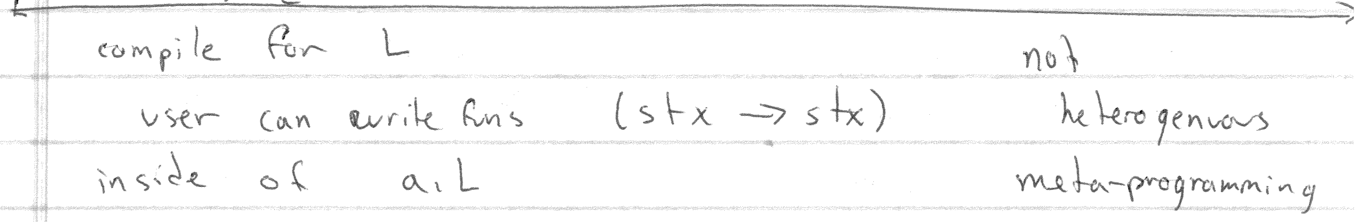
```
#define PI 3.14          →  CPP, C pre-processor
int f (int x) {
  return x + PI;
}
```

↓ gcc (cpp ⇒ cc0)

```
(int f(int x) {
   return x + 3.14;
 })
```

asm

```
# define MAC   1; }
int f () { return MAC
```



compile for L — not heterogeneous meta-programming

user can write fns (stx → stx)

inside of a L

Lang provides + and numbers

++ or if funs

```
(define (++ x) (+ x 1))
```

Lang provides lambda

↓ macro provides let

```
(let ([x e] ...) b ...)
  => ((lambda (x ...) b ...) e ...)
(let ([a 1] [b 2] [c 3]) (+ a b c))
  => ((λ (a b c) (+ a b c)) 1 2 3)
```

Macros provide DSLs (domain-specific languages)

- ↳ class systems
- ↳ alternative evals like Prolog, Laziness
- ↳ analysis like types
- ↳ automata

Macros-by-Example
1986

```
(define-syntax-rule (let ([x e] ...) b ...)
  ((λ (x ...) b ...) e ...))

let : stx -> stx
(defmacro (let s)
  (cons (cons 'lambda (cons (map car (cadr s))
                            (cddr s)))
        (map cadr (cadr s))))
```

```
(define (expand macs stx)
  (if (atom? stx) stx
      (or (for/or ([m (in-list macs)])
            (expand1 m stx))   ; if this isn't #f, then recur
          (map (λ (s) (expand macs s)) stx))))

(define (expand1 m stx)              (expand1 let-def '(+ 1 2))
  in, out := m
  (and (B in stx)
       (T out (D in stx))))
```

B : pat stx -> bool   // it matches   B(let, abc) = #t

D : pat stx -> env   // what pattern variables   D(let, abc) =

Transcribe : pat env -> stx   // expands the macro

$$p = \begin{bmatrix} x \mapsto \text{'}(a\ b\ c) \\ e \mapsto \text{'}(1\ 2\ 3) \\ b \mapsto \text{'}((+\ a b c)) \end{bmatrix}$$

$p = x$ $[x \mapsto a]\ [x \mapsto b]\ [x \mapsto c]$

$pat = () \mid a \mid (p_1 \,.\, p_2) \mid (p \;\ldots)$

$stx = () \mid a \mid (stx \,.\, stx)$

$B : pat \; stx \to bool$

$B[\![()]\!]\, s = (empty?\; s)$

$B[\![a]\!]\, s = true$

$B[x]\; y$

$B[x]\; (+\;1\;2)$

$B[\![(p_1 \,.\, p_2)]\!]\, s = let\,(s_1 \,.\, s_2) = s \;\; in$
$B[p_1]\, s_1 \;\&\&\; B[p_2]\, s_2$

$B[\![(p \;\ldots)]\!]\, s = (list?\; s)$
$\wedge\; (andmap\;\; B[p]\;\; s)$

$D : pat\; stx \to env \qquad env : id/a \to (nat \times stx)$

$D[\![()]\!]\, s = \emptyset$

$D[\![a]\!]\, s = \{(a \mapsto (0, s))\}$

$D[\![(p_1 \,.\, p_2)]\!]\, s = D[p_1]\,(car\; s) \;\cup\; D[p_2]\,(cdr\; s)$

(assumes no shared vars between p_1 and p_2)

↳ (d-s-r (check x x) #t) — linear pattern variable

(check 1 1) ⇒ #t

(check 1 3) ⇒ invalid syntax

$D[\![(p_1 \;\ldots)]\!]\,(s_1 \;\ldots\; s_n)$

$= combine\,(D[p_1]\, s_1 \;\ldots\; D[p_1]\, s_n)$

$combine = \lambda p_1 \ldots p_n \,.\, \lambda i_n \;\langle (p_1\; i)_1 + 1,$
$((p_1\; i)_2 \;\ldots\; (p_n\; i)_2)\rangle$

1-7/ T : pat env → stx

$$T[()]\rho = ()$$

$$T[(p_1 \cdot p_2)]\rho = (T[p_1]\rho \cdot T[p_2]\rho)$$

$$T[a]\rho = \text{if } a \in Dom(\rho)$$

$$\quad \text{if } (\rho\ a)_1 == 0$$

$$\quad\quad (\rho\ a)_2$$

$$\quad\quad \text{error}$$

$$\quad a$$

$$T[(p_1 \circ\circ\circ)]\rho = \text{if controllable } \rho$$

$$\quad \text{map } T[p_1] \text{ decompose } (\rho \mid fv(p_1))$$

$$\quad \text{error}$$

$$\text{controllable } \rho = \exists v\ (v \in dom(\rho|^{fv(p_1)}) \wedge (\rho\ v)_1 > 0)$$

$$\text{decompose } \rho = \text{cons } (\text{split } hd^{\nearrow car} \circ \rho)$$

$$\quad \text{decompose } (\text{split } tl^{\nearrow cdr} \circ \rho)$$

$$\text{split } f \langle n, c\rangle = \text{if } (n == 0)$$

$$\quad \langle 0, c\rangle$$

$$\quad \langle n-1, fc\rangle$$