University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2003 Instructor: Dan Garcia 2003-10-15

CS3 Midterm

`(define (recursion) (recursion))`

Personal Information

Last name	
First Name	
Student ID Number	
Login	cs3-
The name of your TA (please circle)	Alex Andrew Anil Lauren Clint
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

- You have two 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..
- Please turn off all pagers, cell phones and beepers. Remove all hats & headphones.
- Write the difficulty and fairness ratings in the boxes to the right and please add additional comments below.

Grading Results

Question	*Max. Pts*	*Points Earned*	*Difficulty (0=easy 5=hard)*	*Fairness (0=fair 5=unfair)*
0	**1**			
1	**9**			
2	**10**			
3	**10**			
4	**10**			
5	**10**			
Total	**50**			

Please comment above & left:

Question 1 : Blasts from the Past!! (9 points, 1 point each, we drop lowest)

Assume the following procedures have already been defined:

```
(define (infinite-loop) (infinite-loop))

(define (snack-or-not meal)
  (if (equal? meal (or 'breakfast 'lunch 'dinner))
      'not
      'snack))

(define (my-item num sw)    ;; sw means sentence-or-word
  (if (equal? num 1)
      (first sw)
      (my-item (- num 1) (bf sw))))

(define (mood? people)
  (or (empty? people)
      (and (equal? (first people) 'happy)
           (mood? (bf people)))))
```

Fill in the blanks below. If something is impossible, write **IMPOSSIBLE**. If something runs forever, write **RUNS-FOREVER**. If something will produce an error, write **ERROR**. *You do not have to explain the error*. The symbol ➔ means "evaluates to". The word **IMPOSSIBLE** can only appear in the input and **ERROR** can only appear in the output. For example,

`(+ 3 4)` ➔ __7__

1. `(if #f (infinite-loop) 'yankees)` ➔ ______________
2. `(first (bf (last (bl '(cs3 is a great class)))))` ➔ _____________
3. `(last ____________________________________________)` ➔ `(recursion)`
4. `(snack-or-not 'lunch)` ➔ _______________
5. `(my-item 1.5 '(one two three four))` ➔ _______________
6. `(my-item ______________________ '(recursion))` ➔ `e`
7. `(cond ('false 'yes!) (else 'no!))` ➔ _______________
8. `(count (se (se 'fun) '() 'recursion "" (word 'mid "" 'term)))` ➔ ________
9. To what recursion pattern is `mood?` closest? Circle one below:

   `MAPPING     FINDING     FILTERING     COUNTING     TESTING     COMBINING`

10. What is a better name for the `mood?` predicate? Circle one below:

    `all-unhappy?     any-unhappy?     all-happy?     any-happy?`

Name: ______________________________

Question 2: 1,2,3,4,5! Wow! I have the same combo on my luggage! (10 points)

The function `combos-for` produces a sentence of 'integer-integer' combinations. It takes in a positive integer `n` as its only argument and returns a sentence of all the possible combinations of `n` and all numbers `1` through `n` separated by a –. E.g.,

```
(combos-for 1) ➔ (1-1)
(combos-for 3) ➔ (3-1 3-2 3-3)
```

Below is an attempt at `combos-for`:

```
;; INPUT:   A positive integer
;; RETURNS: Returns a sentence of all combinations of n and 1-through-n

   (define (combos-for n)
1.   (combos-for-helper n 1))

   (define (combos-for-helper n i)
2.    (if (= n i)
3.        (word n '- i)
4.        (se (word n '- i)
5.            (combos-for-helper n (- i 1)))))
```

There are two bugs. Provide the answers to the following blanks:
(You may include **ERROR** or **RUNS-FOREVER** in your answers)

a) A call to `(combos-for 2)` results in ____________________ when it should return `(2-2 2-1)` or `(2-1 2-2)`. Replacing line ___ with ____________________________ fixes that particular bug.

b) After the fix, one bug still exists. Provide a valid call to `combos-for` that reveals the bug: `(combos-for ________ )` ➔ ________________. Replacing line ___ with ____ fixes the final bug.
After this fix, the function should perform as advertised on all valid input.

c) What type of recursion does the *original* `combos-for-helper` employ? Circle one:

TAIL EMBEDDED

Name: ______________________________

Question 3: Spin that wheel, cut that pack, and roll those loaded dice! (10 pts)

