depicting the dominator relationships
- Lengauer-Tarjan algorithm ($O(n \log n)$)
- See more:

https://www.llvm.org/devmtg/2017-10/slides/Kuderski-Dominator_Trees.pdf

Dominator tree (example)

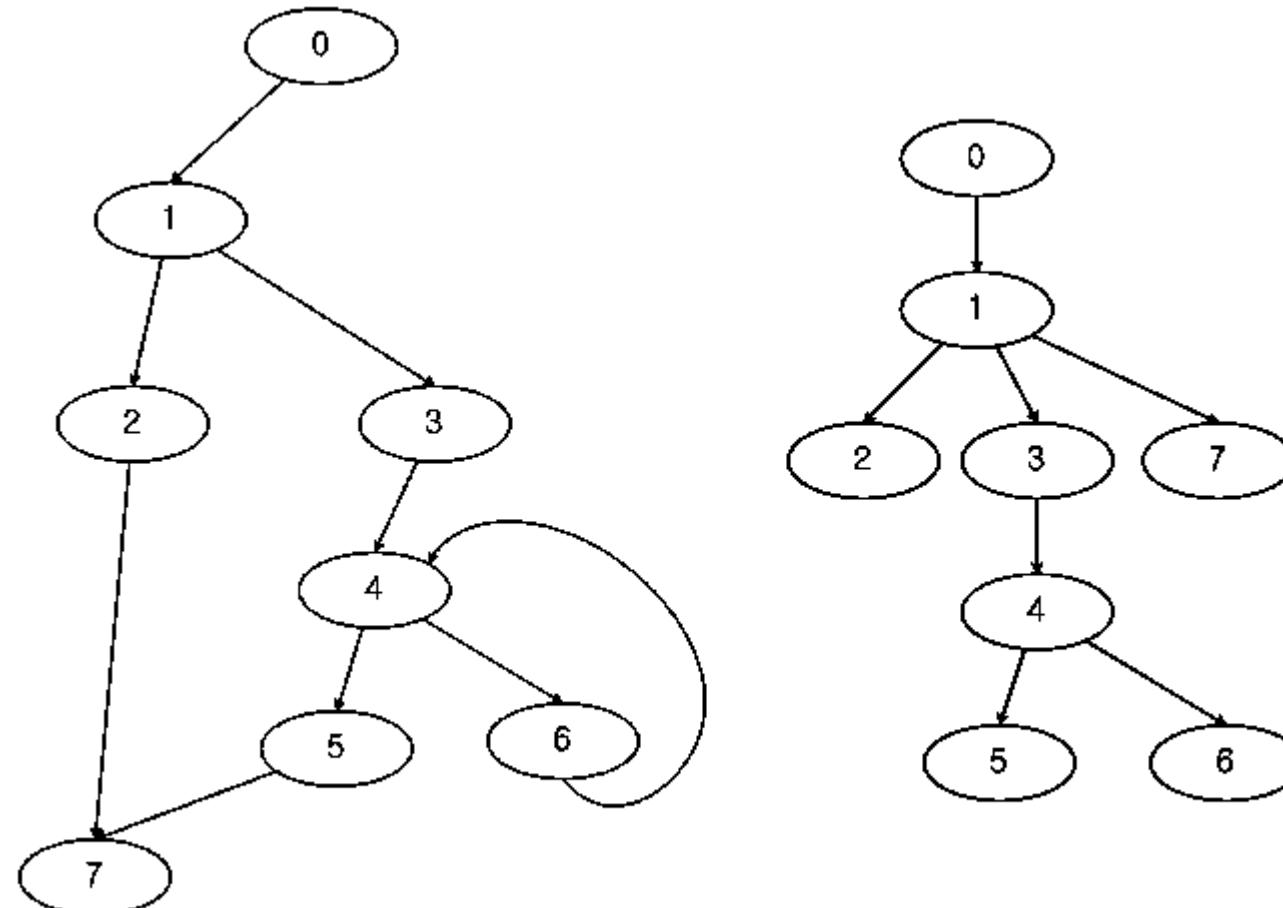


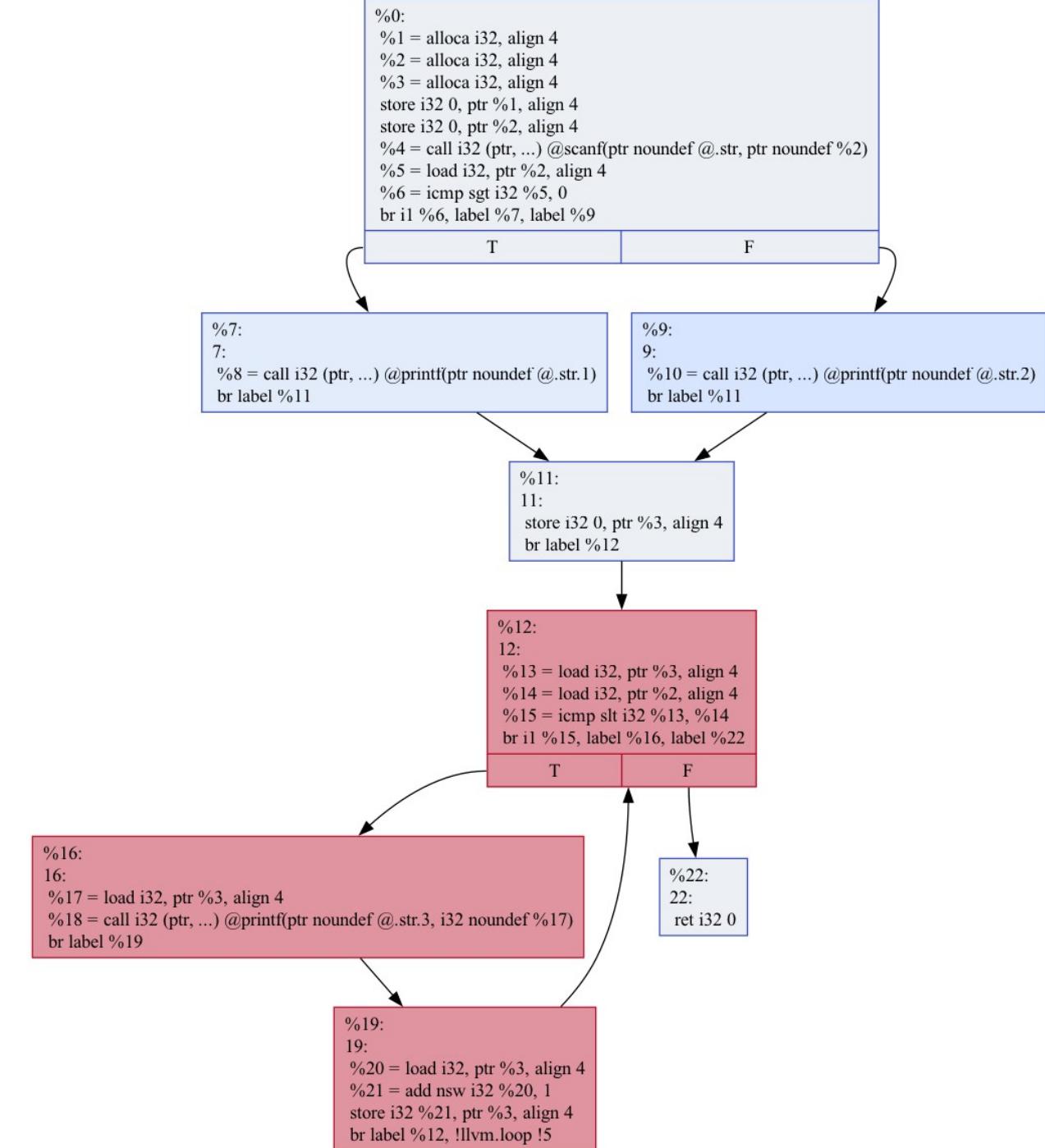
Figure 1: An example graph and its dominator tree.

