20-1/ Macros

#define N 1024 1024";

#define ASSERT(x) if(!x){ exit(1);}

return N + password; ⇒ return 1024 + pass;

src compiler

gcc = cpp → cc0 → as → ld

m4

---

string → tokens → parse tree → ast → code

"1 + 2" NUM(1) ADD NUM(2) ADD (+ 1 2)

N(1) N(2)

flex

yacc

bison

antlr

20-2/ e = (+ e e) | (* e e) | (- e e)
| (define ...)

(define (- x y) (+ x (* -1 y)))

compiler ⇒ ∈ compile.c

stdlib ⇒ ∈ std.jaylang ← a user could have made it

while
for
do while

for (init; cond; step) { body }
⇒
init; → "int x=0;"
while (cond) { body; step; }

forasfun (int x=0, ...)

```
(define-syntax-rule
  (for init cond step body ...)
  (begin init (while cond (begin body ... step))))
(define-syntax-rule
  (while cond body)
  (letrec ([loop (λ () (when cond (begin body (loop))))])
    (loop)))
```

20-3/
```
(dsr (when cond body)
     (if cond body (void)))
(dsr (unless cond body) (when (not cond) body))
(dsrs begin
  [(begin) (void)]
  [(begin e) e]
  [(begin e more ...)
   (let ([_ e]) (begin more ...))])
(dsr (letrec ([x e] ...) body ...)
  (let ([x #f] ...)
    (begin (set! x e) ...
           body ...)))
(dsr (let ([x e] ...) body ...)
  ((λ (x ...) body ...)
   e ...))
(dsrs let*
  [(let* () body ...) (begin body ...)]
  [(let* ([x e] more ...) body ...)
   (let ([x e]) (let* (more ...) body ...))])
```

2-4/ (dsr (class (carg ...)
[(method marg ...)
mbody ...]
...)
(λ (carg ...)
(λ (method name)
(case methodname
[method (λ (marg ...) mbody ...)]
...))))

(define posn% (class (x y) ....))
(define p1 (posn% 3 4))
((p1 'set-x!) 5)
((p1 'get-x)) => 5
(dsr (send obj method args ...)
((obj 'method) args ...))

20-5/ macros-by-example (80s)

- Macro = set (pat x templates)

Syntax-case (90s) & ffexpr (60s)

- Macro = stx ⇒ stx

→ in the program being compiled

macros that work together / syntax-local-value (00s)

Macro = stx x env ⇒ stx

local-expand (00s)

in meta-lang, the macro sys is callable

set-of-scopes (10s)

- Macro = stx' x env ⇒ stx'