University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2000 Instructor: Dan Garcia 2000-12-15

CS 3 Final Exam

Last name____________________________ First name ____________________________

SID Number__________________________ TAs name ____________________________

(Sorry to ask this next question, but with almost 375 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

- Please 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.
- Do <u>every</u> question, as we will not be dropping your lowest-scoring one.
- You have three hours, so relax. We have indicated how long you should take for each question at the top of every page.
- Good skill!

Grading Results

Question	*Max. Points*	*Points Given*
1	**20**	
2	**20**	
3	**20**	
4	**20**	
5	**20**	
6	**20**	
7	**20**	
Total	**140**	

Name: ______________________________

Question 1 – I'm drawing a blank… (20 points, 2 points each; 20 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) `(cdr (cons '() '()) )` ➔ ______________________________.

b) `(map reverse '((a) (b) (c)) )` ➔ ______________________________.

c) `(map – '(4 5 6) 3)` ➔ ______________________________.

d) `(equal? (quote (1)) (list (1)) )` ➔ ______________________________.

```
;; For parts e-g
(define (mystery L)
     (cond ((null? L) #t)
           ((list? (car L)) #f)
           (else (mystery (cdr L)))))
```

e) `mystery` is a type of ______________________________ recursion.

f) `(mystery '(a (b) c))` ➔ ______________________________.

g) A good name for `mystery` would be ______________________________.

```
;; For parts h-i
;; wackawacka is not defined anywhere
(define (broken L)
   (wackawacka L))
```

h) Does entering the above definition into scheme result in an error? __________.

i) `(if 'f (broken (/ 1 0)) 3)` ➔ ______________________________.

j) ~~The Turing test is a test of ______________________________.~~

Name: ________________________________

Question 2 – Ready, set, code! (20 points; 25 minutes)

Assume for this problem that sets and lists only contain numbers. You want to write a function `pretty-print` that prints either a set or a linear list to the user:

```
(define (pretty-print set-or-list)
  (cond ((set? set-or-list)
         (display "Your set is: ")  (display (set-contents set-or-list)))
        (else
         (display "Your list is: ") (display set-or-list))))

(define (set-contents set) set)
```

The problem is how do we write the `set?` predicate? Lists and sets look the same! Therefore, we are going to create a new *representation* for sets (called a `new-set`). While we are doing that, we might as well keep track of the set's *cardinality*, or number of elements in the set. The new representation will be a three-element list, whose first element is the symbol `set`, the second element is the cardinality, & the third element is the set's contents. I.e., `new-set = (set` *`cardinality contents`*`)`. Thus, a set containing the numbers `1`, `4`, `9`, `16`, and `25` could be represented as `(set 5 (16 4 9 25 1))`. `*null-new-set*` is defined as `(set 0 ())`. We also need to redefine `set-contents` as follows:`(define (set-contents new-set) (third new-set))`

We need to write `set?`, and realize that since lists can only contain numbers, all we have to do is to test is if the first element is the symbol `set`. Our first attempt is the following: `(define (set? new-set-or-list) (= (first new-set-or-list) set))`

a) Notice that `set?` has at least one error. Write it correctly below. (6 points)

```
(define (set? new-set-or-list) ;; Returns #t for new-sets and #f for lists
```

b) Since the format of sets have changed, we now need to rewrite `new-set-adjoin`. Feel free to use any of the `new-set-` functions on this page. (14 points)

```
;; Here are some more new-set functions
(define (new-set-null? new-set) (equal? new-set *null-new-set*))
(define (new-set-member elt new-set) (member elt (set-contents new-set)))
(define (new-set-length new-set) (second new-set))
```

```
(define (new-set-adjoin elt new-set)
```

Name: ______________________________

Question 3 – set in tree's clothing (20 points; 20 minutes)

Now something else has cropped up. Recall the general definitions for trees:

```
(define (root tree)  (first  tree))
(define (left tree)  (second tree))
(define (right tree) (third  tree))
(define (leaf? tree) (atom?  tree))
(define (make-tree root left right)
   (list root left right))
```

