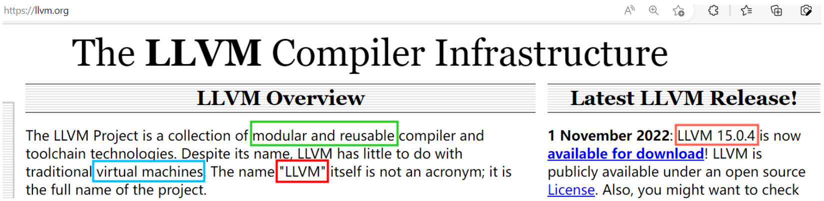
LLVM IR 简介

魏恒峰

hfwei@nju.edu.cn

2022 年 12 月 19 日


https://llvm.org

The **LLVM** Compiler Infrastructure

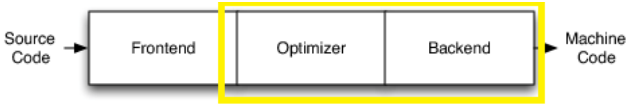
LLVM Overview

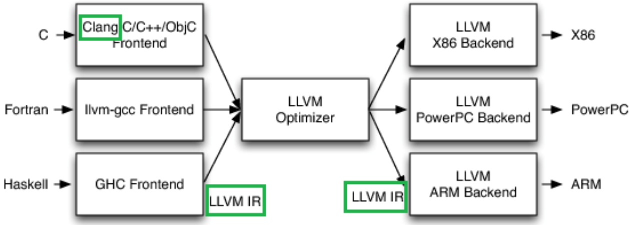
The LLVM Project is a collection of modular and reusable compiler and toolchain technologies. Despite its name, LLVM has little to do with traditional virtual machines. The name "LLVM" itself is not an acronym; it is the full name of the project.

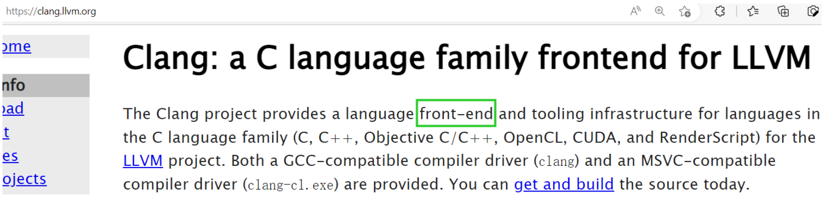
Latest LLVM Release!

1 November 2022: LLVM 15.0.4 is now **available for download**! LLVM is publicly available under an open source License. Also, you might want to check

Chris Lattner (1978)


Source Code
Frontend
Optimizer
Backend
Machine Code


C
Clang C/C++/ObjC Frontend
Fortran
llvm-gcc Frontend
Haskell
GHC Frontend
LLVM IR
LLVM Optimizer
LLVM IR
LLVM X86 Backend
LLVM PowerPC Backend
LLVM ARM Backend
X86
PowerPC
ARM



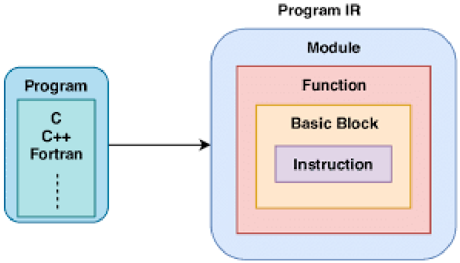
Clang: a C language family frontend for LLVM

