Table of Contents

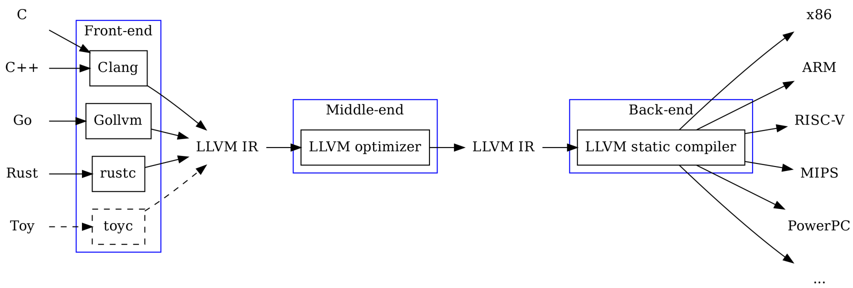
miniSysY □□□□

□□□□

SysY[1] □□□□□□□□□□□c□□□□□SysY □□□□□□□ .sy □□□□□□c □□□ include/define/pointer/struct □□□□□sysy □□□□□□ IO □□□□□ □□□□□□□ IO□

LLVM □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ SSA □□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□ gcc □□□□□

LLVM IR □ LLVM □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□



□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ issue □□□□□□□□□□□

□□□□

□□□□□□□□□□□ SysY □□□□□□□□□□□□□□□□□□□□-□□-□□-□□□ □□□□□□□□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

- □□□□□□□□□□□
- □□□□□□□□□□□□□□□
- □□□□□□□□□□□□□
- □□□□□□□□
- □□□□□□□□□
- □□□ gcc, llvm □□□□□□□□□□□

□□□□□□□□□

- □□□□□
- □□□□
- □ ABI □□□
- □□□□□□□

[illegible] LLVM IR [illegible] GIMPLE [illegible]

GIMPLE [illegible]

[illegible] CISC [illegible] RISC [illegible]LLVM IR [illegible]~~[illegible]~~

[illegible] LLVM IR [illegible] LLVM IR [illegible]

> 1. [illegible] ↩

[illegible]

[illegible] lab [illegible] lab ,[illegible]

[illegible] PDF

[illegible]

[illegible]_[illegible]_labx [illegible]x[illegible] lab [illegible]

[illegible]

- [illegible]+[illegible]
- [illegible] lab [illegible] part [illegible]
- [illegible] part [illegible]
- [illegible]

[illegible]

[illegible] [illegible]1000[illegible] part [illegible]

[illegible]

[illegible]

[illegible]

1. [illegible]
2. [illegible]
3. [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

- NONONO [illegible]
- NONONO “[illegible]”
- YESYESYES [illegible]part[illegible]
- YESYESYES [illegible]llvm,[illegible]

LLVM □ LLVM IR

□□□□□□□□□

- LLVM □□□□□□□□
- LLVM □□□□□□□□□ LLVM Project □□□□□□□□□□□□□□□□□□
- LLVM IR □□□□□□□□ LLVM IR □□□□□□□□□□□□□□□□□ LLVM IR □□□□□
- LLVM IR SSA □□□□□□ LLVM IR □□ SSA □□□□□□□□□□□□□□ IR □□□
- LLVM □□□□□□□□□□□□□□□□□□□□□□□□ LLVM □□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□

LLVM □□□□□□□

□□□□□□□□□□□□ Clang □ LLVM □□□□□□ 10.0□

Ubuntu

20.04 □□□□□

□□ Ubuntu 20.04 □□□□□□□□□□□ LLVM □□□□□□□ 10+□□□□□□□□□□□□□□□

```
$ sudo apt-get install llvm
$ sudo apt-get install clang
```

□□□□□□□□□□□□□□□□□□□□

```
$ clang -v # □□□□□□□□□□□□□□□□□□□□
$ lli --version # □□□□□□□□□□□□□□□□□□□□
```

18.04

□□ Ubuntu 18.04□□□□□□ LLVM □□□□□□□ 6.0□□□□□□□□□□□□□□□□□□□

```
$ sudo apt-get install llvm-10
$ sudo apt-get install clang-10
```

□□□□□□□□□□□□□□□□□□ `-10` □□□□□□□□ `clang-10` □ `lli-10` □□□□□ □□□□ `alias` □□□□□

□□□□□□□□□□□□□□□□□□□□

```
$ clang-10 -v # □□□□□□□□□□□□□□□□□□□□
$ lli-10 --version # □□□□□□□□□□□□□□□□□□□□
```

□□□□

□□□□□

□□□□ apt □□□□□□□□□□□□□□□

□

```
# i386 not available
deb http://apt.llvm.org/focal/ llvm-toolchain-focal main
deb-src http://apt.llvm.org/focal/ llvm-toolchain-focal mai
# 9
deb http://apt.llvm.org/focal/ llvm-toolchain-focal-9 main
deb-src http://apt.llvm.org/focal/ llvm-toolchain-focal-9 m
# 10
deb http://apt.llvm.org/focal/ llvm-toolchain-focal-10 main
deb-src http://apt.llvm.org/focal/ llvm-toolchain-focal-10
```

□□ `/etc/apt/sources.list`

□□□□□□□□

`wget -O - https://apt.llvm.org/llvm-snapshot.gpg.key|sudo apt-key add -`

□□□□□□□□

`apt-get install clang-10 lldb-10 lld-10`

Redhat/Arch/...□Ubuntu/Debian □□□□

□□□□□□□□□ 2 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ Clang □ LLVM □□□□□□□ □□□□□□□

> ~~Fly B***h~~

macOS

Clang

□ macOS □□□□□□□□□□ XCode □ XCode Command Line Tools□□□□□□□ □□□ `clang` □

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
$ clang -v # □□ Clang □□□□□□□□□□□□□□□□□□
```

□□□□□□□□ XCode□□□□□□□□□□□ XCode Command Line Tools□

```
$ xcode-select --install
```

LLVM

□□ XCode □□□ LLVM □□□□□□□□□□□□□□□□□□ LLVM □□□□□

```
$ brew install llvm
```

□□□□□□□□□□□□□□□□□ LLVM □□□□□□□ `$PATH` □

```
echo 'export PATH="/usr/local/opt/llvm/bin:$PATH"' >> ~/.ba
```

□□□□□□□ `zsh` □□□□ shell□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
$ lli --version # □□ LLVM □□□□□□□□□□□□□□□□□
```

Windows

> “□□□□□□□□□□□□□□ *nix □□□□□□□□ Windows □□□□”
>
> ——□□□

□□□ Windows □□□□□□□□□□ Clang+LLVM □□□□□□□□□□□

□□□□□□□□□□□□ `libsysy` □□□ □□ □□□□□□□□□□□□□□□ IO □□□□□□□

LLVM □□□□□

□□□□□□□□□□□□□□□□□□□□□ **LLVM □□□□□** □□□

LLVM □□ LLVM □□□□ LLVM IR □□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□ a + b □□□□□□□□□□

```
//main.c
int main(){
    int a = 19260817;
    int b = 42;
    return a + b;
}
```

Clang

Clang □□ LLVM project □□□□□□□□ LLVM □□□ `c/c++/obj-c` □□□□□□□□□□ GCC □□□□□

□□□□□ `clang -help` □□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□

```
# □□□□□□□
$ clang main.c -o main
# □□□□□□□
$ clang -ccc-print-phases main.c

# □□ tokens
$ clang -E -Xclang -dump-tokens main.c
# □□□□□
$ clang -fsyntax-only -Xclang -ast-dump main.c
# □□ llvm ir□□□□□□
$ clang -S -emit-llvm main.c -o main.ll -O0

# □□□□□□□□□□□□□□□
$ clang -S main.c -o main.s
# □□□□□□□□□□□□□□□□□
$ clang -c main.c -o main.o
```

□□□

lli

`lli` □□□□□ JIT □□□□□ `.bc` □ `.ll` □□□□□□ `.bc` □ LLVM IR □□□□ □□□□□□□□□ `.ll` □ LLVM IR □□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `main.c` □□□

```
# 1. □□ main.c □□□ .ll □□□□□
$ clang -S  -emit-llvm main.c -o main.ll -O0

# 2. □ lli □□□□□□□ .ll □□
$ lli main.ll
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
$ echo $?
187  # □ (19260817 + 42) % 256
```

llvm-link

`lli` □□□□□□ `.ll` □ `.bc` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `llvm-link` □

□□□□□□□□□□□ `libsysy` □□□ □□ □□□□□□□□□□□□□□ IO □□□□□□

□□□□□□□□□□□ `int` □□□□□□□□ `putint()` □□□□□□□□□□□□□ `main.c` □□□□□

```
int main() {
    int a = 19260817;
    int b = 42;

    putint(a);

    return a + b;
}
```

□□□□□□□□□□□□□□□ `lli` □□□□□ `main.c` □□□□□ `main.ll` □□□□□□□ □□□□□

```
$ lli main.ll
PLEASE submit a bug report to https://bugs.llvm.org/ and in
Stack dump:
1.      Program arguments: lli main.ll
zsh: segmentation fault (core dumped)  lli main.ll
```

□□□□ `lli` □□□□ `main.ll` □□□□□□□□□□□□□□□ `putint` □□□□□□□□□
`llvm-link` □□□□□□

`llvm-link` □□□□□ `.ll` □ `.bc` □□□□□□□□□□□□□□□□□

```
# 1. □□□□ libsysy □ main.c □□□□ .ll □□
$ clang -emit-llvm -S libsysy.c -o lib.ll
$ clang -emit-llvm -S main.c -o main.ll

# 2. □□ llvm-link □□□□□□□□□□□□ IR □□
$ llvm-link main.ll lib.ll  -o out.ll

# 3. □ lli □□□□
$ lli out.ll
19260817
```

□□□□□□□

- **llc**□□□□ `.ll` □□□ IR □□□□□□□□□□□□
- **opt**□LLVM □□□□□□□□□□□□□□□□□□□ LLVM □□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

LLVM IR □□□□

□□□□

□□□□□□□□□□LLVM □□□□□□□LLVM □□□□□□□□□□

□□□□ **LLVM IR** □□□□□□□□□□□

□□ □□□□□□ LLVM Lang Ref □□ LLVM Programmer Manual □□□

LLVM IR □□

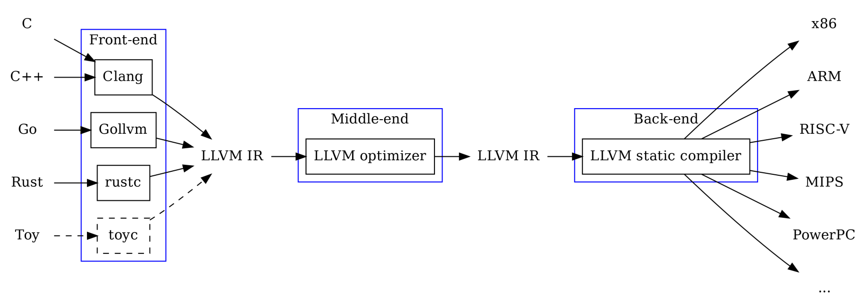
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Intermediate Representation□□□□□ IR□□□□□□ IR □□□□□□□□□□□□□□□□ MIPS □ X86□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- □□ IR □□
- □□□□□□□□ `m` □□□□□ `n` □□□ `m * n` □□□□□□□□□□□□□□□□□□□□□□ IR □□□□□□□□□□□□□□□□□□ IR □□□□□ IR □□□□□□□□□□□□□□□□□□□□□□□□□□□ `m + n` □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□front-end□□□□□□middle-end□□□□□□□back-end□□□□□□□□□□□□□□□□□

- □□□□□□□□□□□ IR
- □□□□ IR □□□□
- □□□□ IR □□□□□□□

□□□LLVM □□□□□□□□□□□□□



LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□ `.bc` □

- □□□□□□□□□□□□□□□ `.ll` □

□□□□□□□□□ `.ll` □□□ LLVM IR□

LLVM IR □□□□

□□□□□□□□□□□□□□□□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ LLVM IR □□ □□□□□□

□□□□□□□□□□□□□□□□□□□□ C □□□ `.ll` □□ LLVM IR□

```
// main.c
int foo(int first, int second) {
    return first + second;
}

int a = 5;

int main() {
    int b = 4;
    return foo(a, b);
}
```

□□□□□□□□□□ `clang -emit-llvm -S main.c -o main.ll -O0` □□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□LLVM □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `main.ll` □□□□□□□□□□□□□□□□□□□□

```
; ModuleID = 'main.c'
source_filename = "main.c"
target datalayout = "e-m:e-p270:32:32-p271:32:32-p272:64:64
target triple = "x86_64-pc-linux-gnu"

@a = dso_local global i32 5, align 4

; Function Attrs: noinline nounwind optnone sspstrong uwtab
define dso_local i32 @foo(i32 %0, i32 %1) #0 {
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  store i32 %0, i32* %3, align 4
  store i32 %1, i32* %4, align 4
  %5 = load i32, i32* %3, align 4
  %6 = load i32, i32* %4, align 4
  %7 = add nsw i32 %5, %6
  ret i32 %7
}

; Function Attrs: noinline nounwind optnone sspstrong uwtab
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 4, i32* %2, align 4
  %3 = load i32, i32* @a, align 4
  %4 = load i32, i32* %2, align 4
  %5 = call i32 @foo(i32 %3, i32 %4)
  ret i32 %5
}

attributes #0 = { noinline nounwind optnone sspstrong uwtab

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

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 7, !"PIC Level", i32 2}
!2 = !{i32 7, !"PIE Level", i32 2}
!3 = !{!"clang version 12.0.1"}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `target triple` □ `target datalayout` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `align` □□□□□□□□□□□ □□□□ `dso_local` □□□□□□□□□□□□□□□□□□□□□□ `;` □□□□□□□ LLVM IR □□□ □□□□□□□□……□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□ `.ll` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
; □□□□□□□□□ @ □□□□□□□ global □□□□□□□□□□□□□□□
@a = global i32 5 ; □□□@a □□□□ i32* □□□□□□□□

; □□□□□ `define` □□□i32 □□□□□□□□□□□□□□ `foo`□□□□□□□`@` □□□□
; □□□□ (i32 %0, i32 %1) □□□□□□□□□□□□□□□□□□□□□□□□□
define i32 @foo(i32 %0, i32 %1)  { ; □□□□□□□□□ %0□□□□ i32□□
  ; □ % □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
  %3 = alloca i32 ; □ %3 □□□□□□□□□□□□ i32 □□□□□□□%3 □□□□ i3
  %4 = alloca i32 ; □□□%4 □□□ i32*

  store i32 %0, i32* %3 ; □ %0□i32□□□ %3□i32*□
  store i32 %1, i32* %4 ; □ %1□i32□□□ %4□i32*□

  %5 = load i32, i32* %3 ; □ %3□i32*□□ load □□□□□□□□ i32□□□
  %6 = load i32, i32* %4 ; □□□□ %4□i32*□ □ load □□□□□ %6□i3

  %7 = add nsw i32 %5, %6 ; □ %5□i32□ □ %6□i32□□□□□□□□□□□ %7

  ret i32 %7 ; □□ %7□i32□
}

define i32 @main() {
  ; □□□□□□□□□ %1□%2……□□□□□□□□□□□□□□□□□□□□□□□□□□
  %1 = alloca i32
  %2 = alloca i32

  store i32 0, i32* %1
  store i32 4, i32* %2

  %3 = load i32, i32* @a
  %4 = load i32, i32* %2

  ; □□□□ @foo □i32 □□□□□□□□□□□
  ; □□□□□□□ %3□i32□□□□□□□□□ %4□i32□□□□□□□□□□□□□ %5
  %5 = call i32 @foo(i32 %3, i32 %4)

  ret i32 %5
}
```

□□

- □□□ `;` □□
- LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□ `@main` □□ `%1` □□□ `i32*` □□□□□□□ `@foo` □□ `%1` □□□ `i32` □□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□ `@main` □□□□ `%1` □ `%2` □ `%3` □
- □□□□□□□□□□□□□□□□□□□□□□□□□ `@` □□□□□□□□□ `%` □□□

- □□□□□□□□□□□□□□ `alloca` □□□□□□□□□□□ `load` □□□□□□□□ `store` □□□□□□ `add` □□□□□□□

LLVM IR □□□

□□□□□□□□□□□□□□ LLVM IR □□□□□□□□[LLVM Programmer Manual] □□□□
□□□□□□□□

□□□□

1. LLVM IR □□□□□□□□□ `module` □□□□□□□□□□□□□□ `module` □□□□□
□□□□□□□□□□□
2. □□ `module` □□□□□□□□□□□□□□ `function` □ `global variavle`
3. □□ `function define` □□□□□□ `basicblock`
4. □□ `basicblock` □□□□ `instruction` □□□□□ `terminator instruction` □□

□□□□□□□□□□ (Define&Delcare)

□□□
□□□□□ LLVM □□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `main` □□□□□□□□□

```
define i32 @main() {
    ret i32 0 ; □□ i32 □□□□ 0
}
```

□□□□□□□□□□□□□□□□□□

```
define i32 @foo(i32 %a,i32 %b) {
    ret i32 0 ; □□ i32 □□□□ 0
}
```

□□□□□□□□□□□□□□□ `define + □□□ (i32) + □□□ (@foo) + □□□□ ((i32 %a,i32 %b)) +□□□ ({ret i32 0})`

□□ `main.ll` □□□
□□ `main` □□□□□□□□ `define dso_local i32 @main() #0...` □□□□
`#0` □ `main.ll` □□□□ `attributes #0 = ...` □□□□□□□□□□□□□□□□□□
□□□ □□ □□□□□□□)

[illegible]

[illegible] `module` [illegible] `.ll` [illegible] `declare` [illegible] `define` [illegible]

```
declare i32 @getint()
declare i32 @getarray(i32*)
declare i32 @getch()
declare void @putint(i32)
declare void @putch(i32)
declare void @putarray(i32,i32*)
```

[illegible]（Basic Block）

[illegible]

[illegible]

[illegible]

[illegible]

[illegible] `if-else` [illegible] `if` [illegible] `then` [illegible] `else` [illegible] `if` [illegible]

[illegible]（Instruction）

[illegible] LLVM IR [illegible]（non-branching Instruction）[illegible] `add` [illegible] `load` [illegible]

[illegible] `call` [illegible] `call` [illegible]

[illegible]（Terminator instruction）

[illegible] `ret` [illegible] `return` [illegible] `br` [illegible] `if` [illegible]

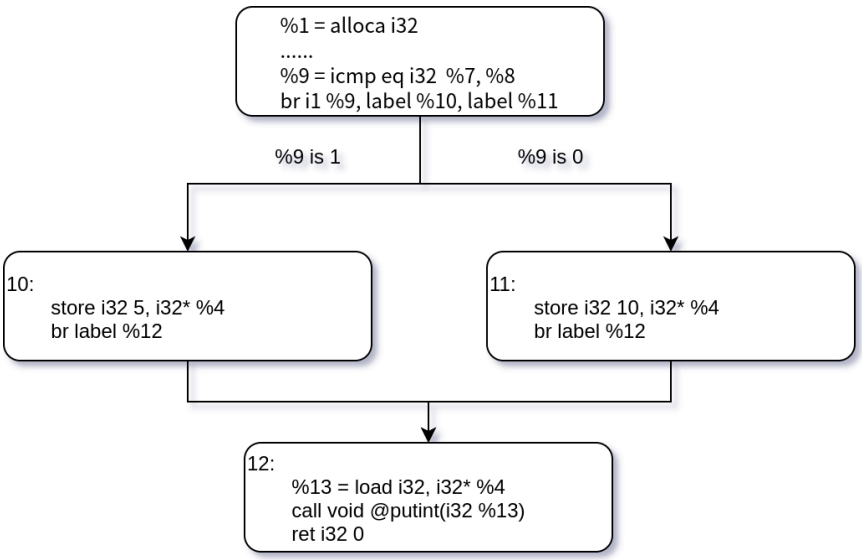
[illegible]

```
//if.c
int main() {
    int a = getint();
    int b = getint();
    int c = 0;
    if (a == b) {
        c = 5;
    } else {
        c = 10;
    }
    putint(c);
    return 0;
}
```

□ `if.c` □□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□

```
declare i32 @getint()
declare void @putint(i32)
define  i32 @main() {
  %1 = alloca i32
  %2 = alloca i32
  %3 = alloca i32
  %4 = alloca i32
  store i32 0, i32* %1
  %5 = call i32 () @getint()
  store i32 %5, i32* %2
  %6 = call i32 () @getint()
  store i32 %6, i32* %3
  store i32 0, i32* %4
  %7 = load i32, i32* %2
  %8 = load i32, i32* %3
  %9 = icmp eq i32 %7, %8
  br i1 %9, label %10, label %11

10:                                               ; preds =
  store i32 5, i32* %4
  br label %12

11:                                               ; preds =
  store i32 10, i32* %4
  br label %12

12:                                               ; preds =
  %13 = load i32, i32* %4
  call void @putint(i32 %13)
  ret i32 0
}
```

□□□□□□□□□□□□□



□□□□□ `br` □□□□□□□□□□□□□

```
br i1 %9, label %10, label %11 ; A
br label %12 ; B
br label %12 ; C
```

□□□□□□□□□□□ `br` □□□□□□ `br` □□□□□□ `br + □□□ + truelabel + falselabel` □□□ `br + label`

□□□□□□□ `A` □□□□□□□□□□□□□□□□□□□□□ `1` □□□□□□ `truelabel` □□□ `basicblock` □□□□□□□ `0` □□□□□□ `falseblock` □□□ `basicblock` □□□□□□□ `br i1 %9, label %10, label %11` □□□□ `%9` □□□ `1` □□□□□□□□□ `%10` □□□□ `0` □□□□□□□□□ `%11` □

□□□□□□□ `B,C` □□ `B,C` □□□□□□□□□□□□□□□□ `%12` □

□□□□□□ `%9` □ `icmp eq` □□ `%7` □□ `%8` □□□ `%9` □□□□□ `1` □□□□ `0` □□□□□□□□□□□□□ `if(a == b)` □ `c=5` □□□□□□ `%10` □ `c=10` □□□□□□ `%11` □□□□□□□□□□□□□□□□□□□□□□□ `%12` □□□□□□□ `putint(c)` □□ `return 0` □

□□□□

□□□ LLVM Lang Ref:type-system □□□□□□

□□□□□□□ LLVM IR □□□□□□ □□□□□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□ IR □□□□□□□□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Void Type

□□□□□□□□□□□□□□□□□□□□ □□

```
define void @foo(){
  ret void
}
```

Integer Type

□□□□□□□□□□□□□□□□□□□□□□□□□□ `i1` □□□□□ `1bit` □□ integer□□□□□□ `bool` □□ `i32` □□ `32bit` □□ integer, □□□□□□□□□□□□□□ `i1` □ `i32` □□□ □□

```
ret i32 0
br i1 %2,label  %3,label  %4
```

Label Type

□□□□□□□□□□□□□□□□

```
br i1 %9, label %10, label %11
br label %12
```

□□ Type

□□□□□□□□□□□□□□□□ `Array Type` □□□□□□□□□ `Pointer Type` □□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□

LLVM IR □□□ □□□ Lang Ref□□□□□□ □□□□□□□□□□□ Clang □□□□□□□□□□□□ github □□ issue □□□□□□□□□□□□□□□□□ □□□□□□□□□ 300 □□□

LLVM □□ SSA

□□□□□□□□□□□□□□□□□□□□□□□□□□

B □□ □□□

□□□□□□□□□□□□□□□□□ LLVM IR □□□□ □□□

□□□□□□□□□□□

□ LLVM IR □□□□□□ SSA □□□□□□□□□□□□□ LLVM IR □□□□□□□□□ SSA □□□□□□□□□□LLVM IR □□□□□□□□□□□□□□□□□□□ IR □□□□ `phi` □□□ SSA IR□□□ lab2 □□□□□□□□□□□□□□□□□□□□□□□ SSA □□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□ IR□□□□□□ `phi` □□□ SSA IR □□□□□□ □□□□□□□

SSA □□

□□□□□□Static Single Assignment, **SSA**□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ SSA □□□□

□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C++ □□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□ IR □□□□□□□□

□□□□□□□□□□□□ `1 * 2 + 3` □□□□□□□□□□□□□□□□

```
%0 = mul i32 1, 2
%0 = add i32 %0, 3
ret i32 %0
```

□□□□□□□□□□□□□□□□□□□□□□□ `%0` □□□□□□□□□□□□□□

```
%0 = mul i32 1, 2
%1 = add i32 %0, 3
ret i32 %1
```

SSA □□□□□□□□□□

□□□

SSA □□□□□□□□□□□□□□□□□□□□□□□□

```
d1: y := 1
□□□□□□
d2: y := 2
□□□□□□
d3: x := y
```

□□□□□□□□□□□□□□ `y` □□□□□□□□□□□ `x` □□□□□□ `y` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Reaching definition□□ look ahead □□□□□□□□□□□□□□□□□□□□□

