CS 3 Midterm #1

Wednesday, September 27, 2000

Instructor: Dan Garcia

Personal Information	
Last name	
First Name	
Student ID Number	
Lab Section Time & Location ***you attend***	
All the work is my own and 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 to complete this quiz. The quiz is open book and open note, no computers.
- **You'll get credit for your best four out of five 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 functionals.
- 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.

Grading Results

Question	*Max. Points*	*Points Given*
1	**10**	
2	**10**	
3	**10**	
4	**10**	
5	**10**	
Subtotal	**50**	
- Lowest	**10**	
Total	**40**	

Question 1 : Olympic Fever (10 points)

(Thanks to Brendan Ferguson for a great question…imitation is the best form of flattery!)

Your grandmother tells you that back in her day they didn't have those fancy computers to evaluate all of their computer programs for them. No, they evaluated all of their programs themselves. Prove to your grandmother that you could do the same if you had to. For each of the following Scheme expressions, write the value that would be returned if we were to evaluate it in the Scheme interpreter. Write your answers next to each expression in the space provided. If an expression results in an error, just write the word ERROR. Assume that the following `defines` have already been made:

```
(define cal 'number-1)
```

```
          (define L ’( ((cal)) (bears) second-to () ))

          (define (bronze x y)

          (list x ‘(* x y)) )

          (define (silver element list)

          (cons element (append ‘() list)))

          (define (gold list)

          (list list))
```

L	
(length L)	
(cdaar L)	
(bronze ‘() 5)	
(bronze #f (/ 1 (truncate 2/3)))	
(silver ‘list ‘(list))	
(silver cal (silver ‘() ‘()))	
(gold ‘cal)	
(gold ‘list)	
‘(‘these-were-hard)	

Question 2 : part A – Berkeley, 1967: sum-mer of love (4 points)

Write a function `sum-ends` that takes one argument, a list of at least two numbers, and replaces the last two elements with their sum. Example:

```
: (sum-ends '(1 2 4 8 16))

(1 2 4 24)
```

```
;; Write your definition for sum-ends here

;; (Feel free to define any helper functions you may need)
```

```
(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, part B – sum-four score and seven years ago (3 points)

Write a function `sum-four` that takes a list of four numbers as an argument and returns a list containing the sum of their elements. The only problem is now a Stanford student has stolen the letters: `C A L T +` from your keyboard, so for this part you can't use any functions that have those letters in them! However, feel free to use any other list-creation functions you've written so far on this exam. Here's an example of `sum-four`:

```
: (sum-four '(8 4 2 1))

(15)
```

```
;; Write your definition for sum-four here

;; Remember, you can’t have the letters C, A, L, T and + in your answer!
```

```
(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, part C – Calling all cars, er, functions! (3 points)

Suppose we wish to compute `f(x) = 7*(x - 2)*(x - 2)` and we have already been given the following function definition: `(define (f x) (g (h (j x))))` Your job is to define `g`, `h` and `j` so that `f` is computed correctly. Each of them should perform at most **one** computation of the form: `+, -, *,` or `/`.

```
(define (g x) )



(define (h x) )



(define (j x) )
```

Question 3, part A – The Meaning of Life (6 points)

Fill in the blanks to get the given results. The only functions allowed are `cons`, `append`, and `list`.

```
:(__________ '(why are) ____________________________________)
```

```
((why are) we here)

:(__________ '(why are) ________________________________________)

(why are we here)

:(__________ '(why are) ________________________________________)

(why are (we here))

:(__________ '(why are) ________________________________________)

(why are we here ())

:(__________ '(why are) __________________) ;; This solution and

((why are) (we here))

:(__________ '(why are) __________________) ;; this solution must be different

((why are) (we here))
```

Question 3, part B – Eighth is enough (4 points)

Write the procedure `eighth` (which, as you can guess, returns the eighth element of a list) three different ways. You can't use a procedure in more than one box. So if you use `append` in your first box, you can't use it in your second two boxes. The first answer should only include functions that begin with `c` and end with `r`. The remaining answers should NOT include functions that begin with `c` and end with `r`. You may not use the function `rest` in any of your answers.

```
(define (eighth L) ; Fill in the body below with c__r functions, not rest
(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))
```

```
(define (eighth L) ; Fill in the body below without c__r functions or rest
```

```
(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))
```

```
(define (eighth L) ; Fill in the body below without c__r functions or rest
(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 4, part A – A little sum-thing sum-thing (2 points)

It has been hinted that the procedures `if`, `cons` and (`and`, `or` & `not`) are interchangeable. Let's see if that's the case. Your goal is to write a procedure `even-sum?` of two integer arguments that returns `#t` if and only if their sum is even (otherwise `#f`). In this part, do NOT use `if`, `cond`, `and`, `or`, `not`. In none of these problems are you allowed to call another `even-sum` function.

