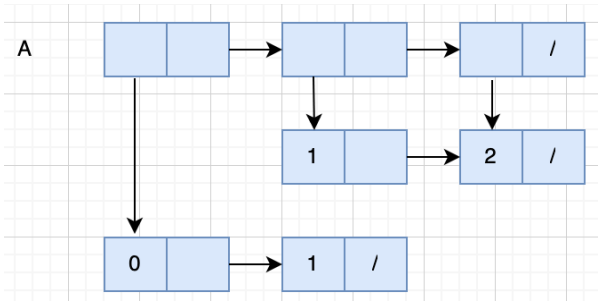
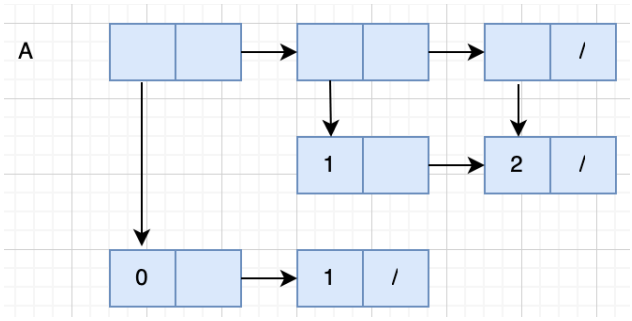
Final Exam

1. Adding a new function to *racket-1*
2. HOP always receives a procedure as an input and returns a procedure as output
    a. True
    b. False
3. Using environment model, applying *let* creates a new frame like a procedure
    a. True
    b. False
4. Loops and recursion achieve the same thing in racket
    a. True
    b. False
5. (set! x (+ 1 5)) cannot be evaluated with the substitution model
    a. True
    b. False
6. A variable defined in a procedure can only be accessed by that procedure
    a. True
    b. False
7. Racket can be evaluated using 3 models: substitution, lambda, environment
    a. True
    b. False
8. Racket uses lexical and dynamic scoping
    a. True
    b. False
9. (define (area W D) (* W D)) binds *area* to a procedure
    a. True
    b. False
10. Does (cdr A) result in: '((1 2) 2)



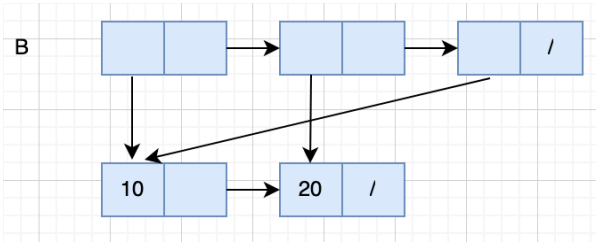
    a. True
    b. False

11. Does (cons 2 A) result in: ‘(2 (0 1) (0 1) 2 2)



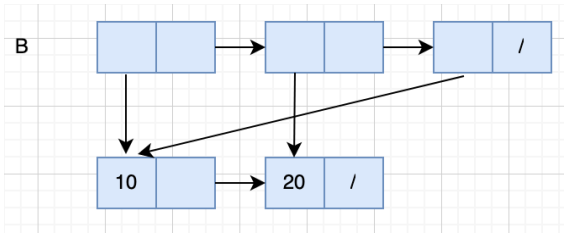
    a. True
    b. False

12. Does (car (cdr B)) result in: ‘(20)



    a. True
    b. False

13. Does the following box-arrow diagram result in: ‘((10 20) 20 (10 20))



    a. True
    b. False

14. Environment diagram:
    a. A points to Global
    b. B points to = Global
    c. C points to = E1
    d. E1 is binded to = x:2
    e. E2 is binded to = y:6
    f. E3 is binded to = a2:b6

15. calc
    a. Number of calls to *calc-apply* for (+ 2 3 5) and (- 2) is the same
        i. True
        ii. False
    b. REPL abbreviation

i. True
ii. False

c. (*foldr 1 args*) means the function only takes 1 argument
i. True
ii. False

d. Adding a new operation requires modifying both *calc-eval* and *calc-apply*
i. True
ii. False

e. Increasing amounts of arguments, increases *calc-eval* function calls exponentially
i. True
ii. False

16. Make-odometer function
a. # of local procedures
i. 3

b. Increment procedure returns a pair of updated values
i. True
ii. False

c. Increment procedure increments by 1/10 of a km
i. True
ii. False

d. # of local state variables for each odometer object
i. 2

e. Evaluating (define *odo* (make-odometer) binds *odo* to a procedure
i. True
ii. False

17. *proc1* and *proc2*
a. Changing the order of what *proc2* returns can be done by modifying *proc1*
i. True
ii. False

b. *proc1* returns a procedure
i. True
ii. False

c. *proc1* is an interactive process
i. True
ii. False

d. *proc2* is not a recursive process
i. True
ii. False

e. *proc1* could be made local to *proc2*

i. True
ii. False

18. Are the following statements equivalent (assuming *y* is defined):

```
(let ((x y)) (+ x 10))

((lambda (x y) (+ x y)) y 10)
```

a. True
b. False

19. Are the following two procedures equivalent?

```
(define d 1000)

(define (a x)
  (lambda (d) (* x d) (+ d d)))

(define (b x)
  (let ((d (+ d d))) (* x d)))
```

a. True
b. False

20. Write the cumulative-sum function:

```
(define (cumulative-sum lst)
  (define (help lst s)
    (if (null? lst)
        '()
        (cons (+ s (car lst)) (help (cdr lst) (+ s (car lst))))))
  (help lst 0))
```