- □□□□□□ `x` □□□□□□□□□□□□□□/□□□ `x` □□□□□□□□□□□ `d1` □□□□□□ `y` □□□□
- kill□□□□□□□□□□□□□□□□□□□□□□□□□ kill□□□□□□□□□ `d2` □ kill □ `d1` □□ `y` □□□
- □□□□□□□□□□ `p` □□□□ `q` □□□□□□□□□□□□□□□□□□□□□□ `p` □□□□□ `q` □□□□□ kill□
- reaching definition□ `a` □ `b` □ reaching definition □□□□□□□□□□ `a` □□ `b` □□□□□□□□□□□□□□□□□□□ `a` □□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□ `d1` □□□□ `d3` □ reaching definition, □□ `d2` □□□□□□□□□□□□

□□□ `d3` □ reaching definition □ `d1` □□ `d2` □□□□□□ `d1` □ `d2` □□□□ kill □□□□□□□□□□□□□□□□ SSA □□□□□□□□□□□□□□□□

```
d1: y1 := 1
□□□□□□
d2: y2 := 2
□□□□□□
d3: x := y2
```

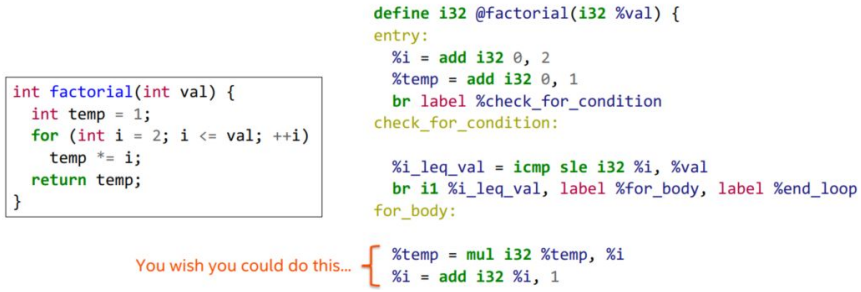
□□□□□□□□□□□□ `x` □□ `y2` □□□□□□ `y2` □□□□ 2□□ `x` □ `y2` □□□□□□□□□□□□□□ `x` □□□□□□□□□□□□□□□ `d2` □□ `d3` □ reaching definition□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

SSA □□□□□□□

□□□□□ IR □□□□□□□□□□ factorial □□□

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

□□ C □□□□□□□□□□□□□□□□□□



□□□□□□□ `%temp` □ `%i` □□□□□□□□□□□□□

□□□□

plan a —— `phi`

`phi` □□□□□□□□□□□□□□□□□□□□□□□□ `mem2reg` □□□□□□ `load/store` □□□ IR □□□ `phi` □□□ SSA□

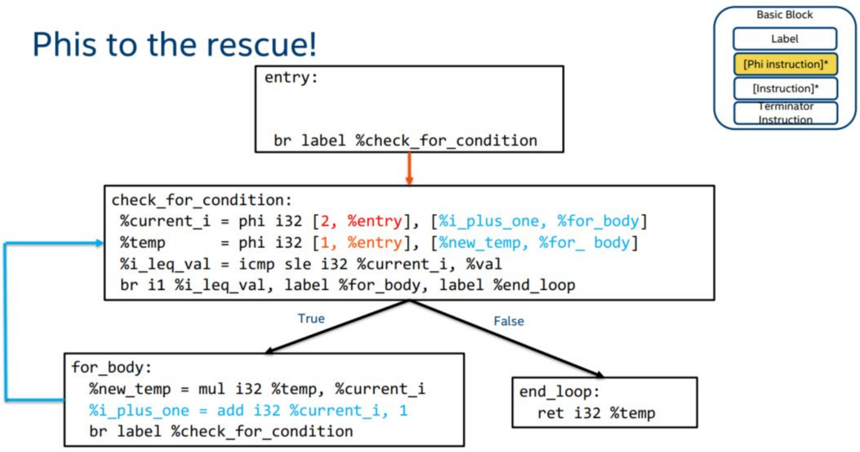
□ clang `-O1` □□□□□□□□□□ `.ll` □□□□□□□□□□□□□□□□□□□□□□□□□□

```llvm
define dso_local i32 @factorial(i32 %0) local_unnamed_addr
  %2 = icmp slt i32 %0, 2
  br i1 %2, label %3, label %5

3:                                                ; preds =
  %4 = phi i32 [ 1, %1 ], [ %8, %5 ]              ; □□□□□ %
  ret i32 %4                                      ; □ %5 □□

5:                                                ; preds =
  %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
}
```

`phi` □□□□□□□

```
<result> = phi <ty> [<val0>, <label0>], [<val1>, <label1>] ...
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `phi` □□□□□
□□□□



□□□□□□□□□□□□□□□□□□□□□

> □□□□□□□□SSA □□□□□□□□□□□□□□□□□□□□□□□□□
> □□□□

plan b —— `alloca` □ `load` □ `store`

□□□□□□□□□□□□ `alloca` □ `load` □ `store` □□□□□□□□□

□ `-O0` □□□□□□□ `.ll` □□□□□□□□□□□□□□□

```
define dso_local i32 @factorial(i32 %0) #0 {
  %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:
  %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:
  %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

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

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

`alloca` □□

□□□□□□ `.ll` □□□□□ IR □□□□□□□□□ `store` □□□□□□□□□□□□□□□□□□□□ LLVM IR □□□□□□□□□□□□□□□ LLVM IR □□□□□□□□□ SSA □□□□□□□□ SSA □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ SSA □□□□□□□□□□□ SSA □□□□□□□□□□□□□□□□□□□□□□□□□□□□ SSA □□□ IR □□□□□

□□□□□ trick□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ SSA □□□□□□□□□□□□□□ `phi` □□□□□□□□

□ LLVM □□□□□□□□□□□□□ `load/store` □□□□□□□LLVM □□□□□“□□□”□□□□□□□□□ `%3` □□□□□□□□□□□

```
entry:
    %3 = alloca i32, align 4
    ...

9:
    ...
    %11 = load i32, i32* %3, align 4
    ...
    store i32 %12, i32* %3, align 4
```

□□ `%3` □□ `alloca` □□□□□□□□ `i32` □□□□□□□□ `%3` □□□□ `i32*` □□□□□□ `%3` □□□□□□□□□□□□ `load` □ `%3` □□□□□□□□□□□□□□ `store` □□□□□□ `%3` □□□□□□ `alloca` □□□□□□□□□□□□□□□□□□□□□ `alloca` □ `load` □ `store` □□□□□□ `phi` □□□□□□

□□□□□□□□□□□□□ LLVM IR □□□□□□□□□□

1. □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `alloca` □□□
2. □□□ `load` □□□□□□□□□□
3. □□ `store` □□□□□□□□□□
4. □□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□ `phi` □□LLVM □□□□□□□ `mem2reg` □□□□□□□□□□□ `alloca` □□□□□□□□□□□ SSA □□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□ `phi` □□□

LLVM □□□□□□□□□□ `alloca + mem2reg` □□□clang □□□□□□□□□□□□□□□□□□□ IR□alloca □□□□□□□□□□□ SSA □□□□□□□□□□□ mem2reg □□□□□□□□□□□ SSA □□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□ IR□□□□□□□ `phi` □□□ SSA IR □□□□□□□□□□□□□□□

LLVM [illegible]

[illegible]

[illegible] lab2 [illegible] lab1——[illegible] [illegible] [illegible] `.ll` [illegible] LLVM IR, [illegible] LLVM IR [illegible]——[illegible] LLVM [illegible] LLVM IR [illegible]~~[illegible]~~

> [illegible]

[illegible]
[illegible]
[illegible] LLVM [illegible]

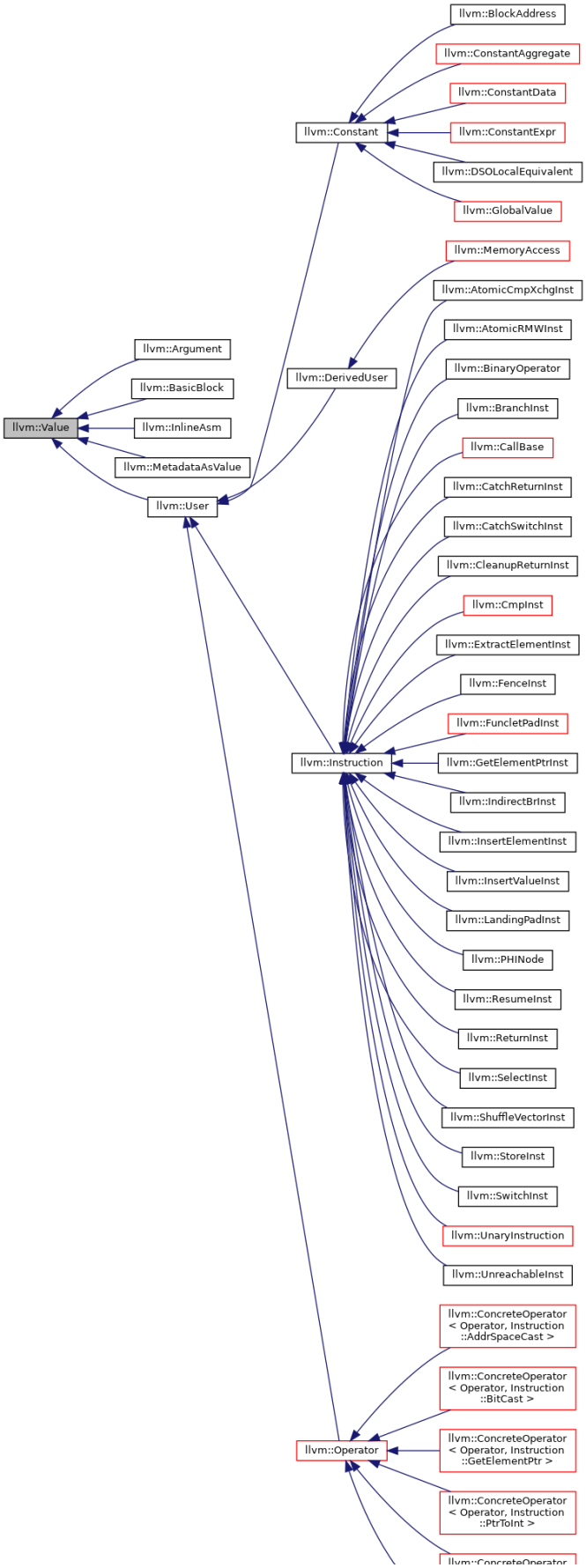
[illegible] `Value` , `Use` , `User`

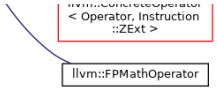
[illegible] LLVM IR [illegible]

[illegible] `Value`

□□□□□□□□□□□□ LLVM □□□□□□□□□□□□□□□□□□ `Value` □□□□□□□□□□□□

□src:https://llvm.org/doxygen/classllvm_1_1Value.html□





□□□□□□□□□□

- `BasicBlock` □ `Argument` □ `User` □□□□ `Value` □
- `Constant` □ `Instruction` □□□ `User`
- □□□□ `Function` □□□□□□ `Function` □□□□□□□□□□ `Constant` □□□□ `Function` □□ `Value` □ `User`

`BasicBlock` □□□□□□□□□ `Arugument` □□□□□□□□□□ `Constant` □□□□ □□ `i32 4` □□□□ `Instruction` □□□□□□ `add i32 %a,%b` □□□

`Value` □□□□□□□□□□□□□□□□□ `Value` □□□□□□□□□□□□□□□□□□□□ □□□□ □ `User` □□□□□□□□□□□□□□ `Value` □□ □□ `Value` □ `User` □□□□□□□ □□□□□ use-def □□ def-use □□□□□□□use-def □□□□□□ `User` □□□ `Value` □□□def-use □□□□□□ `Value` □ `User` □□□□□□□LLVM □□□□□ □□ `Use` □□ `Use` □□□□□□□□□□□□□□□□ □□□□ LLVM 2.0.0 □□□□□□□□□□ □□□□□□□□□□□□□□□

```
class Value {
    // □□□□
  private:
    Use *UseList;
}
```

```
class User : public Value {
    // □□□□
  protected:
    Use *OperandList;
    unsigned NumOperands;
}
```

```
class Use {
    // □□□□
    Use *Next, **Prev;
    Value *Val;
    User *U;
}
```

- `class Value` □□ `UseList` □□□□□□□□ **`Value`** □ **`User`** □□□□□ □□ def-use □□□
- `class User` □□ `OperandList` □□□□□ **`User`** □□□ **`Value`** □□ □□□□□□ use-def □□□

- `class Use` [illegible] `Value, User` [illegible] `User` [illegible] `OperandList` [illegible] `User` [illegible] `Value` , [illegible] `Value` [illegible] `Value` [illegible] `User` [illegible]

[illegible] `.ll` [illegible]

```
define dso_local i32 @main(){
    %x0 = add i32 5, 0
    %x1 = add i32 5, %x0
    ret i32 %x1
}
```

[illegible]

- `%x0` [illegible] `Instruction` [illegible] `OperandList` [illegible] `Constant` [illegible] `5` [illegible] `Constant` [illegible] `0` [illegible]
- `%x1` [illegible] `Instruction` [illegible] `OperandList` [illegible] `Constant` [illegible] `5` [illegible] `Instruction` [illegible] `%x0` [illegible]
- `ret` [illegible] `Instruction` [illegible] `OperandList` [illegible] `Instruction` [illegible] `%x0` [illegible]

[illegible]clang [illegible]LLVM [illegible] 0 [illegible]

[illegible] LLVM IR [illegible] lli [illegible]

```
; □□□□□□□
define dso_local i32 @main(){
    %1 = sub i32 0, 15
    %2 = sub i32 0, %1
    %3 = add i32 0, %2
    ret i32 %3
}

; □□□□□□□
define dso_local i32 @main(){
    %1 = sub i32 0, 15
    %x = sub i32 0, %1
    %2 = add i32 0, %x
    ret i32 %2
}

; □□□□□□□
; lli: test.ll:2:5: error: instruction expected to be numbe
;     %0 = sub i32 0, 15
define dso_local i32 @main(){
    %0 = sub i32 0, 15
    %1 = sub i32 0, %0
    %2 = add i32 0, %1
    ret i32 %2
}

; □□□□□□□
define dso_local i32 @main(){
  _entry:
  ; □□□□□□□□□□□□□□□
    %0 = sub i32 0, 15
    %1 = sub i32 0, %0
    %2 = add i32 0, %1
    ret i32 %2
}
```

□□□□□□□

□□□□□□□□□□□□□□LLVM□□□□□□

□□□□□□□□□□□□□□□ LLVM IR □□□□□□□□□□□□□□□ `load` □□□□□□□□

`` `<result> = load [volatile] <ty>, <ty>* <pointer>[, align <alignment>][, !nontemporal !][, !invariant.load !<empty_node>][, !invariant.group !][, !nonnull !<empty_node>][, !dereferenceable !][, !dereferenceable_or_null !<deref_bytes_node>][, !align !][, !noundef !<empty_node>** **□

□□□ LLVM IR □□□□□□□□□□ □□ LLVM Lang Ref □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□

instructions

llvm ir	usage	intro
add	`<result> = add <ty> <op1>, <op2>`	/
sub	`<result> = sub <ty> <op1>, <op2>`	/
mul	`<result> = mul <ty> <op1>, <op2>`	/
sdiv	`<result> = sdiv <ty> <op1>, <op2>`	□□□□□
icmp	`<result> = icmp <cond> <ty> <op1>, <op2>`	□□□□
and	`<result> = and <ty> <op1>, <op2>`	□
or	`<result> = or <ty> <op1>, <op2>`	□
call	`<result> = call [ret attrs] <ty> <fnptrval> (<function args>)`	□□□□
alloca	`<result> = alloca <type>`	□□□□
load	`<result> = load <ty>, <ty>* <pointer>`	□□□□
store	`store <ty> <value>, <ty>* <pointer>`	□□□
getelementptr	`<result> = getelementptr <ty>, * {, [inrange] <ty> <idx>}*` `<result> = getelementptr inbounds <ty>, <ty>* <ptrval>{, [inrange] <ty> <idx>}*`	□□□□□□□□□□□□□□□□
phi	`<result> = phi [fast-math-flags] <ty> [ <val0>, <label0>], ...`	
zext..to	`<result> = zext <ty> <value> to <ty2>`	□□□□□□□ `ty` □ `value` □ type□□□ `ty2`

terminator insts

llvm ir	usage	intro
br	`br i1 <cond>, label <iftrue>, label <iffalse>` `br label <dest>`	□□□□□
ret	`ret <type> <value>` , `ret void`	□□□□□□□□ □□□□□□□ □□

PRE

pre [illegible] pre [illegible]

[illegible]

1. [illegible] LLVM IR [illegible]
2. [illegible]
3. [illegible]
4. LLVM IR [illegible]LLVM IR [illegible] C[illegible]Java[illegible]
5. [illegible]
6. ANTLR Visitor [illegible]

flex/bison/ANTLR [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
expr -> term | expr '+' term | expr '-' term
term -> factor | term '*' factor | term '/' factor
factor -> '(' expr ')' | number
number -> [0-9]+ | [0-9]+ '.' [0-9]* | [0-9]* '.' [0-9]+
```

flex

flex□the Fast Lexical Analyzer Generator□□ Lawrence Berkeley □□ Lex□

□□ flex

flex □□□□□□ 2.6.4□□□□ 2017 □ 5 □ 6 □□

Ubuntu

```
$ sudo apt install flex
```

MacOS

```
$ brew install flex
```

Homebrew □ flex □□□□□□ 2.6.4□MacOS □□□□ flex □□□ 2.5.35□□□□□□□□□□□□□□□□□ flex□

```
$ echo 'export PATH="/usr/local/opt/flex/bin:$PATH"' >> ~/.
```

□□ □□□□□□□ `zsh` □□□□□□□ shell□□□□□□□□□□□□□□□□□□□□ `.zshrc` □□

Windows & other Linux

□□□

flex □□□

flex □□□□□□ `.l` □□ flex □□□

□□ C □□

flex □□□□ `*.l` □□ `*.lex` □□□□□□□□□□□□□□□□□□ `lex.yy.c` □

□□□□□□□□□□ `%{` □ `%}` □□□□□□□□□□□ flex □□□□□ `lex.yy.c` □□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
/* word_char_counter.l */
/* □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ */
%{
#include <string.h>
int chars = 0;
int words = 0;
%}
```

□□□□□□□□□□□ `%%` □ `%%` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ C
□□

```
/* word_char_counter.l */
/* □□□□□□□□□□□□□□□□□□□ */
%%
[a-zA-Z]+ { chars += strlen(yytext); words++; }
. { }
%%
```

□□□□□□□□□□□ `yytext` □□□□□□□□□□□□□□□□□□□□□□

□□□□ flex □ bison□□

```
/* word_char_counter.l */
int main(int argc, char **argv) {
    yylex();
    printf("I found %d words of %d chars.\n", words, chars)
    return 0;
}
```

□□ `yylex()` □ flex □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ stdin□□

□□□□□□□□□□□□ flex □□□□□ C □□□□□

```
$ flex word_char_counter.l
$ gcc lex.yy.c -o word_char_counter
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
/usr/bin/ld: /tmp/cc1qil64.o: in function `yylex':
lex.yy.c:(.text+0x4b8): undefined reference to `yywrap'
/usr/bin/ld: /tmp/cc1qil64.o: in function `input':
lex.yy.c:(.text+0x10c7): undefined reference to `yywrap'
collect2: error: ld returned 1 exit status
```

□□□□□□□□□□□□□□□ `yywrap()` □□□□□□□□□□□□□□□ flex 2.5.4 □□□□□□□□□□ □□ `EOF` □□□□□ `yywrap()` □□□□□□□□□□□□□□□□□□□□ `yywrap()` □□ 0□□ □□□□□□□□□□□□□□□ `yywrap()` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `yywrap()` □□ 1□□□□□□□□□□□□□

□□□□□□□□□□ `fl` □□□□□□□□□□□□ 1 □ `yywrap()` □□□□□□□□□□□□□□□□□ stdin□□□□□ `gcc lex.yy.c -o word_char_counter -lfl` □□□□□□□□□ □□□□□□□□□□ `%option noyywrap` □□□□□□□ `yywrap()` □□□□□□□□□□□

```
$ ./word_car_counter
Hello, flex.
^D
I found 2 words of 9 chars.
```

□□ C++ □□

□□ flex □□□□□□□

```
/* word_char_counter_cpp.l */
%option c++
%option noyywrap

%{
#include <cstring>
int chars = 0;
int words = 0;
%}

%%
[a-zA-Z]+ { chars += strlen(yytext); words++; }

. { }
%%

int main(int argc, char **argv) {
    FlexLexer* lexer = new yyFlexLexer();
    lexer->yylex();
    std::cout << "I found " << words << " words of " << cha
    return 0;
}
```

□□ flex □□ C++ □□□□□□□□□□□□□□□□□□□□ `%option c++` □□□□□□□□□□□□□□□□ `-+` □□ `flex word_char_counter_cpp.l -+` □

□□□□□□□□□□□□□ flex □□□□□ C++ □□□□□

```
$ flex word_char_counter_cpp.l
$ g++ lex.yy.cc -o word_char_counter_cpp
```

□□□□□□□□□□□□□□□ C □□□□□□□□□□□□□□□

```
$ ./word_car_counter_cpp
Hello, flex.
^D
I found 2 words of 9 chars.
```

bison

bison □□□□□□□□□□□□□□□□□□□□ yacc□yacc □□□□□□□□ S.C.Johnson □□ LR □□□□□□□□□□□□□ 1975 □ 1978 □□□□□□□□ 1985 □□UC Berkeley □□□□ Bob Corbett □□□□□□□□□□□□□□□□ yacc□□□□□ FSF □ Richard Stallman □ □□□□□ yacc□□□□□□□□□□□□□□□□□□ GNU □□□□□□□□□□□□ GNU bison□

□□ bison

bison □□□□□□ 3.7.90□□□□ 2021 □ 8 □ 13 □□

Ubuntu

Ubuntu 20.04 □□□□□ bison □□□□□□ 3.5.1□Ubuntu 18.04 □□□□□ bison □ □□□□□ 3.0.4□□□□□□□□□□□□□□ bison□□□□□□□

```
$ sudo apt install bison
```

MacOS

```
$ brew install bison
```

Homebrew □ bison □□□□□□ 3.8.1□MacOS □□□□ bison □□□ 2.3□□□□□□□ □□□□□□□□□□ bison□

```
$ echo 'export PATH="/usr/local/opt/bison/bin:$PATH"' >> ~/
```

□□ □□□□□□□ `zsh` □□□□□□□ shell□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `.zshrc` □□

Windows & other Linux

□□□□

bison □□□

bison □□□ flex □□□□□flex □□□□□□□bison □□□□□□□□□□□□□□□□□□ flex □□ □□□□□□□□□□□□ token □□□□□ bison □□□□□□□□□□□□□□□□□□ token □□□ □□□□□□□□□□□□□□□ token □□

bison □□□□□□□□□ `.y` □□ □□□□□□□□□□ bison □ flex □□□□□□□□□□□□□□□□□□□□□□□□□□□

```
expr -> term | expr '+' term | expr '-' term
term -> factor | term '*' factor | term '/' factor
factor -> '(' expr ')' | number
number -> [0-9]+ | [0-9]+ '.' [0-9]* | [0-9]* '.' [0-9]+
```

□□ C □□

□□□□□ flex □□□ `calc.l` □□□

```
/* calc.l */
%option noyywrap

%{
#include "calc.tab.h"
%}

/* □□□□□□□□□□□□ token */
%%
\( { return LPAREN; }
\) { return RPAREN; }
"+"|"-" { yylval.op = yytext[0]; return ADDOP; }
"*"|"/" { yylval.op = yytext[0]; return MULOP; }
[0-9]+|[0-9]+\.[0-9]*|[0-9]*\.[0-9]+ { yylval.num = atof(yy
" "|\t {  }
\r\n|\n|\r { return RET; }
%%
```

- `LPAREN, RPAREN, ADDOP` □ token □□□□□□□□ bison □□□□□□□□□□□ □□bison □□□□□□□□□□□□□□□□□□□□□ `calc.tab.h` □□□
- `yylval` □ flex □□□□□□□□□□ flex □ bison □□□□□□□□□□□□ `int` □□□□ `yylval` □□□□ `union { char op; double num; }` □□□□□□□□□ `char` □□□□□□□ `num` □□□□□□□ bison □□□□□□□□□□□□□

□□□□□□□ bison □□□□ `calc.y` □

□□□□□□□□□□□ `%{` □ `%}` □□□□□□ bison □□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□

```
/* calc.y */
%{
#include <stdio.h>
int yylex(void);
void yyerror(const char *s);
%}
```

□□□□□ `yylex()` □ `yyerror(const char *)` □□□□□□□ `yylex` □□□□ flex □□□ `yyerror` □□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□ bison □□□□□□□□□□□

```
/* calc.y */
%union {
    char   op;
    double num;
}

%token RET
%token <num> NUMBER
%token <op> ADDOP MULOP LPAREN RPAREN
%type <num> line expr term factor
```