Source: https://www.boost.org/doc/libs/1_85_0/libs/graph/doc/lengauer_tarjan_dominator.htm

Dominator trees (demo)

How to build:

```
clang -S -emit-llvm dominator.cpp -o dominator.ll
opt -passes='dot-cfg' dominator.ll
dot -Tpng .main.dot -o main.png
```



Hacking on Clang

- LibTooling – interface to write standalone tools based on clang
- LibFormat – library for automatic source code formatting
- LibClang – stable C interface for Clang
- Clang Plugins – do extra actions during compilation
- More on that:

<https://clang.llvm.org/docs/index.html#using-clang-as-a-library>

Working with AST

- AST is a graph, and you can use any graph theory algorithms
- Examining AST: -Xclang -ast-dump -fsyntax-only
- Clang framework provides a few helpers to work with AST
 - ASTConsumer
 - RecursiveASTVisitor
 - ASTAction
- More on that: <https://www.youtube.com/watch?v=VqCkCDFLSsc>

RecursiveASTVisitor

- CRTP class
- Provides methods to act on certain AST node classes
- Basically, an implementation of DFS

```
class FindNamedClassVisitor
    : public RecursiveASTVisitor<FindNamedClassVisitor> {
public:
    explicit FindNamedClassVisitor(ASTContext *Context)
        : Context(Context) {}

    bool VisitCXXRecordDecl(CXXRecordDecl *Declaration) {
        if (Declaration->getQualifiedNameAsString() == "n::m::C") {
            FullSourceLoc FullLocation = Context->getFullLoc(Declaration->getBeginLoc());
            if (FullLocation.isValid())
                llvm::outs() << "Found declaration at "
                << FullLocation.getSpellingLineNumber() << ":"
                << FullLocation.getSpellingColumnNumber() << "\n";
        }
        return true;
    }

private:
    ASTContext *Context;
};
```

LLVM IR generation

Typical use case for AST node visitors. Also can be referred as CodeGen, but do not confuse with LLVM CodeGen

- Uses AST visitors, IRBuilder, and TargetInfo classes
- 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.
- The output can be inspected using the `-emit-llvm` (to force `-S` for textual representation) option to clang. You can also use `-emit-llvm-bc` to write an LLVM bitcode file which can be processed by the suite of LLVM tools like `llvm-dis`, `llvm-nm`, etc.

ASTConsumer

- An interface used to write generic actions on AST
- Provides many different entry points

```
class FindNamedClassConsumer : public clang::ASTConsumer {  
public:  
    virtual void HandleTranslationUnit(clang::ASTContext &Context) {  
        // Traversing the translation unit decl via a RecursiveASTVisitor  
        // will visit all nodes in the AST.  
        Visitor.TraverseDecl(Context.getTranslationUnitDecl());  
    }  
private:  
    // A RecursiveASTVisitor implementation.  
    FindNamedClassVisitor Visitor;  
};
```

FrontendAction

- FrontendActions are entry points for Clang-based tools (including Clang compiler)

```
class FindNamedClassAction : public clang::ASTFrontendAction {  
public:  
    virtual std::unique_ptr<clang::ASTConsumer> CreateASTConsumer(  
        clang::CompilerInstance &Compiler, llvm::StringRef InFile) {  
        return std::make_unique<FindNamedClassConsumer>(&Compiler.getASTContext());  
    }  
};
```

Source manager

- Source location
file:line:col
- AST does not contain information on source locations
 - It may be required for code re-writing
- SourceManager is owned by ASTContext:

```
FullSourceLoc FullLocation = Context->getFullLoc(Declaration->getBeginLoc());
```
- More info: <https://clang.llvm.org/docs/RAVFrontendAction.html>

Matching AST

- Sometimes there's a need to find certain patterns in AST
- Clang provides matchers interface:
 - Node matchers – find specific node type
 - Narrowing matchers – match attributes on AST nodes
 - Traversal matchers – traversal between AST nodes
- More info on syntax:

<https://clang.llvm.org/docs/LibASTMatchersReference.html#decl-matchers>

Node matchers	Narrowing matchers	Traversal matchers
match a specific type	match attributes	traversal between nodes
<ul style="list-style-type: none">- varDecl- callExpr- Expr- forStmt- functionDecl- ...	<ul style="list-style-type: none">- equals- isArray- hasSize- hasName- isIntegral- ...	<ul style="list-style-type: none">- has- hasParent- hasCondition- hasLHS- hasAnyParameter- ...

