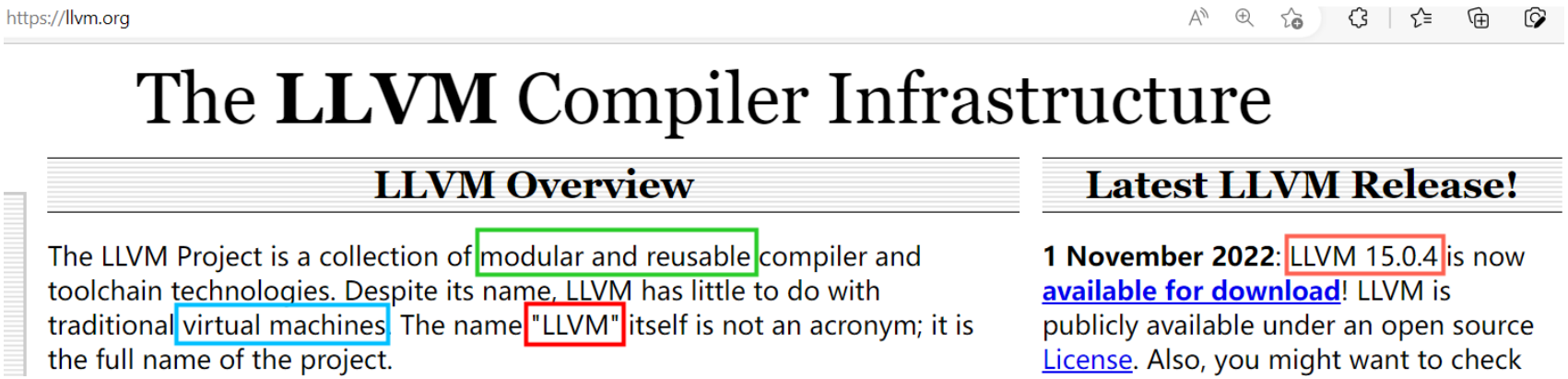
LLVM IR 简介

魏恒峰

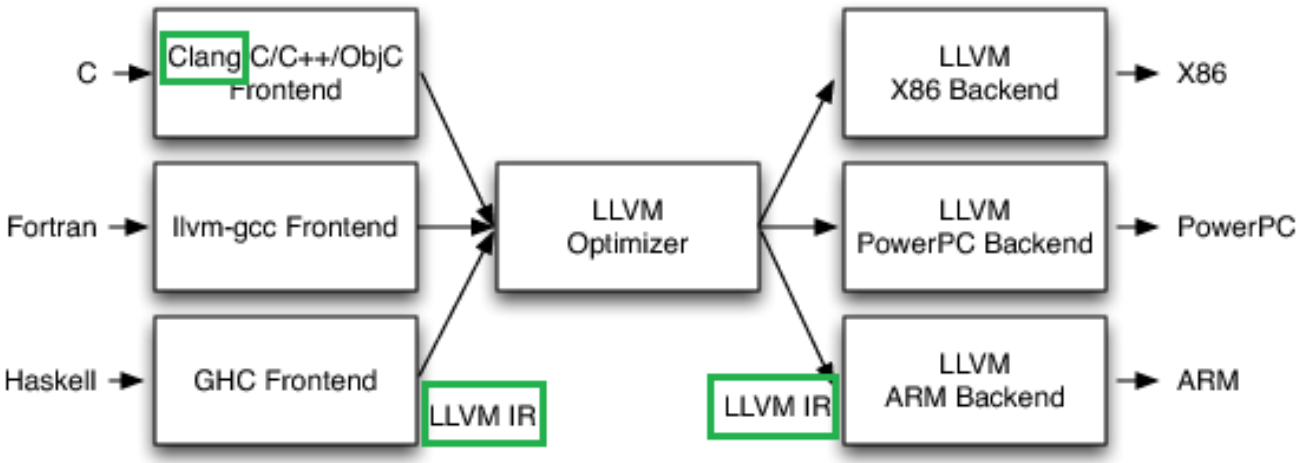
hfwei@nju.edu.cn

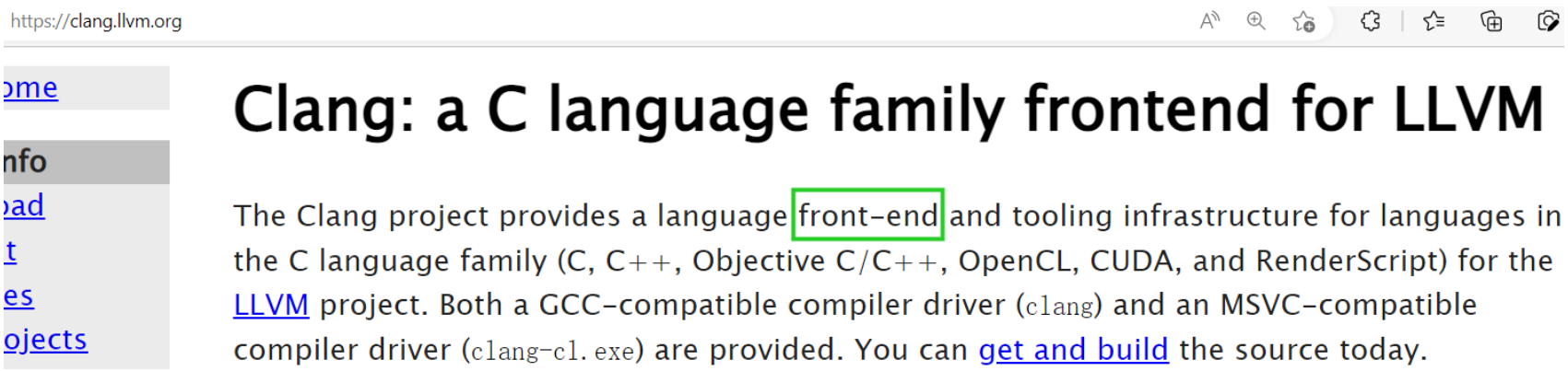
2022 年 12 月 19 日


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

ACM
Software
System
Award
2012

Source Code
Frontend
Optimizer
Backend
Machine Code

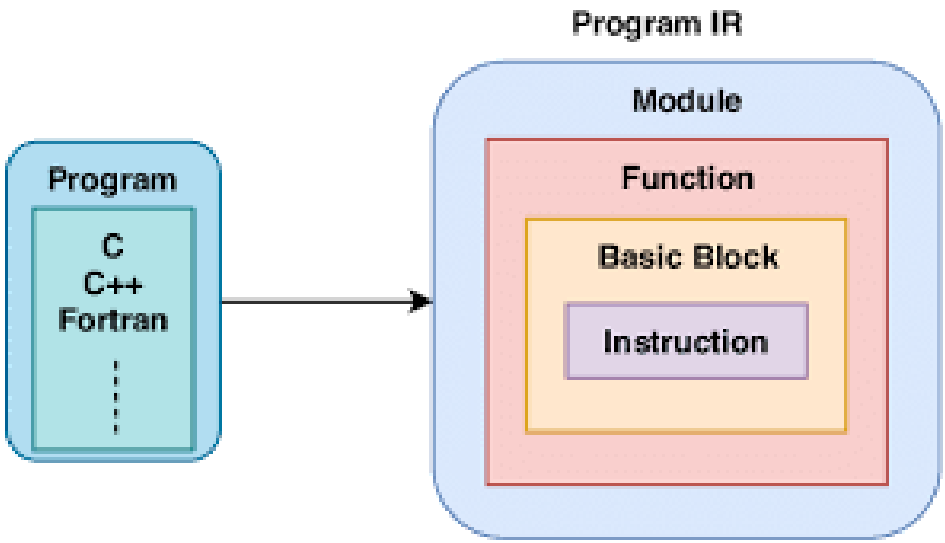

C
Clang C/C++/ObjC Frontend
Fortran
llvm-gcc Frontend
Haskell
GHC Frontend
LLVM IR
LLVM Optimizer
LLVM IR
LLVM X86 Backend
