```
#lang racket
(define (mini-eval expr env)
  (display "  going to evaluate expression: ")
  (displayln expr)
  (cond
    ((number? expr) expr)
    ((symbol? expr) (lookup-variable-value expr env))

    ((add? expr) (eval-add expr env))
    ((subtract? expr) (eval-subtract expr env))
    ((multiply? expr) (eval-multiply expr env))
    ((exponent? expr) (eval-exponent expr env))

    ((definition? expr) (eval-definition expr env))
    ((ifzero? expr) (eval-ifzero expr env))
    ;((lambda? expr) (eval-lambda expr env))
    ;((call? expr) (eval-call expr env))

    (#t (error "kablooie: " expr))
    ))

(define (add? expr) (equal? 'add (car expr)))
(define (subtract? expr) (equal? 'sub (car expr)))
(define (multiply? expr) (equal? 'mul (car expr)))
(define (exponent? expr) (equal? 'exp (car expr)))
(define (definition? expr) (equal? 'define (car expr)))
(define (ifzero? expr) (equal? 'ifzero (car expr)))

; expr = (add expr1 expr2)
(define (eval-add expr env)
  (+ (mini-eval (cadr expr) env) (mini-eval (caddr expr) env)))

; expr = (sub expr1 expr2)
(define (eval-subtract expr env)
  (- (mini-eval (cadr expr) env) (mini-eval (caddr expr) env)))

; expr = (mul expr1 expr2)
(define (eval-multiply expr env)
  (* (mini-eval (cadr expr) env) (mini-eval (caddr expr) env)))

; expr = (exp expr1 expr2)
(define (eval-exponent expr env)
  (expt (mini-eval (cadr expr) env) (mini-eval (caddr expr) env)))

; expr = (define varname expr1)
(define (eval-definition expr env)
  (hash-set! (car env) (cadr expr) (mini-eval (caddr expr) env)))

; expr = (ifzero expr1 expr2 expr3)
(define (eval-ifzero expr env)
  (if (= 0 (mini-eval (cadr expr) env))
      (mini-eval (caddr expr) env)
      (mini-eval (cadddr expr) env)))

; expr = (lambda argname body)
;(define (eval-lambda expr env)

; expr = (call expr1 expr2)
;(define (eval-call expr env)

(define (lookup-variable-value var env)
  (cond ((hash-has-key? (car env) var) (hash-ref (car env) var))
        ((null? env) (error "unbound variable" var))
        (#t (lookup-variable-value var (cdr env)))))

;(define (mini-apply closure argval)

(define (run)
  (let ((global-env (list (make-hash))))
    (define (read-eval-print-loop)
      (display "mini-eval input: ")
      (let ((input (read)))
        (if (not (equal? input 'end))
            (let ((output (mini-eval input global-env)))
              (display "mini-eval output: ")
              (displayln output)
              (read-eval-print-loop))
            'done)))
    (read-eval-print-loop)))
```