Clang Transformer

- Framework for writing C++ diagnostics and source transformations
- Built on Matchers interface and LibTooling
- Rule examples:

```
makeRule(functionDecl(hasName("MkX")).bind("fun"),
         noopEdit(node("fun")),
         cat("The name ``MkX`` is not allowed for functions; please rename"));
```

```
makeRule(declRefExpr(to(functionDecl(hasName("MkX")))),
           changeTo(cat("MakeX")),
           cat("MkX has been renamed MakeX));
```

clang-tidy

- Clang-based linter tool
- Provides rules for many standards, including:
 - C++ core guidelines
 - Abseil
 - Google
 - CERT
 - Linux Kernel
 - LLVM

Extending clang-tidy

- Most useful source of knowledge: look at existing code
- Example (
<https://github.com/llvm/llvm-project/blob/main/clang-tools-extra/clang-tidy/llvm/>):

```
/// Looks for local `Twine` variables which are prone to use after frees and
/// should be generally avoided.
class TwineLocalCheck : public ClangTidyCheck {
public:
    TwineLocalCheck(StringRef Name, ClangTidyContext *Context)
        : ClangTidyCheck(Name, Context) {}
    void registerMatchers(ast_matchers::MatchFinder *Finder) override;
    void check(const ast_matchers::MatchFinder::MatchResult &Result) override;
};
```

```
void TwineLocalCheck::registerMatchers(MatchFinder *Finder) {
    auto TwineType =
        qualType(hasDeclaration(cxxRecordDecl(hasName("::llvm::Twine"))));
    Finder->addMatcher(
        varDecl(unless(parmVarDecl()), hasType(TwineType)).bind("variable"),
        this);
}
```

Case study: modularize

- modularize is a standalone tool that checks whether a set of headers provides the consistent definitions required to use modules

```
(...)/SubHeader.h:11:5:  
#if SYMBOL == 1  
^  
error: Macro instance 'SYMBOL' has different values in this header,  
depending on how it was included.  
'SYMBOL' expanded to: '1' with respect to these inclusion paths:  
(...)/Header1.h  
(...)/SubHeader.h  
(...)/SubHeader.h:3:9:  
#define SYMBOL 1  
^  
Macro defined here.  
'SYMBOL' expanded to: '2' with respect to these inclusion paths:  
(...)/Header2.h  
(...)/SubHeader.h  
(...)/SubHeader.h:7:9:  
#define SYMBOL 2  
^  
Macro defined here.
```

Case study: C++ core guidelines

- C++ Core Guidelines provide helper types for memory management
 - `gsl::owner<...>`
 - `gsl::not_null<...>`
- clang-tidy will warn if you're not using any of these and to suspicious things
 - <https://clang.llvm.org/extra/clang-tidy/checks/cppcoreguidelines/owning-memory.html>

Hands-on

<https://github.com/NN-complir-tech/compiler-course-2025>

The screenshot shows the GitHub repository page for `llvm-nnsu-2024`. The repository is public and forked from `NN-complir-tech/llvm`. The main interface includes a navigation bar with links for Code, Pull requests, Actions, Projects, Security, Insights, and Settings. The repository name is displayed prominently at the top left, along with a fork icon and the original repository link. A search bar is located at the top right. Below the header, there's a summary section showing 1 branch and 0 tags, and a message indicating the branch is up-to-date with the original repository. The central area displays a list of commits from various contributors, such as `aobolensk`, `7623aab`, and `469,448 Commits`. The commits are organized by folder, with details like commit message, author, date, and file changes. To the right of the commit list, there's an "About" section providing information about the LLVM fork, including links to `llvm.org`, a Readme, View license, Activity, Custom properties, and reporting issues. There are also sections for Releases and Packages, both of which currently have no content.

