15-1

Ty += (Ty ... -> Ty)

Expr += (Expr Expr ...)

Prog = Definition ... Expr

Definition = (define (id [id : Ty] ...) : Ty
expr)

[let (x 17)
y)

```
(define (even? [i : Int]) : Bool
  (if (= i 0) true (odd? (- i 1))))
(define (odd? [i : Int]) : Bool
  (if (= i 0) false (even? (- i 1))))
(even? 1000)
```

$\Gamma \vdash$ (fun arg_0 ... arg_n) : res Γ(odd?) =

if $\Gamma \vdash fun : ty_0 \ldots ty_n \to res$ Int → Bool

and $\Gamma \vdash arg_i : ty_i$

$\vdash$ (program defs ... expr) iff

$\Gamma_D = \{(id \mapsto dom \to rng) \ldots\}$ where

defs = (id (... :dom] : rng)

$\Gamma_D \vdash expr : Ty$

$\Gamma_D \vdash def_i : \Gamma_D(id)$

$\Gamma \vdash$ (define (f $[a_0 : ty_0] \ldots [a_n : ty_n]$) : res
body) iff

$\Gamma[a_0 \mapsto ty_0] \ldots [a_n \mapsto ty_n] \vdash body : res$

5-2 | parse → typec → ~~Expose~~ uniquify → expose → flatten → reg → patch → print

⇓ newforms ⇓ new stuff ⇓ boring (but needed)

callq read-int callq even?

callq *%rax
read %rax as codeptr, and jump to it (indirect jump)

callq f

```
                                    subl
(define (add1 [x : Int]) : Int (+ x 1))
(define (fit [x : Int] [f : (Int -> Int)])
  : Int (f x))
(fit (fit (fit 12 add1) sub1)
     (if (= (fib 5) 125)
         add1
         sub1))
```

_fun1: .... code ...
_fun2: .... code ...
_main: .... code ...

movq _fun1, %rax (7mil) / _fun1(%rip) (3000)
callq *%rax
LEAQ → load effective addr → 64-bit

ARGS: rdi, rsi, rdx, rcx, r8, r9

Caller	Callee	Contents		E
8(%rbp)		return addr		
0(%rbp)		old rbp	← callee-saves (for E)	≠
-8(%rbp)		local 1	Caller	
-8k(%rbp)		local k	← caller-saves (for F)	
8n-8(%rsp)	8n+8(%rbp)	arg n+6	caller code	
0(%rsp)	16(%rbp)	arg 1+6		
	8(%rbp)	return addr	Callee	
	0(%rbp)	old rbp	← callee-saves (for F)	↺
	-8(%rbp)	local 1		
		local k		

15-3 | expose : turn (vector e...) into (alloc) (set! ...)

[1] (let ([f 12]) f) ⟹ $f

[2] (define (f ...) ...) f ⟹ leaq f(%rip), %rax

[(:ref x)
 (if x is a fun ⟹ (:function-ref x)
 (:ref x))]

1. tag name with !
   ⟹ horrendous
2. Look at Γ and if fun
   ⟹ borken
3. track funs
   compute a set of names

expose : e * fs ⇒ e

expose' : {id} e*fs ⇒ e

global funs

(prog def... expr)
⇒ (prog' def... (define (main) : Γ(expr) expr) ~~(main)~~ 'main)

(prog e) ⇒ (flat-prog vs flat-e)

(prog d... m) ⇒ (flat-prog fd... m)

expr — vs, flat-e

(movq argn-storage, argn-var)

(define (add1 x) ...)
 (movq %rdi, $x)
 ... assume $x is in ~~$r11~~ or -8(rbp)

(set! lhs (app fun args...))
⇒ fun-stmt...
 args-stmts...
 save caller-saves
 move args into place
 (indirect call fun-arg)
 (movq rax lhs)

lhs would be arg

restore caller + restor arg stacks