□□□□□□ token □□□□□□□ token □□□□□□□□□□□□□□□□□□ □ `yylval` □□□□□□

□□□□□□□□□□ `%%` □ `%%` □□□□□□□□□□□□□□□□□□□□□□□□ `:` □□□□ BNF □□□□ `->` □ `::=` □□□□□□□□□□□□□□ `|` □□□□□ `;` □□□□□□□□□□□□□□□□□□□□□□□□□□□□ C □□□□□□□□□□□□□□□□□□□□□

bison □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ bison □□ bison □□□□□□□□□

□□□□□□□□□□□ bison □□□□

```
/* calc.y */
%%

calculator
: calculator line { }
| { }

line
: expr RET
{
    printf(" = %f\n", $1);
}

expr
: term
{
    {% math %} = $1;
}
| expr ADDOP term
{
    switch ($2) {
        case '+': {% endmath %} = $1 + $3; break;
        case '-': {% math %} = $1 - $3; break;
    }
}

term
: factor
{
    {% endmath %} = $1;
}
| term MULOP factor
{
    switch ($2) {
        case '*': {% math %} = $1 * $3; break;
        case '/': {% endmath %} = $1 / $3; break;
    }
}

factor
: LPAREN expr RPAREN
{
    {% math %} = $2;
}
| NUMBER
{
    {% endmath %} = $1;
}

%%
```

□□

- `$$` □□□□□□□□□□□□
- `$1, $2, ..., $n` □□□□□□□□□□□□□□□□□□□□□□□□□□ `calc.l` □□□□ `yylval` □□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `yyerror` □□□

```
/* calc.y */
void yyerror(const char *s) {
    fprintf(stderr, "%s\n", s);
}
```

□□□□□□□□□□□□□ flex □ bison □□□□□□□□□□□□□□□flex □□□□□□□□□□ □ `lex.yy.c` □bison □□□□□□□□□□□□ `calc.tab.c, calc.tab.h` □

```
$ flex calc.l
$ bison -d calc.y
# -d □□□□□□□□□□□□□□ flex □□
```

□□□□□□□□□□□□□ bison □□□ `yyparse()` □□□□□□□□□□□□

```
/* driver.c */
int yyparse();

int main() {
    yyparse();
    return 0;
}
```

□□□□□□□□□□□□□□□□□□□

```
$ gcc lex.yy.c calc.tab.c driver.c -o calc
```

□□□□□□□□□□□□□□□□□□□□□□

```
$ ./calc
1919 * 810
 = 1554390.000000
123.456 - 654.321
 = -530.865000
4. * .6
 = 2.400000
1 + 1 * 4
 = 5.000000
(5 - 1) * 4
 = 16.000000
6 * 0 -
syntax error
```

□□ C++ □□

> □□ □□□□

ANTLR

ANTLR□ANother Tool for Language Recognition□□□□□□□□□□□□□□□□□□□□ LL(*) □□□□□ANTLR □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

ANTLR □□□□□□□□□□□□□ Java□C++□C#□Python□Go□JavaScript□Swift □□

□□ ANTLR

ANTLR □□ Java □□□□□□ ANTLR □□□□ ANTLR □□ □□□□□ jar □□□□□□□□□□□□□ jar □□□□ ANTLR □□□□□□□ ANTLR □□□ java □□□□□□□□□□□□□□□□□ ANTLR □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□ jar □□□□□□□□□□□□□

```
$ mkdir antlr && cd antlr
$ curl -O https://www.antlr.org/download/antlr-4.9.2-comple
```

□□□□□□□ Java □□ jar □□□□□□ ANTLR□□□

```
$ java -jar antlr-4.9.2-complete.jar
```

ANTLR □□□□□□ `.g4`

ANTLR □□□□□□□□ `.g4` □ANTLR □□□ `.g4` □□□□□□□□□□□□□□□□□□□□□□□□

□ `.g4` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
// calc.g4
grammar calc;
```

ANTLR □□□□□ C □□□□□□□□ `//` □ `/**/` □□□□□□□

□□□□□□□□□□□ANTLR □□□□□□□□□□□□□□□□□□ bison □□□ANTLR □□ `:` □□ BNF □□□□ `->` □ `::=` □□□□□□/□□□□□□□□□□□□ `|` □□□□□ `;` □□□□□□□/□□□□□□□□□□□□

```
// calc.g4
LPAREN: '(';
RPAREN: ')';
ADD: '+';
SUB: '-';
MUL: '*';
DIV: '/';
NUMBER: [0-9]+ | [0-9]+ '.' [0-9]* | [0-9]* '.' [0-9]+;
RET: '\r\n' | '\n' | '\r';
WHITE_SPACE: [ \t] -> skip; // -> skip □□□□□□□□□□□
```

□□□□□□□□□□□ANTLR □□□

```
// calc.g4
calculator: line*;
line: expr RET;
expr: expr ADD term | expr SUB term | term;
term: factor | term MUL factor | term DIV factor;
factor: LPAREN expr RPAREN | NUMBER;
```

□□□□□ANTLR □□ LL(*) □□ ANTLR □□□ `expr: expr ADD term | expr SUB term | term;` □□□□ `term` □□□ ANTLR □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

> ANTLR □□
>
> ```
> // calc.g4
> calculator: line*;
> line: expr RET;
> expr: expr MUL expr | expr DIV expr | expr ADD expr |
> ```
>
> ANTLR □□□□□□□□□□□□□□□□□□□□□□□□□ https://github.com/antlr/grammars-v4

□□ ANTLR □□□□

□□□□ ANTLR □□□□□□□□□ `.g4` □□□□□□□□□□□ ANTLR□□□□□□□□□□□□□□□□□□□ Java □□□

```
$ java -jar antlr-4.9.2-complete.jar calc.g4
```

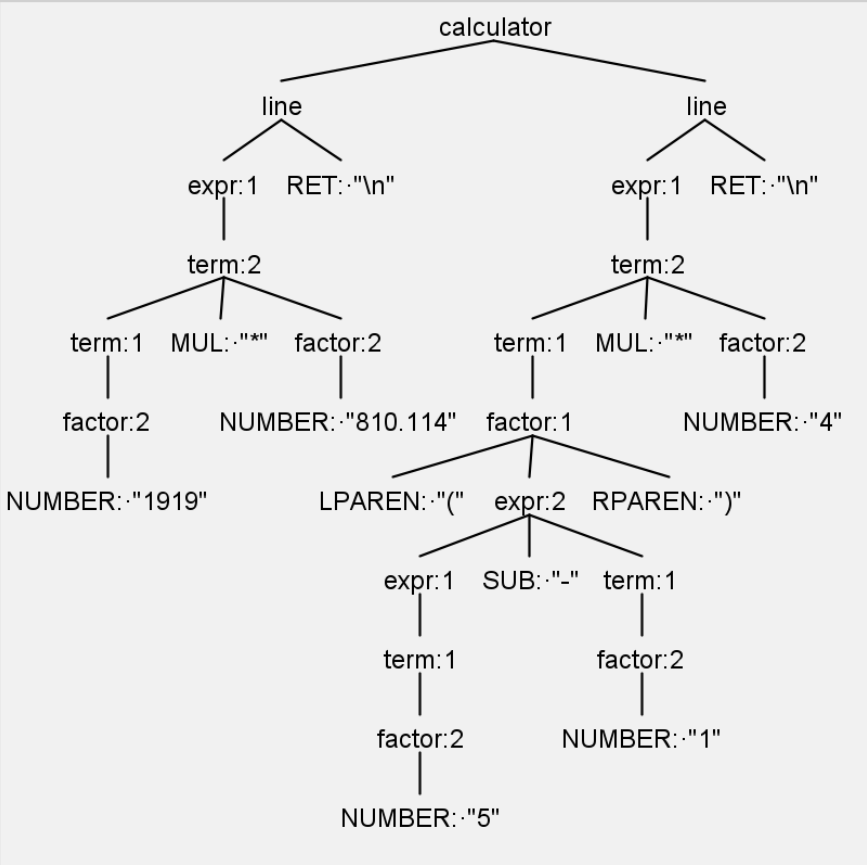
□□□□□□□□□□□□□□□□□□□□□ ANTLR □□□ `-Dlanguage=` □□□□□□

```
# □□ C++ □□
$ java -jar antlr-4.9.2-complete.jar -Dlanguage=Cpp calc.g4
```

ANTLR □□□

```
1919 * 810.114
(5 - 1) * 4
```

□□□□□□□□□□□□



□□□□□

Listener □□□ Visitor □□

ANTLR □□□ Listener □ Visitor □□□□□□□□□□□□□□□□□□□□□□□ Listener □□□□ □□□□□□□□ Vistor □□□□□□□□□□□□□□□□ `-visitor` □□□□□□□□□ Listener □□ □□□□□□□□□□□□□□□ `-no-listener` □

□□□□□ Java □□□□□□□□□□

□□□ANTLR □□□ `calc.interp` □ `calc.tokens` □ `calcBaseListener.java` □ `calcLexer.interp` □ `calcLexer.java` □ `calcLexer.tokens` □ `calcListener.java` □ `calcParser.java` □□□ Visitor □□□ `calcBaseVisitor.java` □ `calcVisitor.java` □□ `calcLexer.java` □□□ `calcParser.java` □□□ `*.tokens` □□□ token □□□ `*.interp` □□□ ANTLR □□□ IDE □□□

□□□ `calcListener.java` □ `calcVisitor.java` □□□ Listener □□□ Visitor □□□ `calcBaseListener.java` □ `calcBaseVisitor.java` □□□

□□□ ANTLR □□□ `BaseListener` □ `BaseVisitor` □□□ IR □□□

Listener □□□ enterXXX □□□ exitXXX □□□ `void enterExpr(calcParser.ExprContext ctx)` □ `void exitExpr(calcParser.ExprContext ctx)` □□□ enter □□□ exit □□□ enter □ exit □□□

Vistor □□□ visitXXX □□□ `T visitExpr(calcParser.ExprContext ctx)` □□□ `Visitor` □□□ `visit` □□□ `visit` □□□ visitXXX □□□ visitXXX □□□ `visit` □□□

□□□Listener □□□ Vistor □□□

Listener □□□ Vistor □□□ Visitor □□□

□□ ANTLR □□□

□□ ANTLR □□ Java □□

□□□ ANTLR □□□ `CLASSPATH` □□□ ANTLR □□□Java □□□ `antlr-4.9.2-complete.jar` □□□

□□□

```
// Main.java
import org.antlr.v4.runtime.tree.ParseTree;
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;

public class Main {
    public static void main(String[] args) {
        String input = "1919 * 810\n" + "123.456 - 654.321\

        CharStream inputStream = CharStreams.fromString(inp
        calcLexer lexer = new calcLexer(inputStream);
        CommonTokenStream tokenStream = new CommonTokenStre
        calcParser parser = new calcParser(tokenStream);
        ParseTree tree = parser.calculator(); // □□□□□□□□□
        System.out.println(tree.toStringTree(parser)); // □
    }
}
```

□ `CLASSPATH` □□□□ ANTLR □□□□□□□□□□□□□□□ `javac` □□□ `Main.java` □

```
$ echo $CLASSPATH
.:/usr/local/lib/antlr-4.9.2-complete.jar:

$ ls
Main.java    calc.tokens           calcBaseVisitor.java  c
calc.interp  calcBaseListener.java  calcLexer.interp      c
$ javac Main.java
```

□□□□□□□□□□□□□□□□□□□□

```
$ java Main
(calculator (line (expr (term (term (factor 1919)) * (facto
```

> □ IDE □□□□□□□□□□□□□□□

□□ ANTLR □□□ C++ □□

□□□□ ANTLR C++ Runtime □□□□□ `runtime/src/` □□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□

```
// main.cpp
#include "calcLexer.h"
#include "calcParser.h"
#include <iostream>

using namespace std;
using namespace antlr4;

int main(int argc, char *argv[]) {
    string input("1919 * 810\n123.456 - 654.321\n4. * .6\n1

    ANTLRInputStream inputStream(input);
    calcLexer lexer(&inputStream);
    CommonTokenStream tokenStream(&lexer);
    calcParser parser(&tokenStream);
    tree::ParseTree *tree = parser.calculator();
    cout << tree->toStringTree(&parser) << endl;

    return 0;
}
```

□□ `CMakeLists.txt` □□□

```
# CMakeLists.txt
project(antlr-calculator CXX)
cmake_minimum_required(VERSION 3.1)
file(GLOB_RECURSE DIR_SRC "src/*.cpp")
file(GLOB_RECURSE DIR_LIB_SRC "third_party/*.cpp")
include_directories(src/)
include_directories(third_party/antlr-runtime)
add_executable(main ${DIR_SRC} ${DIR_LIB_SRC})
```

□□ cmake □□□□□□

```
$ mkdir build && cd build
$ cmake ..
$ make
```

□□□□□□□□□□□□□□□□□□□□

```
$ ./main
(calculator (line (expr (term (term (factor 1919)) * (facto
```

> □□□ https://github.com/kobayashi-compiler/kobayashi-compiler

□□ ANTLR □□□□□□□□□□□□□□ Java □□□

□□□□□□ Java □□□□□□□□□ ANTLR □□□□□□□□□□□□□□□□□

□□□□□□ `Visitor` □□□□ `calcBaseVisitor` □□□□□□□□□□ visitXXX □□□□□□□□□□□□□ `Void` □□□□ visitXXX □□□□□□□□□□□□□□□□

```
// Visitor.java
public class Visitor extends calcBaseVisitor<Void> {
    @Override
    public Void visitCalculator(calcParser.CalculatorContex
        return super.visitCalculator(ctx);
    }

    @Override
    public Void visitLine(calcParser.LineContext ctx) {
        return super.visitLine(ctx);
    }

    @Override
    public Void visitExpr(calcParser.ExprContext ctx) {
        return super.visitExpr(ctx);
    }

    @Override
    public Void visitTerm(calcParser.TermContext ctx) {
        return super.visitTerm(ctx);
    }

    @Override
    public Void visitFactor(calcParser.FactorContext ctx) {
        return super.visitFactor(ctx);
    }
}
```

□□

```
public class Visitor extends calcBaseVisitor<Void> {
    private double nodeValue = 0.0;

    @Override
    public Void visitCalculator(calcParser.CalculatorContex
        // □□□□□ visit □□□□
        return super.visitCalculator(ctx);
    }

    @Override
    public Void visitLine(calcParser.LineContext ctx) {
        // visit expr □□□□□□□□□ nodeValue
        visit(ctx.expr());
        System.out.println(" = " + nodeValue);
        return null;
    }

    @Override
    public Void visitExpr(calcParser.ExprContext ctx) {
        switch (ctx.children.size()) {
            case 1 -> {
                // □ 1 □□□□□□□□□□□□□□ expr -> term□□□ visit
                visit(ctx.term());
            }
            case 3 -> {
                // □ 3 □□□□□□□□□□□□□□ expr -> expr ADD term
                // visit expr □ term □□□□□□□□□□□□□□□□□□□□□□□□
                double lhs = 0.0, rhs = 0.0, result = 0.0;

                visit(ctx.expr());
                lhs = nodeValue;

                visit(ctx.term());
                rhs = nodeValue;

                if (ctx.ADD() != null) {
                    result = lhs + rhs;
                } else {
                    result = lhs - rhs;
                }

                nodeValue = result;
            }
        }
        return null;
    }

    @Override
    public Void visitTerm(calcParser.TermContext ctx) {
        switch (ctx.children.size()) {
            case 1 -> {
```

```
                // □ 1 □□□□□□□□□□□□□□□ term -> factor□□□ visi
                visit(ctx.factor());
            }
            case 3 -> {
                // □ 3 □□□□□□□□□□□□□□□ term -> term MUL facto
                // visit term □ factor □□□□□□□□□□□□□□□□□□□□□□
                double lhs = 0.0, rhs = 0.0, result = 0.0;

                visit(ctx.term());
                lhs = nodeValue;

                visit(ctx.factor());
                rhs = nodeValue;

                if (ctx.MUL() != null) {
                    result = lhs * rhs;
                } else {
                    result = lhs / rhs;
                }

                nodeValue = result;
            }
        }
        return null;
    }

    @Override
    public Void visitFactor(calcParser.FactorContext ctx) {
        switch (ctx.children.size()) {
            case 1 -> {
                // □ 1 □□□□□□□□□□□□□□□ factor -> NUMBER□□ NUM
                nodeValue = Double.parseDouble(ctx.NUMBER()
            }
            case 3 -> {
                // □ 3 □□□□□□□□□□□□□□□ factor -> LPAREN expr
                visit(ctx.expr());
            }
        }
        return null;
    }
}
```

□□ `Main.java` □□□□□□□□ Visitor□

```java
// Main.java
import org.antlr.v4.runtime.tree.ParseTree;
import org.antlr.v4.runtime.CharStream;
import org.antlr.v4.runtime.CharStreams;
import org.antlr.v4.runtime.CommonTokenStream;

public class Main {
    public static void main(String[] args) {
        String input = "1919 * 810\n" + "123.456 - 654.321\

        CharStream inputStream = CharStreams.fromString(inp
        calcLexer lexer = new calcLexer(inputStream);
        CommonTokenStream tokenStream = new CommonTokenStre
        calcParser parser = new calcParser(tokenStream);
        ParseTree tree = parser.calculator();
        Visitor visitor = new Visitor();
        visitor.visit(tree);
    }
}
```

□□□□□□□□□□□□□□□□

```
1919*810
 = 1554390.0
123.456-654.321
 = -530.865
4.*.6
 = 2.4
1+1*4
 = 5.0
(5-1)*4
 = 16.0
```

□ ANTLR □□□□□□□□□□ `Context` □□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `ExprContext` □□□□ `expr()` □□□□□□□□ `expr: expr ADD term | expr SUB term | term;` □□□ `ExprContext` □□□□□□□□□ `expr: expr ADD term | expr SUB term;` □□ `expr()` □□□□□□□□□□□□□ `ExprContext` □□□□□□□□□ `expr: term;` □□ `expr()` □□ `null` □

□□ `Context` □□□□□ `accept` □□□□ visitor□□□□□□□□ `calcBaseVisitor` □□□ `AbstractParseTreeVisitor` □□□□ `accept` □□□□□□□□□ `visit` □□□□□□□□□□□□□□□□□□□□□□ □□ `AbstractParseTreeVisitor` □□□□□□□□ `visitChildren` □□□□□□□ □□□□□□□□□□□□

> □□□□ ANTLR □□ □□□□ ANTLR □□□□□

□□□□□□□

Rurikawa □□□□□github□

□□□□□□□

□□□□□□□□□□□□□□□□□□□□ pdf □□□□□□□□□□□ `pre/□□□□□□/` □□□□□□□□□□ □□□□□□□□ `□□_□□_labLexer.pdf` □

□□□□□□□□□□□□□□□□□ 2021 □ 10 □ 10 □ 23:59□

□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ flex/ANTLR □□□□□□□□□□□□□□□□□□ □□□□□ Token □□□□□□□□□□ token □□□□□□□□□stdout□□□□□□□□□ token□

Token □□□□□□□

Token □□	□□□□□□	□□□□□	□□
□□□	□□□□□□□	`Ident($name)`	□ `$name` □□□□□□□□□□ □□□□
□□□□□	□□□□□□□	`Number($number)`	□ `$number` □□□□□□□□□□ □□□□
if	`if`	`If`	
else	`else`	`Else`	
while	`while`	`While`	
break	`break`	`Break`	
continue	`continue`	`Continue`	
return	`return`	`Return`	
□□□□	`=`	`Assign`	
□□	`;`	`Semicolon`	
□□□	`(`	`LPar`	
□□□	`)`	`RPar`	
□□□□	`{`	`LBrace`	
□□□□	`}`	`RBrace`	
□□	`+`	`Plus`	
□□	`*`	`Mult`	
□□	`/`	`Div`	
□□□	`<`	`Lt`	
□□□	`>`	`Gt`	
□□□	`==`	`Eq`	
□□	□□□□□□□ token □□□□□ □	`Err`	□□□□□ `Err` □□□

□□□□□□□□□□□□□□□□□□□□□

```
Letter -> 'a' | 'b' | 'c' | 'd' | 'e' | 'f' | 'g' | 'h' | '
    | 't' | 'u' | 'v' | 'w' | 'x' | 'y' | 'z' | 'A' | 'B' |
    | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R' | 'S' | 'T' | 'U' |

Digit -> '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8

Underline -> '_'

Nondigit -> Letter | Underline

<□□□> -> Nondigit | <□□□> Nondigit | <□□□> Digit

<□□□□□> -> Digit | <□□□□□> Digit
```

□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□ `0 <= number <= 2147483647` □□□□□□□□□□□□□□
- □□□□□□□□□□□□ token □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ gcc □ clang □□ `a = 1 0;` □□□□□□□□□□□□□ `a=10;` □□□□□□□□ □□ token□□□□□□□□□□□□□□ `a = 1 0;` □□□□□□□□ `Number(1)` □ `Number(0)` □
- □□□□□□□□□□□□□□□□ `===` □□□□□□ `Assign\nEq` □ `Eq\nAssign` □ `Assign\nAssign\nAssign` □□□□ □□□□□□□□□□□□□□□□□□□□ token □□□□□□□□□□ `===` □□□□□ `Eq\nAssign` □

□□

□□□□ 1□

```
a = 10;
c = a * 2 + 3;
return c;
```

□□□□ 1□

```
Ident(a)
Assign
Number(10)
Semicolon
Ident(c)
Assign
Ident(a)
Mult
Number(2)
Plus
Number(3)
Semicolon
Return
Ident(c)
Semicolon
```

□□□□ 2□

```
a = 10;
:c = a * 2 + 3;
return c;
```

□□□□ 2□

```
Ident(a)
Assign
Number(10)
Semicolon
Err
```

□□□□ 3□

```
a = 3;
If = 0
while (a < 4396) {
    if (a == 010) {
        ybb = 233;
        a = a + ybb;
        continue;
    } else {
        a = a + 7;
    }
    If = If + a * 2;
}
```

□□□□ 3□

```
Ident(a)
Assign
Number(3)
Semicolon
Ident(If)
Assign
Number(0)
While
LPar
Ident(a)
Lt
Number(4396)
RPar
LBrace
If
LPar
Ident(a)
Eq
Number(010)
RPar
LBrace
Ident(ybb)
Assign
Number(233)
Semicolon
Ident(a)
Assign
Ident(a)
Plus
Ident(ybb)
Semicolon
Continue
Semicolon
RBrace
Else
LBrace
Ident(a)
Assign
Ident(a)
Plus
Number(7)
Semicolon
RBrace
Ident(If)
Assign
Ident(If)
Plus
Ident(a)
Mult
Number(2)
```

```
Semicolon
RBrace
```

□□

□□□ 10 □ 1 □□□□10 □ 10 □ 23:59 □□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□

`Dockerfile` □ `judge.toml` □□□□□□ □□□□□□□□□□□□□

□□□□□□□□□

```
# -- Dockerfile --
# □□□□□□□□□□□□□□□ Docker □□

# □□ Java 12
FROM openjdk:12-alpine
# □□□□□□□□
COPY ./* /app/
# □□□□
WORKDIR /app/
RUN javac -d ./output ./my/path/MyClass.java
# □□□□□□□ /app/output
WORKDIR /app/output
```

```
# -- judge.toml --
# □□□□□□□□□□□□□□□□□□□□□□

# □□□□□□□□□ lexer
[jobs.lexer]
# □□ Dockerfile □□□□□□□□□□□□
image = { source = "dockerfile", path = "." }

# □□□□□□ Java
run = [
  # □□□□
  "java my.path.MyClass $input",
]
```

□□□□□□□□□□□ `$input` □□□□□□□□□□□□□□□□□□□□□□ □□□□ `cat $input | <□□□□>` □□□□□□

□□□□□□miniSysY

miniSysY □□□□ SysY □□ □□□□□□□□□□□□□ C □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `main()` □□□□□□□□□□□ miniSysY □□□ miniSysY □□□□□ LLVM IR□

□□□□□□□□□□□□□□□□□□□□□□ ANTLR□flex/bison □□□

□□□□□□□□□ `$input` □□□□□□□□□□□□□□□□□□□ LLVM IR □□□ `$ir` □□□□□□□□ `lli` □□□□□□□□□□□□□□□□ IR □□□□□□□□□ `judge.toml` □ `run` □□□ `$input` □□□□□□□□□ `$ir` □□□□□□□□□□□□ `./compiler < $input > $ir` □ `./compiler $input $ir` □□

□□□□

miniSysY □□□□ 8 □ lab □□□□□□□□□

8 □ lab □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 17 □□□□2022 □ 1 □ 2 □□□□□□□□□□□□□□□□□ lab □□□□ part□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ part □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ part □□□□□□□□□□ lab□

□□□

- □□ `main` □□□□□□□□□□□10%□
- □□□□□□□□□10%□
- □□□□□□□□□□□10%□
- `if` □□□□□□□□□□10%□
- □□□□□□□□□□10%□
- □□□10%□
- □□□10%□
- □□□10%□
- □□□□□20%□
  - `mem2reg` □20%□
  - □□□□□10%□
  - □□□□□10%□
  - □□□□□10%□

miniSysY □□

□□□□□□□□□□□□ miniSysY □□□□□□□□□