The Clang project provides a language front-end and tooling infrastructure for languages in the C language family (C, C++, Objective C/C++, OpenCL, CUDA, and RenderScript) for the LLVM project. Both a GCC-compatible compiler driver (`clang`) and an MSVC-compatible compiler driver (`clang-cl.exe`) are provided. You can get and build the source today.

```
clang hello.c -o hello
```

hello @ CompilerExplorer

```
clang -Xclang -ast-dump -c hello.c
```

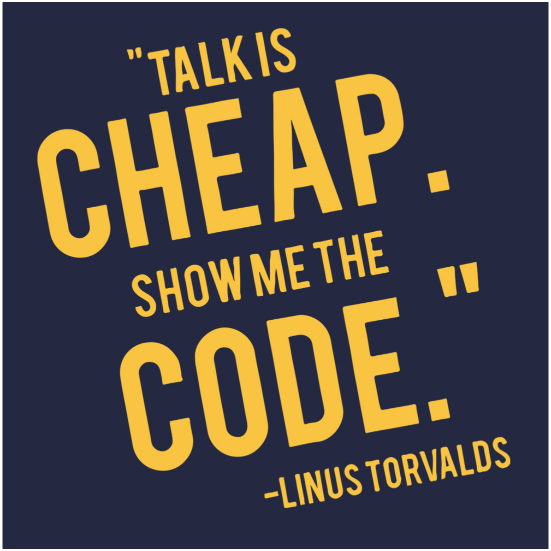


IR: Intermediate Representation

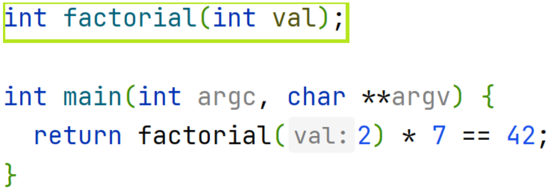
LLVM IR: 带有类型的、介于高级程序设计语言与汇编语言之间

联/袂/推/荐
THE BEAUTY OF HUAWEI OPENARKCOMPILER
史宁宁○编著
华为方舟编译器之美
基于开源代码的架构分析与实现
清华大学出版社

8 章技术内容, 其中 4 章介绍 Maple IR, 另外 4 章基于 Maple IR


"TALK IS CHEAP. SHOW ME THE CODE."

-LINUS TORVALDS



```c
int factorial(int val);

int main(int argc, char **argv) {
  return factorial(val: 2) * 7 == 42;
}
```

factorial0.c

注意: SysY 中没有函数声明语句。

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @main(i32 %0, i8** %1) #0 {
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i8**, align 8
  store i32 0, i32* %3, align 4
  store i32 %0, i32* %4, align 4
  store i8** %1, i8*** %5, align 8
  %6 = call i32 @factorial(i32 2)
  %7 = mul nsw i32 %6, 7
  %8 = icmp eq i32 %7, 42
  %9 = zext i1 %8 to i32
  ret i32 %9
}
```

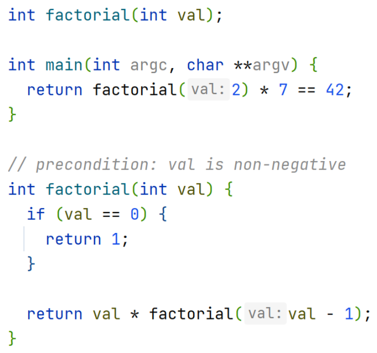
SSA: Static Single Assignment

TAC: Three-Address Code

```
clang -S -emit-llvm factorial0.c f0-opt0.ll
```

```
; Function Attrs: nounwind uwtable
define dso_local i32 @main(i32 %0, i8** nocapture readnone %1)
  %3 = call i32 @factorial(i32 2) #2
  %4 = mul nsw i32 %3, 7
  %5 = icmp eq i32 %4, 42
  %6 = zext i1 %5 to i32
  ret i32 %6
}
```

```
clang -S -emit-llvm factorial0.c f0-opt1.ll -O1 -g0
```

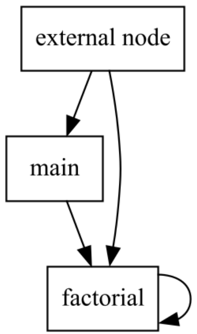


