

University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2009

Instructor: Dan Garcia

2009-03-18

CS3L Midterm

(define (recursion) (recursion))

Personal Information

<i>Last name</i>	ANSWER KEY
<i>First Name</i>	
<i>Student ID Number</i>	
<i>Login</i>	cs3-
<i>The name of your TA (please circle)</i>	Aditi DavidW DavidZ
<i>Name of the person to your Left</i>	
<i>Name of the person to your Right</i>	
<i>All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS3 who have not taken it yet. (please sign)</i>	

Instructions

- Please turn off all cell phones.
Remove all hats & headphones.
- You have three hours to complete this midterm.
It is open book and open notes, no computers.
- Partial credit will be given for incomplete / wrong answers, so please write down as much of the solution as you can.
- Use `true` instead of `#t`, `false` instead of `#f`, since they are equivalent. Handwritten `#t` and `#f` unfortunately look too much alike...
- Feel free to write λ instead of lambda.
- Write the difficulty and fairness ratings in the boxes to the right and please add additional comments below.

Grading Results

<i>Question</i>	<i>Max. Pts</i>	<i>Points Earned</i>	<i>Difficulty (0=easy 5=hard)</i>	<i>Fairness (0=fair 5=unfair)</i>
1	10	10		
2	10	10		
3	15	15		
4	20	20		
5	10	10		
6	15	15		
Total	80			

Please comment above & left:

Login: cs3-_____

Question 1: Aaaaair ball... Aaaaair ball... (10 pts)

You play on the Cal basketball team, and have been keeping track of the free throws you've made and missed for a *game* as a word of 1s (makes) and 0s (misses). You store your season (every game's results) in a sentence. E.g., if your season were the sentence (1000 11 "" "00"), that would mean that you played in 4 games. In the first game (the 1000) you made one, and missed the next three, in the next game you made two, in the third game you didn't shoot free throws at all, and in the last game you missed twice. We want you to write `best-game`, to find out the free throw percentage for your *best game*. You may assume that you shot *at least* one free throw this season.

```
(best-game '(1000 11 "" "00")) → 1      ;; 1 = 100%, for your second game
(best-game '(1011 "" 10 "")) → 0.75    ;; In your first game, you went 3-for-4
(percentage 1011)           → 0.75    ;; A helper; it requires non-empty input
(shot-free-throws? 1011)   → true     ;; This is a helper we wrote for you that
(shot-free-throws? "")     → false    ;; returns true when you shot free throws
```

- You **may not** define any additional helper procedures
- You **may not** use `lambda` or `if` or any explicit recursion
- You **may only** use higher-order functions (and `percentage`, `shot-free-throws?` and other standard scheme built-in functions, like `+`, `-`, `min`, `max`, etc.)

```
(accumulate + game) ;; ← this also works
(/ (appearances 1 game) (count game))
```

```
(define (percentage game) _____ )
```

```
(define (best-game season)
  (accumulate max (every percentage (keep shot-free-throws? season)))
  _____ )
```

Question 2: Magical Mystery Function, step right this way... (10 pts)

```
(define (mystery x)
  (lambda (y)
    (every x y)))
```

Remember: it's not enough to say the domain or range is simply "a function". You have to describe the domain and range of that function! (...and so on if its domain / range is also a function)

What is the *domain* of `mystery`? (3 pts)

`x` is a function whose domain is (at least) words and whose range is words or sentences

What is the *range* of `mystery`? (3 pts)

A function whose domain is words or sentences and whose range is a sentence

Fill in the blanks to complete the interaction with `mystery`. (4 pts).

```
( _____ (λ (w) (word (first w) (last w))) )
```

```
STk> (define bookender _____ mystery _____)
STk> (bookender '(i love cs3)) → (ii le c3) ;; the first and last letters
```

Login: cs3-_____

Question 3: Go Bear! (15 points)

