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!

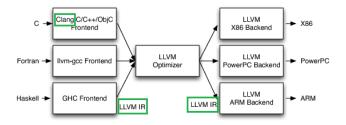
1 November 2022: LLVM 15.0.4 is now available for download! LLVM is publicly available under an open source License. Also, you might want to check





Chris Lattner (1978)





https://clang.llvm.org



<u>ome</u>

nfo oad t es

ojects

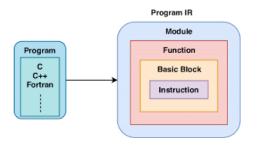
Clang: a C language family frontend for LLVM

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





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



```
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
  %7 = mul nsw i32 %6, 7
  %8 = icmp eq i32 %7, 42
                           SSA: Static-Single Assignment
  %9 = zext i1 \%8 to i32
                         TAC: Three-Address Code
  ret i32 %9
clang -S -emit-llvm factorial0.c f0-opt0.ll
```

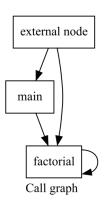
```
; Function Attrs: nounwind uwtable define dso_local i32 @main(\frac{i32 \%0}{i32 \%0}, \frac{i8** nocapture readnone \%1)} \\
\circ 3 = call i32 @factorial(i32 2) #2 \\
\chi4 = mul nsw i32 \%3, 7 \\
\chi5 = icmp eq i32 \%4, 42 \\
\chi6 = zext i1 \%5 to i32 \\
ret i32 \%6 \\
}
```

clang -S -emit-llvm factorial0.c f0-opt1.ll -01 -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;
   }

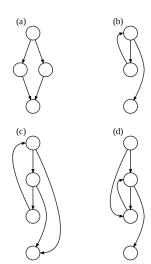
   return val * factorial(val: val - 1);
}
```



factorial1.c



Frances Elizabeth Allen (1932 \sim 2020; 2006 Turing Award)



(Intra-procedure) Control Flow Graph (CFG)

```
%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 = \text{icmp eq i32 } \%4, 0
                                                                         br i1 %5, label %6, label %7
                                                                                                  F
int factorial(int val) {
                                                                                           %7:
  if (val == 0) {
    return 1;
                                                                                           %8 = load i32, i32* %3, align 4
                                                       %6:
                                                                                           \%9 = \text{load i32}, \text{i32* } \%3, \text{align 4}
                                                       6:
                                                                                           %10 = \text{sub nsw i} 32 \% 9, 1
                                                       store i32 1, i32* %2, align 4
  return val * factorial( val: val - 1);
                                                                                           %11 = call i32 @factorial(i32 %10)
                                                       br label %13
                                                                                           %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
```

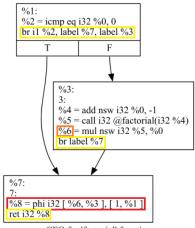
%1:

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

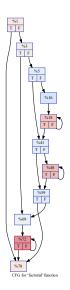
```
int factorial(int val) {
  if (val == 0) {
        return 1;
    }
  return val * factorial(val: val - 1);
}
```



CFG for 'factorial' function

factorial1.c (opt1)

Single-Static Assignment Form and PHI

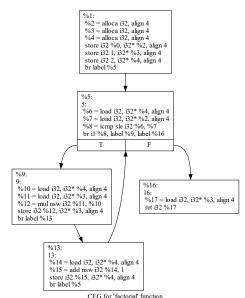


factorial1 (opt3)

18 / 27

```
int factorial(int val) {
   int temp = 1;
   for (int i = 2; i <= val; i++) {
       temp *= i;
   }
   return temp;
}</pre>
```

factorial2.c (opt0)

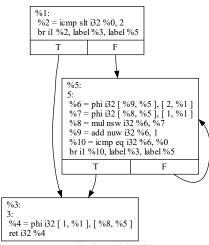


```
int factorial(int val) {
  int temp = 1;

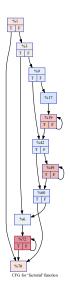
for (int i = 2; i <= val; i++) {
    temp *= i;
  }

  return temp;
}</pre>
```

factorial2.c (opt1)



CFG for 'factorial' function



factorial2 (opt3)



LLVM Home | Documentation » Reference »

LLVM Language Reference Manual

如何用编程的方式生成 LLVM IR?

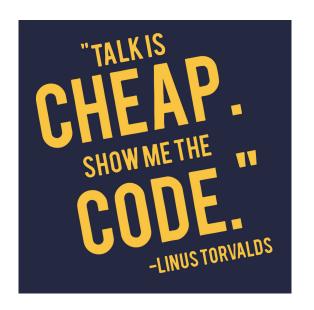
javacpp@github

JavaCPP Presets Platform For LLVM



LLVM JAVA API使用手册

准备工作

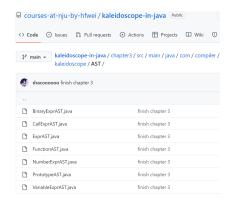


🗸 🖿 llvm.factorial

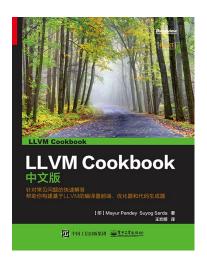
- 😅 🖆 factorial1
- 😅 🖢 factorial2

Kaleidoscope: Implementing a Language with LLVM





kaleidoscope-in-java@github





Thank You!



Office 926 hfwei@nju.edu.cn