```
CompUnit     -> [CompUnit] (Decl | FuncDef)
Decl         -> ConstDecl | VarDecl
ConstDecl    -> 'const' BType ConstDef { ',' ConstDef } ';'
BType        -> 'int'
ConstDef     -> Ident { '[' ConstExp ']' } '=' ConstInitVal
ConstInitVal -> ConstExp
                | '{' [ ConstInitVal { ',' ConstInitVal } ]
VarDecl      -> BType VarDef { ',' VarDef } ';'
VarDef       -> Ident { '[' ConstExp ']' }
                | Ident { '[' ConstExp ']' } '=' InitVal
InitVal      -> Exp
                | '{' [ InitVal { ',' InitVal } ] '}'
FuncDef      -> FuncType Ident '(' [FuncFParams] ')' Block
FuncType     -> 'void' | 'int'
FuncFParams  -> FuncFParam { ',' FuncFParam }
FuncFParam   -> BType Ident ['[' ']' { '[' Exp ']' }]
Block        -> '{' { BlockItem } '}'
BlockItem    -> Decl | Stmt
Stmt         -> LVal '=' Exp ';'
                | [Exp] ';'
                | Block
                | 'if' '(' Cond ')' Stmt [ 'else' Stmt ]
                | 'while' '(' Cond ')' Stmt
                | 'break' ';'
                | 'continue' ';'
                | 'return' [Exp] ';'
Exp          -> AddExp
Cond         -> LOrExp
LVal         -> Ident {'[' Exp ']'}
PrimaryExp   -> '(' Exp ')' | LVal | Number
UnaryExp     -> PrimaryExp
                | Ident '(' [FuncRParams] ')'
                | UnaryOp UnaryExp
UnaryOp      -> '+' | '-' | '!'  // □□□□ '!' □□□□ Cond □
FuncRParams  -> Exp { ',' Exp }
MulExp       -> UnaryExp
                | MulExp ('*' | '/' | '%') UnaryExp
AddExp       -> MulExp
                | AddExp ('+' | '−') MulExp
RelExp       -> AddExp
                | RelExp ('<' | '>' | '<=' | '>=') AddExp
EqExp        -> RelExp
                | EqExp ('==' | '!=') RelExp
LAndExp      -> EqExp
                | LAndExp '&&' EqExp
LOrExp       -> LAndExp
                | LOrExp '||' LAndExp
ConstExp     -> AddExp  // □□□□□□□□□□□□ AddExp □□□□□□□□□□□□□□
```

□□ `Ident` □ `Number` □□□□□□□□

□□□□□□□

□□

miniSysY □□□□□□□□□□□□□ `//` □□□□□□□□□□□□ `/*` □ `*/` □□□□□□□□

- □□□□□□ `//` □□□□□□□□□□□□□□□□□□□□
- □□□□□□ `/*` □□□□□□□□□□ `*/` □□□□□□ `*/` □

□□□ `Ident` □□□

```
Ident    -> Nondigit
          | Ident Nondigit
          | Ident Digit
Nondigit -> '_' | 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... |
Digit    -> '0' | '1' | ... | '9'
```

□□□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□

□□ `Number` □□□

Number □□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
Number             -> decimal-const | octal-const | hexadec
decimal-const      -> nonzero-digit | decimal-const digit
octal-const        -> '0' | octal-const octal-digit
hexadecimal-const  -> hexadecimal-prefix hexadecimal-digit
                    | hexadecimal-const hexadecimal-digit
hexadecimal-prefix -> '0x' | '0X'
nonzero-digit      -> '1' | '2' | ... | '9'
octal-digit        -> '0' | '1' | ... | '7'
digit              -> '0' | nonzero-digit
hexadecimal-digit  -> '0' | '1' | ... | '9'
                    | 'a' | 'b' | 'c' | 'd' | 'e' | 'f'
                    | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
```

□□ `Number` □□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□□□ `Number` □□□□□□□□□□ `0 <= Number <= 2147483647` □□□□□□□□□□□□□□□

[illegible]

`CompUnit`

- [illegible] miniSysY [illegible] EBNF [illegible] `CompUnit` [illegible] `CompUnit` [illegible] `main` [illegible] `int` [illegible] `FuncDef` [illegible]
- `CompUnit` [illegible]/[illegible] `Decl` [illegible] `FuncDef` [illegible] `Ident` [illegible]
- `CompUnit` [illegible]/[illegible]/[illegible]

`ConstInitVal` [illegible] `InitVal`

- [illegible]
- [illegible]
- [illegible] 0[illegible]

`ConstDef`

- `ConstDef` [illegible] `Ident` [illegible] `=` [illegible] `=` [illegible]
- `ConstDef` [illegible] `=` [illegible]
- `ConstDef` [illegible] C [illegible] **miniSysY** [illegible][2*3][8/2][illegible] 6 [illegible] 4[illegible] 0 [illegible] `ConstDef` [illegible] `ConstExp` [illegible]
- [illegible] `ConstDef` [illegible] `=` [illegible] `ConstInitVal` [illegible] `ConstInitVal` [illegible] `ConstExp` [illegible] `ConstInitVal` [illegible] `ConstExp` [illegible] `int` [illegible]
- `ConstInitVal` [illegible]
  - [illegible] `{}` [illegible] 0[illegible]
  - [illegible] `int a[3] = {1, 2, 3};` [illegible] `int a[3][2] = { {1, 2}, {3, 4}, {5, 6} };` [illegible]
  - [illegible] 0[illegible] `int a[5] = {1, 2};` [illegible] `int a[4][3] = { {1, 2, 3}, {4, 5}, {} };` [illegible]

`VarDef`

- `VarDef` [illegible] `=` [illegible]
- `VarDef` [illegible] `ConstDef` [illegible]
- `VarDef` [illegible] `ConstExp` [illegible]

- □□□□□□□ `InitVal` □□ `Exp` □□□□□□□□□□□□□□□ `InitVal` □□ `Exp` □□□□□□□□□□□□□□

`FuncFParam` □□□

- `FuncFParam` □□□□□□□□□□□□□□□ `Ident` □□□□□□□□□□□□□□□□□ □□□□□□□
- □ `FuncFParam` □□□□□□□□□□□□□□□□□□□□□ `[]` □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□ `Exp` □□□ `int` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□ `int a[4][3]` □□□□ `a[1]` □□□□□ □□□□□□□□□□□□□□□□ `int[]` □□□□

`FuncDef`

- `FuncDef` □□□□□□□□□□ `FuncType` □□□□□□□
  - □□□□□□ `int` □□□□□□□□□□□□□□□□ `Exp` □ `return` □□□□□ □ `return` □□□□□□□□□□□□□
  - □□□□□□□ `void` □□□□□□□□□□□□□□□ `return` □□□

`Block`

- `Block` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□□□□□□□□□ `Decl` □□)□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□

`Stmt`

- `Stmt` □□ `if` □□□□□□□□□□□
- □□ `Exp` □□□□ `Stmt` □ `Exp` □□□□□□□□□□□□□□

`LVal`

- `LVal` □□□□□□□□□□□□□□□□□□□□□□□□□
- □ `LVal` □□□□□□□□□□□□□□□□□□□□□□□□

`Exp` □ `Cond`

- `Exp` □□ `int` □□□□□□□ `Cond` □□□□□□□□□□□□□ `!` □□□ `Cond` □□□□
- `LVal` □□□□□□□□□□□ `Exp` □□□□□□□□□□□□□□□□□□□□□□ `LVal` □□□□□□

- miniSysY □□□□□□□□□□□ C □□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- `Cond` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

□ lab3 □□□□ lab □□□□□□□□□ miniSysY □□□□□□□□□□□□□□□□□□□□□□□□□□□ I/O □□□□□□□□□□□□□/□□□□□□□□□□□□□□□□□□□□□□□□□□□□ lab3 □□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□ □□□

□□□□□□□□

1. `int getint();` □□□□□□□□□□□□□□□□□□

```
int n;
n = getint();
```

2. `int getch();` □□□□□□□□□□□□□□□ ASCII □□□

```
int n;
n = getch();
```

3. `int getarray(int []);` □□□□□□□□□ 1 □□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□ `getarray()` □□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□

```
int a[10][10];
int n;
n = getarray(a[0]);
```

4. `void putint(int);` □□□□□□□□□□

```
int n = 10;
putint(n);
putint(11);
```

5. `void putch(int);` □□□□□ ASCII □□□□□□□□□□□□□□□□□□□□ `0~255` □

```
int n = 10;
putch(n);
```

6. `void putarray(int, int[]);` □ 1 □□□□□□□□□□□□□□□□□ N□□□ □□□□□□□□□ N □□□□□□□ `putarray()` □□□□□□□□□□□□□□□□

```
int n = 2;
int a[2] = {2, 3};
putarray(n, a);
```

[illegible]

[illegible]“IDE [illegible]”[illegible]~~[illegible]~~[illegible]

[illegible] CPU [illegible] RGB [illegible]

[illegible]

[illegible][1]

> 1. [illegible] ↩

> [illegible] lab [illegible] lab [illegible]
> [illegible]“[illegible]”[illegible]“[illegible]”[illegible]/[illegible]“[illegible]”[illegible]
> [illegible] [illegible]“[illegible]”[illegible][2]

2: [illegible] https://github.com/rcore-os/rCore-Tutorial-Book-v3/issues/71

[illegible]

- [illegible] (Abstract Syntax Tree, **AST**)[illegible]
- [illegible] **AST** [illegible]Intermediate Representation, **IR**[illegible]LLVM IR [illegible]
- [illegible] IR [illegible] (MIPS, x86, arm, RISC-V, ...)[illegible]

[illegible]

1. [illegible] GCC [illegible] YACC [illegible]BISON[illegible][illegible] 3.x [illegible][illegible] ANTLR/FLEX/BISON/[illegible] [illegible] [illegible] [illegible]
2. [illegible] 0 [illegible]
3. [illegible] LLVM IR [illegible]
4. [illegible] LLVM IR[illegible]

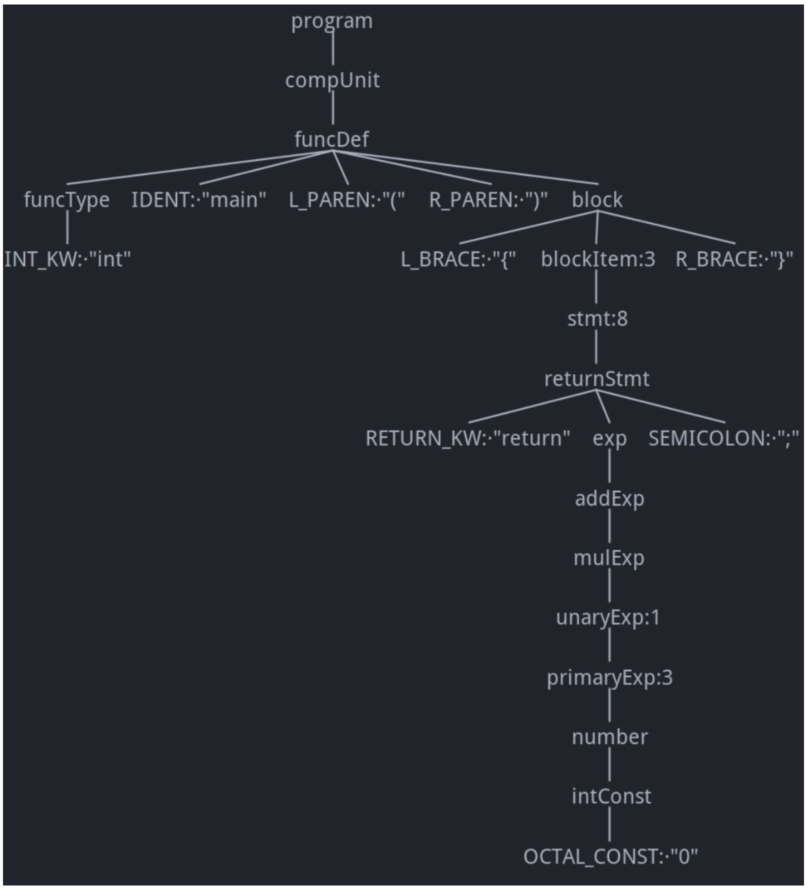
[illegible]

[illegible] [illegible]

[illegible] MiniSysY [illegible]

```
int main() {
    return 0;
}
```

[illegible] Token [illegible] Token [illegible] [illegible]



[illegible]

[illegible] MiniSysY [illegible] ANTLR [illegible] FLEX/BISON[illegible]

[illegible]

[illegible] AST, [illegible] MiniSysY [illegible]

```
int main() {
   return a; //a [illegible]
}
```

[illegible] C [illegible]#%&*[illegible]UB[illegible]

[illegible]

[illegible] AST [illegible]

- [illegible]
- [illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]AST,[illegible]statically-typed[illegible]typing rules[illegible]——[illegible]

> [illegible]MiniSysY[illegible] `int` [illegible] `void` [illegible]
> [illegible] `int` [illegible] `int[]` [illegible]

[illegible]LLVM IR

[illegible]LLVM IR[illegible][LLVM IR [illegible]]

Lab 1: main [illegible]

- [illegible]
  - [illegible] `return` [illegible] `main` [illegible]
  - [illegible]
- [illegible]2022 [illegible] 1 [illegible] 2 [illegible] 23:59
- [illegible] `[illegible]_[illegible]_lab1.pdf`
- [illegible][illegible] `lab1[illegible]main [illegible]/` [illegible]
- [illegible]2022 [illegible] 1 [illegible] 9 [illegible] 23:59

Part 1 □□ main □□□ return □□□□

□□

□ Part 1 □□□□□□□□□□□□□□□ `main` □□□□□□□ `return` □□□□□□ LLVM IR □□□□□

□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□

```
CompUnit -> FuncDef
FuncDef  -> FuncType Ident '(' ')' Block
FuncType -> 'int'
Ident    -> 'main'
Block    -> '{' Stmt '}'
Stmt     -> 'return' Number ';'
```

□□ Number □□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
Number             -> decimal-const | octal-const | hexadec
decimal-const      -> nonzero-digit | decimal-const digit
octal-const        -> '0' | octal-const octal-digit
hexadecimal-const  -> hexadecimal-prefix hexadecimal-digit
                    | hexadecimal-const hexadecimal-digit
hexadecimal-prefix -> '0x' | '0X'
nonzero-digit      -> '1' | '2' | '3' | '4' | '5' | '6' | '
octal-digit        -> '0' | '1' | '2' | '3' | '4' | '5' | '
digit              -> '0' | nonzero-digit
hexadecimal-digit  -> '0' | '1' | '2' | '3' | '4' | '5' | '
                    | 'a' | 'b' | 'c' | 'd' | 'e' | 'f'
                    | 'A' | 'B' | 'C' | 'D' | 'E' | 'F'
```

□□ `Number` □□□ LLVM IR □□□□□□□□□□□□□□□□□□□□□□□□□□□□ `Number` □□ □□□□□□□□□ `0 <= Number <= 2147483647` □□□□□□□□□□□□□□□

□□

□□□□ 1□

```
int main() {
    return 123;
}
```

□□□□ 1□

```
define dso_local i32 @main(){
    ret i32 123
}
```

□□□□ 2□

```
int main() {
    return 0
}
```

□□□□ 2□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 0 □□□□□□□

Part 2 □□□□

□□

miniSysY □□□□□□□□□□□□□ `//` □□□□□□□□□□□□ `/*` □ `*/` □□□□□□□□

- □□□□□□□ `//` □□□□□□□□□□□□□□□□□□□□□
- □□□□□□□ `/*` □□□□□□□□□□□ `*/` □□□□□□□ `*/` □

□ Part 2 □□□

□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

□□□□ 1□

```
int main() { // mian
    return /* 123 */ 234;
}
```

□□□□ 1□

```
define dso_local i32 @main(){
    ret i32 234
}
```

□□□□ 2□

```
int main() {
    /*
    return 123;
}
```

□□□□ 2□

□□□□□□□□ **0** □□□□□□□□

Lab 1 [illegible]

Lab1 [illegible] miniSysY [illegible]——[illegible] `main` [illegible] **LLVM IR** [illegible]

[illegible] miniSysY [illegible] `putint()` [illegible] LLVM IR [illegible] Unix & Linux [illegible] `echo $?` [illegible] `echo $?` [illegible] 256 [illegible]

□□□□

□□□□□□□□□□□ `lab1` □□□□ `judge.toml` □□□□□□ `[jobs.lab1]` □

□□□□ `$input` □□□□□□□□□□□□□□□□□□ LLVM IR □□□ `$ir` □□□□□□□□ `lli` □□□□□□□□□□□□□□ IR □□□□□□□□□ `judge.toml` □ `run` □□□ `$input` □□□□□□□□□ `$ir` □□□□□□□□□□□ `./compiler < $input > $ir` □ `./compiler $input $ir` □□

```toml
# □□□□
[jobs.lab1]

image = { source = "dockerfile", path = "." }

run = [
    "./compiler $input $ir",
]
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 0 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

Lab 2□□□□□□

- □□□□□□□□□□□□□□□□□□□□□ lab 1 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
  - □□□□□□□□
  - □□□□□□□□□□□□
- □□□□□□□□□□2022 □ 1 □ 2 □ 23:59
- □□□□□□□□□□ `□□_□□_lab2.pdf`
- □□□□□□□□□□□□ `lab2□□□□□□/` □□□□□□□□
- □□□□□□□□□□□□□2022 □ 1 □ 9 □ 23:59

Part 3 □□□□□□□

□ Part 3 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□□

```
CompUnit    -> FuncDef
FuncDef     -> FuncType Ident '(' ')' Block
FuncType    -> 'int'
Ident       -> 'main'
Block       -> '{' Stmt '}'
Stmt        -> 'return' Exp ';'
Exp         -> AddExp
AddExp      -> MulExp
MulExp      -> UnaryExp
UnaryExp    -> PrimaryExp | UnaryOp UnaryExp
PrimaryExp -> '(' Exp ')' | Number
UnaryOp     -> '+' | '-'
```

□□

□□□□ 1□

```
int main() {
    return ---(-1);
}
```

□□ IR 1□

```
define dso_local i32 @main() {
    %1 = sub i32 0, 1
    %2 = sub i32 0, %1
    %3 = sub i32 0, %2
    %4 = sub i32 0, %3
    ret i32 %4
}
```

□□□□□ 1□lli □□□□□□□□

```
1
```

□□□□ 2□

```
int main() {
    return +-+-010;
}
```

□□ IR 2□

```
define dso_local i32 @main() {
    %1 = sub i32 0, 8
    %2 = sub i32 0, %1
    ret i32 %2
}
```

□□□□□ 2□lli □□□□□□:

```
8
```

□□□□ 3□

```
int main() {
    return -+(+-((-+(-+(1))));
}
```

□□ IR 3□

□□□□□□□ **0** □□□□□□□

Part 4 [illegible]

[illegible] Part 4 [illegible]

miniSysY [illegible] C [illegible]

[illegible]

[illegible] `CompUnit` [illegible]

```
CompUnit    -> FuncDef
FuncDef     -> FuncType Ident '(' ')' Block
FuncType    -> 'int'
Ident       -> 'main'
Block       -> '{' Stmt '}'
Stmt        -> 'return' Exp ';'
Exp         -> AddExp
AddExp      -> MulExp
               | AddExp ('+' | '−') MulExp
MulExp      -> UnaryExp
               | MulExp ('*' | '/' | '%') UnaryExp
UnaryExp    -> PrimaryExp | UnaryOp UnaryExp
PrimaryExp  -> '(' Exp ')' | Number
UnaryOp     -> '+' | '-'
```

[illegible] C [illegible] int [illegible]

[illegible]

[illegible] 1[illegible]

```
int main() {
    return 1 + (-2) * (3 / (4 - 5));
}
```

[illegible] IR 1[illegible]

```
define dso_local i32 @main() {
    %1 = sub i32 0, 2
    %2 = sub i32 4, 5
    %3 = sdiv i32 3, %2
    %4 = mul i32 %1, %3
    %5 = add i32 1, %4
    ret i32 %5
}
```

□□□□□ 1□lli □□□□□□□□

```
7
```

□□□□ 2□

```
int main() {
    return 1 +-+ (- - - - - - - - -1);
}
```

□□ IR 2□

```
define dso_local i32 @main() {
    %1 = sub i32 0, 1
    %2 = sub i32 0, %1
    %3 = sub i32 0, %2
    %4 = sub i32 0, %3
    %5 = sub i32 0, %4
    %6 = sub i32 0, %5
    %7 = sub i32 0, %6
    %8 = sub i32 0, %7
    %9 = sub i32 0, %8
    %10 = sub i32 0, %9
    %11 = add i32 1, %10
    ret i32 %11
}
```

□□□□□ 2□lli □□□□□□□□

```
2
```

□□□□ 3□

```
int main() {
    return 4 * (1 / 5) - 4 + 1 ** 1;
}
```

□□ IR 3□

□□□□□□□ **0** □□□□□□□

Lab 2 □□□□

□□□□ clang □□□□□□□□□□□□□□□□□□ C □□□□□ `++` □ `--` □□□□□□□ `+` □ `-` □□□□□□□□□□□ `++` □ `--` □□□□□□□□□□□□□□□□□□ `+` □ `-` □□□□□□□miniSysY □□□□□□□□ `++` □ `--` □□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□

```
int main() {
    return 1 +-+ (- - -15) / 0x5;
}
```

```
define dso_local i32 @main(){
    %x0 = sub i32 0, 15
    %x1 = sub i32 0, %x0
    %x2 = sub i32 0, %x1
    %x3 = sub i32 0, %x2
    %x4 = sdiv i32 %x3, 5
    %x5 = add i32 1, %x4
    ret i32 %x5
}
```

□□ LLVM IR □□□□□□□LLVM □□ SSA□LLVM IR □□□□□□□□□□□□□□□□□□□

□□□□

□□□□□□□□□□□□ `lab2` □□□□ `judge.toml` □□□□□□ `[jobs.lab2]` □

□□□□ `$input` □□□□□□□□□□□□□□□□□□□□ LLVM IR □□□ `$ir` □□□□□□□□□ `lli` □□□□□□□□□□□□□□□ IR □□□□□□□□□□ `judge.toml` □ `run` □□□ `$input` □□□□□□□□□□ `$ir` □□□□□□□□□□□□ `./compiler < $input > $ir` □ `./compiler $input $ir` □□

```
# □□□□
[jobs.lab2]

image = { source = "dockerfile", path = "." }

run = [
    "./compiler $input $ir",
]
```

□□□ 0 □□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□

Lab 3□□□□□

- □□□□□□□□□□□□□□□□□□□□ lab 2 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
  - □□□□□□□□□□□□□□□□□□□□
  - □□□ miniSysY □□□□□□
- □□□□□□□□ Part 5 □ Part 6 □□□ miniSysY □□□□□□□□□□Part 5 □□□□□□ `putint()` □□□□ Part 5 □ Part 6 □□□□□□□□□□
- □□□□□□□□□2022 □ 1 □ 2 □ 23:59
- □□□□□□□□□ `□□_□□_lab3.pdf`
- □□□□□□□□□□□□ `lab3□□□□□/` □□□□□□□□
- □□□□□□□□□□□□2022 □ 1 □ 9 □ 23:59

Part 5 □□□□□□□

~~□□□□□□□□□□□□□□~~

□ Part 5 □□

□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□□

```
CompUnit     -> FuncDef
Decl         -> ConstDecl | VarDecl
ConstDecl    -> 'const' BType ConstDef { ',' ConstDef } ';'
BType        -> 'int'
ConstDef     -> Ident '=' ConstInitVal
ConstInitVal -> ConstExp
ConstExp     -> AddExp
VarDecl      -> BType VarDef { ',' VarDef } ';'
VarDef       -> Ident
                | Ident '=' InitVal
InitVal      -> Exp
FuncDef      -> FuncType Ident '(' ')' Block // □□□□ Ident
FuncType     -> 'int'
Block        -> '{' { BlockItem } '}'
BlockItem    -> Decl | Stmt
Stmt         -> LVal '=' Exp ';'
                | [Exp] ';'
                | 'return' Exp ';'
LVal         -> Ident
Exp          -> AddExp
AddExp       -> MulExp
                | AddExp ('+' | '−') MulExp
MulExp       -> UnaryExp
                | MulExp ('*' | '/' | '%') UnaryExp
UnaryExp     -> PrimaryExp | UnaryOp UnaryExp
PrimaryExp   -> '(' Exp ')' | LVal | Number
UnaryOp      -> '+' | '-'
```

- □□□ `Ident` □□□

```
Ident    -> Nondigit
            | Ident Nondigit
            | Ident Digit
Nondigit -> '_' | 'a' | 'b' | ... | 'z' | 'A' | 'B' | ... |
Digit    -> '0' | '1' | ... | '9'
```

□□□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□
- □□□□□□□□□□

□□□□

`ConstInitVal`

- `ConstInitVal` □□ `ConstExp` □□□□□□□□□ `int` □□□□□□□ □□□□□□□

`VarDef`

- `VarDef` □□□□□□□□□ `=` □□□□□□□□□□□□□□□
- □ `VarDef` □□ `=` □□□□□ `=` □□□ `InitVal` □ `ConstInitVal` □□□□□□□□□□□□ `ConstInitVal` □□□□□ `ConstExp` □□□□□□ `InitVal` □□□□□□□□□□□□□□□ `Exp` □