LLVM PowerPC Backend
LLVM ARM Backend
X86
PowerPC
ARM


https://clang.llvm.org
ome
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
```


Program IR
Module
Function
Basic Block
Instruction
Program
C
C++
Fortran

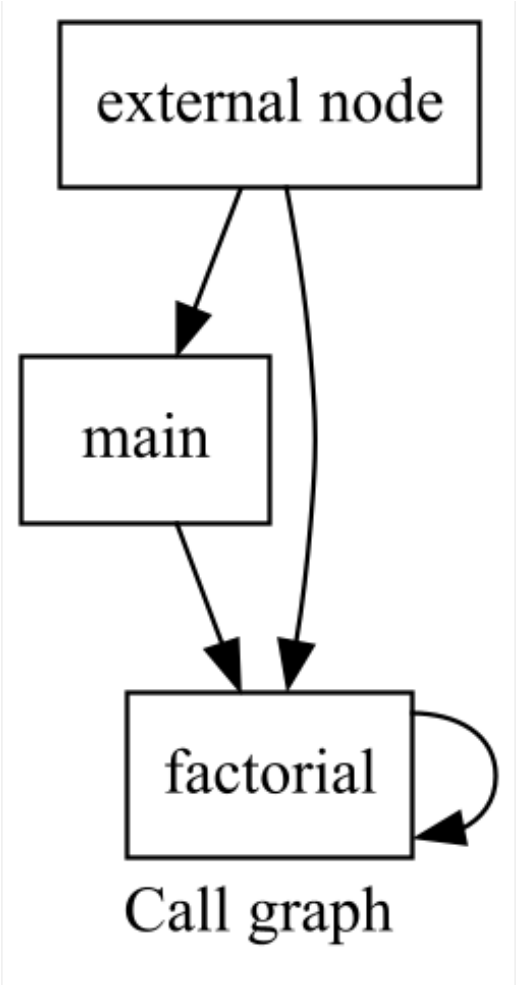
```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @main(i32 %0, i8** %1) #0 {
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i8**, align 8
  store i32 0, i32* %3, align 4
  store i32 %0, i32* %4, align 4
  store i8** %1, i8*** %5, align 8