```
int factorial(int val);

int main(int argc, char **argv) {
  return factorial( val: 2) * 7 == 42;
}

// precondition: val is non-negative
int factorial(int val) {
  if (val == 0) {
    return 1;
  }

  return val * factorial( val: val - 1);
}
```

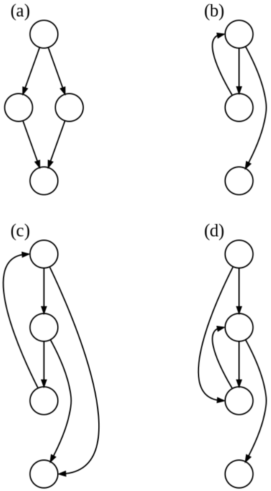


$$\begin{tikzcd}
 & \text{external node} \arrow[dl] \arrow[d, bend left] \\
\text{main} \arrow[r] & \text{factorial} \arrow[loop right]
\end{tikzcd}$$

Call graph

factorial1.c

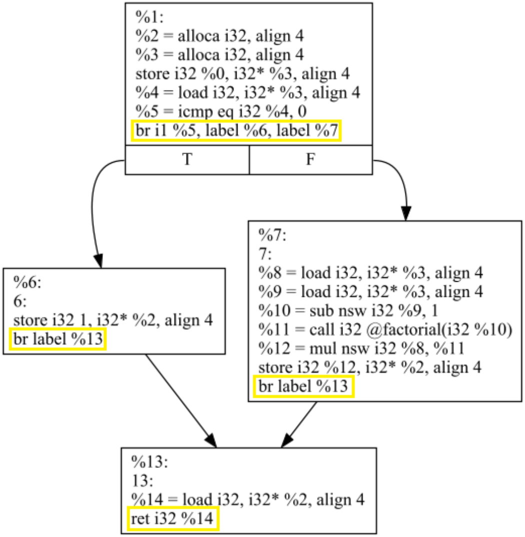
Frances Elizabeth Allen
(1932 ~ 2020; 2006 Turing Award)



(Intra-procedure) Control Flow Graph (CFG)

```
int factorial(int val) {
  if (val == 0) {
    return 1;
  }

  return val * factorial(val: val - 1);
}
```



%1:
%2 = alloca i32, align 4
%3 = alloca i32, align 4
store i32 %0, i32* %3, align 4
%4 = load i32, i32* %3, align 4
%5 = icmp eq i32 %4, 0
br i1 %5, label %6, label %7

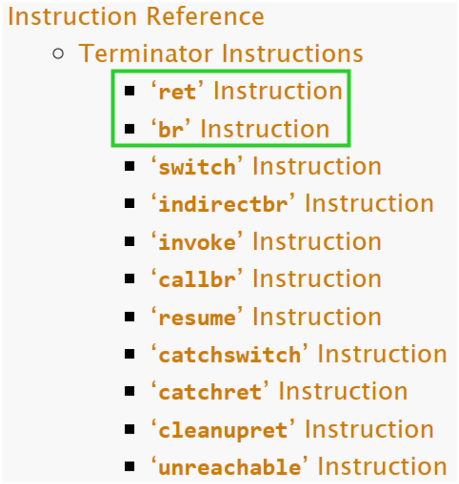
%6:
6:
store i32 1, i32* %2, align 4
br label %13

%7:
7:
%8 = load i32, i32* %3, align 4
%9 = load i32, i32* %3, align 4
%10 = sub nsw i32 %9, 1
%11 = call i32 @factorial(i32 %10)
%12 = mul nsw i32 %8, %11
store i32 %12, i32* %2, align 4
br label %13

%13:
13:
%14 = load i32, i32* %2, align 4
ret i32 %14

CFG for 'factorial' function

`factorial1.c (opt0)`

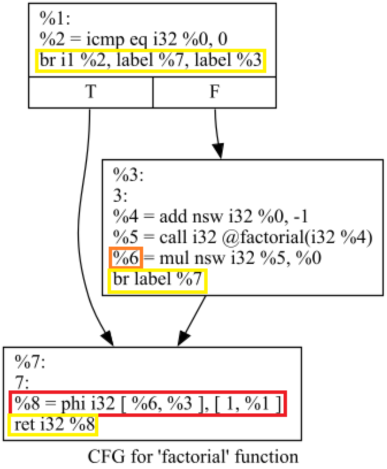


Instruction Reference

- Terminator Instructions
  - '**ret**' Instruction
  - '**br**' Instruction
  - '**switch**' Instruction
  - '**indirectbr**' Instruction
  - '**invoke**' Instruction
  - '**callbr**' Instruction
  - '**resume**' Instruction
  - '**catchswitch**' Instruction
  - '**catchret**' Instruction
  - '**cleanupret**' Instruction
  - '**unreachable**' Instruction