You decide to simulate your stock portfolio as it plummets downward. The function `stock` takes in the starting value `n` and returns a sentence that simulates what happens to it over time. When the stock reaches 1, you sell it.

```
(define (stock n)
  (se n
    (if (= n 1)
        'sell!
        (stock (if (zero? (remainder n 3)) ;; is n a multiple of 3? (3,6,9,...)
                    (/ n 3)
                    (+ n 1) )))))
```

a) What happens to stocks that were valued at 2? I.e., what will `(stock 2)` return? If it is an error, say what it is. If it is an infinite loop, write “∞ loop”. (2 pts)

(2 3 1 sell!)

b) What happens to stocks that were valued at 4? I.e., what will `(stock 4)` return? If it is an error, say what it is. If it is an infinite loop, write “∞ loop”. (4 pts)

(4 5 6 2 3 1 sell!)

c) Now let’s do some analysis of a blue chip stock valued at 2,999. What are the first three and last three elements of `(stock 2999)`? Fill in the blanks below. (3 pts)

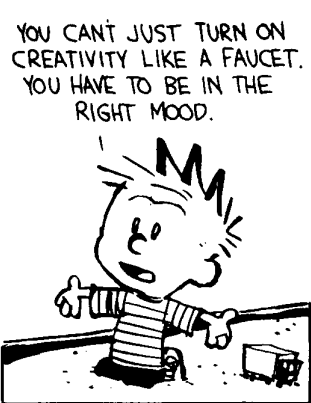
(2999 3000 1000 3 1 sell!)

d) If we changed the last line from “(+ n 1)” to “(+ n 3)”, what values of `n` would be unaffected (i.e., `stock` would return the same value)? (3 pts)

Powers of 3: $3^0=1$, $3^1=3$, $3^2=9$, etc. because the else line wouldn't be hit

e) For those values of `n` that *would* be affected by the change in (d), what would `stock` return? If it’s an error, say what it is. If it is an infinite loop, write “∞ loop”. (3 pts)

∞ loop



Login: cs3-_____

Question 4 : Does recurs-in occurs-in? recursion? No! (20 points)

You want to write the predicate `occurs-in?` from lab to check if a given *pattern* occurs in a word *w* as a contiguous subsequence:

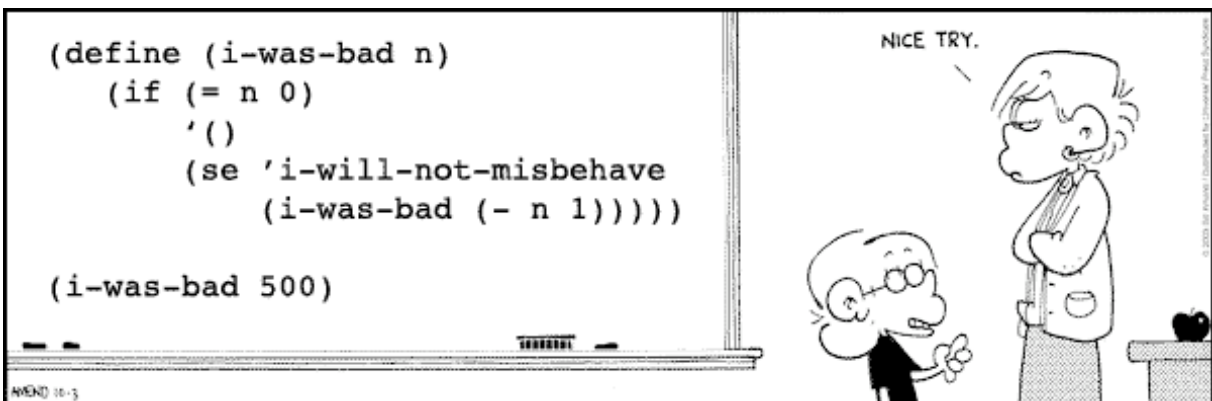
	<i>pattern</i>	<i>w</i>	
(occurs-in?	'ello	'hello)	→ true
(occurs-in?	'cat	'hello)	→ false
(occurs-in?	'hlo	'hello)	→ false
(occurs-in?	'hello	"")	→ false
(occurs-in?	"	'hello)	→ true