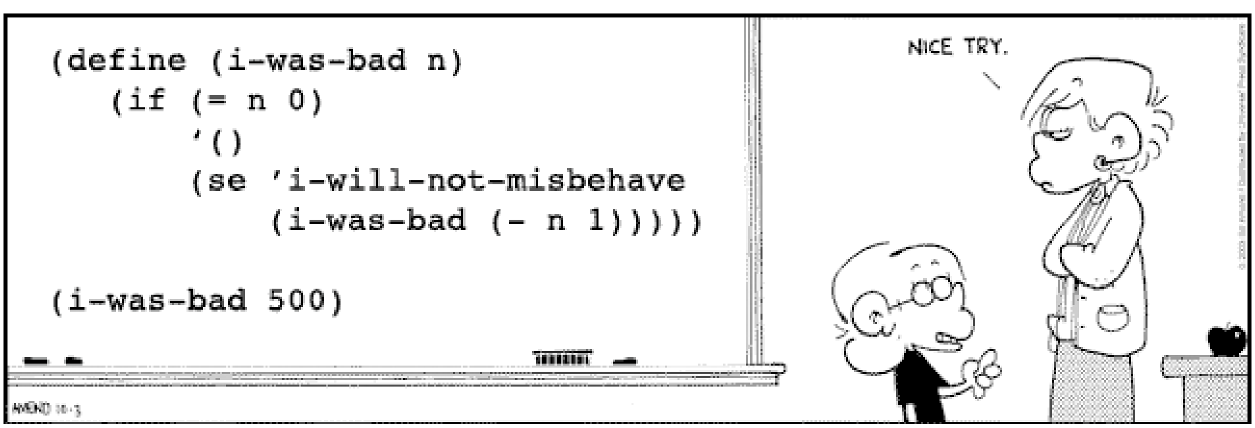
We want the function `dice-combos` to return all the possible combinations of rolling two `n`-sided dice. The format for a combination is the same as in Question 2, where all combinations come in "integer-integer" format. Order does not matter within a word or sentence, so `1-2` and `2-1` mean the same thing, & `(2-2 2-1 1-1)` and `(1-1 2-2 1-2)` mean the same thing. However, `i-j` and `j-i` (where `j` and `i` are any numbers) should never *both* appear in the answer (see the examples below).

`dice-combos` takes in a positive integer `n` which represents the number of sides on one die and returns a sentence of *all* the combinations that can result from rolling two `n`-sided dice. Here are some examples:

```
(dice-combos 1) → (1-1)                          ;; order doesn't matter
(dice-combos 2) → (2-2 2-1 1-1)                  ;; within word or sentence
(dice-combos 3) → (3-3 3-2 3-1 2-2 2-1 1-1)
```

Fill in the blanks to complete the code below. Feel free to call any other procedures found on this test (called "software re-use"); *assume they work as intended.*

```
(define (dice-combos n)
   (dice-combos-helper ________________   ____________________________________ )


(define (dice-combos-helper n so-far)
   (if (= n 1)

       _________________________________________________________________________

       (dice-combos-helper  _______  ________________________________________)))
```



Name: ______________________________

Question 4: Putting the FUN back in function! (10 pts)

You're given the following two functions:

```
(define (get-back w)          ;; send the first letter to the back
  (word (bf w) (first w)))

(define (word-fun n w)        ;; have some fun with words
  (if (= n 0)
      ""
      (word w (word-fun (- n 1) (get-back w)))))
```

a) What does the call to `(word-fun 1 'abc)` return?

b) What does the call to `(word-fun 3 'abc)` return?

c) What would be a good name for the unnamed function below that takes in a number `n` and a *letter* `L`?

```
(define (______________________ n L) (word-fun n L))     ;; L is a letter
```

d) What are the first and last three numbers of: `(word-fun 1000000 1234567890)`?

_________ *…lots-of-numbers-in-the-middle…* _________
first-3 *last-3*

Name: ______________________________

Question 5: Two roads diverged in a wood… (10 pts)

You're lost in the forest. Every *place* in the forest is either a *dead-end* or has exactly 2 *one-way paths*: *left* and *right*. Your goal is to find out if there is a way home.

We introduce an abstract data type called a *place*, but you don't know *(and you don't need to know)* how it is represented; it could be a word, sentence, boolean, or number. You are presented with four operations (all take a *place* as an argument):

`(home?      place)`	returns `#t` if the *place* is your home, `#f` otherwise.
`(dead-end? place)`	returns `#t` is the *place* is a *dead-end* (i.e., no paths from it).
`(go-left    place)`	follows the *left* path, returning a new `place`.
`(go-right   place)`	follows the *right* path, returning a new `place`.

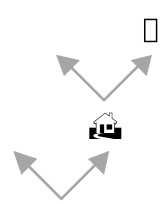
It is an error to `go-left` or `go-right` from a *dead-end* (because it has no paths!). There is no way in this forest to follow a sequence of left paths and/or right paths and end up where you started. I.e., there's no way to walk in circles. Your *home* (if one exists) might be at a *dead-end* or it might not. You might actually start your search at home.

Write `(path-home? place)` which uses the four functions above and returns `#t` if you can get home following a (possibly zero) number of lefts & rights starting from `place` and `#f` otherwise.

```
(define (path-home? place)

        ________________________________________________________

        ________________________________________________________

        ________________________________________________________

        ________________________________________________________ )
```

```
(home?    ) ➔ #f
(home?    ) ➔ #f
(home?  ) ➔ #t
(home?    ) ➔ #f
(home?  ) ➔ #f

(dead-end?    ) ➔ #f
(dead-end?    ) ➔ #t
(dead-end?  ) ➔ #f
(dead-end?    ) ➔ #t
(dead-end?  ) ➔ #t

(go-left    ) ➔
(go-right   ) ➔
(go-left   ) ➔
(go-right  ) ➔
;; go-left or go-right from  ,  or  is an ERROR
```

Example:



```
(path-home?    ) ➔ #t
(path-home?  ) ➔ #t
(path-home?    ) ➔ #f
```

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;
Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,
And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.
I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I--
I took the one less traveled by,
And that has made all the difference.
– Robert Frost