  %6 = call i32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

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

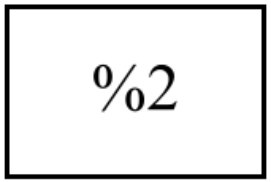
```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @main(i32 %0, i8** %1) #0 {
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i8**, align 8
  store i32 0, i32* %3, align 4
  store i32 %0, i32* %4, align 4
  store i8** %1, i8*** %5, align 8
  %6 = call i32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

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

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



`factorial1.c`

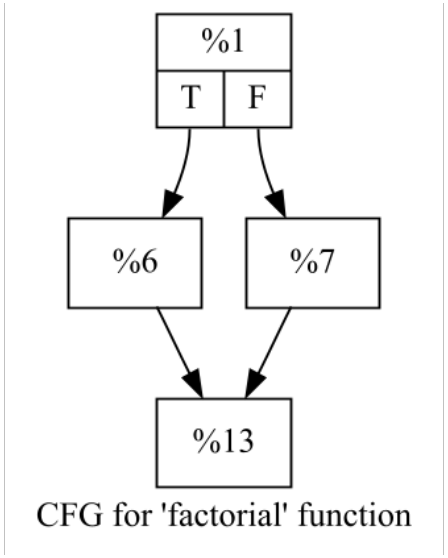


CFG for 'main' function

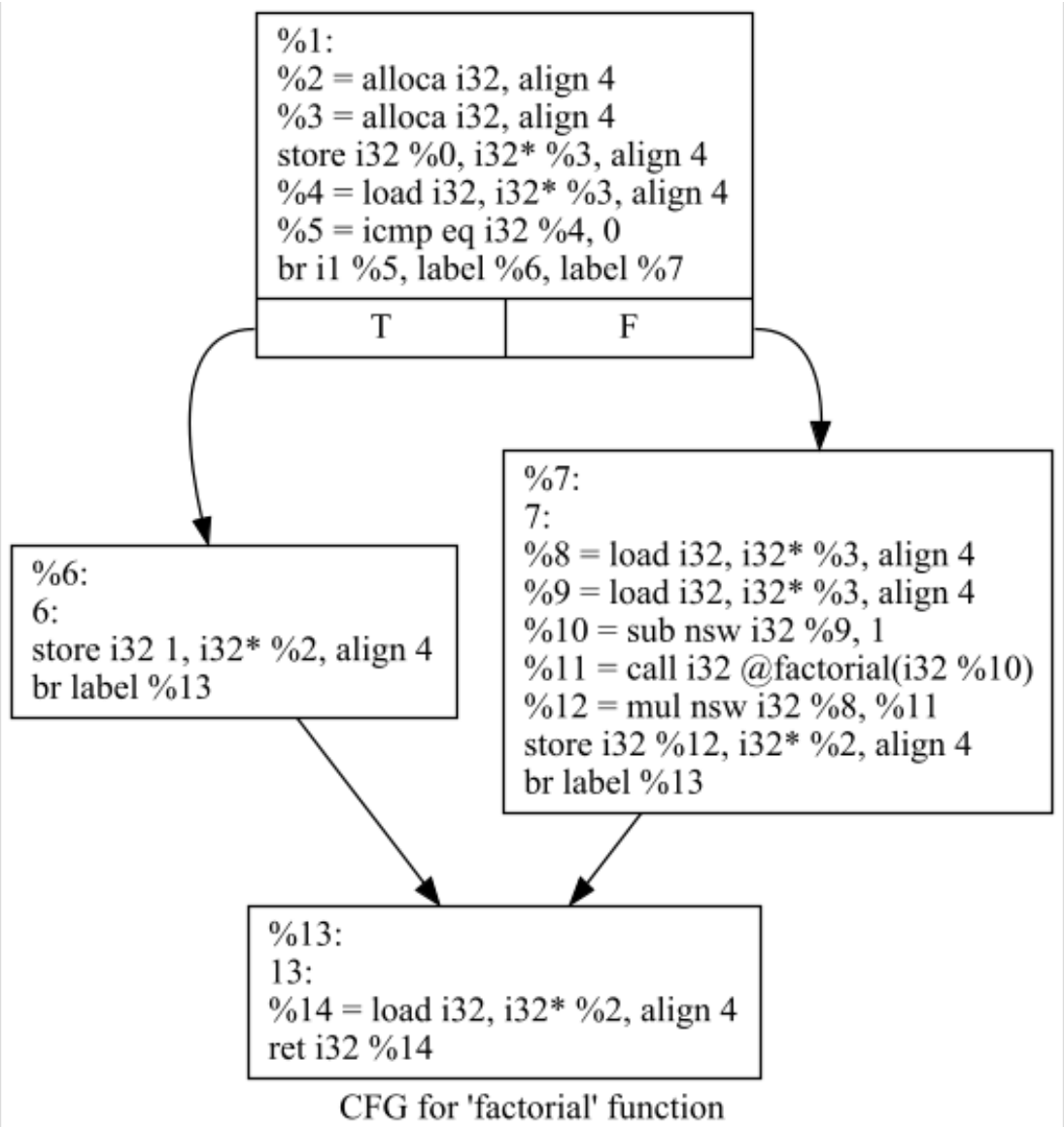
```
%2:
%3 = alloca i32, align 4
%4 = alloca i32, align 4
%5 = alloca i8**, align 8
store i32 0, i32* %3, align 4
store i32 %0, i32* %4, align 4
store i8** %1, i8*** %5, align 8
%6 = call i32 @factorial(i32 2)
%7 = mul nsw i32 %6, 7
