ComS 342
Recitation 2, 10:00 Tuesday
Homework 2

Sean Gordon

October 4, 2019

1.

```
(define X 88) (define Y 89) (define Z 90)
(+ X 32)
$ $ 120
(+ Y 32)
$ 121
(+ Z 32)
$ 122
```

2.

```
(define fib (
        lambda (n) (
                if (> n 1)
                (+ (fib (- n 1)) (fib (- n 2)))
                n
        )
))
```

```
(define fib (lambda (n) (if (> n 1) (+ (fib (- n 1)) ...
                                  (fib (- n 2))) n)))
```

3a.

```
(define count (
        lambda (n lst) (
                if(null? lst) 0
                (
                        if (= n (car lst))
                                (+ 1 (count n (cdr lst)))
                                (+ 0 (count n (cdr lst)))
                )
        )
))
```

```
(define count (lambda (n lst) (if(null? lst) 0(if (= n (car lst)) ...
        (+ 1 (count n (cdr lst)))(+ 0 (count n (cdr lst)))))))
```

3b.

```
(define maxhelp (
        lambda (n lst) (
                if(null? lst)
                        n
                        (if (> n (car lst))
                                (maxhelp n (cdr lst))
                                (maxhelp (car lst) (cdr lst))
                        )
        )
))


(define max (
        lambda (lst) (
                if(null? lst) 0
                        (maxhelp(car lst) lst)
        )
))


1.
(define maxhelp (lambda (n lst) (if(null? lst)n(if (> n (car lst)) ...
                (maxhelp n (cdr lst))(maxhelp (car lst) (cdr lst))))))
2.
(define max (lambda (lst) (if(null? lst) 0 (maxhelp(car lst) lst))))
```

3c.

```
(define max2 (
        lambda (a b) (
                if(> a b) a b
        )
))


(define maxrepeat (
        lambda (lst) (
                if(null? lst) 0
                        (max2
                                (count (car lst) lst)
                                (maxrepeat (cdr lst))
                        )
        )
))


1.
(define count (lambda (n lst) (if(null? lst) 0(if (= n (car lst)) ...
                (+ 1 (count n (cdr lst)))(+ 0 (count n (cdr lst)))))))
2.
(define max2 (lambda (a b) (if(> a b) a b)))
3.
(define maxrepeat (lambda (lst) (if(null? lst) 0 ...
(max2 (count (car lst) lst)(maxrepeat (cdr lst))))))
```

4a. Should "cons" be used here? It is specified this should be done "using list expression"

```
(define pairs (list (list 1 5) (list 6 4) (list 7 8) (list 15 10)))
```

4b.

```
(define secondSum (
        lambda (lst) (
                + (car (cdr (car lst))) (
                        + (car (cdr (car (cdr lst)))) (
                                + (car (cdr (car (cdr (cdr lst)))))
                                (car (cdr (car (cdr (cdr (cdr lst))))))
                        )
                )
        )
))
```

```
(define secondSum (lambda (lst) (+ (car (cdr (car lst))) ...
(+ (car (cdr (car (cdr lst)))) ...
(+ (car (cdr (car (cdr (cdr lst))))) ...
(car (cdr (car (cdr (cdr (cdr lst))))))) ))))
```

5a.

```
(define second (
        lambda (apr) (
                apr #f
        )
))
```

```
(define second (lambda (apr) (apr #f)))
```

5b. Honestly I'm just really confused. There's like 6 ways to do this and nobody has given a clear answer, only cutely dodged questions and withheld important information.

```
(define pair (
    lambda (fst snd) (
        lambda (op) (
            if (= op 1) (+ fst snd) (
                if (= op 2) (- fst snd) (
                    if (= op 3) (* fst snd) (
                        if (= op 4) (/ fst snd)
                                                0
                    )
                )
            )
        )
    )
))
```

```
(define pair (lambda (fst snd) (lambda (op) (if (= op 1) ...
(+ fst snd) (if (= op 2) (- fst snd) ...
(if (= op 3) (* fst snd) (if (= op 4) (/ fst snd)0)))))))
```

5c.

```
(define add (lambda (p) (p 1)))
    ⟹
(add apair)
5
```

6a.

```
(define mylist (list (list 1 3) (list 4 2) (list 5 6)))
```

6b.

```
(define applyonnth (
        lambda (op lst n) (

                if(null? lst) -1
                        (if (= n 1)
                                (op (car (car lst)) (car (cdr (car lst))))
                                (applyonnth op (cdr lst) (- n 1))
                        )
        )
))


(define applyonnth (lambda (op lst n) (if(null? lst) -1 ...
        (if (= n 1)(op (car (car lst)) (car (cdr (car lst)))) ...
        (applyonnth op (cdr lst) (- n 1))))))


Assuming operators are defined as such:
(define add (lambda (x y)(+ x y)))
(define subtract (lambda (x y)(- x y)))
```

6c.

```
(define applyonnth (
    lambda (op) (
        lambda(lst) (
            lambda(n) (

                if(null? lst) -1
                    (if (= n 1)
                        (op (car (car lst)) (car (cdr (car lst))))
                        (((applyonnth op) (cdr lst)) (- n 1))
                    )
                )
            )
        )
))

(define applyonnth (lambda (op) (lambda(lst) (lambda(n) ...
(if(null? lst) -1(if (= n 1)(op (car (car lst)) ...
(car (cdr (car lst))))(((applyonnth op) (cdr lst)) (- n 1))))))))
```