University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2002 Instructors: Clint Ryan & Dan Garcia 2002-09-20

CS 3 Midterm #1

Personal Information

Last name	
First Name	
Student ID Number	
The name of the TA for the Discussion ***you attend***	
Name of the person to your Left	
Name of the person to your Right	
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

We will drop your lowest score for questions 1 through 4. Question 0 is compulsory.

You have 50 minutes to complete this quiz. The quiz 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.

For these questions you only need the functions from the following sections (listed in the back page of the book): **Words and Sentences**, **Arithmetic**, **True and False** and **Variables**.

Use `true` instead of `#t`, and `false` instead of `#f`. We have found that handwritten `#t` and `#f` unfortunately look too much alike.

Please comment on the exam on the right. Rate its difficulty (0 = cake, 5 = impossible), fairness (0 = unfair, 5 = fair), and **feel free to add any other comments that come to mind.**

Please turn off all pagers, cell phones and beepers. Remove all hats & headphones.

Grading Results

Question	*Max. Points*	*Points Earned*
0	**2**	
1	**6**	
2	**6**	
3	**6**	
4	**6**	
Subtotal	**26**	
Min (of 1-4)	**6**	
Total	**20**	

Comments:

Difficulty (0=easy, 5=hard):
Fairness (0=unfair, 5=fair):
Other comments? (write here)

Question 0 : Compulsories... (2 points)

Assume **you don't know what recursion or higher-order functions are.** Could you write a function whose domain is a sentence names containing the first names of all the people seated in this room right now and which... (circle YES or NO)

a) ...returns the number of people with the same name as you? YES NO

b) ...returns the number of people whose first name ends in "*ing*", like "Ming" and "Yaping"? YES NO

Question 1 : I'm drawing a blank... (6 points)

Fill in the blanks below. When you see the symbol "➔", this means you should write down what the interpreter would return if the expression were typed in. If any of the following displays an error, write ERROR and describe (in your own words) what the error is. If the answer is a procedure, write #<PROCEDURE *procedurename*>, for example: #<PROCEDURE square>. (1 pt each)

```
(define (mystery a b c)
   (let ((b 7))
      (if (= a b)
          (se (word "" a) '() 'b)
          c)))

(define (evil) good)

(define (good) 'night)
```

a) `(mystery 7 10 'split)` ➔ _____.

b) `(mystery 9 2 mystery)` ➔ _____.

c) `(mystery mystery mystery mystery)` ➔ _____.

d) `((and not evil))` ➔ _____.

e) Describe, **<u>as precisely as possible</u>**, the *domain* and *range* of `mystery`. (1 each)

	Domain	Range
a:		
b:		
c:		

Name: _____

Question 2 : Down at the `swap-args` meet... (6 points)

Your lab partner wishes to write a function `swap-args` that swaps (i.e., switches) the arguments of a two-argument function. For example, you know that (in `stk`)

```
> (/ 2 6) → 0.333333333333
```

Your partner would *somehow* like to call `swap-args` with some arguments and have the result be *as if* the computation were:

```
> (/ 6 2) → 3
```

...but they are a little confused how to write it and how to call it. They start with:

```
(define (swap-args x y)     ;; version 1 of swap-args
  (y x))
```

a) Then they attempt to call it several ways. Fill in the blanks below. When you see the symbol "→", this means you should write down what the interpreter would return if the expression were typed in. If any of the following displays an error, write ERROR and describe (in your own words) what the error is. If the answer is a procedure, write #<PROCEDURE *procedurename*>, for example: #<PROCEDURE square>. (1 point each)

> (swap-args / 2 6) → _____.

> (/ (swap-args 2 6)) → _____.

> (swap-args (/ 2 6)) → _____.

b) Now provide a call to `swap-args` (version 1, above) which *actually returns* 3 (1 pt)

> (swap-args _____) → 3

c) Fix `swap-args` so that it can be used in a general fashion to reverse the arguments of any two-argument procedure (i.e., it shouldn't have subtraction hard-coded). *Hint: When you are through, one of the calls from part (a) will return 3.* (2 points)

```
;; Version 2 of swap-args, fixed

(define (swap-args _____)

   (_____))
```

Name: _____

Question 3 : Beethoven wasn't the only great composer... (6 points)

You are told `inner-ends` **doesn't contain** `if`, `cond`, `and`, `or`, `not`. The intent is to grab the last letter of the first word and the first letter of the last word from a sentence and smush them together. E.g.,

```
> (inner-ends '(abcde fghij klmno pqrst uvwxy)) → eu
> (inner-ends '(123 456 789)) → 37
```

a) Write the *simplest* (most straight-forward, fewest function calls) definition for `inner-ends`. When you see the symbol "→", show what the example call to `inner-ends` would return. If it displays an error, write `ERROR` and describe (in your own words) what the error is. If the answer is a procedure, write `#<PROCEDURE procedurename>`, e.g., `#<PROCEDURE square>`. (2,1,1 points)

```
(define (inner-ends s)
   (_____))

> (inner-ends '(ucb)) → _____

> (inner-ends  'ucb)  → _____
```

b) Now, given the following functions

```
(define (unend ws)          ;; Remove the ends of a word/sentence
  (bf (bl ws)))

(define (duplicate ws)      ;; Duplicate a word/sentence
  (if (word? ws)
      (word     ws ws)
      (sentence ws ws)))
```

Fill in the blanks below with **three English words** so that the expression evaluates correctly. E.g., "Raise the roof" or "Snoop Doggy Dogg". (2 points)

```
(inner-ends (unend (duplicate '( _____ _____ _____ )))) → el
```

Question 4 : Now you write `my-if`, no `if`s, `and`s or `not`s!... (6 points)

Given the function: `(define (my-if X Y Z) (if X Y Z))`

Rewrite `my-if` using `and`, `or` & `not` (you may **NOT** use `if` & `cond` in your solution).	Rewrite `my-if` using `cond` (you may **NOT** use `if`, `and`, `or` & `not` in your solution).

`(define (my-if X Y Z)`
`  (_____ (and X Y)`
`  (_____)))`
	`(define (my-if X Y Z)`
	`  (cond (_____)`
	`  (_____)))`