```c
int factorial(int val) {
  if (val == 0) {
    return 1;
  }

  return val * factorial( val: val - 1);
}
```

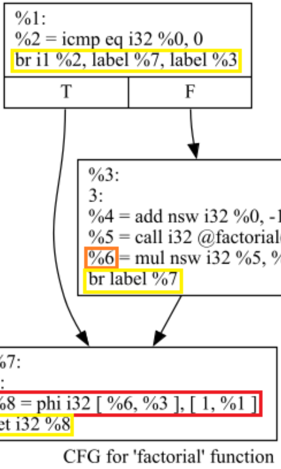

```
%1:
%2 = icmp eq i32 %0, 0
br i1 %2, label %7, label %3
```

```
%3:
3:
%4 = add nsw i32 %0, -1
%5 = call i32 @factorial(i32 %4)
%6 = mul nsw i32 %5, %0
br label %7
```
```
%7:
7:
%8 = phi i32 [ %6, %3 ], [ 1, %1 ]
ret i32 %8
```

factorial1.c (opt1)

```
int factorial(int val) {
  if (val == 0) {
    return 1;
  }

  return val * factorial( val: val - 1);
}
```

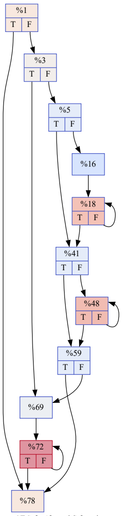

%1:
%2 = icmp eq i32 %0, 0
br i1 %2, label %7, label %3

%3:
3:
%4 = add nsw i32 %0, -1
%5 = call i32 @factorial(i32 %4)
%6 = mul nsw i32 %5, %0
br label %7

%7:
7:
%8 = phi i32 [%6, %3], [1, %1]
ret i32 %8

`factorial1.c (opt1)`

Single-Static Assignment Form and PHI (How to implement it?)



CFG for 'factorial' function

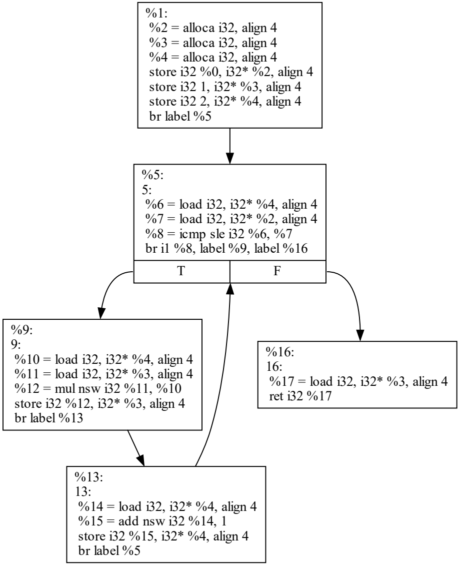
factorial1 (opt3)

```
int factorial(int val) {
  int temp = 1;

  for (int i = 2; i <= val; i++) {
    temp *= i;
  }

  return temp;
}
```

factorial2.c (opt0)



```
%1:
 %2 = alloca i32, align 4
 %3 = alloca i32, align 4
 %4 = alloca i32, align 4
 store i32 %0, i32* %2, align 4
 store i32 1, i32* %3, align 4
 store i32 2, i32* %4, align 4
 br label %5

%5:
5:
 %6 = load i32, i32* %4, align 4
 %7 = load i32, i32* %2, align 4
 %8 = icmp sle i32 %6, %7
 br i1 %8, label %9, label %16