A tree is an atom or a list with three items in it, the root and the two subtrees. We saw expression trees and value trees, which were two particular examples of trees, but not necessarily all the valid trees that exist. Since new-sets are now lists of three items, they could possibly be confused with trees.

a) For this part, assume that a new-set can contain only positive integers (I.e., 1, 2, 3, etc.). Write an expression using only `new-set-adjoin` and `*null-new-set*` that returns a new set which is also a valid tree. Use values that insure that it will have the smallest sum of its set elements. Warning: be sure to re-read your answer several times to make sure it's correct. (10 points)

b) Draw the corresponding tree. (10 points)

Name: ______________________________

Question 4 – Dick Trace-me (20 points; 25 minutes)

Someone writes the strange program `foo`:

```
(define (foo n)
  (cond ((= n 0) 'done)
        (else
         (display (list 'l n))          ;; left
         (foo (- n 1))
         (display (list 'm n))          ;; middle
         (foo (- n 1))
         (display (list 'r n))          ;; right
         (newline)
         'this-was-hard)))
```

a) A call to `(foo 1)` displays a line & returns a value as shown below (we left the return value blank). Fill in the blank with the return value. (4 points)

```
: (foo 1)
(l 1)(m 1)(r 1)
```

____

b) What is displayed and returned when you call `(foo 2)`? (12 points)

```
: (foo 2)
```

Note above that `(foo 1)` displayed 3 pairs: `(l 1)`, `(m 1)` and `(r 1)`.

c) How many pairs will be displayed for `(foo 3)`? (4 points) ____________________

Name: ____________________________________

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

If you followed our suggested pace, you should have 90 min left and be half done.

Name: ________________________________

Question 5 – Did you accumulate CS3 knowledge? (20 points; 30 min.)

For parts (a) & (b), you may not need all the blank lines provided, and that's ok.

a) Provide a call to the HOF `all?` (that returns true if all of the elements in the second argument satisfy the first argument predicate function) which will return `#t` if all of the numbers in the huge list of numbers `*big-number-list*` are greater than 3. E.g., if `*big-number-list*` contained `(7 9 5)`, your call should return `#t`. Do not use any auxiliary function. (5 points)

```
(all?

     ______________________________________________

     ______________________________________________

     *big-number-list*)
```

Note - we don't think you've seen the predicate all? this semester - but the description of all? above should give you most of what you need to solve the problem.

b) You have seen how powerful `accumulate` is; it can be used to sort or find the smallest element in a list. You believe it is underutilized, and want to prove that (like a Swiss-army knife) there are many more uses for it. Fill in the blanks in `my-count-if` which, like `count-if`, will count the number of elements in the linear list `L` that satisfy `pred?`. You may assume L has at least two elements. Do not call any auxiliary functions. *We consider this a hard question*; you are more than welcome to skip this part for now and come back to it later in the exam. (15 points)

```
(define (my-count-if pred? L)
  (accumulate

     ______________________________________________

     ______________________________________________

     ______________________________________________

     ______________________________________________

     (cons 0 L) ))
```

You may need to change the code that is provided in this problem.

Name: ________________________________

Question 6 – Listen to what the flower people say (20 points; 30 min.)

You come up with an idea for a cool fractal flower. The idea is to start with a stem (as in Figure 1) which is a line from the bottom center of the window to the middle. Then make a left and right turn and recurse. Figure 3 shows what happens when you do this a couple of times. We've provided `draw-half-line`:

```
;; Draw a line from (x1,y1) halfway to (x2,y2) as in Figure 4
(define (draw-half-line x1 y1 x2 y2)
  (position pen x1 y1)
```

(draw-line x1 y1 (/ (+ x1 x2) 2) (/ (+ y1 y2) 2)))
;; how we draw a line in 2008 is a little different than in 2000