```
;; Write this without using if, cond, and, or & not

;;

(define (even-sum-a? x y) ; fill in body below
```

Question 4, part B – sum-shine on my shoulders (2 points)

Now write it a different way, *without* calling "numerical combiners" (i.e., no `+`, `-`, `/`, `*`, `truncate`, `sqrt`, `abs`, `expt` & `remainder`). Study the four cases when two inputs are the different combinations of even and odd. Write this by using combinations of `and`, `or`, `not`, `even?`, & `odd?`. Do NOT use `if` or `cond` here.

```
;; Write this without using +, -, /, *, truncate, sqrt, abs, expt & remainder

;; Hint: use and, or, not, even? and odd?. Do NOT use if and cond.

;;

(define (even-sum-b? x y) ; fill in body below
```

Question 4, part C – You sum my battleship! (3 points)

Now write it a different way, again without calling numerical combiners. However, this time you are also restricted from using `and`, `or` & `not`. Use only `if`, `even?` & `odd?`. If you use `#t` or `#f` (or any expression that always evaluates to one of them) in your function, you will lose one point.

```
;; Write using only if, even? & odd? Do not use #t or #f in your function.

;;

(define (even-sum-c? x y) ; fill in body below
```

Question 4, part D – I cond take it anymore! (3 points)

Now write it a different way, again without calling numerical combiners. However, this time you are also restricted from using `and`, `or` & `not`. Use only `cond`, `even?` & `odd?`. If you use `#t` or `#f` (or any expression that always evaluates to one of them) in your function, you will lose one point.

```
;; Write using only cond, even? & odd? Do not use #t or #f in your function.

;;

(define (even-sum-d? x y) ; fill in body below
```

Question 5 – (list 'Superman!) (10 points)

One of the things you might have realized is that there are some things you <u>can</u> do with arbitrarily-long lists and some things you <u>can't</u> given what we've learned so far in class (i.e., no recursion or functionals; if you don't know what those are, that's ok – it'll be easier for you to answer the question!). For example, you <u>can</u> remove the first two elements from the front, but you <u>can't</u> add 1 to each of the numbers in a list of numbers. This question is intended to test whether your know the limits of your arbitrary-length list manipulation "power". Remember that you have access to some very powerful functions: `position`, `member`, `count`, `remove`, `reverse`, `list-ref`, `subseq`, and `length`. We will ask you to perform operations on lists, and you need to either show us how you do it, or say "not in my power…yet!".

Problem	**Solution** (If possible, just fill in the body of the function. Assume the argument was `L`)

Remove the first two elements from the front of an arbitrarily-long linear list, and return the remaining part. E.g.,
 	
`: (remove-first-two-elements '(1 2 3))`
 	
`(3)`
	`(rest (rest L))`

Add 1 to each of the numbers in an arbitrarily-long linear list of numbers.	**not in my power...yet!**

Return the middle element in a linear list of arbitrary length (assume the length is odd). E.g.,
 	
`: (middle-element ‘(1 2 3 4 5 6 7))`
 	
`4`
 	
`: (middle-element ‘(cindy jan marsha))`
 	
`jan`	

Given a linear list of arbitrary length, return a list of all the 1s. E.g.,
 	
`: (get-ones ‘(3 1 foo 1 5 a 2 6 5 1))`
 	
`(1 1 1)`	

Given an arbitrarily-long linear list of face card values (one of the set of `a`, `k`, `q`, `j`), return the list of aces (the `a`’s). E.g.,
 	
`: (get-aces ‘(a k a a a a q))`
 	
`(a a a a a)`	

Given an arbitrarily-long linear list of face card values, return the number of them which are not aces). E.g.,
 	
`: (how-many-not-aces ‘(a k a a a a q))`
 	
`2`	

Reverse both levels of an arbitrarily-long list of arbitrarily-long lists. E.g.,
 	
`: (reverse-both ‘((1 2 3 4)`
 	
`(x y)`
 	
`(a b c)) )`
 	
`((c b a) (y x) (4 3 2 1))`	