%9:
9:
 %10 = load i32, i32* %4, align 4
 %11 = load i32, i32* %3, align 4
 %12 = mul nsw i32 %11, %10
 store i32 %12, i32* %3, align 4
 br label %13

%16:
16:
 %17 = load i32, i32* %3, align 4
 ret i32 %17

%13:
13:
 %14 = load i32, i32* %4, align 4
 %15 = add nsw i32 %14, 1
 store i32 %15, i32* %4, align 4
 br label %5
```

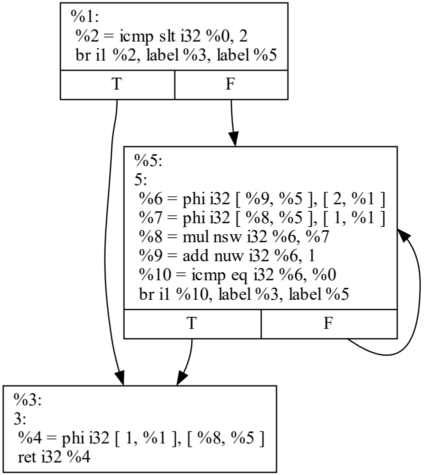
CFG for 'factorial' function

```c
int factorial(int val) {
  int temp = 1;

  for (int i = 2; i <= val; i++) {
    temp *= i;
  }

  return temp;
}
```

factorial2.c (opt1)

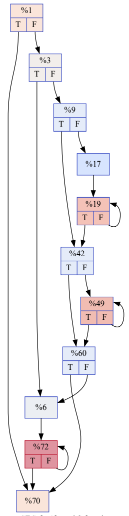


```
%1:
 %2 = icmp slt i32 %0, 2
 br i1 %2, label %3, label %5
T | F

%5:
5:
 %6 = phi i32 [ %9, %5 ], [ 2, %1 ]
 %7 = phi i32 [ %8, %5 ], [ 1, %1 ]
 %8 = mul nsw i32 %6, %7
 %9 = add nuw i32 %6, 1
 %10 = icmp eq i32 %6, %0
 br i1 %10, label %3, label %5
T | F

