University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Spring 2001 Instructor: Dan Garcia 2001-05-14

CS3 Final Exam

Last name__________________________ First name __________________________

SID Number__________________________ TA's name __________________________

(Sorry to ask this next question, but with 300 students, there may be a wide range of behavior.)

The student on my left is ____

The student on my right is ____

I certify that my answers are all my own work. I certify that I shall not discuss the exam questions or answers with anyone in CS3 who has yet to take it until after the scheduled exam time.

Signature ____

Instructions

- Question 0: Fill in this front page and write your name on every page! The exam is open book and open notes (no computers). Put all answers on these pages; don't hand in any stray pieces of paper.
- 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.
- 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.
- You have three hours, so relax. We estimate 30 minutes per question.
- Each question is worth almost the same amount, so don't spend all your time on one problem; if you're stuck, move on.
- Good skill!

Grading Results

Question	*Max. Points*	*Points Given*
0	**1**	
1	**20**	
2	**20**	
3	**20**	
4	**20**	
5	**20**	
6	**19**	
Total	**120**	

Name: ______________________________

Question 1 – I'm drawing a blank... (20 points, 2 points each; 30 minutes)

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 display an error, write down what the error is.

a) `(word)` ➔ ______________________________.

b) `(every (lambda (L) (if (equal? L 'r) 'w L)) 'rabbit)` ➔ ______________.

c) `(caaddr '(cs3 ((is)) (the) coolest (class)) )` ➔ ______________.

d) `(keep ______________________________ '(3 2 7 99 21))` ➔ `(2 21)`

e) `(or (and ______________ 'spam ) 'water-buffalo)` ➔ `water-buffalo`

f) `( (lambda (f) (f (f 2))) (lambda (x) (* x x)) )` ➔ ______________.

g) Clint discussed the micromechanical flying insect (MFI). *Aside from the examples he mentioned*, name an interesting use for a swarm of these little "buggers". Your answer can be whimsical and fun if you wish. Limit your answer to one sentence.

"I would use MFIs for ______________________________".

h) `'(a (b))` is a (circle *as many as apply*): SENTENCE TREE LIST

i) You can use fractals to model the path of a river (circle one): TRUE FALSE

j) Oliver discussed the early troubles with natural language processing. (E.g., "*The spirit is willing but the flesh is weak*", when (mis)-translated from English to Russian and back to English, resulted in the phrase "*The vodka is good but the meat is rotten*".) Since then, they have fixed all the bugs and can now translate between English to Russian with almost perfect accuracy. (circle one) TRUE FALSE

Name: ______________________________

Question 2 – I'm makin' a sentence…I'm checkin' it twice… (20 pts; 30 min.)

You decide to store the names of all your friends in a sentence, and convert each name into a single word by sticking a dash "-" in-between. For example, "pat diaz" would be converted to `pat-diaz`. (Assume *nobody* has a dash in their first or last names and *everyone* has one first and one last name.) You write the constructor `make-name` as follows:

```
(define (make-name firstname lastname)
  (word firstname '- lastname))
```

a) Using recursion, write `last-name` which takes a name and returns the last name. **DON'T** write any auxiliary functions. (10 points)

```
(define (last-name name)

    ______________________________________________________

    ______________________________________________________

    ______________________________________________________ )
```

b) What type of recursion did you use above? Circle one. (4 points)

EMBEDDED TAIL

c) Next, write a function `make-keeper` that takes in a name predicate `good-name?` and returns a function that, when applied to a sentence of names, only keeps the names of people whose names are "good". (6 points) **Use no explicit recursion.** Here's an example:

```
: (define (diaz? name) (equal? 'diaz (last-name name)))
: (define keep-diaz-friends (make-keeper diaz?))
: (keep-diaz-friends '(pat-diaz jameel-gupta raoul-diaz petra-rabinowitz))
(pat-diaz raoul-diaz)
```

```
(define (make-keeper good-name?)

   ____________________________________________________________ )
```

Name: ______________________________

Question 3 – King of your castle, ruler of your domain (20 points; 30 minutes)

For this problem, assume the Scheme interpreter evaluates all arguments in a numeric expression from left-to-right. E.g., if `bar` and `foo` are undefined, the following expression: `(* foo bar)` will always yield "`Unbound variable: foo`" (and not `bar`) because `foo` will be evaluated first since it is to the left of `bar`.

