Problem 1.

```
1. (+ 25 9 16) => 50
2. (/ 24 4) => 6
3. (+ (* 3 28) (- 2 2)) => 84
4. (define a 8) ; evaluation unspecified, but env has a = 8
5. (define b (+ a 7) ; evaluation unspecified, but env has b = 15
6. (+ a b (* a b)) => 127
7. (= a b) => #f
8. (if (and (> b a) (< b (* a b))) b a) => 15
9. (cond ((= a 9) 6) ((= b 3) (+ 6 7 a)) (else 25)) => 25
10.(+ 10 (if (> b a) b a)) => 25
11.(* (cond ((> a b) a) ((< a b) b) (else -1)) (+ a 15)) => 345
```

Problem 2A.

```
; checks an empty list
(define (empty? lst) (= (length lst) 0))

; gets the list element at index i
(define (idx_getter lst i)
  (cond ((empty? lst) lst)
        ((> i 0) (idx_getter (cdr lst) (- i 1)))
        ((= i 0) (car lst))))

; checks if 0 <= i < length and 0 < j <= length
(define (index_in_range lst i j)
  (define len (length lst))
  (if (and (>= i 0) (< i len) (> j 0) (<= j len)) #t #f))

; creates a slice from the beginning of a list
(define (slice: lst i)
  (cond ((empty? lst) lst)
        ((not (index_in_range lst i (length lst))) '())
        ((> i 0) (slice: (cdr lst) (- i 1)))
        ((= i 0) lst)))

; creates a slice from the end of a list
(define (:slice lst j)
  (cond ((empty? lst) lst)
        ((not (index_in_range lst 0 j)) '())
        ((> j 0) (cons (car lst) (:slice (cdr lst) (- j 1))))
        (else '())))

; creates a slice within a list
(define (slice lst i j)
  (cond ((empty? lst) lst)
        ((not (index_in_range lst i j)) '())
        ((= i j) (cons (idx_getter lst i) '()))
        (else (slice: (:slice lst j) i))))

; tests
(define lst '(1 2 3 4 5 6 7))
(display (slice: lst 2)) ; => (3 4 5 6 7)
(display (:slice lst 6)) ; => (1 2 3 4 5 6)
(display (slice lst 2 6)) ; => (3 4 5 6)
```

Problem 2B.

```
(define (square x) (* x x))
(define (seq n)
    (cond ((< n 0) 0)
          ((= n 0) 1)
          (else (+ (square (seq (- n 1))) 4))))
```

Problem 2C.

```
(define (prime? x)
  (define root (sqrt x))
  (define (try i)
    (cond ((> i root) #t)
          ((= (modulo x i) 0) #f)
          (else (try (+ i 1)))))
  (cond ((= x 2) #t)
        ((< x 2) #f)
        (else (try 2))))
```