%3:
3:
 %4 = phi i32 [ 1, %1 ], [ %8, %5 ]
 ret i32 %4
```

CFG for 'factorial' function



CFG for 'factorial' function

factorial2 (opt3)

https://llvm.org/docs/LangRef.html

LLVM Home | Documentation » Reference »

LLVM Language Reference Manual

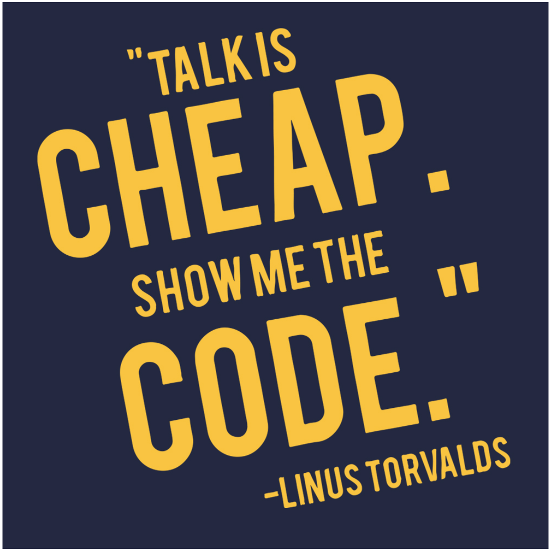
如何用编程的方式生成 LLVM IR?

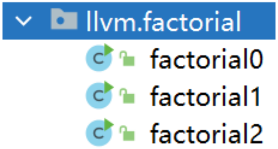
javacpp@github

JavaCPP Presets Platform For LLVM

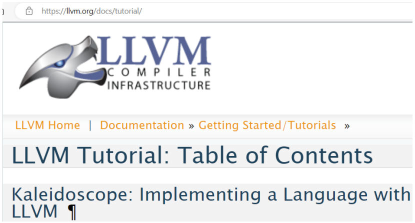
LLVM JAVA API使用手册

准备工作


"TALK IS
CHEAP.
SHOW ME THE
CODE."
-LINUS TORVALDS


llvm.factorial
factorial0
factorial1
factorial2

Kaleidoscope: Implementing a Language with LLVM

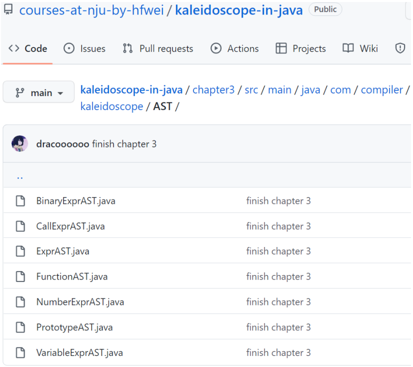


https://llvm.org/docs/tutorial/

LLVM Home | Documentation » Getting Started/Tutorials »

LLVM Tutorial: Table of Contents

Kaleidoscope: Implementing a Language with LLVM ¶



courses-at-nju-by-hfwei / kaleidoscope-in-java Public

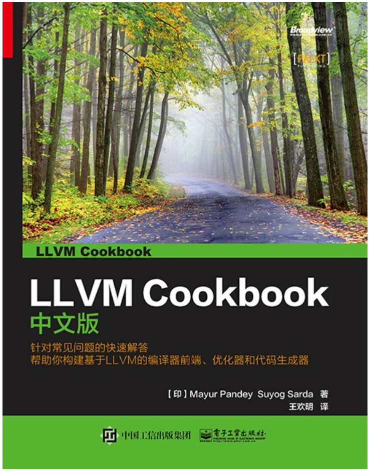
kaleidoscope-in-java / chapter3 / src / main / java / com / compiler / kaleidoscope / AST /

dracoooooo finish chapter 3

..

BinaryExprAST.java	finish chapter 3
CallExprAST.java	finish chapter 3
ExprAST.java	finish chapter 3
FunctionAST.java	finish chapter 3
NumberExprAST.java	finish chapter 3
PrototypeAST.java	finish chapter 3
VariableExprAST.java	finish chapter 3

kaleidoscope-in-java@github



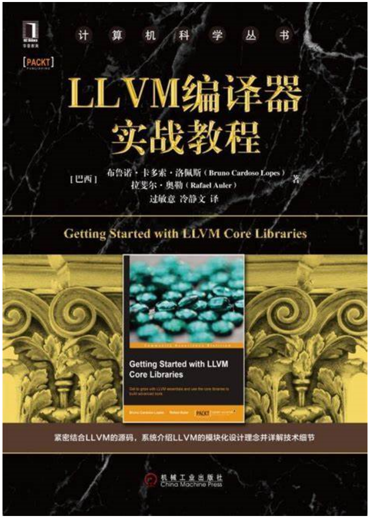
LLVM Cookbook

LLVM Cookbook
中文版

针对常见问题的快速解答
帮助你构建基于LLVM的编译器前端、优化器和代码生成器

[印] Mayur Pandey Suyog Sarda 著
王欢明 译

中国工信出版集团 電子工業出版社



计 算 机 科 学 丛 书

LLVM编译器
实战教程

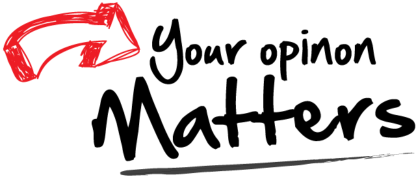
[巴西] 布鲁诺·卡多索·洛佩斯（Bruno Cardoso Lopes）
拉斐尔·奥勒（Rafael Auler） 著
过敏意 冷静文 译

Getting Started with LLVM Core Libraries

紧密结合LLVM的源码，系统介绍LLVM的模块化设计理念并详解技术细节

机械工业出版社
China Machine Press

Thank
You!



Office 926

hfwei@nju.edu.cn