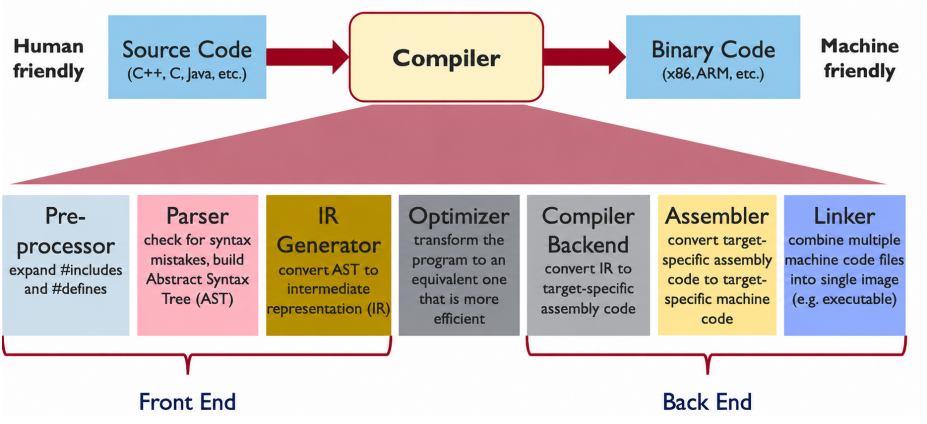
L6: Program Representation

Software Analysis

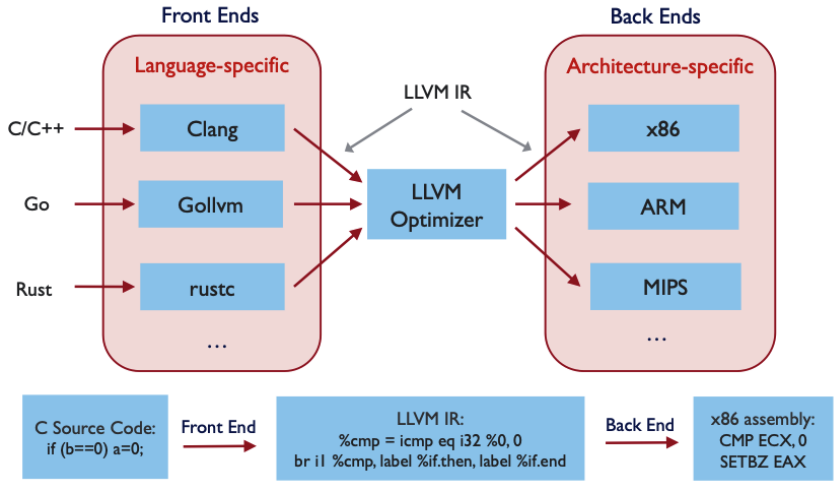
IIT Guwahati · August 2026

What is LLVM?

LLVM is a modular and reusable compiler framework supporting multiple front-ends and back-ends.



LLVM Compiler Architecture



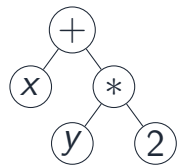
What is an Abstract Syntax Tree?

An **Abstract Syntax Tree (AST)** is a tree representation of a program that captures its **essential syntactic structure** while omitting unnecessary syntactic details.

Example

```
x + y * 2
```

$\Longrightarrow$



Source: Clang — Introduction to the Clang AST

The Clang AST

Clang AST

Clang turns C/C++ source code into a rich AST

- **Inspect** — declarations, expressions, types, calls
- **Detect** — bugs, patterns, coding violations
- **Transform** — refactoring, instrumentation, assertions

Example: Automatically Instrument Branches

Original Program

```
if (x > 0) {
    foo();
} else {
    bar();
}
```

Instrumented Program

```
if (x > 0) {
    printf("Branch: 1\n");
    foo();
} else {
    printf("Branch: 2\n");
    bar();
}
```

Find `IfStmt` $\rightarrow$ insert instrumentation $\rightarrow$ modified program

Inspecting the AST

`clang -ast-dump`

The running example, `test.cpp`:

```
int f(int x) {
    int result = x + 42;
    return result;
}
```

Inspecting the AST

clang -ast-dump

The running example, `test.cpp`:

```
int f(int x) {
    int result = x + 42;
    return result;
}
```

