1. Define e and p structures
2. (optional) write interpreter for p
3. write test infrastructure w/ dummy compile func
4. write lots of tests

compiler : p → x86-64

R_0 p = (program e) e = num | (- e) | (+ e e) | (read)
var | (let var e e)

compiler : R_0 → X_0

X_0 prog := ".globl main\n"
"main:" INSTR+

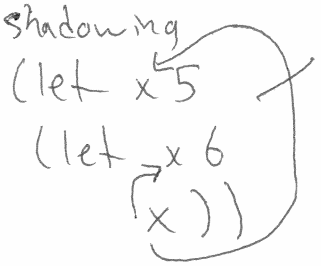
arg = constant | register | reg+offset
(con) (reg) (mem)

INSTR := "addq" arg "," arg // dest ← ~~dest~~ src + dest $\%2^{64}$
(src) (dest)
subq arg, arg
negq arg // dest ← negate dest
movq arg, arg // dest ← src
callq label (label = any string)
pushq arg // push arg on to stack
popq arg
retq

compiler (+ 10 32)

```
movq $10, %rax
addq $32, %rax
movq %rax, %rdi
callq _print_int
retq
```

R_0	X_0
operations take expressions (+ (+ 1 1) (+ 2 3))	inst take 1 or 2 arg and arg are dest or srcs (ie atomic) inst take dest and maybe src
expr evaluate to a num Ans = num	x expr evaluates to an effect (a change in memory) Ans = void
infinite variables (map to mem?)	finite registers + finite memory
tree-shaped	sequential
shadowing (let x 5 (let x 6 x))	register + memory are globally named



uniquify

$R_0 \Rightarrow I_0 \Rightarrow I_1 \Rightarrow I_2 \Rightarrow I_3 \Rightarrow$
$\Rightarrow I_4 \Rightarrow X_0 \xrightarrow[\text{print}]{} \text{asmstring} \xrightarrow[\text{gas/gcc}]{} \text{binary}$

$X_0 \Rightarrow$ asm-string — print
job: turn a tree into a string easy

$R_0 \Rightarrow I_0$ — uniquify $I_0 = R_0$ but no dupe vars
job: remove shadowing

(let ([x 10]) (+ x (let ([x 20]) x)))
↓ ↓ ↓ ↓
x_1 x_1 x_2 x_2

uniquify : ((dupe var) ⇒ (no-dupe var)) e

2-3/

(+ (let ([x 7]) 5) x)

↓ x0

↓ consult mapping but it's empty so we error

$I_0 \Rightarrow I_1$ (R_0 w/o dupe vars → C_0)

job: remove nested expressions — flatten

C_0 = (program (var *) stmt+)

stmt = (:= var exp) | (ret arg)

exp = arg | (read) | (- arg) | (+ arg arg)

arg = int | var

(+ (+ 1 2) (+ 3 4)) ⇒

x = 1+2

y = 3+4

z = x+y

ret z

add_lhs91 = 1+2

add_rhs92 = 3+4

ans_to_prog93 = ...

ret ?...

continue next time ...