`Block`

- `Block` □□□□□□□□□□□

`Stmt`

- □□ `Exp` □□□ `Stmt` □□ `Exp` □□□□□□□□□□□□

`LVal`

- □□□□□ `LVal` □□□□□ `Exp` □□ `LVal` □□□□□□□□□ `Exp` □□ □□□□□□□□□□

□□

□□ 1

□□□□ 1□

```
int main() {
    int a = 123 - 122;
    return a;
}
```

□□ IR 1□

```
define dso_local i32 @main(){
    %1 = alloca i32
    %2 = sub i32 123, 122
    store i32 %2, i32* %1
    %3 = load i32, i32* %1
    ret i32 %3
}
```

□□□□ 1□

```
1
```

□□ 2

□□□□ 2□

```
int main() {
    const int Nqn7m1 = 010;
    int yiersan = 456;
    int mAgIc_NuMbEr;
    mAgIc_NuMbEr = 8456;
    int a1a11a11 = (mAgIc_NuMbEr - yiersan) / 1000 - Nqn7m1
    _CHAOS_TOKEN = 2;
    a1a11a11 = a1a11a11 + _CHAOS_TOKEN;
    return a1a11a11 - _CHAOS_TOKEN + 000;
}
```

□□ IR 2□

```
define dso_local i32 @main(){
    %1 = alloca i32
    %2 = alloca i32
    %3 = alloca i32
    %4 = alloca i32
    store i32 456, i32* %4
    store i32 8456, i32* %3
    %5 = load i32, i32* %3
    %6 = load i32, i32* %4
    %7 = sub i32 %5, %6
    %8 = sdiv i32 %7, 1000
    %9 = sub i32 %8, 8
    store i32 %9, i32* %2
    store i32 2, i32* %1
    %10 = load i32, i32* %2
    %11 = load i32, i32* %1
    %12 = add i32 %10, %11
    store i32 %12, i32* %2
    %13 = load i32, i32* %2
    %14 = load i32, i32* %1
    %15 = sub i32 %13, %14
    %16 = add i32 %15, 0
    ret i32 %16
}
```

□□□□ 2:

```
0
```

□□ 3

□□□□ 3□

```
int main() {
    const int sudo = 0;
    int rm = 5, r = 3, home = 5;
    sudo = rm -r /home*       0;
    return 0;
}
```

□□□□ 3□

□□□□□□□ **0** □□□□□□□□□□□□□□□ `LVal` □□□□□□□

Part 6 □□□□

□ Part 6 □□□□□□□□□□□□ miniSysY □□□□□□□□

□□□□□□□□□□□□□□□□□□□□ miniSysY □□□□□□□□□□□□□□□□□□□□□□□□□□□□ I/O □□□□□□□□□□□□□□□/□□ LLVM IR □□

□ **Lab 3** □□□□□□□ `getarray` □ `putarray` □□□

□□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□□

```
CompUnit     -> FuncDef
Decl         -> ConstDecl | VarDecl
ConstDecl    -> 'const' BType ConstDef { ',' ConstDef } ';'
BType        -> 'int'
ConstDef     -> Ident '=' ConstInitVal
ConstInitVal -> ConstExp
ConstExp     -> AddExp
VarDecl      -> BType VarDef { ',' VarDef } ';'
VarDef       -> Ident
              | Ident '=' InitVal
InitVal      -> Exp
FuncDef      -> FuncType Ident '(' ')' Block // □□□□ Ident
FuncType     -> 'int'
Block        -> '{' { BlockItem } '}'
BlockItem    -> Decl | Stmt
Stmt         -> LVal '=' Exp ';'
              | [Exp] ';'
              | 'return' Exp ';'
Exp          -> AddExp
LVal         -> Ident
PrimaryExp   -> '(' Exp ')' | LVal | Number
AddExp       -> MulExp
              | AddExp ('+' | '−') MulExp
MulExp       -> UnaryExp
              | MulExp ('*' | '/' | '%') UnaryExp
UnaryExp     -> PrimaryExp
              | Ident '(' [FuncRParams] ')'
              | UnaryOp UnaryExp
FuncRParams  -> Exp { ',' Exp }
UnaryOp      -> '+' | '-'
```

□□□□

[illegible]

[illegible]

1. `int getint();` [illegible]

```
int n;
n = getint();
```

2. `int getch();` [illegible] ASCII [illegible]

```
int n;
n = getch();
```

3. `int getarray(int []);` [illegible] 1 [illegible] `getarray()` [illegible]

```
int a[10][10];
int n;
n = getarray(a[0]);
```

4. `void putint(int);` [illegible]

```
int n = 10;
putint(n);
putint(11);
```

5. `void putch(int);` [illegible] ASCII [illegible] `0~255` [illegible]

```
int n = 10;
putch(n);
```

6. `void putarray(int, int[]);` [illegible] 1 [illegible] N[illegible] N [illegible] `putarray()` [illegible]

```
int n = 2;
int a[2] = {2, 3};
putarray(n, a);
```

[illegible]

[illegible] 1

□□□□ 1□

```
int main() {
    int n = getint();
    putint(n + 4);
    return 0;
}
```

□□ IR 1□

```
declare i32 @getint()
declare void @putint(i32)
define dso_local i32 @main(){
    %1 = alloca i32
    %2 = call i32 @getint()
    store i32 %2, i32* %1
    %3 = load i32, i32* %1
    %4 = add i32 %3, 4
    call void @putint(i32 %4)
    ret i32 0
}
```

□□□□ 1□

```
4
```

□□□□ 1□

```
8
```

□□ 2

□□□□ 2□

```
int main() {
    int a = getch(), b;
    b = getch();
    putch(a);
    putch(b);
    putch(10);
    putch(a - 16);
    putch(b + 6);
    return 0;
}
```

□□ IR 2□

```
declare i32 @getch()
declare void @putch(i32)
define dso_local i32 @main(){
    %1 = alloca i32
    %2 = alloca i32
    %3 = call i32 @getch()
    store i32 %3, i32* %2
    %4 = call i32 @getch()
    store i32 %4, i32* %1
    %5 = load i32, i32* %2
    call void @putch(i32 %5)
    %6 = load i32, i32* %1
    call void @putch(i32 %6)
    call void @putch(i32 10)
    %7 = load i32, i32* %2
    %8 = sub i32 %7, 16
    call void @putch(i32 %8)
    %9 = load i32, i32* %1
    %10 = add i32 %9, 6
    call void @putch(i32 %10)
    ret i32 0
}
```

□□□□ 2□

```
tl
```

□□□□ 2□

```
tl
dr
```

□□ 3

□□□□ 3□

```
int main() {
    int a = getint();
    putint();
    return 0;
}
```

□□□□ 3□

□□□□□□□ **0** □□□□□□□

Lab 3 □□□□

□□□□□ IR □□□ `alloca` □ `store` □ `load`

LLVM IR SSA □□

□□□□□□□□□□□ LLVM IR

`lli` □□□□□□□ `.ll` □□□□□□□□□□□□□□□□□□□□□□□□□□□□ `llvm-link` □

□□□□□□□□□□□□ `libsysy` □□□ □□ □□□□□□□□□□□□□□□ IO □□□□□□□

```c
/* libsysy.c */
#include "libsysy.h"
#include <stdio.h>
/* Input & output functions */
int getint() {
    int t;
    scanf("%d", &t);
    return t;
}
int getch() {
    char c;
    scanf("%c", &c);
    return (int)c;
}
int getarray(int a[]) {
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);
    return n;
}
void putint(int a) { printf("%d", a); }
void putch(int a) { printf("%c", a); }
void putarray(int n, int a[]) {
    printf("%d:", n);
    for (int i = 0; i < n; i++)
        printf(" %d", a[i]);
    printf("\n");
}
```

```
/* libsysy.h */
#ifndef __SYLIB_H_
#define __SYLIB_H_

#include <stdarg.h>
#include <stdio.h>
#include <sys/time.h>
/* Input & output functions */
int  getint(), getch(), getarray(int a[]);
void putint(int a), putch(int a), putarray(int n, int a[]);
#endif
```

□□□□□ `clang` □ `libsysy.c` □□□ `.ll` □□□□□□□ `llvm-link` □□□ □□□□□□ miniSysY □□□□□□□ `.ll` □□□□□□□□□ LLVM IR□□□□ `lli` □□ □□□

□□□

```
; main.ll
declare i32 @getint()
declare void @putint(i32)
define dso_local i32 @main(){
    %1 = alloca i32
    %2 = call i32 @getint()
    store i32 %2, i32* %1
    %3 = load i32, i32* %1
    %4 = add i32 %3, 4
    call void @putint(i32 %4)
    ret i32 0
}
```

```
$ clang -emit-llvm -S libsysy.c -o lib.ll
$ ./□□□□□ main.sy -o main.ll
$ llvm-link main.ll lib.ll -S -o out.ll
$ lli out.ll
```

□□□□

□□□□□□□□□□□□ `lab3` □□□□ `judge.toml` □□□□□□ `[jobs.lab3]` □

□□□□ `$input` □□□□□□□□□□□□□□□□□□ LLVM IR □□□ `$ir` □□□□□□□□ `lli` □□□□□□□□□□□□□□ IR □□□□□□□□□ `judge.toml` □ `run` □□□ `$input` □□□□□□□□□ `$ir` □□□□□□□□□□□ `./compiler < $input > $ir` □ `./compiler $input $ir` □□

```
# □□□□
[jobs.lab3]

image = { source = "dockerfile", path = "." }

run = [
    "./compiler $input $ir",
]
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 0 □□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□

Lab 4：`if` □□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□ lab 3 □□□□□□□ `if` □ `else` □□□□□□□□□□□□
- □□□□□□□□□□2022 □ 1 □ 2 □ 23:59
- □□□□□□□□□□ `□□_□□_lab4.pdf`
- □□□□□□□□□□□□ `lab4□□□□□/` □□□□□□□□
- □□□□□□□□□□□□2022 □ 1 □ 9 □ 23:59

Part 7 `if` □□□□□□□□

□□□□□□□□□□□□□□□□ `if` □ `else` □□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□□

```
CompUnit     -> FuncDef
Decl         -> ConstDecl | VarDecl
ConstDecl    -> 'const' BType ConstDef { ',' ConstDef } ';'
BType        -> 'int'
ConstDef     -> Ident '=' ConstInitVal
ConstInitVal -> ConstExp
ConstExp     -> AddExp
VarDecl      -> BType VarDef { ',' VarDef } ';'
VarDef       -> Ident
              | Ident '=' InitVal
InitVal      -> Exp
FuncDef      -> FuncType Ident '(' ')' Block // □□□□ Ident
FuncType     -> 'int'
Block        -> '{' { BlockItem } '}'
BlockItem    -> Decl | Stmt
Stmt         -> LVal '=' Exp ';'
              | Block
              | [Exp] ';'
              | 'if' '(' Cond ')' Stmt [ 'else' Stmt ]
              | 'return' Exp ';' // [changed]
Exp          -> AddExp
Cond         -> LOrExp // [new]
LVal         -> Ident
PrimaryExp   -> '(' Exp ')' | LVal | Number
UnaryExp     -> PrimaryExp
              | Ident '(' [FuncRParams] ')'
              | UnaryOp UnaryExp
UnaryOp      -> '+' | '-' | '!'  // □□ '!' □□□□ Cond □ [cha
FuncRParams  -> Exp { ',' Exp }
MulExp       -> UnaryExp
              | MulExp ('*' | '/' | '%') UnaryExp
AddExp       -> MulExp
              | AddExp ('+' | '-') MulExp
RelExp       -> AddExp
              | RelExp ('<' | '>' | '<=' | '>=') AddExp
EqExp        -> RelExp
              | EqExp ('==' | '!=') RelExp  // [new]
LAndExp      -> EqExp
              | LAndExp '&&' EqExp  // [new]
LOrExp       -> LAndExp
              | LOrExp '||' LAndExp  // [new]
```

□□

- `Cond` □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□ `Stmt -> Block` □□□□□□□□□□□□□□□
- `Stmt` □□ `if` □□□□□□□□ `if if else` □□□ `if { if else }` □
- □□□ 0 □ `Cond` □□ `true` □□□ 0 □ `Cond` □□ `false` □

□□

~~□□ IR □□□□□□□□□□□□□□□~~

□□ 1

□□□□ 1□

```
int main() {
    int a = getint();
    int b = getint();
    if (a <= b) {
        putint(1);
    }
    else {
        putint(0);
    }
    return 0;
}
```

□□ IR 1□

```
declare i32 @getint()
declare void @putint(i32)
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    %3 = call i32 @getint()
    store i32 %3, i32* %2
    %4 = call i32 @getint()
    store i32 %4, i32* %1
    %5 = load i32, i32* %2
    %6 = load i32, i32* %1
    %7 = icmp sle i32 %5, %6
    br i1 %7,label %8, label %10

8:
    call void @putint(i32 1)
    br label %9

9:
    ret i32 0

10:
    call void @putint(i32 0)
    br label %9
}
```

□□□□ 1□

```
9 12
```

□□□□ 1□

```
1
```

□□ 2

□□□□ 2□

```
int main() {
    int a, b, c = 1, d;
    int result;
    a = 5;
    b = 5;
    d = -2;
    result = 2;
    if (a + b + c + d == 10) {
        result = result + 1;
    } else if (a + b + c + d == 8) {
        result = result + 2;
    } else {
        result = result + 4;
    }
    putint(result);
    return 0;
}
```

□□ IR 2□

```
declare void @putint(i32)
define dso_local i32 @main(){
    %1 = alloca i32
    %2 = alloca i32
    %3 = alloca i32
    %4 = alloca i32
    %5 = alloca i32
    store i32 1, i32* %3
    store i32 5, i32* %5
    store i32 5, i32* %4
    %6 = sub i32 0, 2
    store i32 %6, i32* %2
    store i32 2, i32* %1
    %7 = load i32, i32* %5
    %8 = load i32, i32* %4
    %9 = add i32 %7, %8
    %10 = load i32, i32* %3
    %11 = add i32 %9, %10
    %12 = load i32, i32* %2
    %13 = add i32 %11, %12
    %14 = icmp eq i32 %13, 10
    br i1 %14, label %29, label %20

15:
    %16 = load i32, i32* %1
    %17 = add i32 %16, 1
    store i32 %17, i32* %1
    br label %18

18:
    %19 = load i32, i32* %1
    call void @putint(i32 %19)
    ret i32 0

20:
    %21 = load i32, i32* %5
    %22 = load i32, i32* %4
    %23 = add i32 %21, %22
    %24 = load i32, i32* %3
    %25 = add i32 %23, %24
    %26 = load i32, i32* %2
    %27 = add i32 %25, %26
    %28 = icmp eq i32 %27, 8
    br i1 %28, label %37, label %34

29:
    br label %15

30:
    %31 = load i32, i32* %1
    %32 = add i32 %31, 2
```

```
    store i32 %32, i32* %1
    br label %33

33:
    br label %18

34:
    %35 = load i32, i32* %1
    %36 = add i32 %35, 4
    store i32 %36, i32* %1
    br label %33

37:
    br label %30
}
```

□□□□ 2□

```
6
```

□□ 3

□□□□ 3□

```
int main() {
    int a, b, c = 1, d;
    int result;
    a = 5;
    b = 5;
    d = -2;
    result = 2;
    if (a + b == 9 || a - b == 0 && result != 4)
        result = result + 3;
    else if (c + d != -1 || (result + 1) % 2 == 1)
        result = result + 4;
    putint(result);
    return 0;
}
```

□□ IR 3□

```
declare void @putint(i32)
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    %3 = alloca i32
    %4 = alloca i32
    %5 = alloca i32
    store i32 1, i32* %3
    store i32 5, i32* %5
    store i32 5, i32* %4
    %6 = sub i32 0, 2
    store i32 %6, i32* %2
    store i32 2, i32* %1
    %7 = load i32, i32* %5
    %8 = load i32, i32* %4
    %9 = add i32 %7, %8
    %10 = icmp eq i32 %9, 9
    br i1 %10, label %27, label %22

  11:
    %12 = load i32, i32* %1
    %13 = add i32 %12, 3
    store i32 %13, i32* %1
    br label %14

  14:
    %15 = load i32, i32* %1
    call void @putint(i32 %15)
    ret i32 0

  16:
    %17 = load i32, i32* %3
    %18 = load i32, i32* %2
    %19 = add i32 %17, %18
    %20 = sub i32 0, 1
    %21 = icmp ne i32 %19, %20
    br i1 %21, label %41, label %36

  22:
    %23 = load i32, i32* %5
    %24 = load i32, i32* %4
    %25 = sub i32 %23, %24
    %26 = icmp eq i32 %25, 0
    br i1 %26, label %28, label %16

  27:
    br label %11

  28:
    %29 = load i32, i32* %1
    %30 = icmp ne i32 %29, 4
```

```
    br i1 %30, label %31, label %16

  31:
    br label %11

  32:
    %33 = load i32, i32* %1
    %34 = add i32 %33, 4
    store i32 %34, i32* %1
    br label %35

  35:
    br label %14

  36:
    %37 = load i32, i32* %1
    %38 = add i32 %37, 1
    %39 = srem i32 %38, 2
    %40 = icmp eq i32 %39, 1
    br i1 %40, label %42, label %35

  41:
    br label %32

  42:
    br label %32
}
```

□□□□ 3□

```
5
```

□□ 4

□□□□ 4□

```
int main() {
    int a;
    a = 10;
    if (+-!!!a) {
        a = - - -1;
    }
    else {
        a = 0;
    }
    putint(a);
    return 0;
}
```

□□ IR 4□

```
declare void @putint(i32)
define dso_local i32 @main() {
    %1 = alloca i32
    store i32 10, i32* %1
    %2 = load i32, i32* %1
    %3 = icmp eq i32 %2, 0
    %4 = zext i1 %3 to i32
    %5 = icmp eq i1 %3, 0
    %6 = zext i1 %5 to i32
    %7 = icmp eq i1 %5, 0
    %8 = zext i1 %7 to i32
    %9 = zext i1 %7 to i32
    %10 = sub i32 0, %9
    %11 = icmp ne i32 %10, 0
    br i1 %11, label %12, label %18

12:
    %13 = sub i32 0, 1
    %14 = sub i32 0, %13
    %15 = sub i32 0, %14
    store i32 %15, i32* %1
    br label %16

16:
    %17 = load i32, i32* %1
    call void @putint(i32 %17)
    ret i32 0
18:
    store i32 0, i32* %1
    br label %16
}
```

□□□□ 4□

```
0
```

Lab 4 [illegible]

[illegible]

> [illegible]“[illegible]”[illegible]

[illegible]

[illegible] LLVM IR [illegible] `ret` [illegible] Lab 4 [illegible] `if` [illegible] `else` [illegible] IR [illegible]IR [illegible]

[illegible]

```
int main() {
    int a = getint();
    if (a == 1) {
        putint(1);
    } else {
        putint(2);
    }
    return 0;
}
```

```
declare i32 @getint()
declare void @putint(i32)
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = call i32 @getint()
    store i32 %2, i32* %1
    %3 = load i32, i32* %1
    %4 = icmp eq i32 %3, 1
    br i1 %4, label %5, label %6

5:
    call void @putint(i32 1)
    br label %7

6:
    call void @putint(i32 2)
    br label %7

7:
    ret i32 0
}
```

□□□□□ 4 □□□□□□□□□□□□□□□□□□□□ `0` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□ `icmp` □□□□□ `br` □□□ `icmp` □□□ `a` □ `1` □□□□□□□□ `%4` □□ `br` □□□□□□□□□□□□□□□ `%4` □ `true` □□□□ `5` □ `%4` □ `false` □□ □□ `6` □ `5` □ `6` □ `br` □□□□□□□□□□□□□□□□□□□□□□□□ `7` □

□□□□□clang □□□□□□□□□□□□□□□□□□□□□□LLVM □□□□□□□□□□□□□□□□□□□□□□□ 0 □□□

LLVM IR □□□□

□□□ □□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□ lab □□□□□□□□□□□□□□□□□□□□□□□□□□ `zext` , `and` , `or` □ `icmp` □□□□□□ □□□□□□□□□□□□□□

LLVM IR □□□□□□□□□ `i1` □□□□□□□□ `i32` □□□□□□□□□□□ `zext` □□□□□ □□□□□□□□□

`zext` □□□□□□□□ `<result> = zext <ty> <value> to <ty2>` , □□□□ □□□□□□

```
define i32 @main() {
%x = add i1 0,0
%x1 = zext i1 %x to i32
ret i32 %x1
}
```

`icmp` □□□□□□□□□□□□□□□ `<result> = icmp <cond> <ty> <op1>, <op2>` □□□□□□□□□ `icmp` □ `<result>` □ `i1` □□

`and` □ `or` □□□□/□□□ `<result> = and/or <ty> <op1>, <op2>` □□□□□□□□□□□□□□□□□□□□

`br` □□□□□□□□□□□□□□□□□□□□□□□□ `br i1 <cond>, label <iftrue>, label <iffalse>` □□□□□□□□□ `br label <dest>` □□□□□□□□ □□□□□□□□□□□□

```
define i32 @main() {
block_a:
    %x=add i32 0,123
    %y=add i32 0,321 ;
    %m=add i32 0,123
    %n=add i32 0,123 ;
    %res_xy = icmp eq i32 %x,%y
    %res_mn = icmp eq i32 %m,%n
    %cond = and i1 %res_xy,%res_mn; □□□□ and □□ or □□□□□□
    br i1 %cond ,label %block_true,label %block_false
block_true:
    ret i32 0
block_false:
    ret i32 1
}
```

□ `c` □□□□□□□□□□□□□□□□□□□□□□□

```
if((x==y)&&(m==n)){
    return 0;
}
return 1;
```

[illegible]

[illegible] `lab4` [illegible] `judge.toml` [illegible] `[jobs.lab4]` [illegible]

[illegible] `$input` [illegible] LLVM IR [illegible] `$ir` [illegible] `lli` [illegible] IR [illegible] `judge.toml` [illegible] `run` [illegible] `$input` [illegible] `$ir` [illegible] `./compiler < $input > $ir` [illegible] `./compiler $input $ir` [illegible]

```
# [illegible]
[jobs.lab4]

image = { source = "dockerfile", path = "." }

run = [
    "./compiler $input $ir",
]
```

[illegible] 0 [illegible]

Lab 5□□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□ lab 4 □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□□2022 □ 1 □ 2 □ 23:59
- □□□□□□□□□ `□□_□□_lab5.pdf`
- □□□□□□□□□□□□ `lab5□□□□□□□□□/` □□□□□□□□
- □□□□□□□□□□□□2022 □ 1 □ 9 □ 23:59

Part 8 [illegible]

[illegible] Part 8 [illegible]

[illegible]

[illegible]

`Block`

- `Block` [illegible]
- [illegible]
- [illegible] `Decl` [illegible])[illegible]
  [illegible]

[illegible]

[illegible] 1

[illegible] 1[illegible]

```
int main() {
    int a = getint();
    {
        int b = 2;
        putint(a + b);
        int a = getint();
        putint(a + b);
    }
    int b = a + 2;
    putint(a + b);
    return 0;
}
```

[illegible] IR 1[illegible]

```
declare i32 @getint()
declare void @putint(i32)
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    %3 = alloca i32
    %4 = alloca i32
    %5 = call i32 @getint()
    store i32 %5, i32* %4
    store i32 2, i32* %3
    %6 = load i32, i32* %4
    %7 = load i32, i32* %3
    %8 = add i32 %6, %7
    call void @putint(i32 %8)
    %9 = call i32 @getint()
    store i32 %9, i32* %2
    %10 = load i32, i32* %2
    %11 = load i32, i32* %3
    %12 = add i32 %10, %11
    call void @putint(i32 %12)
    %13 = load i32, i32* %4
    %14 = add i32 %13, 2
    store i32 %14, i32* %1
    %15 = load i32, i32* %4
    %16 = load i32, i32* %1
    %17 = add i32 %15, %16
    call void @putint(i32 %17)
    ret i32 0
}
```

□□□□ 1□

```
1 5
```

□□□□ 1□

```
374
```

□□ 2

□□□□ 2□

```
int main() {
    const int c1 = 10 * 5 / 2;
    const int c2 = c1 / 2, c3 = c1 * 2;
    if (c1 > 24) {
        int c1 = 24;
        putint(c2 - c1 * c3);
        putch(10);
    }
    {
        int c2 = c1 / 4;
        putint(c3 / c2);
        {
            int c3 = c1 * 4;
            putint(c3 / c2);
        }
    }
    putch(10);
    putint(c3 / c2);
    return 0;
}
```

□□ IR 2□

```
declare void @putint(i32)
declare void @putch(i32)
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    %3 = alloca i32
    %4 = icmp sgt i32 25, 24
    br i1 %4, label %5, label %9

5:
    store i32 24, i32* %3
    %6 = load i32, i32* %3
    %7 = mul i32 %6, 50
    %8 = sub i32 12, %7
    call void @putint(i32 %8)
    call void @putch(i32 10)
    br label %9

