Functions

- Top-level functions (program D ... E)
- Different kinds of variable reference
  - ① x was 5 => "argument" in assembler
  - ② x was function => leaq
  - ③ (except if already did it)

```
(let ([x 5])
  (let ([f add1])
    (f x)))
```

add1 → #2 (proc-ref 'add1)

f → #3 (ref 'f)

x → #1 (ref 'x)

- callq *%rax (my advice: in X* have a late call inst
  (callq label (arg ...) ret)

Closures

PL: P = (program E)

E = ... | (λ (X ...) E)

((λ (x) (+ x x)) 5) => 10

```
(let ([y 10])
  (let ([f (λ (x) (+ x y))])
    (f 5)))
```

=> 15

```
(let y d f f)
  5)
```

=> 15

```
(let ([v (vector 0 1)])
  (let ([f (λ (x) (vector-set! v 0 x))])
    (begin (f 5) (vector-ref v 0))))
```

=> 5

Type-Checking

1) NOT RECURSIVE

(if you want, add that)

(letrec ([f (λ (x) [...])])

OR (rec-λ f (x) [...])

use 'f'

2) E = ... | (λ ([x : T] ...) : T E)

input type

output type

Closures for Compilers:

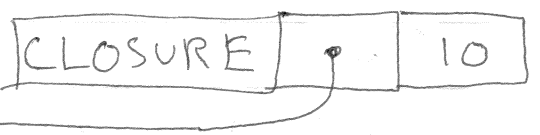
Closure = code + data (object)

body of fun

(+ x y)

free-variables of body

TODO write a free-variable : E => set(variables)

(λ (x) (+ x y)) => 

(λ (clo x)

(+ x y)))

= let ([y (vector-ref clo 1)])

(vector new-fun 10)

Closure language R^C

↓ new pass (de-closure pass)

Function language R^F

16-3

Closure Conversion (Defunctionalization) lambda lifting

$$CC : E^C \longrightarrow E^F \times \boxed{(D \ldots)}$$

$$CC[(\lambda\ (X \ldots)\ E)] = \qquad Y \ldots = FV(E) - \{X \ldots\}$$

let $(E', D \ldots) = CC[E]$ in

(vector code-ptr $Y \ldots$) $\quad |Y \ldots| = n$

$\times$ $(1+n)$-fields

$$[\,(\text{define}\ (\text{code-ptr clo}\ X \ldots)\ (\text{let}\ ([\,Y_i\ (\text{vector-ref clo}\ i)]) \ldots\ E')\ \ldots))\,] \ +\!\!+\ [D \ldots]$$

$$CC[(+\ E_1\ E_2)] = (+\ E_1'\ E_2') \times [D_1 \ldots, D_2 \ldots]$$

where $E_1', D_1 \ldots = CC[E_1]$

$E_2', D_2 \ldots = CC[E_2]$

$$CC[(f\ a_0 \ldots a_n)] =$$

$f', D_f \ldots = CC[f]$

$a_0', D_{a_0} \ldots = CC[a_0]$

...

~~X~~ $(f'\ a_0' \ldots)$

~~X~~ $((\text{vector-ref}\ f'\ 0)\ f'\ a_0' \ldots)$

○ $(\text{let}\ ([\text{fclo}\ f'])\ ((\text{vector-ref}\ \text{fclo}\ 0)\ \text{fclo}\ a_0' \ldots))$

$$CC[(\text{fun-ref TLF})] = (\text{vector TLF})$$

(and change TLD to add unused clo arg)

16-4/ C: (λ (x) (+ x x))

→ F: (define (F0 clo x) (+ x x))

~~(vector F0 clo)~~ (vector F0)

C: (λ ([x : Int]) : Int (+ x x)) : Int→Int

→ F: (define (F0 [clo : ①]
[x : Int])
: Int
(+ x x))
(vector F0) : (Vector (→ ① INT INT))

① = (vector (→ ① INT INT))

Infinite type !!!

OPTion 1

T = ... | X | μX. T

① = μX. (Vector (→ X Int Int))

OPT 2 ① = Trust Me