%8 = icmp eq i32 %7, 42
%9 = zext i1 %8 to i32
ret i32 %9
```

CFG for 'main' function

`factorial1.c (opt0)`



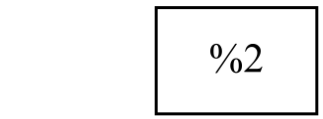
CFG for 'factorial' function



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

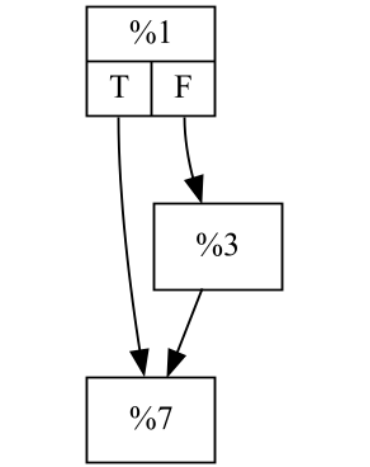


CFG for 'main' function

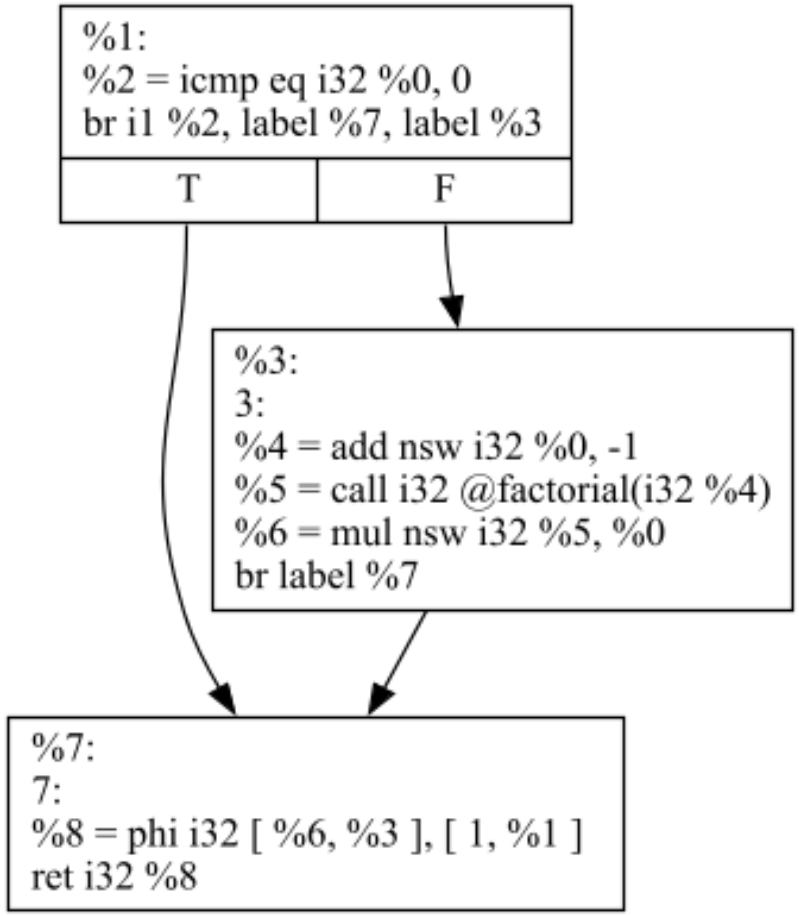
```
%2:
%3 = call i32 @factorial(i32 2)
%4 = mul nsw i32 %3, 7
%5 = icmp eq i32 %4, 42
%6 = zext i1 %5 to i32
ret i32 %6
```

CFG for 'main' function

