University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2000 Instructor: Dan Garcia 2000-11-01

CS 3 Midterm #2

Personal Information	
Last name	
First Name	
Student ID Number	
Lab Section Time & Location ***you attend***	
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)	

Instructions

- Please write your name on every page! You have three hours (or more if you need!) to complete this quiz. The quiz is open book and open notes, no computers.
- **You'll get credit for your best five out of six questions.** Partial credit will be given for incomplete/ wrong answers, so please write down as much of the solution as you can**.**
- You may always write auxiliary functions for a problem unless they are specifically prohibited in the question.
- Feel free to use any Scheme function that was described in sections of the textbook we have read without defining it yourself. Do not use functions or constructs that we have not covered, such as let or lambdas.
- You do not need to write comments for functions you write unless you think the grader will not understand what you are trying to do otherwise.
- Please comment on the exam in the margins on this page. Rate its difficulty (0 = cake, 5 = impossible), fairness (0 = unfair, 5 = fair), and feel free to add any other comments that come to mind.

Grading Results & Your Comments

Question	*Max. Points*	*Points Given*	*Liked or disliked?*
1	**10**		
2	**10**		
3	**10**		
4	**10**		
5	**10**		
6	**10**		
Subtotal	**60**		
- Lowest	**10**		
Total	**50**		

Question 1 – Magical Mystery function, cons right this way (10 points)

Consider the following `mystery` function:

```
(define (mystery grade-list a-plus-min)
(if (null? grade-list)
#t
(and (>= (first grade-list) a-plus-min)
(mystery (rest grade-list) a-plus-min))))
```

a. Just by looking at the code above, which of the four types of recursion we've seen would you say is being done here and why? (1.5 points)

b. What is a good name for `mystery` procedure? (1.5 points)

c. Fill in the specifications for `mystery`. (3 points)

```
;; INPUTS : è

;; REQUIRES : è

;; SIDE-EFFECTS : è

;; RETURNS : è

;; EXAMPLE : è

;; EXAMPLE : è

;; (have your two examples highlight mystery's different return values)
```

d. Rewrite `mystery` (still using recursion) so that it doesn't have "work to do" once the base case is reached. If you use a helper function you will lose one point. (4 points)

```
(define (mystery grade-list a-plus-min)
```

```
(define (hour-part time-list) (first time-list))

(define (minute-part time-list) (second time-list))

(define (minutes-in-day time-list)

(+ (* (hour-part time-list) 60) (minute-part time-list)))

(define (time-parked time-in time-out)

(- (minutes-in-day time-out) (minutes-in-day time-in)))

(define (parking-charge time-in time-out)

(* (truncate (/ (+ (time-parked time-in time-out) 19) 20)) 0.25))
```

Question 2 – 20,000 Leagues Under the Sea (10 points)

You want to write a procedure `depth` to return the depth of the deepest sub-list of your argument. E.g.,

```
(depth 'a) ==> 0
```

```
(depth '() ) ==> 1

(depth '(a b c) ) ==> 1

(depth '(a (b) c) ) ==> 2

(depth '(a ((b)) ((c (d)))) ) ==> 4
```

a. Just by looking at the skeleton code below in part (b), which of the four types of recursion we've seen would you say is being done here and why? (2 points)

b. We've started the procedure for you. Fill in the blanks to finish it. (8 points)

```
(define (depth L)

(cond

((atom? L)

_____ )

((null? L)

_____)

((list? (first L))

_____)

(else

_____)))
```

Question 3 – When in Rome… (10 points)

Your malicious lab partner "accidentally" changes one of the symbols or numbers in the `roman-sum` function from Appendix B of the Roman Numerals case study. When using the modified code, you find that a call to `roman-sum` with the argument `'(1 5)` returns `9`, instead of the correct answer `4`. Here is the original code for `roman-sum` from the case study for your reference:

```
1 (define (roman-sum number-list)

2 (cond

3 ((null? number-list) 0)

4 ((null? (cdr number-list)) (car number-list))

5 ((not (starts-with-prefix? number-list))

6 (+ (car number-list)

7 (roman-sum (cdr number-list)) ) )

8 ((starts-with-prefix? number-list)

9 (+ (- (cadr number-list) (car number-list))

10 (roman-sum (cddr number-list)) ) ) ) )
```

a. Fill in the blanks below to specify a possible cause of the bug, that is, a substitution of a single symbol that would produce the behavior described above. (6 points)