9:
    %10 = sdiv i32 25, 4
    store i32 %10, i32* %2
    %11 = load i32, i32* %2
    %12 = sdiv i32 50, %11
    call void @putint(i32 %12)
    %13 = mul i32 25, 4
    store i32 %13, i32* %1
    %14 = load i32, i32* %1
    %15 = load i32, i32* %2
    %16 = sdiv i32 %14, %15
    call void @putint(i32 %16)
    call void @putch(i32 10)
    %17 = sdiv i32 50, 12
    call void @putint(i32 %17)
    ret i32 0
}
```

□□□□ 2:

```
-1188
816
4
```

□□ 3

□□□□ 3□

```
int main() {
    int a = 1;
    int a = 2;
    return 0;
}
```

□□□□ 3□

□□□□□□□ **0** □□□□□□□

Part ⑨ □□□□

□ Part ⑨ □□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□□

```
CompUnit     -> Decl* FuncDef // [changed]
Decl         -> ConstDecl | VarDecl
ConstDecl    -> 'const' BType ConstDef { ',' ConstDef } ';'
BType        -> 'int'
ConstDef     -> Ident '=' ConstInitVal
ConstInitVal -> ConstExp
ConstExp     -> AddExp
VarDecl      -> BType VarDef { ',' VarDef } ';'
VarDef       -> Ident
                | Ident '=' InitVal
InitVal      -> Exp
FuncDef      -> FuncType Ident '(' ')' Block // □□□□ Ident
FuncType     -> 'int'
Block        -> '{' { BlockItem } '}'
BlockItem    -> Decl | Stmt
Stmt         -> LVal '=' Exp ';'
                | Block
                | [Exp] ';'
                | 'if' '(' Cond ')' Stmt [ 'else' Stmt ]
                | 'return' Exp ';'
Exp          -> AddExp
Cond         -> LOrExp
LVal         -> Ident
PrimaryExp   -> '(' Exp ')' | LVal | Number
UnaryExp     -> PrimaryExp
                | Ident '(' [FuncRParams] ')'
                | UnaryOp UnaryExp
UnaryOp      -> '+' | '-' | '!'  // □□ '!' □□□□ Cond □
FuncRParams  -> Exp { ',' Exp }
MulExp       -> UnaryExp
                | MulExp ('*' | '/' | '%') UnaryExp
AddExp       -> MulExp
                | AddExp ('+' | '-') MulExp
RelExp       -> AddExp
                | RelExp ('<' | '>' | '<=' | '>=') AddExp
EqExp        -> RelExp
                | EqExp ('==' | '!=') RelExp
LAndExp      -> EqExp
                | LAndExp '&&' EqExp
LOrExp       -> LAndExp
                | LOrExp '||' LAndExp
```

□□□□

`ConstDef` □ `VarDef`

- □□□□□□□□□□□□

`ConstInitVal` □ `InitVal`

- □□□/□□□□□□□□□□□□□□□□□□□□□□
- □□□
- □□□ 0□

□□

□□ 1

□□□□ 1□

```
int a = 5;
int main() {
    int b = getint();
    putint(a + b);
    return 0;
}
```

□□ IR 1□

```
@a = dso_local global i32 5
declare i32 @getint()
declare void @putint(i32)
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = call i32 @getint()
    store i32 %2, i32* %1
    %3 = load i32, i32* @a
    %4 = load i32, i32* %1
    %5 = add i32 %3, %4
    call void @putint(i32 %5)
    ret i32 0
}
```

□□□□ 1□

```
4
```

□□□□ 1□

```
9
```

□□ 2

□□□□ 2□

```
const int a = 6;
int b = a + 1;
int main() {
    int c = b;
    int b = 8;
    putint(b + c);
    return 0;
}
```

□□ IR 2□

```
@b = dso_local global i32 7
declare i32 @getint()
declare void @putint(i32)
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    %3 = load i32, i32* @b
    store i32 %3, i32* %2
    store i32 8, i32* %1
    %4 = load i32, i32* %1
    %5 = load i32, i32* %2
    %6 = add i32 %4, %5
    call void @putint(i32 %6)
    ret i32 0
}
```

□□□□ 2□

```
15
```

□□ 3

□□□□ 3□

```
int a = 6;
int b = a + 1;
int main() {
    int c = b;
    int b = 8;
    putint(b + c);
    return 0;
}
```

□□□□ 3□

□□□□□□□ **0** □□□□□□□

□□□□

□□□□□□□□□□□□ `lab5` □□□□ `judge.toml` □□□□□□ `[jobs.lab5]` □

□□□□ `$input` □□□□□□□□□□□□□□□□□□ LLVM IR □□□ `$ir` □□□□□□□□ `lli` □□□□□□□□□□□□□□□□ IR □□□□□□□□□ `judge.toml` □ `run` □□□ `$input` □□□□□□□□□□ `$ir` □□□□□□□□□□□□ `./compiler < $input > $ir` □ `./compiler $input $ir` □□

```toml
# □□□□
[jobs.lab5]

image = { source = "dockerfile", path = "." }

run = [
    "./compiler $input $ir",
]
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 0 □□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□

Lab 6□□□

- □□□□□□□□□□□□□ lab 5 □□□□□□□ `while` □□□ `continue` □ `break` □□□
- □□□□□□□□□□2022 □ 1 □ 7 □ 23:59
- □□□□□□□□□□ `□□_□□_lab6.pdf`
- □□□□□□□□□□□□ `lab6□□□/` □□□□□□□□
- □□□□□□□□□□□□□2022 □ 1 □ 9 □ 23:59

Part 10 □□□□

□ Part 10 □□□□□□□□□□□ `while` □□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□□

```
CompUnit     -> Decl* FuncDef
Decl         -> ConstDecl | VarDecl
ConstDecl    -> 'const' BType ConstDef { ',' ConstDef } ';'
BType        -> 'int'
ConstDef     -> Ident '=' ConstInitVal
ConstInitVal -> ConstExp
ConstExp     -> AddExp
VarDecl      -> BType VarDef { ',' VarDef } ';'
VarDef       -> Ident
              | Ident '=' InitVal
InitVal      -> Exp
FuncDef      -> FuncType Ident '(' ')' Block // □□□□ Ident
FuncType     -> 'int'
Block        -> '{' { BlockItem } '}'
BlockItem    -> Decl | Stmt
Stmt         -> LVal '=' Exp ';'
              | Block
              | [Exp] ';'
              | 'if' '(' Cond ')' Stmt [ 'else' Stmt ]
              | 'while' '(' Cond ')' Stmt
              | 'return' Exp ';' // [changed]
Exp          -> AddExp
Cond         -> LOrExp
LVal         -> Ident
PrimaryExp   -> '(' Exp ')' | LVal | Number
UnaryExp     -> PrimaryExp
              | Ident '(' [FuncRParams] ')'
              | UnaryOp UnaryExp
UnaryOp      -> '+' | '-' | '!'  // □□ '!' □□□□ Cond □
FuncRParams  -> Exp { ',' Exp }
MulExp       -> UnaryExp
              | MulExp ('*' | '/' | '%') UnaryExp
AddExp       -> MulExp
              | AddExp ('+' | '-') MulExp
RelExp       -> AddExp
              | RelExp ('<' | '>' | '<=' | '>=') AddExp
EqExp        -> RelExp
              | EqExp ('==' | '!=') RelExp
LAndExp      -> EqExp
              | LAndExp '&&' EqExp
LOrExp       -> LAndExp
              | LOrExp '||' LAndExp
```

□□

~~□□ IR □□□□□□□□□□□□□□□□□□□□~~

□□ 1

□□□□ 1□

```
int main() {
    int n = getint();
    int i = 0, sum = 0;
    while (i < n) {
        i = i + 1;
        sum = sum + i;
        putint(sum);
        putch(10);
    }
    return 0;
}
```

□□ IR 1□

```
declare i32 @getint()
declare void @putint(i32)
declare void @putch(i32)
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    %3 = alloca i32
    %4 = call i32 @getint()
    store i32 %4, i32* %3
    store i32 0, i32* %2
    store i32 0, i32* %1
    br label %5

5:
    %6 = load i32, i32* %2
    %7 = load i32, i32* %3
    %8 = icmp slt i32 %6, %7
    br i1 %8, label %9, label %16

9:
    %10 = load i32, i32* %2
    %11 = add i32 %10, 1
    store i32 %11, i32* %2
    %12 = load i32, i32* %1
    %13 = load i32, i32* %2
    %14 = add i32 %12, %13
    store i32 %14, i32* %1
    %15 = load i32, i32* %1
    call void @putint(i32 %15)
    call void @putch(i32 10)
    br label %5

16:
    ret i32 0
}
```

□□□□ 1□

```
5
```

□□□□ 1□

```
1
3
6
10
15
```

□□ 2

□□□□ 2□

```
int main() {
    const int ch = 48;
    int i = 1;
    while (i < 12) {
        int j = 0;
        while (j < 2 * i - 1) {
            if (j % 3 == 1) {
                putch(ch + 1);
            } else {
                putch(ch);
            }
            j = j + 1;
        }
        putch(10);
        i = i + 1;
    }
    return 0;
}
```

□□ IR 2□

```
declare void @putch(i32 )
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    store i32 1, i32* %2
    br label %3

3:
    %4 = load i32, i32* %2
    %5 = icmp slt i32 %4, 12
    br i1 %5, label %8, label %7

6:
    store i32 0, i32* %1
    br label %9

7:
    ret i32 0
8:
    br label %6

9:
    %10 = load i32, i32* %1
    %11 = load i32, i32* %2
    %12 = mul i32 2, %11
    %13 = sub i32 %12, 1
    %14 = icmp slt i32 %10, %13
    br i1 %14, label %24, label %21

15:
    %16 = load i32, i32* %1
    %17 = sdiv i32 %16, 3
    %18 = mul i32 %17, 3
    %19 = sub i32 %16, %18
    %20 = icmp eq i32 %19, 1
    br i1 %20, label %31, label %30

21:
    call void @putch(i32 10)
    %22 = load i32, i32* %2
    %23 = add i32 %22, 1
    store i32 %23, i32* %2
    br label %3

24:
    br label %15

25:
    %26 = add i32 48, 1
    call void @putch(i32 %26)
    br label %27
```

```
27:
    %28 = load i32, i32* %1
    %29 = add i32 %28, 1
    store i32 %29, i32* %1
    br label %9

30:
    call void @putch(i32 48)
    br label %27

31:
    br label %25
}
```

□□□□ 2:

```
0
010
01001
0100100
010010010
01001001001
0100100100100
0100100100100100
0100100100100100100
0100100100100100100100
0100100100100100100100100
```

Part 11 continue□break □□□□□

□ Part 11 □□□□□□□□□□□□□□□□ `continue` □ `break` □□□

`continue` □ `break` □□□□□□□□□

□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□

```
CompUnit     -> Decl* FuncDef
Decl         -> ConstDecl | VarDecl
ConstDecl    -> 'const' BType ConstDef { ',' ConstDef } ';'
BType        -> 'int'
ConstDef     -> Ident '=' ConstInitVal
ConstInitVal -> ConstExp
ConstExp     -> AddExp
VarDecl      -> BType VarDef { ',' VarDef } ';'
VarDef       -> Ident
              | Ident '=' InitVal
InitVal      -> Exp
FuncDef      -> FuncType Ident '(' ')' Block // □□□□ Ident
FuncType     -> 'int'
Block        -> '{' { BlockItem } '}'
BlockItem    -> Decl | Stmt
Stmt         -> LVal '=' Exp ';'
              | Block
              | [Exp] ';'
              | 'if' '(' Cond ')' Stmt [ 'else' Stmt ]
              | 'while' '(' Cond ')' Stmt
              | 'break' ';'
              | 'continue' ';'
              | 'return' Exp ';' // [changed]
Exp          -> AddExp
Cond         -> LOrExp
LVal         -> Ident
PrimaryExp   -> '(' Exp ')' | LVal | Number
UnaryExp     -> PrimaryExp
              | Ident '(' [FuncRParams] ')'
              | UnaryOp UnaryExp
UnaryOp      -> '+' | '-' | '!'  // □□ '!' □□□□ Cond □
FuncRParams  -> Exp { ',' Exp }
MulExp       -> UnaryExp
              | MulExp ('*' | '/' | '%') UnaryExp
AddExp       -> MulExp
              | AddExp ('+' | '-') MulExp
RelExp       -> AddExp
              | RelExp ('<' | '>' | '<=' | '>=') AddExp
EqExp        -> RelExp
              | EqExp ('==' | '!=') RelExp
LAndExp      -> EqExp
              | LAndExp '&&' EqExp
LOrExp       -> LAndExp
              | LOrExp '||' LAndExp
```

□□

~~□□ IR □□□□□□□□□□□□□□□□□□□□~~

□□ 1

□□□□ 1□

```
int main() {
    int n = getint();
    int i = 0, sum = 0;
    while (i < n) {
        if (i % 2 == 0) {
            i = i + 1;
            continue;
        }
        i = i + 1;
        sum = sum + i;
        putint(sum);
        putch(10);
    }
    return 0;
}
```

□□ IR 1□

```
declare i32 @getint()
declare void @putint(i32 )
declare void @putch(i32 )
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    %3 = alloca i32
    %4 = call i32 @getint()
    store i32 %4, i32* %3
    store i32 0, i32* %2
    store i32 0, i32* %1
    br label %5

5:
    %6 = load i32, i32* %2
    %7 = load i32, i32* %3
    %8 = icmp slt i32 %6, %7
    br i1 %8, label %14, label %13

9:
    %10 = load i32, i32* %2
    %11 = srem i32 %10, 2
    %12 = icmp eq i32 %11, 0
    br i1 %12, label %25, label %18

13:
    ret i32 0
14:
    br label %9

15:
    %16 = load i32, i32* %2
    %17 = add i32 %16, 1
    store i32 %17, i32* %2
    br label %5

18:
    %19 = load i32, i32* %2
    %20 = add i32 %19, 1
    store i32 %20, i32* %2
    %21 = load i32, i32* %1
    %22 = load i32, i32* %2
    %23 = add i32 %21, %22
    store i32 %23, i32* %1
    %24 = load i32, i32* %1
    call void @putint(i32 %24)
    call void @putch(i32 10)
    br label %5

25:
```

```
    br label %15
}
```

□□□□ 1□

```
10
```

□□□□ 1□

```
2
6
12
20
30
```

□□ 2

□□□□ 2□

```
int main() {
    const int ch = 48;
    int i = 1;
    while (i < 12) {
        int j = 0;
        while (1 == 1) {
            if (j % 3 == 1) {
                putch(ch + 1);
            } else {
                putch(ch);
            }
            j = j + 1;
            if (j >= 2 * i - 1)
                break;
        }
        putch(10);
        i = i + 1;
        continue; // something meaningless
    }
    return 0;
}
```

□□ IR 2□

```
declare void @putch(i32 )
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    store i32 1, i32* %2
    br label %3

3:
    %4 = load i32, i32* %2
    %5 = icmp slt i32 %4, 12
    br i1 %5, label %8, label %7

6:
    store i32 0, i32* %1
    br label %9

7:
    ret i32 0

8:
    br label %6

9:
    %10 = icmp eq i32 1, 1
    br i1 %10, label %18, label %15

11:
    %12 = load i32, i32* %1
    %13 = srem i32 %12, 3
    %14 = icmp eq i32 %13, 1
    br i1 %14, label %30, label %29

15:
    call void @putch(i32 10)
    %16 = load i32, i32* %2
    %17 = add i32 %16, 1
    store i32 %17, i32* %2
    br label %3

18:
    br label %11

19:
    %20 = add i32 48, 1
    call void @putch(i32 %20)
    br label %21

21:
    %22 = load i32, i32* %1
    %23 = add i32 %22, 1
    store i32 %23, i32* %1
```

```
    %24 = load i32, i32* %1
    %25 = load i32, i32* %2
    %26 = mul i32 2, %25
    %27 = sub i32 %26, 1
    %28 = icmp sge i32 %24, %27
    br i1 %28, label %33, label %32

29:
    call void @putch(i32 48)
    br label %21

30:
    br label %19

31:
    br label %15

32:
    br label %9

33:
    br label %31
}
```

□□□□ 2□

```
0
010
01001
0100100
010010010
01001001001
0100100100100
010010010010010
01001001001001001
010010010010010010
0100100100100100100
```

Lab 6 [illegible]

> [illegible]

[illegible]

[illegible]

```
while (cond_a) {
    if (cond_b) {
        break;
    }
    do_some_thing();
}
some_other_things();
```

[illegible] AST [illegible] AST [illegible] `some_other_things();` [illegible] `while` [illegible] `break` [illegible] `br` [illegible]“[illegible] `br` ”[illegible]

[illegible] `some_other_things();` [illegible] `br` [illegible]“[illegible]”[illegible]

[illegible]

[illegible] `analyseWhileStmt()` [illegible] `visitWhileStmt()` [illegible] `break` [illegible] `continue` [illegible]

[illegible]

```
Stack<Recorder> stk = new Stack();

visitWhileStmt() {
    stk.push(new Recorder());
    // do some thing
    stk.top.foreach(mark -> {
        update(mark);
    });
    stk.pop();
}

visitBreakStmt() {
    stk.top.record(new Mark("break"));
}

visitContinueStmt() {
    stk.top.record(new Mark("continue"));
}
```

> □□ `Stack` □□□□□□□□□□□□

□□□□

□□□□□□□□□□□ `lab6` □□□□ `judge.toml` □□□□□□ `[jobs.lab6]` □

□□□□ `$input` □□□□□□□□□□□□□□□□□□ LLVM IR □□□ `$ir` □□□□□□□□ `lli` □□□□□□□□□□□□□□ IR □□□□□□□□□ `judge.toml` □ `run` □□□ `$input` □□□□□□□□□ `$ir` □□□□□□□□□□□ `./compiler < $input > $ir` □ `./compiler $input $ir` □□

```
# □□□□
[jobs.lab6]

image = { source = "dockerfile", path = "." }

run = [
    "./compiler $input $ir",
]
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 0 □□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□

Lab 7□□□

- □□□□□□□□□□□□□ lab 6 □□□□□□□□□□□□□□□□□□
- □□□□□□□□□2022 □ 1 □ 7 □ 23:59
- □□□□□□□□□ `□□_□□_lab7.pdf`
- □□□□□□□□□□□□ `lab7□□□/` □□□□□□□□
- □□□□□□□□□□□2022 □ 1 □ 9 □ 23:59

Part 12 □□□□□□□□□

□ Part 12 □□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□□

```
CompUnit     -> Decl* FuncDef
Decl         -> ConstDecl | VarDecl
ConstDecl    -> 'const' BType ConstDef { ',' ConstDef } ';'
BType        -> 'int'
ConstDef     -> Ident { '[' ConstExp ']' } '=' ConstInitVal
ConstInitVal -> ConstExp
              | '{' [ ConstInitVal { ',' ConstInitVal } ]
ConstExp     -> AddExp
VarDecl      -> BType VarDef { ',' VarDef } ';'
VarDef       -> Ident { '[' ConstExp ']' }
              | Ident { '[' ConstExp ']' } '=' InitVal //
InitVal      -> Exp
              | '{' [ InitVal { ',' InitVal } ] '}' // [c
FuncDef      -> FuncType Ident '(' ')' Block // □□□□ Ident
FuncType     -> 'int'
Block        -> '{' { BlockItem } '}'
BlockItem    -> Decl | Stmt
Stmt         -> LVal '=' Exp ';'
              | Block
              | [Exp] ';'
              | 'if' '(' Cond ')' Stmt [ 'else' Stmt ]
              | 'while' '(' Cond ')' Stmt
              | 'break' ';'
              | 'continue' ';'
              | 'return' Exp ';'
Exp          -> AddExp
Cond         -> LOrExp
LVal         -> Ident {'[' Exp ']'} // [changed]
PrimaryExp   -> '(' Exp ')' | LVal | Number
UnaryExp     -> PrimaryExp
              | Ident '(' [FuncRParams] ')'
              | UnaryOp UnaryExp
UnaryOp      -> '+' | '-' | '!'  // □□ '!' □□□□ Cond □
FuncRParams  -> Exp { ',' Exp }
MulExp       -> UnaryExp
              | MulExp ('*' | '/' | '%') UnaryExp
AddExp       -> MulExp
              | AddExp ('+' | '-') MulExp
RelExp       -> AddExp
              | RelExp ('<' | '>' | '<=' | '>=') AddExp
EqExp        -> RelExp
              | EqExp ('==' | '!=') RelExp
LAndExp      -> EqExp
              | LAndExp '&&' EqExp
LOrExp       -> LAndExp
              | LOrExp '||' LAndExp
```

□□□□

`ConstDef`

- `ConstDef` [illegible] C [illegible] `[2 * 3][8 / 2]` [illegible] 6 [illegible] 4[illegible] 0 [illegible] `ConstDef` [illegible] `ConstExp` [illegible]
- [illegible] `ConstDef` [illegible] `=` [illegible] `ConstInitVal` [illegible] `ConstInitVal` [illegible] `ConstExp` [illegible] `ConstInitVal` [illegible] `ConstExp` [illegible] `int` [illegible]
- `ConstInitVal` [illegible]
  - [illegible] `{}` [illegible] 0[illegible]
  - [illegible] `int a[3] = {1, 2, 3};` [illegible] `int a[3][2] = { {1, 2}, {3, 4}, {5, 6} };` [illegible]
  - [illegible] 0[illegible] `int a[5] = {1, 2};` [illegible] `int a[4][4] = { {1, 2, 3}, {4, 5}, {} };` [illegible]

`VarDef`

- `VarDef` [illegible] C [illegible] `VarDef` [illegible] `ConstExp` [illegible]
- [illegible] `InitVal` [illegible] `Exp` [illegible] `InitVal` [illegible] `Exp` [illegible]

`LVal`

[illegible] `LVal` [illegible]

[illegible]/[illegible]

> "[illegible]"[illegible] `int` [illegible]/[illegible]

- [illegible] `int` [illegible]/[illegible]
- [illegible] `int` [illegible] C [illegible]
- [illegible]
- [illegible] `ConstInitVal/InitVal` [illegible] `ConstExp/Exp` [illegible]
- [illegible] `ConstInitVal` [illegible] `ConstExp` [illegible] C [illegible]

[illegible]

[illegible]

LLVM IR □□□□□□□

> □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ clang □□□ LLVM IR□□□□□
> clang □□□ LLVM IR □□□□□□□□□□□

- □□□□□□□ C □□□□□ `memset(pointer, 0, size * sizeof(int))` □□□□□□□□□ 0□□□ `pointer` □□□□□□□□□□□□ `size` □□□□ □□ `sizeof(int)` □ 4□□□□□ `store` □□□□□□□□□□□□□□□□□□□□□□□□□
  - □□□□□ `memset` □□□□□□□□□ IR □□□□□□□□□
- □□□ `@arr = dso_local global [3 x i32] [i32 1, i32 2, i32 3]` □□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□ `@arr = dso_local global [2 x [2 x i32]] [[2 x i32] [i32 1, i32 2], [2 x i32] [i32 3, i32 0]]` □□□□□□ `zeroinitializer` □□□□□□□□□□□□□□□ 0□

□□

□□ 1

□□□□ 1□

```
int main() {
    int a[2][2] = {{1}, {2, 3}};
    int e[2][2] = {{a[0][0], a[1][1]}, {5, 6}};
    putint(e[1][1] + a[1][0]);
    return 0;
}
```

□□ IR 1□

```
declare void @putint(i32)
declare void @memset(i32*, i32, i32)
define dso_local i32 @main() {
    %1 = alloca [2 x [2 x i32]]
    %2 = alloca [2 x [2 x i32]]
    %3 = getelementptr [2 x [2 x i32]], [2 x [2 x i32]]* %2
    %4 = getelementptr [2 x i32], [2 x i32]* %3, i32 0, i32
    call void @memset(i32* %4, i32 0, i32 16)
    store i32 1, i32* %4
    %5 = getelementptr i32, i32* %4, i32 2
    store i32 2, i32* %5
    %6 = getelementptr i32, i32* %4, i32 3
    store i32 3, i32* %6
    %7 = getelementptr [2 x [2 x i32]], [2 x [2 x i32]]* %2
    %8 = add i32 0, 0
    %9 = mul i32 %8, 2
    %10 = getelementptr [2 x i32], [2 x i32]* %7, i32 0, i3
    %11 = add i32 %9, 0
    %12 = getelementptr i32, i32* %10, i32 %11
    %13 = load i32, i32* %12
    %14 = getelementptr [2 x [2 x i32]], [2 x [2 x i32]]* %
    %15 = add i32 0, 1
    %16 = mul i32 %15, 2
    %17 = getelementptr [2 x i32], [2 x i32]* %14, i32 0, i
    %18 = add i32 %16, 1
    %19 = getelementptr i32, i32* %17, i32 %18
    %20 = load i32, i32* %19
    %21 = getelementptr [2 x [2 x i32]], [2 x [2 x i32]]* %
    %22 = getelementptr [2 x i32], [2 x i32]* %21, i32 0, i
    call void @memset(i32* %22, i32 0, i32 16)
    store i32 %13, i32* %22
    %23 = getelementptr i32, i32* %22, i32 1
    store i32 %20, i32* %23
    %24 = getelementptr i32, i32* %22, i32 2
    store i32 5, i32* %24
    %25 = getelementptr i32, i32* %22, i32 3
    store i32 6, i32* %25
    %26 = getelementptr [2 x [2 x i32]], [2 x [2 x i32]]* %
    %27 = add i32 0, 1
    %28 = mul i32 %27, 2
    %29 = getelementptr [2 x i32], [2 x i32]* %26, i32 0, i
    %30 = add i32 %28, 1
    %31 = getelementptr i32, i32* %29, i32 %30
    %32 = load i32, i32* %31
    %33 = getelementptr [2 x [2 x i32]], [2 x [2 x i32]]* %
    %34 = add i32 0, 1
    %35 = mul i32 %34, 2
    %36 = getelementptr [2 x i32], [2 x i32]* %33, i32 0, i
    %37 = add i32 %35, 0
    %38 = getelementptr i32, i32* %36, i32 %37
    %39 = load i32, i32* %38
```