Code Pull requests Actions Projects Security Insights Settings

llvm-nnsu-2024 Public
forked from [NN-complir-tech/llvm](#)

course-spring-2... 1 Branch 0 Tags Go to file Add file Code About

This branch is up to date with [NN-complir-tech/llvm:course-spring-2024](#).

aobolensk Merge pull request [NN-complir-tech#3](#) from NN-complir-tech/dev/... · 7623aab · last week · 469,448 Commits

Commit	Message	Date
<code>.ci</code>	[ci] Make libc++ and Clang CI scripts independent	7 months ago
<code>.github/workflows</code>	win	last week
<code>bolt</code>	[BOLT][Instrumentation] Add test for append-pid option	6 months ago
<code>clang-tools-extra</code>	[clangd] Avoid null result in FindRecordTypeAt()	3 months ago
<code>clang</code>	[Driver] Enable __float128 support on X86 on FreeBSD / ...	3 months ago
<code>cmake</code>	[CMake] Switch the CMP0091 policy (MSVC_RUNTIME_LI...)	7 months ago
<code>compiler-rt</code>	eliminate python SyntaxWarnings from check-all output.	3 months ago
<code>cross-project-tests</code>	[Dexter] XFAIL Dexter tests for Apple Silicon (arm64)	9 months ago
<code>flang</code>	[flang] Add comdat to functions with linkonce linkage (l...)	4 months ago
<code>libc</code>	[libc] Remove test RPC opcodes from the exported header	7 months ago
<code>libclc</code>	Reland "[CMake] Bumps minimum version to 3.20.0."	9 months ago
<code>libcxx</code>	eliminate python SyntaxWarnings from check-all output.	3 months ago
<code>libcxxabi</code>	[libc++] Fix problems with GCC 13 and switch to it in the CI	6 months ago
<code>libunwind</code>	[libc++][libunwind] Fixes to allow GCC 13 to compile libunwind	6 months ago

LLVM fork for compiler course (spring 2024)

[llvm.org](#)

Readme View license Activity Custom properties 0 stars 0 watching 0 forks Report repository

Releases No releases published [Create a new release](#)

Packages No packages published [Publish your first package](#)

Lab assignment #1

- Write a simple Clang AST plugin
- Task list in Google Docs
- Deadline: March, 19
- Where to seek help
 - <https://clang.llvm.org/docs/ClangPlugins.html>
 - <https://github.com/llvm/llvm-project/tree/main/clang/examples/PrintFunctionNames>
- Contact teachers

Acceptance criterias

- The code base with implementation passes clang-format/clang-tidy checks on CI
- The code base with implementation passes build and test stage on CI
- LIT tests are added and passed on CI

FileCheck

- Tool from clang toolchain
- It is widely used for testing in LLVM framework

- How to run FileCheck tool:
 - FileCheck %s - compare stdin with file %s
- string to be verified:
 - CHECK: content
- Manual: <https://llvm.org/docs/CommandGuide/FileCheck.html>
- Run LIT test:
 - llvm-lit path/to/test

llvm-lit

lit is a portable tool for executing LLVM and Clang style test suites, summarizing their results, and providing indication of failures. lit is designed to be a lightweight testing tool with as simple a user interface as possible.

Manual: <https://llvm.org/docs/CommandGuide/lit.html>

llvm-lit simple test demo

```
clang > test > CodeGen > _demo > C++ 1.cpp > ...
1 // RUN: %clangxx %s -emit-llvm -c -S -o - - | FileCheck %s
2 #include <iostream>
3
4
5 // CHECK: define noundef i32 @main() #0
6 int main() {
7     return 0;
8 }
```

```
$PATH_TO_LLVM_BIN/llvm-lit -a clang/test/_demo/1.cpp
```

Next time...

- Intermediate representations
- LLVM IR
- IR generation

Test

<https://forms.gle/KedxuyGUAkYiMmYf6>

Для каких целей используется построение графов (control flow graph и dominator tree) в clang?

Your answer

Зачем нужен статический анализ? Где он используется?

Your answer

Как статический анализ связан с компилятором?

Your answer



Extra materials

- Data flow analysis - <https://www.youtube.com/watch?v=OROXJ9-wUQE>
- AST Matchers:
 - <https://github.com/lac-dcc/llvm-course/tree/master/ast-matcher>
 - <https://www.youtube.com/watch?v=mizS4KnfTtQ>
- A. Homescu “Mutating the clang AST from Plugins”: https://www.youtube.com/watch?v=_rUwW8Awc5s