   On line # _____,

   replacing the symbol or number _____

   with the symbol or number _____

   would cause `(roman-sum '(1 5))` to return `9`.

b. You now evaluate the expression `(roman-sum '(10 5 1))` and it returns `16`. Can you rule out the bug you suggested in part a? Why or why not? (4 points)

Put down your pen or pencil, stretch, take a deep breath, and proceed…

We're talking Baseball… (Questions 4, 5 and 6)

You graduate from Cal and get a job as a Sabermetrician (someone who analyzes baseball statistics). Every baseball game involves (usually) several at-bats for a player. For purposes of calculating batting average statistics, players either get a hit (`h`) or an out (`o`) every time they bat. For this problem, assume all players have unique last names so we need only refer to players by their last names (but it is rumored that there is a player with the last name of `h` and one with the last name of `o`). If World Series MVP Derek Jeter batted three times today and got a hit, an out and a hit, the list would look like: (`Jeter h o h`). We'll call this list a "`days-work`".

You are given a list containing all the days works for all the players for the season, which we'll call a "`days-work-list`". This could look like the following for a short season (in general, a player will be listed multiple times, as `jeter` and `piazza` are here):

```
(define *sample-days-work-list*

((jeter h o h) (sojo h o) (piazza o) (jeter h) (piazza h o)))
```

A player's batting average is computed by dividing all the hits (# `h`'s) over the course of the season by all the at-bats (# `h`'s + # `o`'s). For the sample season above:

a. Jeter's average is .750 (3 hits / 4 at-bats)
b. Sojo's average is .500 (1 hit / 2 at-bats)
c. Piazza's average is .333 (1 hit / 3 at-bats)

You may assume each player's day's work has at least one at-bat (i.e., each `days-work` is at least length 2). We would like to know who had the *highest batting average*, and what it was. That is, we would like you to write the procedure `batting-champion`, which takes a `days-work-list` and returns a list of the player and their batting average. (In the case of a tie, return any winner.) E.g.,

`(batting-champion *sample-days-work-list*)` would return `(jeter .75)`

Recall how we decomposed the difficult bowling problem into simpler procedural components, and specified intermediate data structures (with examples):

```
;; score-reports (a "score-report" list)

((theresa 120) (amelia 150) (theresa 220) (will 165) (amelia 200))

|

| SCORE-REPORTS-TO-TOTALS

v

;; score-totals (a "score-total" list)

((theresa 340 2) (amelia 350 2) (will 165 1))
```

```
|
| SCORE-TOTALS-TO-AVERAGES
V
;; score-averages (a "score-average" list)
((theresa 170) (amelia 175) (will 165))
|
| FIND-HIGHEST-AVERAGE-BOWLER
V
amelia
```

Question 4 – Get top down and boogie… (10 points)

a. Now, decompose the baseball problem similarly. That is, show the process of finding the batting champion from the seasons-work data by:

i. Creating (and naming!) intermediate data structures
ii. Showing us examples of what the data structures look like
iii. Naming the procedures which operate on the data structures.

