2-1 > (read)   (file->value path)

(+ 10 32)

=> '(+ 10 32)

compile : $R_0 \longrightarrow$ asmblely

R_0	asm
operations take expressions (+ 1 (+ 2 3)) └ vs produce new value	inst take 1 or 2 arguments └ must be num, mem, reg addq 1 (addq 2 3) └ one arg must be mem/reg └ that same arg is the dest
infinite variables	few registers └ infinite Memory
shadowing [(let ([x ...]) [(let ([x ...]) ...)	registers are unique

$R_0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5 \longrightarrow \underset{6}{x86} \xrightarrow[\text{gas}]{} \text{binary}$

6 | print | x86-tree | x86-string

job: display asm source   (program int inst+)

frame-size

1 | unique | R_0 | R_0

job: shadowing vs unique

(let ([x 10]) (+ x (let ([x 20]) x)))

=>

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

(gensym 'x) = 'x9021 'x1729

2 | flatten | R_0 | C_0

job: remove nested expressions

```
(+ 1 (+ 2 3))  ==>   "like"   x = 2+3
                              ret 1+x

(+ (+ 1 2)
   (+ 3 4))    =>   x = 1+2
                    y = 3+4
                    ret x+y
```

```
(+ 1 (let ([x 5]) (+ x 3)))
  lhs1  rhs1      body  lhs2 rhs2
```

=> ① lhs1 = 1   ② x = 5

③ lhs2 = x   ④ rhs2 = 3   ⑤ body = lhs2 + rhs2

~~rhs1 = x+3~~   ⑥ rhs1 = body

⑦ ret lhs1 + rhs1

```
=> body = 5 + 3        =>      x = 5
   ret 1 + body                body = x+3
                               ret 1+body
```

```
C0   = (program (var*) stmt+)
stmt = (:= var exp)  | (ret arg)
exp  = arg | (read) | (- arg) | (+ arg arg)
arg  = int | var
```

3 | select-instruction | C_0 | X^* (x86-tree)

job: turn exp of C_0 into X inst assuming everything is a register

```
                      addq $10, !x      addq $10, %rax
[ body = 5+3  ]         [ movq  $3, !body
[ ret 1 + body]   =>    [ addq  $5, !body
                        [ movq  !body, %rax
                        [ addq  $1, %rax
                          retq
```

X^* = X, but !var is a valid arg and

program = (vs*) inst+

2-3 4 | assign-homes | X^* | X°

job : map variablesto memory (on the stack)

X° = X (but may beak some x86-rules)

```
                                   frame setup +
[ movq  $3, !body ]         [ movq  $3,  -8(%rbp) ]
[ addq  $5, !body ]   =>    [ addq  $5,  -8(%rbp) ]
                                 + frame cleanup

[ movq  $3, !body ]         [ movq,  $3,  -8(%rbp) ]
[ movq  $8, !lhs  ]   =>    [ movq   $8, -16(%rbp) ]
```

(calculate the frame size)

5 | patch | X° | X

```
    x*
[ movq !lhs, !rhs ]   =>   [ movq  -8(%rbp), -16(%rbp) ]
                          ⇙
              [ movq  -8,  %rax ]
              [ movq  %rax,  -16 ]
```