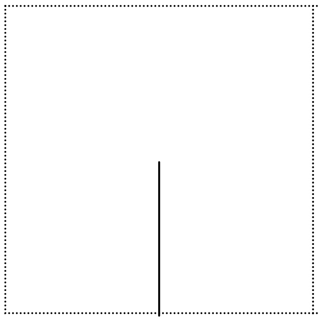
n = 0

Figure 1

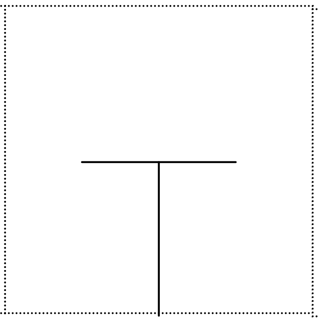
n = 1

Figure 2

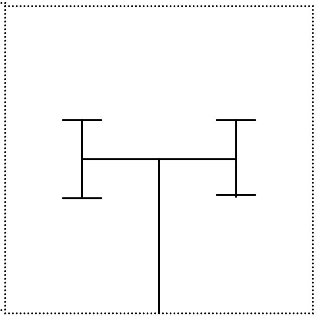
n = ?

Figure 3

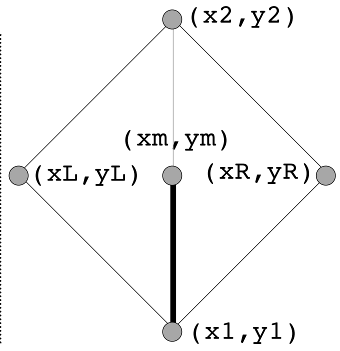

Figure 4

a) Fill in the blanks to complete the `flower` procedure below. Use Figure 4 to help you understand the temporary variables `xm`, `ym`, `xL`, `yL`, `xR` and `yR`. (15 points)

```
(define (flower x1 y1 x2 y2 n)
  (if (= n 0)
      (draw-half-line x1 y1 x2 y2)
      (let ((xm (/ (+ x1 x2) 2))
            (ym (/ (+ y1 y2) 2))
            (xL (/ (- (+ x2 x1 y1) y2) 2))  ;; Do NOT worry about how we
            (yL (/ (- (+ x2 y1 y2) x1) 2))  ;; calculated xL,yL,xR or yR
            (xR (/ (- (+ x2 x1 y2) y1) 2))  ;;
            (yR (/ (- (+ x1 y1 y2) x2) 2))) ;; Simply look at Figure 4

          ________________________________________________________________

          ________________________________________________________________

          ________________________________________________________________ )))

________________
```

b) What was the value of `n` that generated Figure 3? (3 points) __________

c) Modify Figure 3 to show the result of the next generation of `flower` (i.e., with `n` one larger than the correct answer to part (b) above). (2 points)

Name: ________________________________

Question 7 – Getting back at mom (20 points; 30 minutes)

Computer scientists (and most of our moms) always seem interested in `sort`ing things (like lists) and returning them to their proper place. You want to rebel and `unsort` things (like throwing all your clothes on the floor). Someone suggests writing `unsort`, a procedure which takes a list of unique elements and returns the same list with all of its elements rearranged in a random order. E.g.,

```
: (unsort '(a b c d)) → (b d c a)
: (unsort '(a b c d)) → (d b a c)
```

a) Fill in the blanks to complete the `unsort` procedure below. You may not use any auxiliary functions, & your solution may not exceed two lines. (10 points)

```
(define (unsort L)
  (if (null? L)      ;; If L is null, just return it
       L

       _____________________________________________________

       _____________________________________________________ ))
```

b) Your "friend" from Stanford types the following into your scheme interpreter: `:`

```
(define let  'stanford-is-great)
: (define let* 'we-won-the-big-game)
```

…which has the effect of redefining (and rendering unusable) `let` and `let*`. You are now asked rewrite `unsort` given that you cannot use these special forms. Fill in the blanks to complete the `unsort` procedure below. The solution does not require any auxiliary functions. However, if you do choose to use an auxiliary function, you will lose 5 points. Do not use `set!` or a `define` within a `define`. *We consider this the hardest question on the exam.* (10 points)

```
(define (unsort L)
  (if (null? L)     ;; If L is null, just return it
       L

       _____________________________________________________

       _____________________________________________________

       _____________________________________________________ ))

;; If you need a small auxiliary function, write it below.
```

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