Clang provides a built-in AST dump, which is the documented means of inspecting the tree from the command line:

```
clang++ -Xclang -ast-dump -fsyntax-only test.cpp
```

Clang, *Introduction to the Clang AST* · verified with Apple clang 17.0.0 (arm64), exit status 0

Inspecting the AST

clang -ast-dump

The running example, `test.cpp`:

```
int f(int x) {
    int result = x + 42;
    return result;
}
```

Clang provides a built-in AST dump, which is the documented means of inspecting the tree from the command line:

```
clang++ -Xclang -ast-dump -fsyntax-only test.cpp
```

`-fsyntax-only` terminates compilation after semantic analysis, so **no code is generated**.

Clang, *Introduction to the Clang AST* · verified with Apple clang 17.0.0 (arm64), exit status 0

Inspecting the AST

clang -ast-dump

The running example, `test.cpp`:

```
int f(int x) {
    int result = x + 42;
    return result;
}
```

Clang provides a built-in AST dump, which is the documented means of inspecting the tree from the command line:

```
clang++ -Xclang -ast-dump -fsyntax-only test.cpp
```

`-fsyntax-only` terminates compilation after semantic analysis, so **no code is generated**.

Clang, *Introduction to the Clang AST* · verified with Apple clang 17.0.0 (arm64), exit status 0

Structure of an AST Dump

Output:

```
‘-FunctionDecl 0x13c83d108 <test.cpp:1:1, line:4:1> line:1:5 f ’int (int)’
  |-ParmVarDecl 0x13c83d030 <col:7, col:11> col:11 used x ’int’
  ‘-CompoundStmt 0x13c83d358 <col:14, line:4:1>
    |-DeclStmt 0x13c83d2f8 <line:2:5, col:24>
    | ‘-VarDecl 0x13c83d218 <col:5, col:22> col:9 used result ’int’ cinit
    | ‘-BinaryOperator 0x13c83d2d8 <col:18, col:22> ’int’ ’+’
    | |-ImplicitCastExpr 0x13c83d2c0 <col:18> ’int’ <LValueToRValue>
    | | ‘-DeclRefExpr 0x13c83d280 <col:18> ’int’ lvalue ParmVar ’x’ ’int’
    | ‘-IntegerLiteral 0x13c83d2a0 <col:22> ’int’ 42
    ‘-ReturnStmt 0x13c83d348 <line:3:5, col:12>
      ‘-ImplicitCastExpr 0x13c83d330 <col:12> ’int’ <LValueToRValue>
        ‘-DeclRefExpr 0x13c83d310 <col:12> ’int’ lvalue Var ’result’ ’int’
```

Structure of an AST Dump

Output:

```
‘-FunctionDecl 0x13c83d108 <test.cpp:1:1, line:4:1> line:1:5 f ’int (int)’
  |-ParmVarDecl 0x13c83d030 <col:7, col:11> col:11 used x ’int’
  ‘-CompoundStmt 0x13c83d358 <col:14, line:4:1>
    |-DeclStmt 0x13c83d2f8 <line:2:5, col:24>
    | ‘-VarDecl 0x13c83d218 <col:5, col:22> col:9 used result ’int’ cinit
    | ‘-BinaryOperator 0x13c83d2d8 <col:18, col:22> ’int’ ’+’
    | |-ImplicitCastExpr 0x13c83d2c0 <col:18> ’int’ <LValueToRValue>
    | | ‘-DeclRefExpr 0x13c83d280 <col:18> ’int’ lvalue ParmVar ’x’ ’int’
    | ‘-IntegerLiteral 0x13c83d2a0 <col:22> ’int’ 42
    ‘-ReturnStmt 0x13c83d348 <line:3:5, col:12>
      ‘-ImplicitCastExpr 0x13c83d330 <col:12> ’int’ <LValueToRValue>
        ‘-DeclRefExpr 0x13c83d310 <col:12> ’int’ lvalue Var ’result’ ’int’
```

Each line gives a **node kind**, an address, a **source range** and a **type**.