There are four total steps we'd like you to use, with each step performing a small, non-trivial task. We've done the first two steps for you. (5 points)

```
;; days-work-list (a list of all the "days-work"s for each player)
((jeter h o h) (sojo h o) (piazza o) (jeter h) (piazza h o)))
|
| STEP I : ACCUMULATE-SEASONS-WORK
V
;; seasons-work-list (a list of all the "seasons-work"s for each player)
((jeter h o h h) (sojo h o) (piazza o h o))
|
| STEP II : TOTAL-HITS-OUTS
```

V

;; **hits-outs-list** (a list of all the "hits-outs" for each player)

((jeter 3 1) (sojo 1 1) (piazza 1 2))

|

| **STEP III :** è

V

è

è

|

| **STEP IV :** è

V

(jeter .75)

a. Just as we were able to write `highest-average-bowler` without writing any of the helper functions, write `batting-champion` using the procedures defined above. (5 points)

```
(define (batting-champion days-work-list)
```

```
(define (hour-part time-list) (first time-list))

(define (minute-part time-list) (second time-list))

(define (minutes-in-day time-list)

(+ (* (hour-part time-list) 60) (minute-part time-list)))

(define (time-parked time-in time-out)

(- (minutes-in-day time-out) (minutes-in-day time-in)))

(define (parking-charge time-in time-out)

(* (truncate (/ (+ (time-parked time-in time-out) 19) 20)) 0.25))
```

Question 5 setup – Vader grabbed a light Sabermetrician… (10 points)

Someone has been kind enough to implement Step I (`accumulate-seasons-work`) for us. Your job for Question 5 is to implement Step II (`total-hits-outs`) two different ways, to impress your Sabermetrician boss. You have been provided a few selector and constructor functions and the specifications for `total-hits-outs`. Do you work for the functions on the following page.

`;; Selectors and Constructors (use them or lose style points)`

```
(define *hit* 'h)

(define *out* 'o)

(define (hit? at-bat) (equal? at-bat *hit*))

(define (out? at-bat) (equal? at-bat *out*))

(define (get-name seasons-work) (first seasons-work))

(define (get-atbats seasons-work) (rest seasons-work))

(define (make-hits-outs player hits outs) (list player hits outs))

;; We will make the assumption that get-name (defined above)

;; works on hits-outs

;; AS WELL AS

;; seasons-work and we won't need to define it. If we did,

;; it would be defined as shown below.

;;

;; (define (get-name hits-outs) (first hits-outs))

(define (get-hits hits-outs) (second hits-outs))

(define (get-outs hits-outs) (third hits-outs))
```

```
;; total-hits-outs

;;

;; INPUTS : seasons-work-list, a list of seasons-work for each player,

;; : which is their last name and h's and o's for their batting.

;; : (The order of the players does not matter)

;; :

;; REQUIRES : seasons-work-list to contain only one entry for each player
```

```
;; :

;; SIDE-EFFECTS : none

;; :

;; RETURNS : hits-outs-list - for each player, a list of their name,

;; : and the totals hits (h's) and outs (o's).

;; : The order of the players does not matter.

;; :

;; EXAMPLE : (total-hits-outs

;; : '((jeter h o h h) (sojo h o) (piazza o h o)) )

;; : returns

;; : ((jeter 3 1) (sojo 1 1) (piazza 1 2))
```

Question 5 – Baseball has been vedy vedy good to me… (10 points)

a. Implement `total-hits-outs` from Question 4a **using recursion with no functionals** – feel free to use helper functions if you wish. (5 points)

```
;; Recursive version of Step II to impress the boss

;;

(define (total-hits-outs seasons-work-list)
```

b. Implement `total-hits-outs` from Question 4a **without explicit recursion of your own** – feel free to use helper functions if you wish. (5 points)

```
;; Functional version of Step II to impress the boss

(define (total-hits-outs seasons-work-list)
```

Question 6 – (Boys and) Girls just wanna have FUNctionals (10 points)

a. Implement Step III from Question 4a **using functionals and no explicit recursion of your own**. You should write a constructor for the data structure you designed in Question 4a. You may use helper functions if you wish. (5 pts)

```
;; Write your definition for the Step III function here

(define ( hits-outs-list)
```

b. Implement Step IV from Question 4a **using functionals and no explicit recursion of your own**. You should write a selector for the data structure you designed in Question 4a and constructed in Question 6a. You may use helper functions if you wish. (5 points)

```
;; Write your definition for the Step IV function here

(define (
```