`factorial1.c (opt1)`

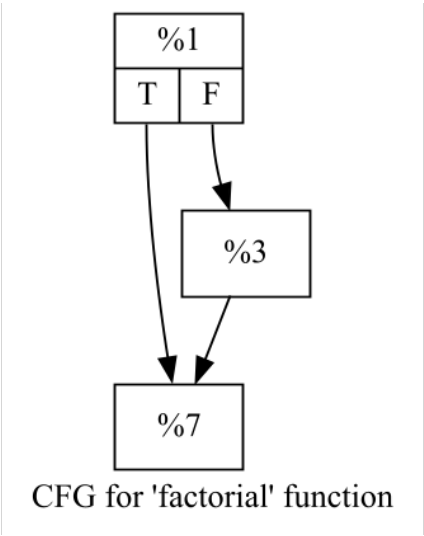


CFG for 'factorial' function

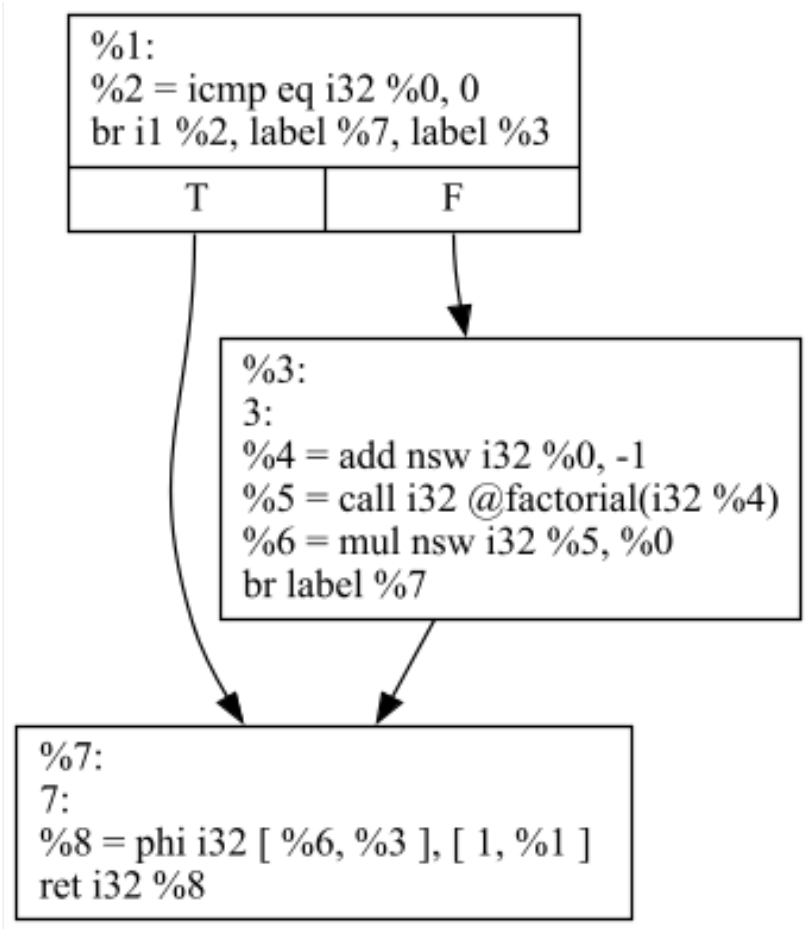


CFG for 'factorial' function

`factorial1.c (opt1)`



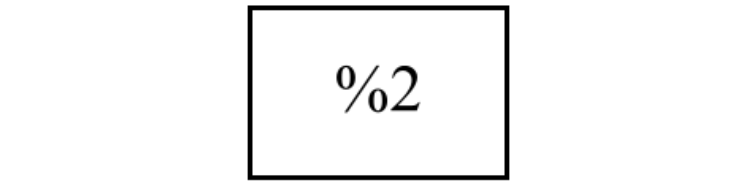
CFG for 'factorial' function



CFG for 'factorial' function

`factorial1.c (opt1)`

Single-Static Assignment Form and PHI

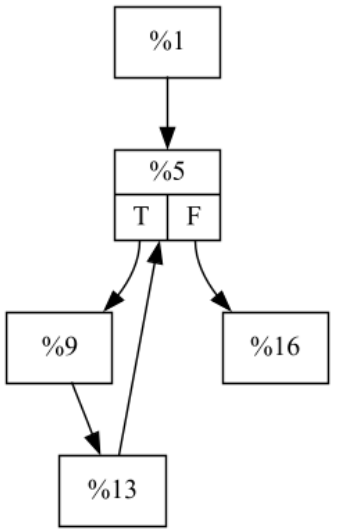


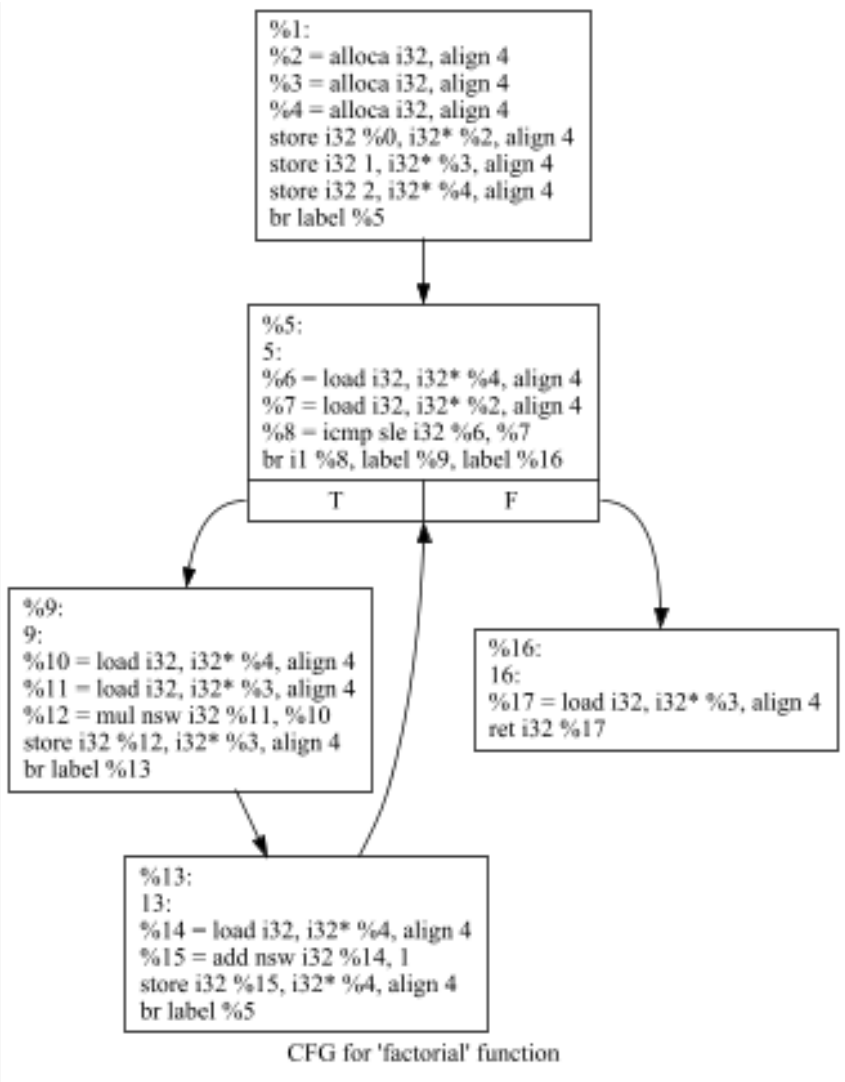
CFG for 'main' function

```
%2:
%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
```

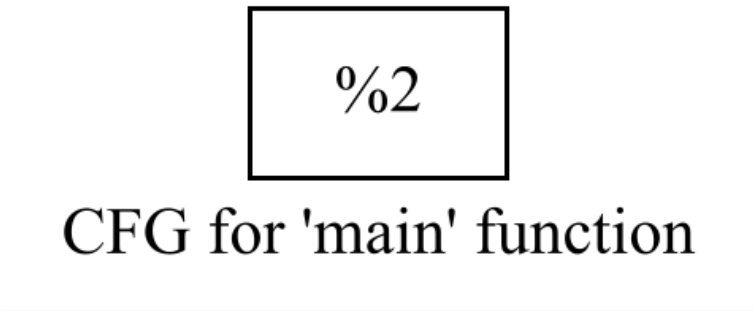
CFG for 'main' function

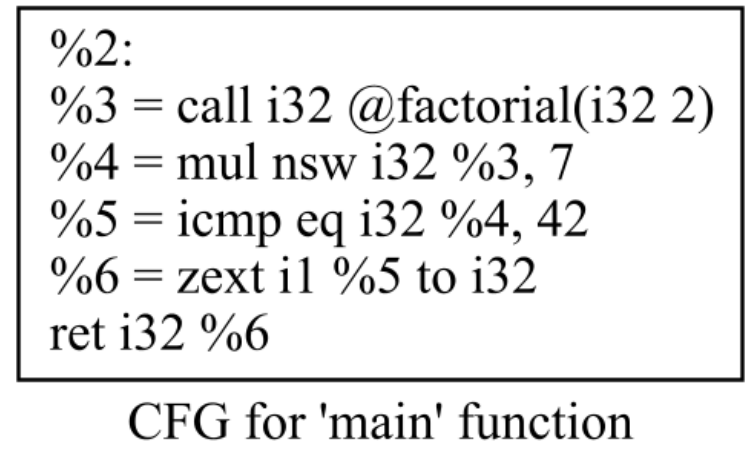
`factorial2.c (opt0)`



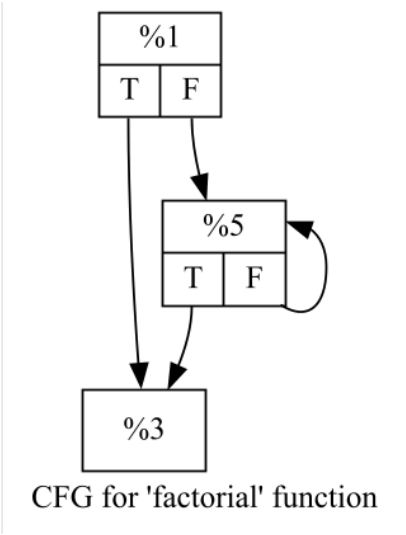


factorial2.c (opt0)

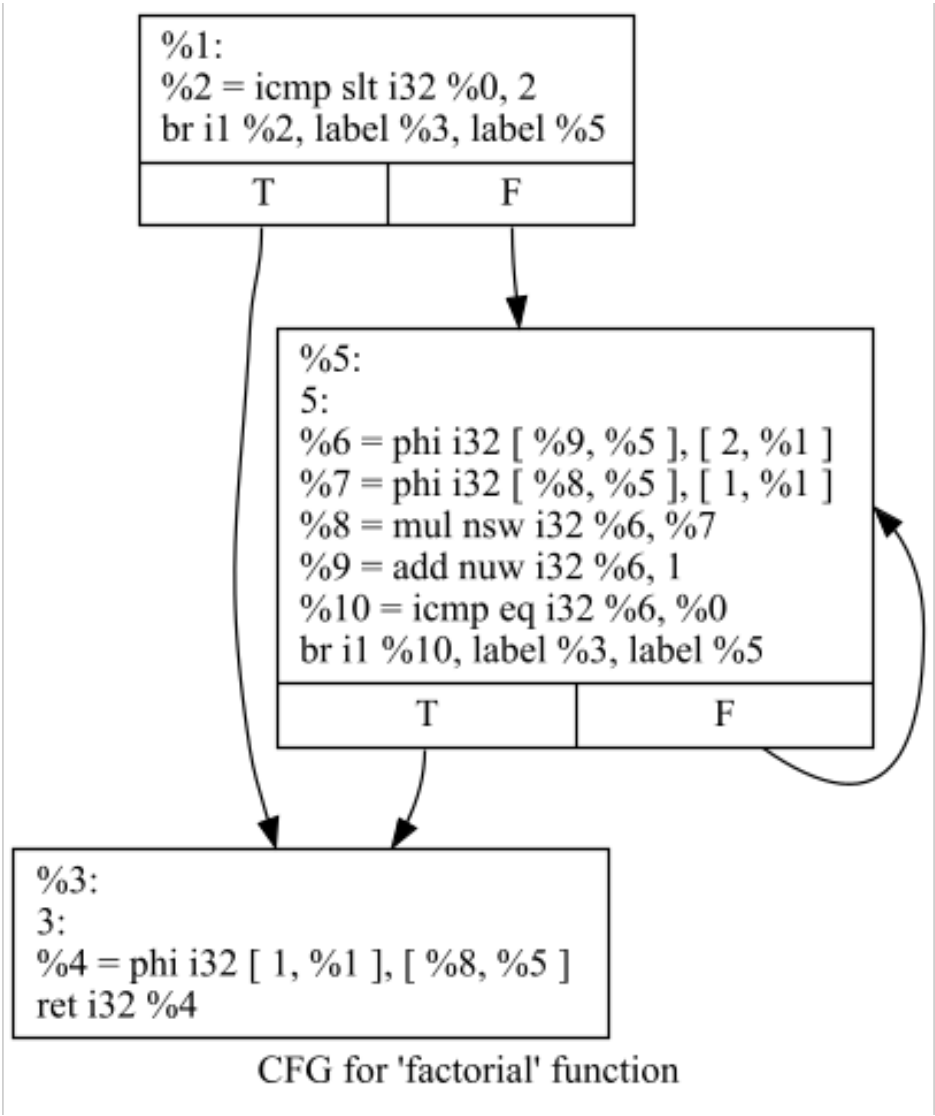




`factorial2.c (opt1)`



CFG for 'factorial' function



CFG for 'factorial' function

`factorial2.c (opt1)`

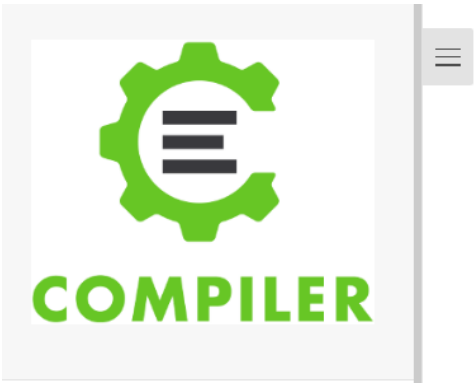
https://llvm.org/docs/LangRef.html