Structure of an AST Dump

Output:

```
‘-FunctionDecl 0x13c83d108 <test.cpp:1:1, line:4:1> line:1:5 f ’int (int)’
  |-ParmVarDecl 0x13c83d030 <col:7, col:11> col:11 used x ’int’
  ‘-CompoundStmt 0x13c83d358 <col:14, line:4:1>
    |-DeclStmt 0x13c83d2f8 <line:2:5, col:24>
    | ‘-VarDecl 0x13c83d218 <col:5, col:22> col:9 used result ’int’ cinit
    | ‘-BinaryOperator 0x13c83d2d8 <col:18, col:22> ’int’ ’+’
    | |-ImplicitCastExpr 0x13c83d2c0 <col:18> ’int’ <LValueToRValue>
    | | ‘-DeclRefExpr 0x13c83d280 <col:18> ’int’ lvalue ParmVar ’x’ ’int’
    | ‘-IntegerLiteral 0x13c83d2a0 <col:22> ’int’ 42
    ‘-ReturnStmt 0x13c83d348 <line:3:5, col:12>
      ‘-ImplicitCastExpr 0x13c83d330 <col:12> ’int’ <LValueToRValue>
        ‘-DeclRefExpr 0x13c83d310 <col:12> ’int’ lvalue Var ’result’ ’int’
```

Each line gives a **node kind**, an address, a **source range** and a **type**.

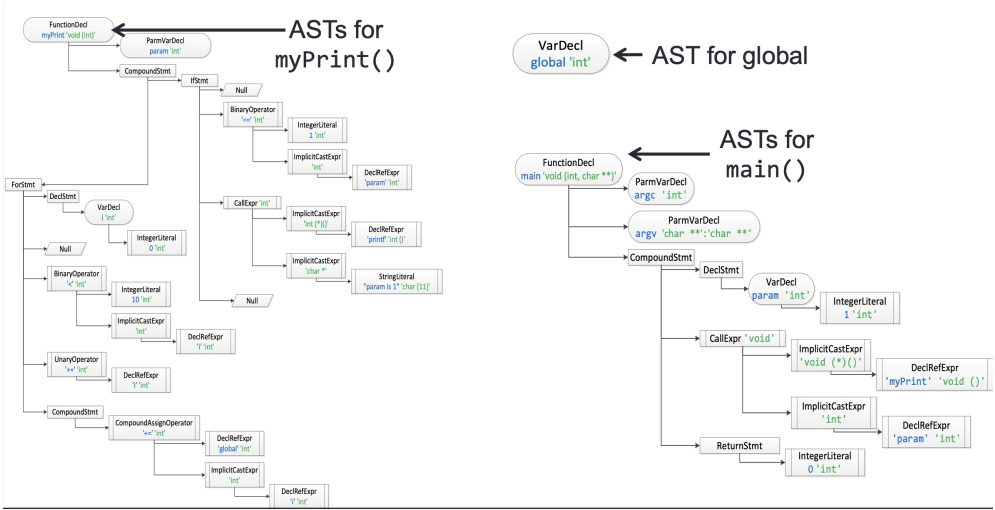
The **ImplicitCastExpr** nodes do not appear in the source. **The AST contains constructs the programmer did not write.**

Example: Clang AST Analysis

```c
#include <stdio.h>

int global;

void myPrint(int param) {
    if (param == 1)
        printf("param is 1");

    for (int i = 0; i < 10; i++) {
        global += i;
    }
}

int main(int argc, char *argv[]) {
    int param = 1;
    myPrint(param);
    return 0;
}
```

- 2 functions are declared:
  - `myPrint`
  - `main`
- `main` calls `myPrint` and returns 0.
- `myPrint` calls `printf`.
- `myPrint` contains:
  - an `if` statement
  - a `for` statement
- 1 global variable is declared:
  - `global`

Clang AST: `myPrint`



Clang AST: Declarations and Statements

- `Decl` represents declarations in the program.
  - Different declaration types have different subclasses.
  - `FunctionDecl` — function declaration.
  - `ParmVarDecl` — function parameter declaration.
  - `VarDecl` — variable declaration.
- `Stmt` represents statements in the program.
  - Different statement types have different subclasses.
  - `IfStmt` — `if` statement.
  - `ReturnStmt` — function return statement.
  - `ForStmt` — `for` loop.
- **Comments** such as `/* ... */` and `// ...` are not represented as ordinary AST nodes.