```
    %40 = add i32 %32, %39
    call void @putint(i32 %40)
    ret i32 0
}
```

□□□□ 1□

```
8
```

□□ 2

□□□□ 2□

```
const int c[2][1] = {{1}, {3}};
int b[2][3] = {{1}}, e[4][4];
int d[5], a[3] = {1, 2};
int main() {
    putint(c[1][0] + b[0][0] + c[0][0] + a[1] + d[4]);
    return 0;
}
```

□□ IR 2□

```
declare void @memset(i32*  ,i32 ,i32 )
declare void @putint(i32 )

@c = dso_local constant [2 x [1 x i32]] [[1 x i32] [i32 1],
@b = dso_local global [2 x [3 x i32]] [[3 x i32] [i32 1, i3
@e = dso_local global [4 x [4 x i32]] zeroinitializer
@d = dso_local global [5 x i32] zeroinitializer
@a = dso_local global [3 x i32] [i32 1, i32 2, i32 0]

define dso_local i32 @main() {
    %1 = getelementptr [2 x [1 x i32]], [2 x [1 x i32]]* @c
    %2 = add i32 0, 1
    %3 = mul i32 %2, 1
    %4 = getelementptr [1 x i32], [1 x i32]* %1, i32 0, i32
    %5 = add i32 %3, 0
    %6 = getelementptr i32, i32* %4, i32 %5
    %7 = load i32, i32* %6
    %8 = getelementptr [2 x [3 x i32]], [2 x [3 x i32]]* @b
    %9 = add i32 0, 0
    %10 = mul i32 %9, 3
    %11 = getelementptr [3 x i32], [3 x i32]* %8, i32 0, i3
    %12 = add i32 %10, 0
    %13 = getelementptr i32, i32* %11, i32 %12
    %14 = load i32, i32* %13
    %15 = add i32 %7, %14
    %16 = getelementptr [2 x [1 x i32]], [2 x [1 x i32]]* @
    %17 = add i32 0, 0
    %18 = mul i32 %17, 1
    %19 = getelementptr [1 x i32], [1 x i32]* %16, i32 0, i
    %20 = add i32 %18, 0
    %21 = getelementptr i32, i32* %19, i32 %20
    %22 = load i32, i32* %21
    %23 = add i32 %15, %22
    %24 = getelementptr [3 x i32], [3 x i32]* @a, i32 0, i3
    %25 = add i32 0, 1
    %26 = getelementptr i32, i32* %24, i32 %25
    %27 = load i32, i32* %26
    %28 = add i32 %23, %27
    %29 = getelementptr [5 x i32], [5 x i32]* @d, i32 0, i3
    %30 = add i32 0, 4
    %31 = getelementptr i32, i32* %29, i32 %30
    %32 = load i32, i32* %31
    %33 = add i32 %28, %32
    call void @putint(i32 %33)
    ret i32 0
}
```

□□□□ 2□

```
7
```

□□ 3

□□□□□ 3□

```
int a = 1;
int b[2] = {1, a};
int main() {
    putint(b[1]);
    return 0;
}
```

□□□□□ 3□

□□□□□□□□ **0** □□□□□□□□□

□□ 4

□□□□□ 4□

```
int arr[2][2] = {{1, 1}, {4, 5}};
int main() {
    arr[1] = 2;
    putint(arr[1][0]);
    return 0;
}
```

□□□□□ 4□

□□□□□□□□ **0** □□□□□□□□□

Lab 7 □□□□

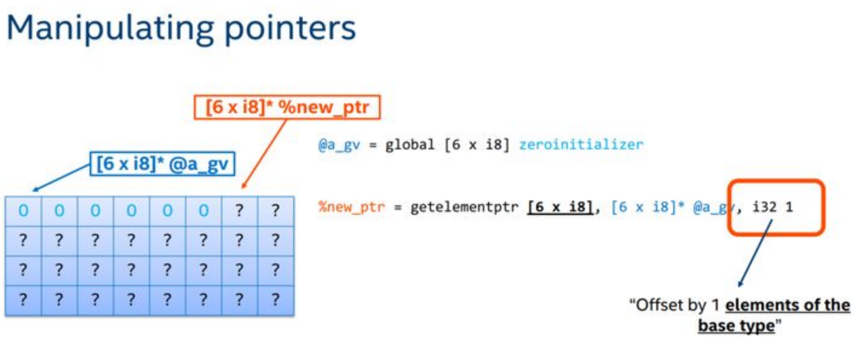
□□ `getElementPtr` □□

□□□□□□□□□ lab □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ LLVM IR □□□□□ □□□ clang □□□□□□□□□□□□□□ `getElementPtr` □□□□□□ GEP) □□□□□□□ □□□ □□□

LLVM □□□□□ GEP □□□□□□□□□□□□□□□□□

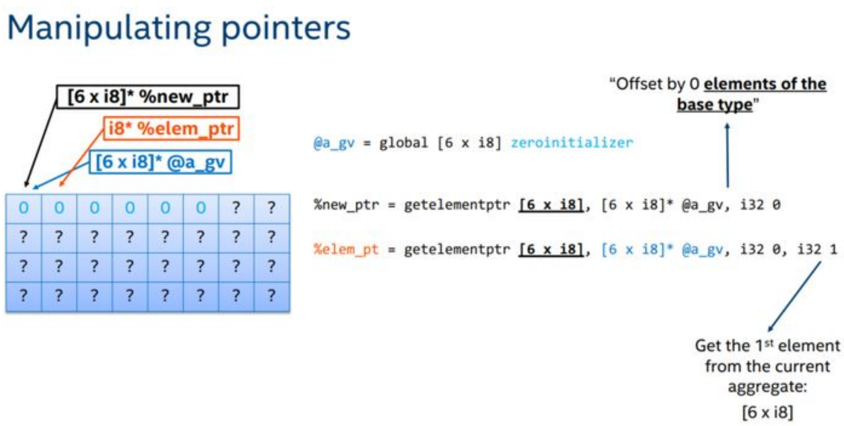
GEP □□□□□□□“□□□□”□□□□□□□□□□□□□□□□□□□□□GEP □□□□□□□□□ `<result> = getelementptr <ty>, <ty>* <ptrval>, {<ty> <index>}*` □□□□□□ `<ty>` □□□□□□□□□□□□□□□□□□ `<ty>` □□□□□□□□□ `<ptrval>` □□□□ `<ty> <index>` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□ `i32` □ `i64` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□

□□□□□□□□□□□□□□□□□□□□□□□□□ `ptrval` □□□ `<ty>*` □□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `ty` □□□□□□□□□□□□□□□



□□□□□□□□□□□□□□□□□ `[6 x i8]` □□□□□ 1□□□ GEP □□□□□□□□□□□ `@a_gv` □□□ 6 □□□□□□□□□□□□□□□□□□□□□□□□ `[6 x i8]*` □□□

□□



□□□ `%elem_ptr = getelementptr [6 x i8], [6 x i8]* @a_gv, i32 0, i32 1` □□□□□□□□□□□□ 0□□□□□□□□ `[6 x i8]` □□□□□□□□□□□□□□□ 0 x 6 □□□□□□□□□□□□□□□□□□□□□□ 1□□□□□□□□□□□□ `i8` □ `[6 x i8]` □□□□□ 6□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `i8*` □

GEP □□ `[4][5]` □□□□□□□□□□□□□□□□ `[20]` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□ `a[5][4]` □□□ GEP □□□□ `a[2][3]` □□□□□□□□□□□□ GEP □□□□□□

```
; 1
@a = global [5 x [4 x i32]] zeroinitializer
%1 = getelementptr [5 x [4 x i32]], [5 x [4 x i32]]* @a, i3

; 2
@a = global [5 x [4 x i32]] zeroinitializer
%1 = getelementptr [5 x [4 x i32]], [5 x [4 x i32]]* @a, i3
%2 = getelementptr [4 x i32], [4 x i32]* %1, i32 0, i32 3 ;

; 3
@a = global [5 x [4 x i32]] zeroinitializer
%1 = getelementptr [5 x [4 x i32]], [5 x [4 x i32]]* @a, i3
%2 = getelementptr [4 x i32], [4 x i32]* %1, i32 2, i32 3 ;

; 4
@a = global [20 x i32] zeroinitializer
%1 = mul i32 2, 4
%2 = add i32 %1, 3
%3 = getelementptr [20 x i32], [20 x i32]* @a, i32 0, i32 %

; 5
@a = global [20 x i32] zeroinitializer
%1 = getelementptr [20 x i32], [20 x i32]* @a, i32 0, i32 0
%2 = mul i32 2, 4


%3 = getelementptr i32, i32* %1, i32 %2 ; %3 □□□ i32*
%4 = getelementptr i32, i32* %3, i32 3 ; %4 □□□ i32*
```

□□□□

□□□□□□□□□□□□ `lab7` □□□□ `judge.toml` □□□□□□ `[jobs.lab7]` □

□□□□ `$input` □□□□□□□□□□□□□□□□□□□ LLVM IR □□□ `$ir` □□□□□□□□□ `lli` □□□□□□□□□□□□□□□□ IR □□□□□□□□□□ `judge.toml` □ `run` □□□ `$input` □□□□□□□□□□ `$ir` □□□□□□□□□□□□ `./compiler < $input > $ir` □ `./compiler $input $ir` □□

```toml
# □□□□
[jobs.lab7]

image = { source = "dockerfile", path = "." }

run = [
    "./compiler $input $ir",
]
```

□□□ 0 □□

Lab 8□□□

- □□□□□□□□□□□□□ lab 7 □□□□□□□□□□□□
- □□□□□□□□□2022 □ 1 □ 7 □ 23:59
- □□□□□□□□□ `□□_□□_lab8.pdf`
- □□□□□□□□□□□□ `lab8□□□/` □□□□□□□□
- □□□□□□□□□□□2022 □ 1 □ 9 □ 23:59

Part 13 □□

□ Part 13 □□□□□□□□□□□□□□□□□□

□□□□ IR □□

□□□□□□□□□□□□□□□ `CompUnit` □□□□□□□□

```
CompUnit     -> [CompUnit] (Decl | FuncDef) // [changed]
Decl         -> ConstDecl | VarDecl
ConstDecl    -> 'const' BType ConstDef { ',' ConstDef } ';'
BType        -> 'int'
ConstDef     -> Ident { '[' ConstExp ']' } '=' ConstInitVal
ConstInitVal -> ConstExp
              | '{' [ ConstInitVal { ',' ConstInitVal } ]
ConstExp     -> AddExp
VarDecl      -> BType VarDef { ',' VarDef } ';'
VarDef       -> Ident { '[' ConstExp ']' }
              | Ident { '[' ConstExp ']' } '=' InitVal
InitVal      -> Exp
              | '{' [ InitVal { ',' InitVal } ] '}'
FuncDef      -> FuncType Ident '(' [FuncFParams] ')' Block
FuncType     -> 'void' | 'int' // [changed]
FuncFParams  -> FuncFParam { ',' FuncFParam } // [new]
FuncFParam   -> BType Ident ['[' ']' { '[' Exp ']' }] // [n
Block        -> '{' { BlockItem } '}'
BlockItem    -> Decl | Stmt
Stmt         -> LVal '=' Exp ';'
              | Block
              | [Exp] ';'
              | 'if' '(' Cond ')' Stmt [ 'else' Stmt ]
              | 'while' '(' Cond ')' Stmt
              | 'break' ';'
              | 'continue' ';'
              | 'return' [Exp] ';' // [changed]
Exp          -> AddExp
Cond         -> LOrExp
LVal         -> Ident {'[' Exp ']'}
PrimaryExp   -> '(' Exp ')' | LVal | Number
UnaryExp     -> PrimaryExp
              | Ident '(' [FuncRParams] ')'
              | UnaryOp UnaryExp
UnaryOp      -> '+' | '-' | '!'  // □□ '!' □□□□ Cond □
FuncRParams  -> Exp { ',' Exp }
MulExp       -> UnaryExp
              | MulExp ('*' | '/' | '%') UnaryExp
AddExp       -> MulExp
              | AddExp ('+' | '-') MulExp
RelExp       -> AddExp
              | RelExp ('<' | '>' | '<=' | '>=') AddExp
EqExp        -> RelExp
              | EqExp ('==' | '!=') RelExp
LAndExp      -> EqExp
              | LAndExp '&&' EqExp
LOrExp       -> LAndExp
              | LOrExp '||' LAndExp
```

□□□□

`CompUnit`

- □□□□□□□□□□□□ `main` □□□□□□□ `int` □ `FuncDef` □ `main` □□□□□□□
- `CompUnit` □□□□/□□□□□□□□□ `Decl` □□□□□□□ `FuncDef` □□□ □□□□□□□□□ `Ident` □□□□□□□□□□□□□
- `CompUnit` □□□/□□/□□□□□□□□□□□□□□□□□
    - □□□□□□□□□□□□□□□□□□□□

`FuncFParam` □□□

- `FuncFParam` □□□□□□□□□□□□ `Ident` □□□□□□□□□□□□□□□□□ □□□□□□
- □ `FuncFParam` □□□□□□□□□□□□□□□□□□□□□ `[]` □□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□ `Exp` □□□ `int` □□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□
    - □□
- □□□□□□□□□□□□□□□□□□□□□□□ `int a[4][3]` □□□□ `a[1]` □□□□□ □□□□□□□□□□□□□□□□ `int[]` □□□□
- □□□□□□□□□□□□□□□□□□□ `Ident` □□□□□□□□□□□□□□□

`FuncDef`

- `FuncDef` □□□□□□□□□□ `FuncType` □□□□□□□
    - □□□□□□ `int` □□□□□□□□□□□□□□□□ `Exp` □ `return` □□□□□ □ `return` □□□□□□□□□□□□□□□
    - □□□□□□□ `void` □□□□□□□□□□□□□□□ `return` □□□

□□□□□□□□

□□□□□□□□□□□□□□□□□□

1. `int getarray(int []);` □□□□□□□□□ 1 □□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□ `getarray()` □□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□

```
int a[10][10];
int n;
n = getarray(a[0]);
```

2. `void putarray(int, int[]);` □ 1 □□□□□□□□□□□□□□□□□□□□ N□□□ □□□□□□□□□□ N □□□□□□□□ `putarray()` □□□□□□□□□□□□□□□□□□

```
int n = 2;
int a[2] = {2, 3};
putarray(n, a);
```

□□

□□ 1

□□□□ 1□

```
int func1() {
    return 555;
}

int func2() {
    return 111;
}

int main() {
    int a = func1();
    putint(a - func2());
    return 0;
}
```

□□ IR 1□

```
declare void @putint(i32)
define dso_local i32 @func1() {
    ret i32 555
}
define dso_local i32 @func2() {
    ret i32 111
}
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = call i32 @func1()
    store i32 %2, i32* %1
    %3 = load i32, i32* %1
    %4 = call i32 @func2()
    %5 = sub i32 %3, %4
    call void @putint(i32 %5)
    ret i32 0
}
```

□□□□ 1□

```
444
```

□□ 2

□□□□ 2□

```
void set1(int pos, int arr[]) {
    arr[pos] = 1;
}

int main() {
    int a[2][5];
    int n;
    n = getarray(a[0]);
    getarray(a[1]);
    int i = 0;
    while (i < n) {
        set1(i, a[i % 2]);
        i = i + 1;
    }
    putarray(n, a[0]);
    putarray(n, a[1]);
    return 0;
}
```

□□ IR 2□

```
declare i32 @getarray(i32*)
declare void @putarray(i32, i32*)

define dso_local void @set1(i32 %0, i32* %1) {
    %3 = alloca i32*
    %4 = alloca i32
    store i32 %0, i32* %4
    store i32*  %1, i32* * %3
    %5 = load i32* , i32* * %3
    %6 = load i32, i32* %4
    %7 = getelementptr i32, i32* %5, i32 %6
    store i32 1, i32* %7
    ret void
}

define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    %3 = alloca [2 x [5 x i32]]
    %4 = getelementptr [2 x [5 x i32]], [2 x [5 x i32]]* %3
    %5 = add i32 0, 0
    %6 = mul i32 %5, 5
    %7 = getelementptr [5 x i32], [5 x i32]* %4, i32 0, i32
    %8 = call i32 @getarray(i32* %7)
    store i32 %8, i32* %2
    %9 = getelementptr [2 x [5 x i32]], [2 x [5 x i32]]* %3
    %10 = add i32 0, 1
    %11 = mul i32 %10, 5
    %12 = getelementptr [5 x i32], [5 x i32]* %9, i32 0, i3
    %13 = call i32 @getarray(i32* %12)
    store i32 0, i32* %1
    br label %14

14:
    %15 = load i32, i32* %1
    %16 = load i32, i32* %2
    %17 = icmp slt i32 %15, %16
    br i1 %17, label %18, label %28

18:
    %19 = load i32, i32* %1
    %20 = getelementptr [2 x [5 x i32]], [2 x [5 x i32]]* %
    %21 = load i32, i32* %1
    %22 = srem i32 %21, 2
    %23 = add i32 0, %22
    %24 = mul i32 %23, 5
    %25 = getelementptr [5 x i32], [5 x i32]* %20, i32 0, i
    call void @set1(i32 %19, i32* %25)
    %26 = load i32, i32* %1
    %27 = add i32 %26, 1
    store i32 %27, i32* %1
```

```
    br label %14

28:
    %29 = load i32, i32* %2
    %30 = getelementptr [2 x [5 x i32]], [2 x [5 x i32]]* %
    %31 = add i32 0, 0
    %32 = mul i32 %31, 5
    %33 = getelementptr [5 x i32], [5 x i32]* %30, i32 0, i
    call void @putarray(i32 %29, i32* %33)
    %34 = load i32, i32* %2
    %35 = getelementptr [2 x [5 x i32]], [2 x [5 x i32]]* %
    %36 = add i32 0, 1
    %37 = mul i32 %36, 5
    %38 = getelementptr [5 x i32], [5 x i32]* %35, i32 0, i
    call void @putarray(i32 %34, i32* %38)
    ret i32 0
}
```

□□□□ 2□

```
5 1 2 3 4 5
5 6 7 8 9 10
```

□□□□ 2□

```
5: 1 2 1 4 1
5: 6 1 8 1 10
```

□□ 3

□□□□ 3□

```
int gcd(int m, int n) {
    if (n == 0) {
        return m;
    }
    return gcd(n, m % n);
}

int main() {
    int a = 100, b = 48;
    putint(gcd(a, b));
    return 0;
}
```

□□ IR 3□

```
declare void @putint(i32 )

define dso_local i32 @gcd(i32 %0, i32 %1) {
    %3 = alloca i32
    %4 = alloca i32
    store i32 %0, i32* %4
    store i32 %1, i32* %3
    %5 = load i32, i32* %3
    %6 = icmp eq i32 %5, 0
    br i1 %6, label %7, label %9

7:
    %8 = load i32, i32* %4
    ret i32 %8

9:
    %10 = load i32, i32* %3
    %11 = load i32, i32* %4
    %12 = load i32, i32* %3
    %13 = srem i32 %11, %12
    %14 = call i32 @gcd(i32 %10, i32 %13)
    ret i32 %14
}

define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    store i32 100, i32* %2
    store i32 48, i32* %1
    %3 = load i32, i32* %2
    %4 = load i32, i32* %1
    %5 = call i32 @gcd(i32 %3, i32 %4)
    call void @putint(i32 %5)
    ret i32 0
}
```

□□□□ 3□

```
4
```

□□ 4

□□□□ 4□

```
int sum2d(int a[][3]) {
    int i = 0, sum = 0;
    while (i < 2) {
        int j = 0;
        while (j < 3) {
            sum = sum + a[i][j];
            j = j + 1;
        }
        i = i + 1;
    }
    return sum;
}

int main() {
    int arr[2][3] = {{1, 2, 3}, {4, 5}};
    putint(sum2d(arr));
    return 0;
}
```

□□ IR 4□

```
declare void @memset(i32*, i32, i32)
declare void @putint(i32 )

define dso_local i32 @sum2d([3 x i32]* %0) {
    %2 = alloca i32
    %3 = alloca i32
    %4 = alloca i32
    %5 = alloca [3 x i32]*
    store [3 x i32]* %0, [3 x i32]* * %5
    store i32 0, i32* %4
    store i32 0, i32* %3
    br label %6

6:
    %7 = load i32, i32* %4
    %8 = icmp slt i32 %7, 2
    br i1 %8, label %12, label %10

9:
    store i32 0, i32* %2
    br label %13

10:
    %11 = load i32, i32* %3
    ret i32 %11

12:
    br label %9

13:
    %14 = load i32, i32* %2
    %15 = icmp slt i32 %14, 3
    br i1 %15, label %16, label %29

16:
    %17 = load i32, i32* %3
    %18 = load [3 x i32]*, [3 x i32]* * %5
    %19 = load i32, i32* %4
    %20 = getelementptr [3 x i32], [3 x i32]* %18, i32 0
    %21 = mul i32 %19, 3
    %22 = load i32, i32* %2
    %23 = add i32 %21, %22
    %24 = getelementptr [3 x i32], [3 x i32]* %20, i32 0, i
    %25 = load i32, i32* %24
    %26 = add i32 %17, %25
    store i32 %26, i32* %3
    %27 = load i32, i32* %2
    %28 = add i32 %27, 1
    store i32 %28, i32* %2
    br label %13
```

```
29:
    %30 = load i32, i32* %4
    %31 = add i32 %30, 1
    store i32 %31, i32* %4
    br label %6
}

define dso_local i32 @main() {
    %1 = alloca [2 x [3 x i32]]
    %2 = getelementptr [2 x [3 x i32]], [2 x [3 x i32]]* %1
    %3 = getelementptr [3 x i32], [3 x i32]* %2, i32 0, i32
    call void @memset(i32* %3, i32 0, i32 24)
    store i32 1, i32* %3
    %4 = getelementptr i32, i32* %3, i32 1
    store i32 2, i32* %4
    %5 = getelementptr i32, i32* %3, i32 2
    store i32 3, i32* %5
    %6 = getelementptr i32, i32* %3, i32 3
    store i32 4, i32* %6
    %7 = getelementptr i32, i32* %3, i32 4
    store i32 5, i32* %7
    %8 = getelementptr [2 x [3 x i32]], [2 x [3 x i32]]* %1
    %9 = call i32 @sum2d([3 x i32]* %8)
    call void @putint(i32 %9)
    ret i32 0
}
```

□□□□ 4□

```
15
```

□□ 5

□□□□ 5□

```
int foo(int a, int b) {
    int t = a + b;
    a = t - a;
    b = t - b;
    return a - b;
}

int main() {
    putint(foo(1, 2, 3));
    return 0;
}
```

□□□□ 5□

□□□□□□□ **0** □□□□□□□

□□□□

□□□□□□□□□□□□ `lab8` □□□□ `judge.toml` □□□□□□ `[jobs.lab8]` □

□□□□ `$input` □□□□□□□□□□□□□□□□□□□□ LLVM IR □□□ `$ir` □□□□□□□□□ `lli` □□□□□□□□□□□□□□□□ IR □□□□□□□□□□ `judge.toml` □ `run` □□□ `$input` □□□□□□□□□□ `$ir` □□□□□□□□□□□□ `./compiler < $input > $ir` □ `./compiler $input $ir` □□

```toml
# □□□□
[jobs.lab8]

image = { source = "dockerfile", path = "." }

run = [
    "./compiler $input $ir",
]
```

□□□ 0 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

[illegible]

[illegible] 4 [illegible]

- mem2reg[illegible]20% [illegible]
- [illegible]10% [illegible]
- [illegible]10% [illegible]
- [illegible]10% [illegible]

[illegible] 20%[illegible]

[illegible] `[illegible]/` [illegible]

[illegible] 7 [illegible]

□□□□□mem2reg

LLVM IR □□ “memory □□ SSA value” □□□□□□□□□□□□□□□□□□ LLVM IR □□□□□□□□□□□□□ SSA □□□□□□□□□□□□□□ `alloca/load/store` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ mem2reg□□□□ `int` □□□□ `alloca/load/store` □□□ LLVM IR □□□□□□ SSA □□□ LLVM IR □□□□□□ mem2reg □□□□□□ LLVM IR □□□□□□□□□□ `int` □□□□□□□□ `alloca/load/store` □□□□□□□□□□□□□□□□□□□□□ `phi` □□□

> □□□□□ `int` □□□□□□□□□□□□□□□□□□□

□□□□□□ mem2reg □□□□□

□□

□□□□□□□□□□□ `mem2reg` □□□□ `judge.toml` □□□□□□□ `[jobs.mem2reg]` □

□□□□ `$input` □□□□□□□□□□□□□□□□□□□ LLVM IR □□□ `$ir` □□□□□□□□□ `lli` □□□□□□□□□□□□□□□ IR □□□□□□□□□□ `judge.toml` □ `run` □□□ `$input` □□□□□□□□□ `$ir` □□□□□□□□□□□□ `./compiler < $input > $ir` □ `./compiler $input $ir` □□

