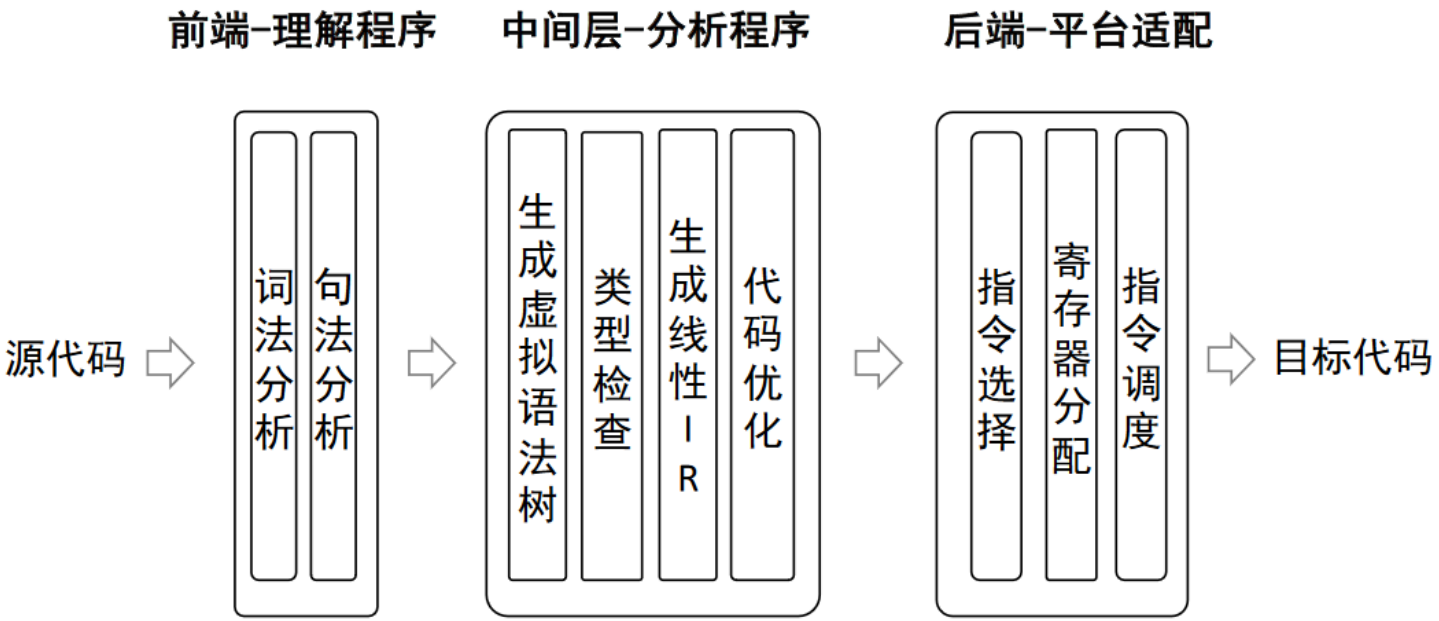
通过 LLVM 了解编译器

LLVM

Low Level Virtual Machine（虽然并不是一个 virtual machine）

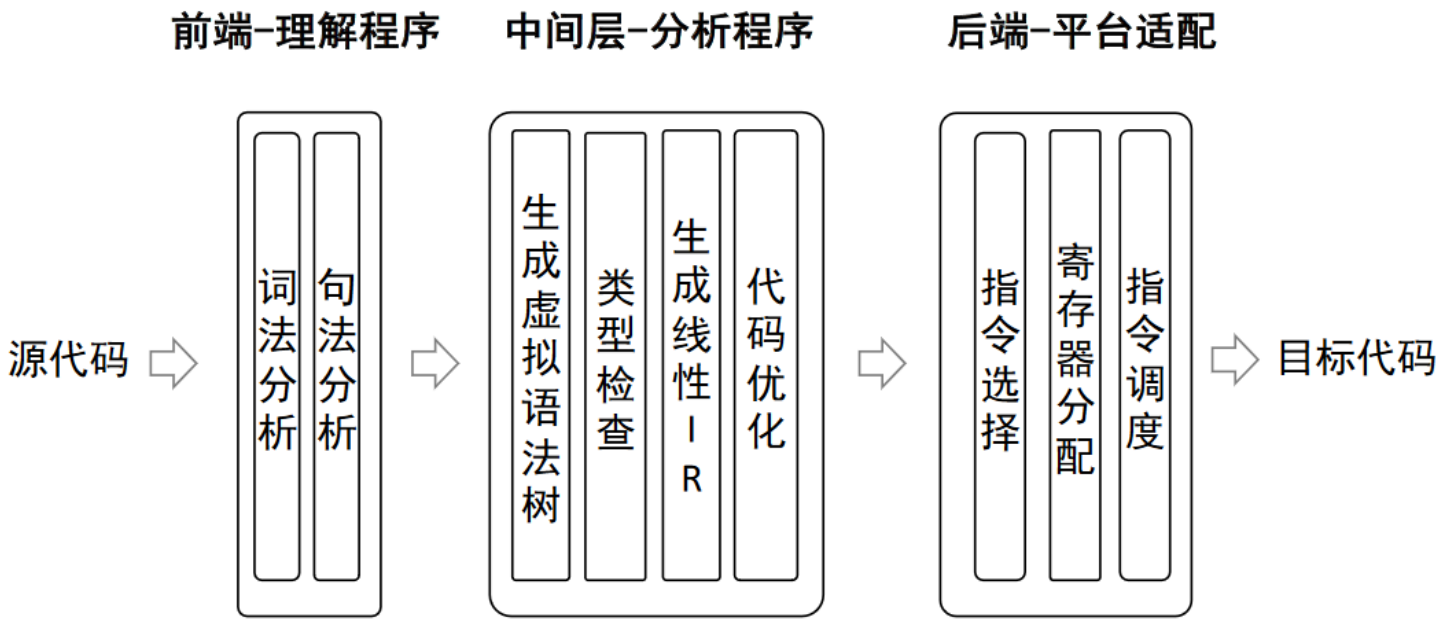
一个开源的编译器框架，包含若干 subprojects 实现了编译的各个阶段。

它有一个原生实现的 C/C++编译器 clang。



LLVM

本学期后续会有5~6个PJ，我们将自己实现下图中的各个阶段



安装

clang --version

apt install llvm

apt install clang-tools（扩展工具，如静态分析等）

前端-词法&语法分析，生成AST

clang **–fsyntax-only** 只分析语法而不进行实际编译 **–Xclang –ast-dump** 要求clang前端输出AST **[xxx].c**

```c
// example.c
int foo(int aa, int bb, int cc){
    int sum = aa + bb;
    return sum / cc;
}

```

```
fdong@compiler:~/Desktop$ clang -c -fsyntax-only -Xclang -ast-dump foo.c
TranslationUnitDecl 0x1708d58 <<invalid sloc>> <invalid sloc>
|-TypedefDecl 0x1709580 <<invalid sloc>> <invalid sloc> implicit __int128_t '__int128'
| `-BuiltinType 0x1709320 '__int128'
|-TypedefDecl 0x17095f0 <<invalid sloc>> <invalid sloc> implicit __uint128_t 'unsigned __int128'
| `-BuiltinType 0x1709340 'unsigned __int128'
|-TypedefDecl 0x17098f8 <<invalid sloc>> <invalid sloc> implicit __NSConstantString 'struct __NSConstantString_tag'
| `-RecordType 0x17096d0 'struct __NSConstantString_tag'
|   `-Record 0x1709648 '__NSConstantString_tag'
|-TypedefDecl 0x1709990 <<invalid sloc>> <invalid sloc> implicit __builtin_ms_va_list 'char *'
| `-PointerType 0x1709950 'char *'
|   `-BuiltinType 0x1708e00 'char'
|-TypedefDecl 0x1709c88 <<invalid sloc>> <invalid sloc> implicit __builtin_va_list 'struct __va_list_tag[1]'