You are asked to write a procedure named `ruler` that draws the marks on a ruler using the dash "-" character and *returns the number of dashes you've printed overall*. As you probably know, the number of the dashes depends on their location: the half-ruler marks have the most, the quarter ruler marks have half as many, etc. The smallest dashes have only one dash. Examples are on the right edge of this page…

a) Using recursion, write `dashes`, a procedure which takes in a non-negative integer `n`, displays `n` dashes, then a `newline`, and *returns the number of dashes it displayed* (that is, `n`). Two example calls are shown on the right. (8 points)

```
: (dashes 0)

0

: (dashes 5)
-----
5
```

```
(define (dashes n)
  (cond ((= n 0)

          ____________________________________________

          ____________________________________________ )

        (else ____________________________________________

          ____________________________________________ )))
```

b) `dashes` uses EMBEDDED TAIL recursion. (Circle one) (2 pts)

c) Finally, write `ruler`. Feel free to use `expt` to specify how many dashes should be drawn on each line. (E.g., 2^3 = `(expt 2 3)` ➔ `8`, useful for `(ruler 4)` in the example on the right.) **Don't define any helper procedures.** Don't forget the return value! (10 pts)

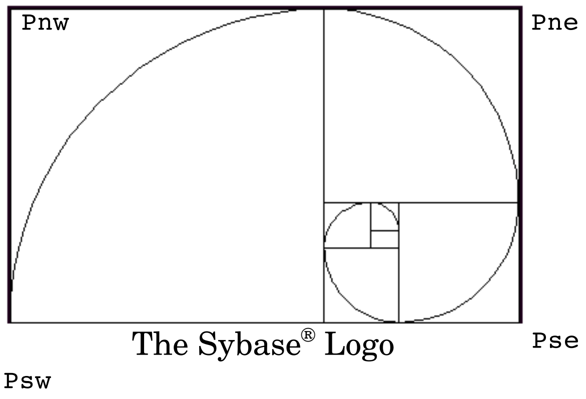
```
(define (ruler n)
  (if (= n 0)

       ____________________________________________

       ____________________________________________

       ____________________________________________

       ____________________________________________ ))
```

```
: (ruler 0)
0

: (ruler 1)
-
1

: (ruler 2)
-
--
-
4

: (ruler 3)
-
--
-
----
-
--
-
12

: (ruler 4)
-
--
-
----
-
--
-
--------
-
--
-
----
-
--
-
32
```

Name: ____________________________________

Question 4 – Those clever graphic designers at Sybase®, Inc. (20 pts; 30 min.)

Those of you who have seen the Sybase® logo (shown on the right) realize it's really a fractal in disguise. Your goal is to draw it. Fortunately, *all* of the hard math will be done for us.



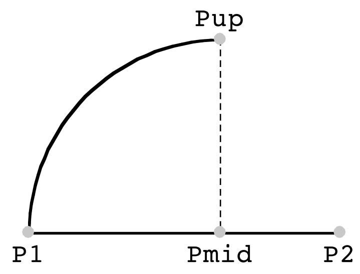
The Sybase® Logo

First of all, take a careful look at the fractal. Note that the bold lines on the outside left, top and right are only drawn once! Let's assume they're taken care of for us and we only have to draw the inner lines and arcs.

Your friend provided a really nice graphic interface for you to use; instead of using `x` and `y` coordinates, you just use *points*. The code, as you'll see, becomes much cleaner.

```
;; Look at the figure below for an example
(draw-lineP P1 P2) ;; Draws a line from point P1 to point P2
(draw-arcP  P1 P2) ;; Draws a 90º arc from P1 to Pup centered around Pmid
(get-Pmid   P1 P2) ;; Return Pmid, a little more than half-way from P1 to P2
(get-Pup    P1 P2) ;; Return Pup, a point "above" the P1-P2 line
                     ;; such that the triangle P1-Pmid-Pup is a right triangle.
```

The arc on the right was created with a call to `(draw-arcP P1 P2)`. The arc angle is 90° and is drawn from `P1` to `Pup` as if one end of a compass were at `Pmid`.



a) Fill in the blanks to complete the `sybase` procedure. Use figure to the right to help you understand the temporary variables `Pmid` and `Pup`. (15 points)

```
(define (sybase P1 P2 n)
  (if (= n 0)
      (draw-lineP P1 P2)
      (let ((Pmid (get-Pmid P1 P2))
            (Pup  (get-Pup  P1 P2)))

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________ )))
```

b) Provide the call to `sybase` that generated the fractal in the Sybase® logo at the top. Assume the corner point labels in the diagram are already defined for us to use. (5 points)

```
(sybase _____ _____ _____ )
```

Name: ______________________________

Question 6 – Fractals on the brain (19 points; 30 min)

You *loooooove* fractals so much that you see them absolutely everywhere you go… even in non-graphic functions like the following:

```
(define (fun n)
  (if (= n 0)
      '()
      (cons (fun (- n 1))
            (fun (- n 1)))))
```

a) Given `fun` as defined above, what does Scheme return if you type: `(fun 1)`? (4 pts)

b) Draw the box-and-pointer diagram for `(fun 2)`. (5 points)

c) How many calls to `cons` will there be for `(fun 5)`? (5 points)

d) In Question 4, you might have noticed you were using data structures (namely, *points*) without even being told how they were implemented! **In one word**, how were you able to do this? Hint – this is one BIG IDEA of the course. (5 points)

You're done! Have a great summer break!!