LLVM Home | Documentation » Reference »

LLVM Language Reference Manual

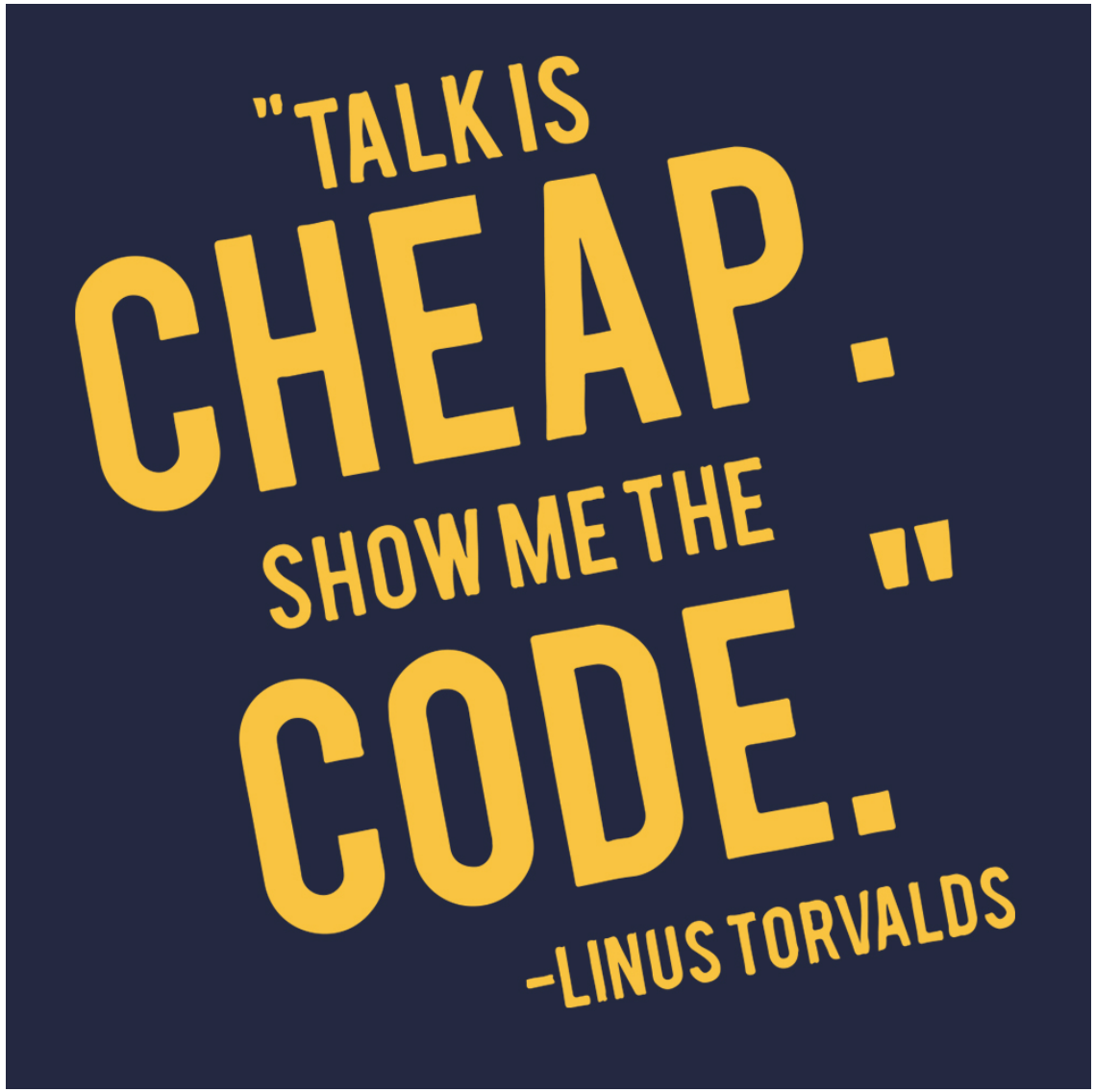
javacpp@github

JavaCPP Presets Platform For LLVM

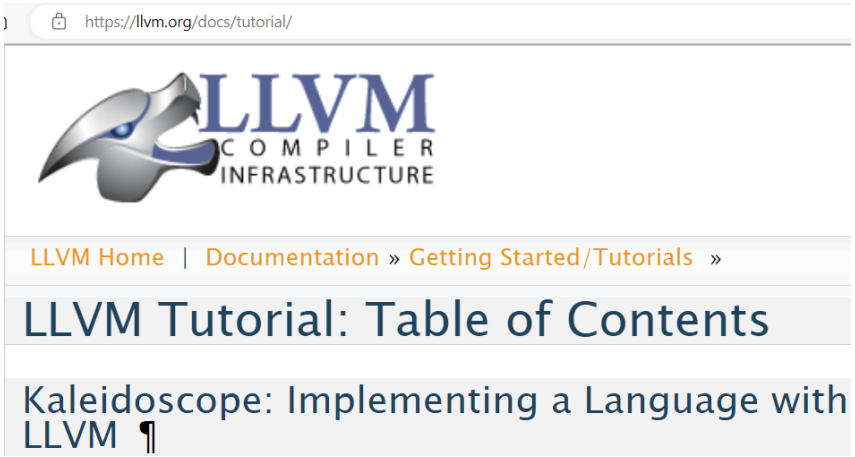


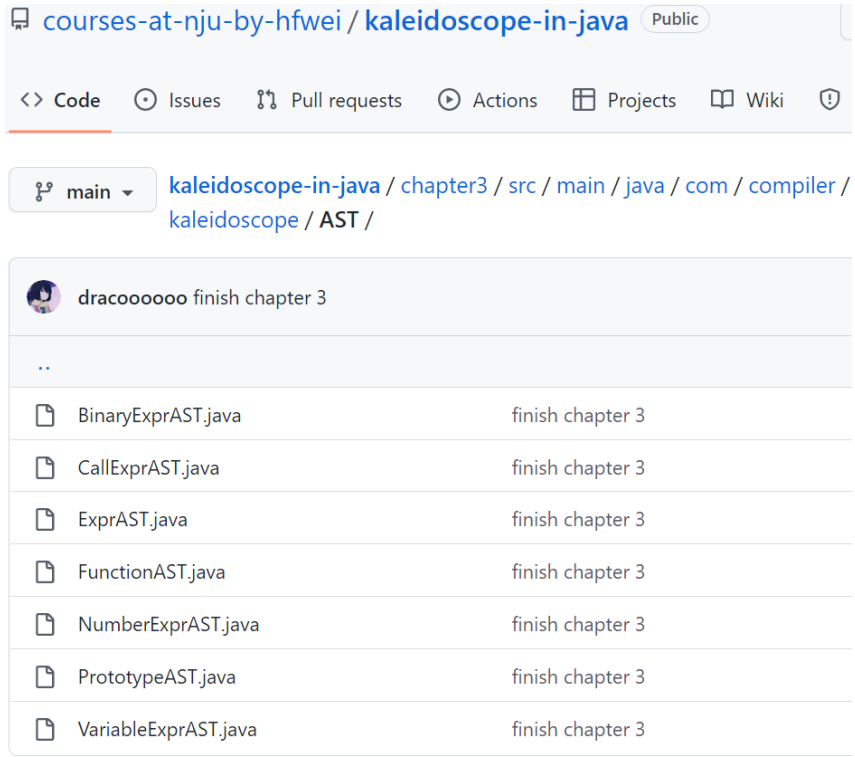
LLVM JAVA API使用手册

准备工作


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

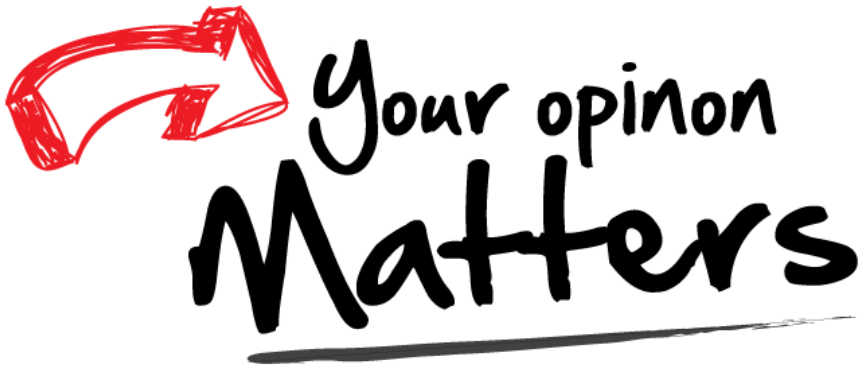
Kaleidoscope: Implementing a Language with LLVM





kaleidoscope-in-java@github

Thank
You!



Office 926

hfwei@nju.edu.cn