You try to write `occurs-in?` but it's buggy:

```
(define (occurs-in? pattern w)
  (cond
    1 ((empty? pattern) true)
    2 ((empty? w) false)
    3 ((equal? (first pattern) (first w))
    4 (occurs-in? (bf pattern) (bf w)) )
    5 (else
    6 (occurs-in? pattern (bf w)) )))
```

- a) Does `occurs-in?` employ (circle one) **TAIL** (no embedding of the recursive call) or **EMBEDDED** recursion? (2 pt)
- b) What is the *smallest input* that reveals the bug? (6 pts) That is, `(occurs-in? 'ac 'abc)` returns **true** instead of **false**.
- c) Aha! Your very smart friend says ... that's because on line 3 you're only testing equality of the *very first* letters. You should change that `cond` clause (lines 3 and 4) to see if the *entire* pattern is at the front of *w*! If so, you're done and return `true`, otherwise fall through to the `else` and keep recursing on *w*. Your friend changes line 4, it's your job to change line 3 (we've split it into 3a & 3b) so that `occurs-in?` will work correctly. Line 3a should check that *pattern* is no bigger than *w*. (12 pts)

```
(define (occurs-in? pattern w)
  (cond
    1 ((empty? pattern) true)
    2 ((empty? w) false)
    3a ((and (<= (count pattern) (count w))
    3b (equal? pattern (repeated bl (- (count w) (count pattern))) w) ) )
    4 true) ;; ← your friend changed this line, you change the ones above
    5 (else
    6 (occurs-in? pattern (bf w)) )))
```



Login: cs3-_____

Question 5 : He's a Unix... He's a Unix... He's dead. (10 points)

The Unix file system we use displays directories with slashes between them. For example, our home directory is `"/home/ff/cs3"`. We'd like to convert that to a sentence of directories by splitting that word `w` into many words in a sentence. Rather than write a splitter specific to `/`, we'll write one that can split on any character `c`.

Fortunately, the split char *will always be* the first character, *won't be* the last character, and *will never appear twice in a row* in the input word (e.g., `"/"`).

```
(unixsplit "/" "/home")      → (home)
(unixsplit "/" "/home/ff/cs3") → (home ff cs3)
(unixsplit ":" ":home:ff:cs3") → (home ff cs3) ;; older Macs used ":" not "/"
```

Fill in the blanks to complete `unixsplit` to split a word `w` by the input character `c`. We've started it for you.

```
(define (unixsplit c w)
  (accumulate
    (lambda (ltr ans)
      (cond ((word? ans) (se ltr) ;; first time through, strip c, return sent
              ((equal? ltr c) (se "" ans))
              _____
              (se (word ltr (first ans)) (bf ans))
            (else _____ )))
    (word (bf w) c) ) ;; strip c from beginning, add it to the end
```

Question 6: This midterm comes in like a lion, goes out like a Lambda (15 pts)

a) Add **only** parens and quotes so the expression returns an empty sentence. (5 pt)

```
_____  
((lambda () _____) _____) _____  
(((lambda () _____) _____) _____) _____  
(((lambda () '()) _____) _____) _____  
((lambda _____) _____) _____  
((lambda _____) _____) _____  
_____ lambda _____ lambda _____
```

b) What are *two* reasons we might use a `let`? (4 pts)

(1,2) To remove redundant code for style/performance reasons, (3) to remember a random #, (4) To make our code easier to read by giving a name to an expr

c) You love recursion so much, you start doing *everything* recursively.

In English, describe an embedded recursive algorithm for eating a candy bar. (Hint: it's similar to the algorithm we used to count the stairs in lecture.) (6 pt)

To eat a candy bar: Ask: is the candy bar finished? If so, stop. Otherwise, take a bite and recurse (i.e., follow these directions to eat a candy bar).