

EECS 111 Quiz #1 February 3, 2010

Don't panic! Read each question through. If any part confuses you, come up and ask me privately. Watch your time. Don't spend forever on any one question. Write cleanly. If you need to make big changes, X out the current code, write "see back" and write your new version on the back, with the number of the question.

1. (10 pts) The following code was supposed to return a person's BMI (body mass index) category, using the formula $BMI = \text{weight} * 703 / \text{height}^2$, where weight is in pounds and height is in inches, and the table on the right. But the code has many errors. Circle each error, and say briefly what's wrong. On the right, write the correct code.

$bmi < 18.5$	underweight
$18.5 \leq bmi < 25$	normal
$25 \leq bmi < 30$	overweight
$30 \leq bmi$	obese

<pre>(define (bmi wgt hgt) (bmi-cat 703 * wgt / hgt * hgt)) (define (bmi-cat x) (cond ((x < 30) "overweight") (x < 25) "normal" (x < 18.5) "underweight" otherwise "obese"))</pre>	<pre>(define (bmi wgt hgt) (bmi-cat (/ (* 703 wgt) (* hgt hgt)))) (define (bmi-cat x) (cond ((< x 18.5) "underweight") ((< x 25) "normal") ((< x 30) "overweight") (else "obese")))</pre>
--	---

- Comment [CKR1]:** infix instead of prefix
- Comment [CKR2]:** infix instead of prefix
- Comment [CKR3]:** missing () around each cond branch
- Comment [CKR6]:** Common mistakes: verbose repeated tests, not handling exactly 18.5 or 25
- Comment [CKR4]:** wrong test order; everyone under 30 is overweight
- Comment [CKR5]:** wrong keyword

2. (10 pts) What does fn1 in the code below return for the following input values?

(fn1 (list 1 2 3 4))	(list 4 3 2 1)
(fn1 (list 4 3 2 1))	(list 1 2 3 4)
(fn1 (list (list 1 2) (list 3 4)))	(list (list 3 4) (list 1 2))
(fn1 (list 1))	(list 1)
(fn1 empty)	empty

Comment [CKR7]: Common mistake: (list 4 3 2 1)

Describe what fn1 returns in general: **reverses top-level of list, not lists inside list**

```
(define (fn1 l)
  (fn2 l empty))

(define (fn2 l1 l2)
  (if (empty? l1) l2 (fn2 (cdr l1) (cons (car l1) l2))))
```

3. (10 pts) What does `fn1` in the code below return for the following input values?

<code>(fn1 (list 1 2 3))</code>	<code>(list -1 0 1)</code>
<code>(fn1 (list 1))</code>	<code>(list 0)</code>
<code>(fn1 (list 4 3 2 1))</code>	<code>(list 1.5 0.5 -0.5 -1.5)</code>
<code>(fn1 empty)</code>	<code>error: division by zero</code>

Comment [CKR8]: Common mistake: saying "empty"

Describe what `fn1` returns in general: **calculates the average of the numbers in a list and returns a list of $x - \text{average}$, for each element x in list; not defined for empty list**

```
(define (fn1 l)
  (fn2 l (/ (fn3 l) (length l))))

(define (fn2 l x)
  (if (empty? l) empty (cons (- (car l) x) (fn2 (cdr l) x))))

(define (fn3 l)
  (if (empty? l) 0 (+ (car l) (fn3 (cdr l)))))
```

4. (10 pts) Define a function `(next-pow m n)` for non-negative integers m and n to return the smallest k such that $m^k \geq n$. Feel free to define helper functions. Some test cases:

```
(check-expect (next-pow 2 8) 3)
(check-expect (next-pow 2 9) 4)
(check-expect (next-pow 2 16) 4)
(check-expect (next-pow 2 1) 0)
(check-expect (next-pow 3 1) 0)
(check-expect (next-pow 3 27) 3)
(check-expect (next-pow 3 80) 4)
```

```
(define (next-pow m n)
  (if (<= n 1) 0 (add1 (next-pow m (/ n m)))))
```

Comment [CKR9]: this is both simpler and more efficient

or

```
(define (next-pow m n)
  (next-pow-iter m n 0))

(define (next-pow-iter m n k)
  (if (<= n (expt m k)) k (next-pow-iter m n (add1 k))))
```

5. (10 pts) Define a function (merge list1 list2) to return the sorted merger of two sorted lists of numbers, including duplicates. Some test cases:

```
(check-expect (merge empty empty) empty)
(check-expect (merge (list 1 2 3) empty) (list 1 2 3))
(check-expect (merge empty (list 1 2 3)) (list 1 2 3))
(check-expect (merge (list 1 2 2 8) (list 1 1 5))
              (list 1 1 1 2 2 5 8))
```

```
(define (merge lst1 lst2)
  (cond ((empty? lst1) lst2)
        ((empty? lst2) lst1)
        ((< (car lst1) (car lst2))
         (cons (car lst1) (merge (cdr lst1) lst2)))
        (else
         (cons (car lst2) (merge lst1 (cdr lst2))))))
```

The following is less efficient, needing more comparisons and more consing.

```
(define (merge lst1 lst2)
  (sort (append lst1 lst2) <))
```

Comment [CKR10]: Common mistake: defining merge, with tests for empty, and a helper that looped and didn't test for empty. This breaks.

Comment [CKR11]: putting an ((and (empty? lst1) (empty? lst2)) empty) branch before this branch is redundant

Comment [CKR12]: putting an ((and (empty? lst1) (empty? lst2)) empty) branch after this branch is worse because it can never be executed

Comment [CKR13]: <= is not really needed

Comment [CKR14]: (< (car lst2) (car lst1)) is redundant

Comment [CKR15]: sort requires a predicate to determine the ordering