| `-ConstantArrayType 0x1709c30 'struct __va_list_tag[1]' 1
|   `-RecordType 0x1709a70 'struct __va_list_tag'
|     `-Record 0x17099e8 '__va_list_tag'
`-FunctionDecl 0x175fa40 <foo.c:2:1, line:5:1> line:2:5 foo 'int (int, int, int)'
  |-ParmVarDecl 0x175f850 <col:9, col:13> col:13 used aa 'int'
  |-ParmVarDecl 0x175f8d0 <col:17, col:21> col:21 used bb 'int'
  |-ParmVarDecl 0x175f950 <col:25, col:29> col:29 used cc 'int'
  `-CompoundStmt 0x175fd08 <col:32, line:5:1>
    |-DeclStmt 0x175fc50 <line:3:5, col:22>
    | `-VarDecl 0x175fb58 <col:5, col:20> col:9 used sum 'int' cinit
    |   `-BinaryOperator 0x175fc30 <col:15, col:20> 'int' '+'
    |     |-ImplicitCastExpr 0x175fc00 <col:15> 'int' <LValueToRValue>
    |     | `-DeclRefExpr 0x175fbc0 <col:15> 'int' lvalue ParmVar 0x175f850 'aa' 'int'
    |     `-ImplicitCastExpr 0x175fc18 <col:20> 'int' <LValueToRValue>
    |       `-DeclRefExpr 0x175fbe0 <col:20> 'int' lvalue ParmVar 0x175f8d0 'bb' 'int'
    `-ReturnStmt 0x175fcf8 <line:4:5, col:18>
      `-BinaryOperator 0x175fcd8 <col:12, col:18> 'int' '/'
        |-ImplicitCastExpr 0x175fca8 <col:12> 'int' <LValueToRValue>
        | `-DeclRefExpr 0x175fc68 <col:12> 'int' lvalue Var 0x175fb58 'sum' 'int'
        `-ImplicitCastExpr 0x175fcc0 <col:18> 'int' <LValueToRValue>
          `-DeclRefExpr 0x175fc88 <col:18> 'int' lvalue ParmVar 0x175f950 'cc' 'int'
```