```
# □□□□
[jobs.mem2reg]

image = { source = "dockerfile", path = "." }

run = [
    "./compiler $input $ir",
]
```

□□□□

□□ 7 □□□□□□□□□□□□□□□□

- □□□□□□□□□2022 □ 1 □ 7 □ 23:59
- □□□□□□□□□ `□□_□□_mem2reg.pdf`
- □□□□□□□□□□□ `□□□□/` □□□□□□□
- □□□□□□□□□□□2022 □ 1 □ 9 □ 23:59

mem2reg □□□□

> □□□□□□□□□ mem2reg □□□□□□□□□□□□□ *Static Single Assignment book* □ *Engineering a Compiler*□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□

□ LLVM □□ SSA □□□□□□□□□□□ SSA □□□□□□□LLVM IR □□□□□□□□□□□□ SSA □□□□□□□□□□□ SSA □□□□□□□□□□□□□□□□□□□□ LLVM IR □□□□□□□□□□ □□□ SSA □□□□□□□□□□□□□□□ `alloca/load/store` □□□

LLVM □ pass□□□□□□□□□□ LLVM IR □□□□□□□□□□□□□□□mem2reg □□□□□□□□ pass□□ mem2reg □□LLVM □□□□□□□□□□ `alloca` □□□□□□□□□□□□□□□□□□□ □□□□ SSA □□□□□□□□□□□□□ `store/load` □□□□□□□□□□□□ def-use/use-def □□□□□□□□□□□□□□ `phi` □□□□□□□□□□□□□□

□□□□□□□□□□□

```
int main() {
    int x, cond = 1;
    if (cond > 0)
        x = 1;
    else
        x = -1;
    return x;
}
```

□□□ `alloca/load/store` □□□ IR □□□

```
define dso_local i32 @main() {
    %1 = alloca i32
    %2 = alloca i32
    store i32 1, i32* %1
    %3 = load i32, i32* %1
    %4 = icmp sgt i32 %3, 0
    br i1 %4, label %5, label %8

5:
    store i32 1, i32* %2
    br label %6

6:
    %7 = load i32, i32* %2
    ret i32 %7

8:
    %9 = sub i32 0, 1
    store i32 %9, i32* %2
    br label %6
}
```

□□ mem2reg □□□□□ IR □□□

```
define dso_local i32 @main() {
    %1 = icmp sgt i32 1, 0
    br i1 %1, label %2, label %5

2:
    br label %3

3:
    %4 = phi i32 [ 1, %2 ], [ %6, %5 ]
    ret i32 %4

5:
    %6 = sub i32 0, 1
    br label %3
}
```

□□□□□ `alloca/load/store` □□□ IR □□ `%1, %2` □□□□ `x` □ `cond` □ □□□□□□ `load` □□□□□□ `%3, %7` □□□□□□□□□□□□ `store` □□□□□□□□ mem2reg □□ IR □ `3` □□□□□□□□□□ `phi` □□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□ `2` □□□□ `3` □ `%4` □ `1` □□□□□□□□ `5` □□ □□ `3` □ `%4` □ `%6` □□□□□□□□□□□□□□□□□□ `phi` □□□□□□□□□□□□□□□□ `phi` □□□

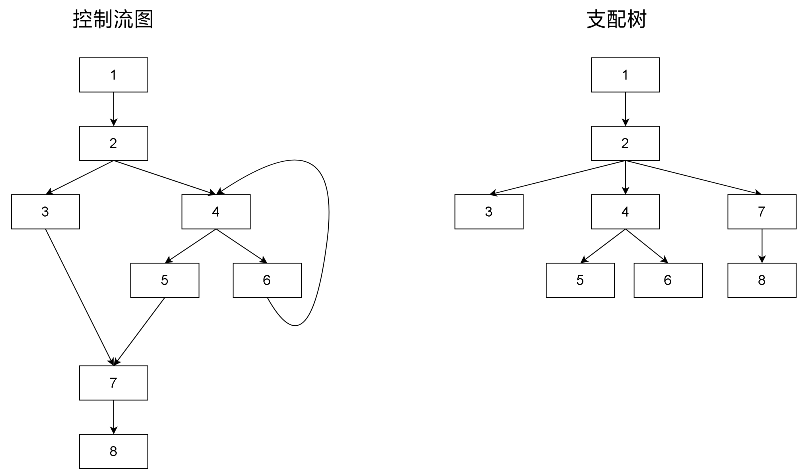
```
define dso_local i32 @main() {
    %1 = icmp slt i32 0, 5
    br i1 %1, label %6, label %2

2:
    %3 = phi i32 [ 0, %0 ], [ %10, %6 ]
    %4 = phi i32 [ 0, %0 ], [ %7, %6 ]
    %5 = phi i32 [ 0, %0 ], [ %11, %6 ]
    ret i32 %5

6:
    %7 = phi i32 [ 0, %0 ], [ %10, %6 ]
    %8 = phi i32 [ 0, %0 ], [ %7, %6 ]
    %9 = phi i32 [ 0, %0 ], [ %11, %6 ]
    %10 = add i32 %7, 1
    %11 = add i32 %9, 1
    %12 = icmp slt i32 %7, 5
    br i1 %12, label %6, label %2
}
```

□□ IR □□□□□□□□□ `6` □□□ `%10 = add i32 %7, 1` □□ `%10 = add i32 %8, 1` □□□□□□□□□ `10` □□□□□□□□□□□□□□ `phi` □□□□□□□□□□□□□□□□□ `phi` □□□□□□□□□□□□□ `6` □□□ `6` □□□ `%8` □□ `%7` □□□□□□□□□ `%7` □□□□□ `%7` □□□ `%10` □□□

□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□/□□□□□□□□□□□□□□□□□□□□□□
- □□□□□Control Flow Graph, CFG□□. CFG □□□□□□□□□□□/□□□□□□□□□□□□□□□□□□□□□□□□□□.
- □□□dominate□□□□ CFG □□□□ n_1 □ n_2□n_1 □□ n_2 □□□□□□□□□□□□□□ n_2 □□□□□□□□ n_1□□ n_1 □□□□□□□□ n_2 □□□□□. □□□□□□□□□□□□.
- □□□□□strictly dominate□□n_1 □□□□ n_2 □□□□ n_1 □□ n_2 □ $n_1 \neq n_2$.
- □□□□□□immediate dominator, idom□□□□ n □□□□□□□□□ n □□□□□□□□ n □□□□□□□□□□□□□□□□ n□□□□□□□□□□□□□□□□ n □□□□□□□□. □□□□□□□□□□□□□□□□□□□. □□□□□□□□□□□□□□□□□□□□□□□□□□dominator tree□.

□□□□□□□□ CFG □□□□□□□□□□



- □□□□□dominance frontier□□□□ n □□□□□□ CFG □□□□□ n □□□□□□□. □□□□□□□□□□□□□ n □□□□□ $DF(n) = \{x|n□□x□□□□□□n□□□□□x\}$.

□□□□□□ `x` □□□□ `a` □□□□□□□□□□□□□□□□□□□□□□□□□□kill□□□□□□□□□□ `x` □□□□□□□□ `a` □□□□□ `x` □□□□□□□□□□ `x` □□□□□□□□□□□□□□□□□□□□□——□□□□□□□□□□□□□□ `x` □ `a` □□□□□□□□□ `x` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `a` □□□□□□□□□□□□□□□□□ `a` □□□ `x` □□□□□□□□□□□□□□□□□□□□□□□ `4` □□ `7` □□□□□ `5` □□□ `4` □□□ `7` □ `4` □□□□□□ `{7}` □□□□□□□ `a` □ `2` □□□□□□ `3` □ `4` □□□□□□□ 0 □ 1□□□□□□□□□□□□□□ `7` □ `a` □□□ 0 □□ 1□

□□ CFG □□□□□□□□□□□□□□□□□□

Algorithm 3.2: Algorithm for computing the dominance frontier of each CFG node.

```
for (a, b) ∈ CFG edges do
    x ← a
    while x does not strictly dominate b do
        DF(x) ← DF(x) ∪ b
        x ← immediate dominator(x)
```

□□

- □□□□□□□iterated dominance frontier□□□□□ S □□□□□□□ $DF^+(S)$ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□. □□□□□□$DF^+(S) = DF_{i\to\infty}(S),\ \ DF_1(S) = DF(S),\ \ DF_{i+1}(S) = DF(S \cup DF_i(S))$.

SSA □□□□

□□□ SSA □□□□□□□ SSA □□□□□□□□□ *Static Single Assignment book* Chapter 3: Standard Construction and Destruction Algorithms □□□

> *Static Single Assignment book* □□□□□□□□□□□□□□□□□□

□□□□□□□□□□□ SSA □□□□□□□ *Static Single Assignment book* □□□□□□□□□□□□

SSA □□□□□□□□□□□□ ϕ □□□□□□□□□□□

□□ phi □□

□□□□□ v □□□□□ d □□□□□□□□□□□□□□□□□□□ p□□□□□□□□□□□□ v □□□□□□□□□□□□□□□ v □□□ d □□□□ p□□□□ p □□ v □□□□□□□ d □□□□□□□□□□□□□□□□□ v □□□□□□□□□□□□□□□□□□□□ d' □□ v □□□□□□□□□□□ d □□ d' "□□"□kill□ □□

□□□single reaching-definition property□□□□□□□□□□□□□ v □□□ d □□□□ p□□□□ p □□ v □□□□□□□ d □□□□□

□□ ϕ □□□ ϕ □□□□ ϕ □□□ ϕ □□□□□□□□□□□□□□□□□□□□□□□□□□□□ ϕ □□□□□□

□□join node□□□□□□□□□□□□ S□□□□□□□□□ S □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ S □□□□□□□ $J(S)$□

□□□□□ n_1 □ n_2 □□□ v □□□□□□□□□□□□□□□□ $J(n_1, n_2)$ □□n_1, n_2 □□□□□□□□□□□□□□□□□□□ v □□□□ ϕ □□□□□□□□□□□□□□ v □□□□□□□□□ $\mathrm{Defs}(v)$□□□□□□ $J(\mathrm{Defs}(v))$ □□□□□□□□□□□□ v □□ ϕ □□□□□ ϕ □□□□□ v □□□□□□□□□ ϕ □□□□□ $J(\mathrm{Defs}(v) \cup J(\mathrm{Defs}(v)))$ □□□□□□□□□□□□□□ $J(S \cup J(S)) = J(S)$□□□□□□□□□□□□ $J(\mathrm{Defs}(v))$ □□ v □□ ϕ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ϕ □□□

□□ $J(\mathrm{Defs}(v))$ □□□□□□□□□□□□□□□□□□□□□□□□□□ $DF^+(S) = J(S \cup \{\mathrm{entry}\})$□□□ entry □□□□□□□□□□□□□□□□□□ entry □□□□□□□□□ $\mathrm{Defs}(v) \cup \mathrm{entry} = \mathrm{Defs}(v)$□□□□□□□ entry □□□□□□□□□□ ϕ □□□□□□□□□□□□□□□□□□□□□ entry □□ ϕ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ $\mathrm{Defs}(v)$ □□□□□□□□□□□□□□□□□□□□□□□□□ v □□□ ϕ □□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ ϕ □□□□□□□□□□□□□□□□□□□ ϕ □□□□□□□□

Algorithm 3.1: Standard algorithm for inserting ϕ-functions

```
for v: variable names in original program do
    F ← {}                                    ▷ set of basic blocks where φ is added
    W ← {}                     ▷ set of basic blocks that contain definitions of v
    for d ∈ Defs(v) do
        let B be the basic block containing d
        W ← W ∪ {B}
    while W ≠ {} do
        remove a basic block X from W
        for Y: basic block ∈ DF(X) do
            if Y ∉ F then
                add v ← φ(...) at entry of Y
                F ← F ∪ {Y}
                if Y ∉ Defs(v) then
                    W ← W ∪ {Y}
```

[illegible] ϕ [illegible]"[illegible]"[illegible] ϕ [illegible]kill [illegible]

[illegible]

[illegible] SSA [illegible] ϕ [illegible]（live-range[illegible] SSA [illegible] DFS[illegible]DFS [illegible] v [illegible] v.reachingDef[illegible]

Algorithm 3.3: Renaming algorithm for second phase of SSA construction

```
    ▷ rename variable definitions and uses to have one definition per variable name
foreach v : Variable do
    v.reachingDef ← ⊥
foreach BB: basic Block in depth-first search preorder traversal of the dom. tree do
    foreach i : instruction in linear code sequence of BB do
        foreach v : variable used by non-φ-function i do
            updateReachingDef(v, i)
            replace this use of v by v.reachingDef in i
        foreach v : variable defined by i (may be a φ-function) do
            updateReachingDef(v, i)
            create fresh variable v′
            replace this definition of v by v′ in i
            v′.reachingDef ← v.reachingDef
            v.reachingDef ← v′
    foreach φ: φ-function in a successor of BB do
        foreach v : variable used by φ do
            updateReachingDef(v, φ)
            replace this use of v by v.reachingDef in φ
```

Procedure updateReachingDef(v,i) Utility function for SSA renaming

Data: v : variable from program
Data: i : instruction from program
▷ *search through chain of definitions for v until we find the closest definition that dominates i, then update v.reachingDef in-place with this definition*

```
r ← v.reachingDef
while not (r == ⊥ or definition(r) dominates i) do
    r ← r.reachingDef
v.reachingDef ← r
```

SSA [illegible]——LLVM IR [illegible] mem2reg

[illegible] mem2reg [illegible] `int` [illegible]

[illegible] mem2reg [illegible] miniSysY [illegible] LLVM IR[illegible] CFG [illegible]

[illegible] `phi` [illegible] `int` [illegible] `alloca` [illegible] `int` [illegible] `store` [illegible] ϕ [illegible] `phi` [illegible] `phi` [illegible]

[illegible] `int` [illegible] `alloca` [illegible] `int` [illegible] `load` [illegible] `load` [illegible] `int` [illegible] `store` [illegible] `store` [illegible] `phi` [illegible]

> [illegible] SSA “[illegible]” [illegible] reachingDef [illegible] reachingDef[illegible] LLVM IR [illegible]“[illegible]”[illegible] LLVM IR [illegible] mem2reg [illegible]

[illegible]

1. [illegible] SSA [illegible] CFG_[illegible]
2. SSA - LLVM-Clang-Study-Notes
3. [illegible] LLVM [illegible] SSA [illegible] IR [illegible] def-use [illegible] - RednaxelaFX [illegible] - [illegible]
4. Static Single Assignment Book: by lots of authors
5. Engineering a Compiler: by Keith D. Cooper, Linda Torczon

□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

□□□□□ lab7□lab8□□□□□□□□

□□□□

`FuncFParam` □□□

- □ `FuncFParam` □□□□□□□□□□□□□□□ `[]` □□□□□□□□□
  □□□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□ `int a[2][3][4][5]` □□□ `a[1][2]` □□□□□□□□ 4□□□□□ 5 □□□□□□□□□□ `int[][5]` □□□□

□□□□

□□□□□□□□ `mdarray` □□□ `judge.toml` □□□□ `[jobs.mdarray]` □

□□□ □□ □□□ 7 □□□□□□□□□□

- □□□□□□□2022 □ 1 □ 7 □ 23:59
- □□□□□□□ `□□_□□_multiDimension.pdf`
- □□□□□□□□ `□□□□/` □□□□□
- □□□□□□□□2022 □ 1 □ 9 □ 23:59

□□□□□□□□□□

□□
□□□□□□□□□□□□□□□□□□□□□□□□□/□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

- □□ `main` □□
- □□□□□□□□□□
- □□□□□□□□□□□□□□

□□/□□□□□□

□□ C □□□□□□□□□□□□□□□ `.ll` □□□□□

□ 1

```
int a() {
    return 1;
}

int b() {
    return 2 + a();
}

int c() {
    return 3 + b();
}

int main() {
    return 4 + c();
}
```

□□□□□□□□□□□

```
int main(){
    return 4 + 3 + 2 + 1;
}
```

□ 2

```
int foo(int t) {
    if (t > 0)
        return foo(t - 1) + t;
    return 0;
}

int bar() {
    return foo(10) + 123;
}

int main() {
   putint(bar());
   return 0;
}
```

□□□□□□□□□□□□□

```
int main() {
    putint(foo(10) + 123);
    return 0;
}
```

□ 3

```
int a = 10;

void foo() {
    a = a + 1;
}

int bar(int c) {
    if (c == 1)
        return 10;
    else
        return 5;
}

int main(){
    int b = 5;
    foo();
    foo();
    b = a + bar(2);
    return b;
}
```

□□□□□□□□□□□□□

```
int a = 10;
int main() {
    int b = 5;
    a = a + 1;
    a = a + 1;
    int a_tmp_var;
    if (2 == 1)
        a_tmp_var = 10;
    else
        a_tmp_var = 5;
    b = a + a_tmp_var;
    return b;
}
```

□ 4

```
int foo(int t) {
    if (t == 1) {
        return 5;
    } else {
        return 10;
    }
}

int main() {
    return foo(1) + foo(0);
}
```

□□□□□□□□□□□□□□

```
int main() {
    int tmp_val_in_llvm_ir_a;
    int tmp_val_in_llvm_ir_b;
    if (1 == 1) {
        tmp_val_in_llvm_ir_a = 5;
    } else {
        tmp_val_in_llvm_ir_a = 10;
    }
    if (0 == 1) {
        tmp_val_in_llvm_ir_b = 5;
    } else {
        tmp_val_in_llvm_ir_b = 10;
    }
    return tmp_val_in_llvm_ir_a + tmp_val_in_llvm_ir_b;
}
```

□□□□□□□□□ `.ll` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `.ll` □□□□□□□ `lli` □□□□□□□□□□□□□□□□□□□□

[illegible] [illegible][illegible]

[illegible]

[illegible] `.ll` [illegible]

[illegible] `.ll` [illegible]

[illegible] `foo,main` [illegible] `define @foo` , `define @main` [illegible] `foo` [illegible] `main` [illegible] `foo` [illegible]

[illegible] `inline` [illegible] `judge.toml` [illegible] `[jobs.inline]` [illegible]

[illegible] [illegible] [illegible] 7 [illegible]

- [illegible]2022 [illegible] 1 [illegible] 7 [illegible] 23:59
- [illegible] `[illegible]_[illegible]_funcInline.pdf`
- [illegible][illegible] `[illegible]/` [illegible]
- [illegible]2022 [illegible] 1 [illegible] 9 [illegible] 23:59

[illegible]

[illegible]

1. [illegible]

   [illegible] miniSysY [illegible] `main` [illegible] miniSysY [illegible]“[illegible]”[illegible]

2. [illegible]“[illegible]”[illegible]

   [illegible] `call` [illegible] `return` [illegible] `call` [illegible]

□□□□□□□□□□

□□

□□

□□□

```
int global_variable = 0;  // □□□□

int add() {
    global_variable = global_variable + 1;
    return 1;
}

int main() {
    if (1 == 1 || 1 == add()) {} // □□□□□ add() □□□□□
    return global_variable;
}
```

□□□□□□□□□□□□□□□□□□□□ `1 == 1` □□□□ `1 == add()` , □□□□□□□□□□□□□□□ `add()` □□□□□□□□□□ `global_variable` □□□□□□□□□□□□□

□□□□□□□□□□□□□□□□ miniSysY □□□□□□□

```
// □□ c □□□□□□
void * p = NULL;
int ret;
/* ... */
if (p && ret = func(p) ){
}
/* ... */
```

□□□□□□□ C □□□□□□□□□□□□□

□□□□ □□□□□□□

□□□□

□□□□□□□□ `short_circuit` □□□ `judge.toml` □□□□ `[jobs.short_circuit]` □

□□
□□□□□ lab□□□□□□□□□□□□□□□□□□□□

□□
□□
□□□□□□□□ □□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□ 7 □□□□□□□□□□□

- □□□□□□□2022 □ 1 □ 7 □ 23:59
- □□□□□□□ `□□_□□_shortCircuit.pdf`
- □□□□□□□□ `□□□□/` □□□□□□
- □□□□□□□□2022 □ 1 □ 9 □ 23:59

□□□□□□□

> □□□□（Short-circuit evaluation）□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ AND □□□□□□ □□□ `false` □□□□□□ `false` □□ OR □□□□□□□ `true` □□□□ □□□□ `true` □□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

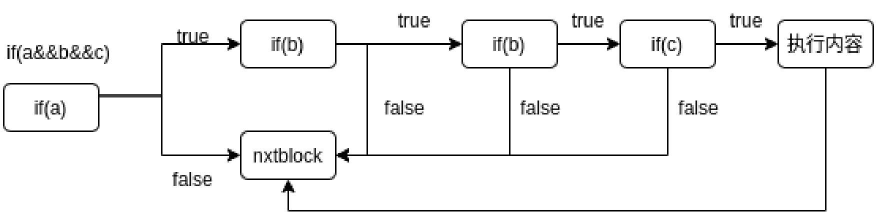
□□□

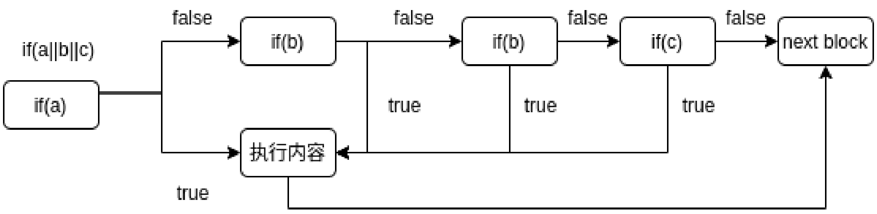
□□□□“□□□”□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□

```
if (a && b && c) {
    do_something();
}
do_others();
```

□□□□□□□□□□□□ `true` □ `false` □ `br` □□□□





□□ `if(a && b && c)` □□□□□□□□□□□□□□□□□□□□□□□

```
;
; □□□□□%a, %b, %c □□□□□
;
block_entry:
    br label %block_a

block_a:
    %res_a = icmp ne i32 %a, 0
    br i1 %res_a label block_b, label %block_out

block_b:
    %res_b = icmp ne i32 %b, 0
    br i1 %res_b label block_c, label %block_out

block_c:
    %res_c = icmp ne i32 %c, 0
    br i1 %res_a label block_exec, label %block_out

block_exec:
    do_sth()
    ; a, b, c □□□□□□□□□□□

block_out:
    ; a, b, c □□□□□□□□□□
    do_others()
```

□□ `a, b, c` □□□□□□□ □□□□□□□□□ `block_out` □□□□ `a / b` □□□□□□□□□ `block_b/c` □□□ `b/c`,□□□□□ `block_c` □□□□ `c` □□□□□□□□□□□□ `block_exec` □□□□ □□□□

□□□□

□□□□□

1. R □□□□□□□
2. □□□□

□□□□

□□□□ https://decaf-lang.github.io/minidecaf-tutorial/

SYSY □□□□□□
https://gitlab.eduxiji.net/nscscc/compiler2021/-/blob/master/SysY%E8%AF%AD%E8%A8%80%E5%AE%9A%E4%B9%89.pdf

LLVM IR lang ref https://llvm.org/docs/LangRef.html

LLVM Programmer Manual
https://llvm.org/docs/ProgrammersManual.html#the-core-llvm-class-hierarchy-reference

Often misunderstood GEP Instruction
https://llvm.org/docs/GetElementPtr.html

Mem2reg □□□□□□ https://llvm-clang-study-notes.readthedocs.io/en/latest/ssa/Mem2Reg.html

LLVM IR □□
https://www.cnblogs.com/Five100Miles/category/1438128.html

LLVM □□□ https://llvm.liuxfe.com/docs/man/lli

LLVM IR □□□□ https://github.com/Evian-Zhang/llvm-ir-tutoria

□□□□□A Tour to LLVM IR
□ https://zhuanlan.zhihu.com/p/66793637
□ https://zhuanlan.zhihu.com/p/66909226

LLVM SSA □□
https://blog.csdn.net/qq_29674357/article/details/78731713

□□□□□□□□ https://pandolia.net/tinyc/

Implementing a JIT Compiled Language with Haskell and LLVM
https://www.bookstack.cn/read/stephendiehl-llvm/spilt.1.llvm.md

[illegible]

[illegible]

rynco [illegible]

holmium [illegible]

roife [illegible]

dglr [illegible]

□□□□□

rust + □□□□□□□□ □□

https://github.com/roife/racoon

JAVA + ANTLR □□

https://github.com/BUAA-SE-Compiling/miniSysY_example_compiler