中间层-类型检查、静态分析和生成IR

clang **–fsyntax-only** 只分析语法而不进行实际编译 **–Wall** 开启所有常用警告 **–Wextra** 开启额外警告 **-pedantic** 严格遵循 ISO C++标准，报告一切不符合标准的警告 **[xxx].c**

类型检查往往并入了前端

```c
// example.c
int foo(int aa, int bb, int cc){
    int sum = aa + bb;
    return sum / cc;
}

```

```
fdong@compiler:~/Desktop$ clang -fsyntax-only -Wall -Wextra -pedantic foo.c
foo.c:5:2: warning: no newline at end of file [-Wnewline-eof]
}
 ^
1 warning generated.
```

中间层-类型检查、静态分析和生成IR

clang **-S** **–emit-llvm** **[xxx].c** **–o -**

只执行到
生成汇编代码阶段

生成llvm IR

输出到标准输出
（控制台）

```c
// main.c
int main(){
    int aa = 2024;
    return 0;
}

```

```
fdong@compiler:~/Desktop$ clang -S -emit-llvm main.c -o -
; ModuleID = 'main.c'
source_filename = "main.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-pc-linux-gnu"

; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 0, i32* %1, align 4
  store i32 2024, i32* %2, align 4
  ret i32 0
}

attributes #0 = { noinline nounwind optnone uwtable "frame-pointer"="all" "min-legal-vector-w
idth"="0" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-cpu"="x86-64" "
target-features"="+cx8,+fxsr,+mmx,+sse,+sse2,+x87" "tune-cpu"="generic" }

!llvm.module.flags = !{!0, !1, !2, !3, !4}
!llvm.ident = !{!5}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{i32 7, !"uwtable", i32 1}
!4 = !{i32 7, !"frame-pointer", i32 2}
!5 = !{!"Ubuntu clang version 14.0.0-1ubuntu1.1"}
fdong@compiler:~/Desktop$
```

后端-指令选择、寄存器优化和生成目标代码

clang **-O** **–S –emit-assembly** **[xxx].c** **–o -**

-O：代码优化等级 [O1, O2, O3]

–S –emit-assembly：生成汇编代码

–o -：输出到标准输出（控制台）

```c
// main.c
int main(){
    int aa = 2024;
    return 0;
}

```

```
fdong@compiler:~/Desktop$ clang -O -S -emit-assembly main.c -o -
clang: warning: -e mit-assembly: 'linker' input unused [-Wunused-command-line-argument]
        .text
        .file   "main.c"
        .globl  main                            # -- Begin function main
        .p2align        4, 0x90
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# %bb.0:
        xorl    %eax, %eax
        retq
.Lfunc_end0:
        .size   main, .Lfunc_end0-main
        .cfi_endproc
                                        # -- End function
        .ident  "Ubuntu clang version 14.0.0-1ubuntu1.1"
        .section        ".note.GNU-stack","",@progbits
        .addrsig
```

后端-生成目标代码

clang **-c** **[xxx].c** **–o [xxx].o**

只生成目标代码

输出到二进制文件

```c
// main.c
int main(){
    int aa = 2024;
    return 0;
}

```