

You must turn in Part 1 before you use your computer for anything. During the entire exam you may not use email, IM, cell phone, PDA, headphones, ear buds, or any other communication device or software. Efficiency and elegance will not affect your scores (except for the bonus problem), provided that we can understand your code.

On both parts, you may assume for any procedure you are asked to write that all input arguments will be of the correct types; you do not need to write code to check for illegal input data. Unless specified for a specific problem, mutation is not allowed in code that you write.

Part 1, written. Allowed resources: pencil or pen, and an eraser.

Time for entire exam: 150 minutes.

Suggestion: Spend no more than 50 minutes on this part, so that you have a lot of time for the computer part. Ideally, finish Part 1 in 40 minutes or less. But you can vary this time based on how hard you think it will be for you to do the computer part.

Problem	Possible	Earned
1 - x	2	
1 - y	4	
1 - print	3	
2	6	
3	14	
4	4	
5	7	
bonus	(3)	
Total	40	

Procedures:

Arithmetic: +, -, *, /, modulo, max, min, =, <, ≤, >, ≥

Predicates and logic: not, eq?, equal?, null?, zero?, procedure?, positive?, negative?, pair?, list?, even?, odd?, number?, symbol?, integer?, member

Lists: cons, list, append, length, reverse, reverse!, set-car!, set-cdr!, car, cdr, cadr, caddr, etc, list-recur, snlist-recur.

Functional: map, apply, andmap, ormap

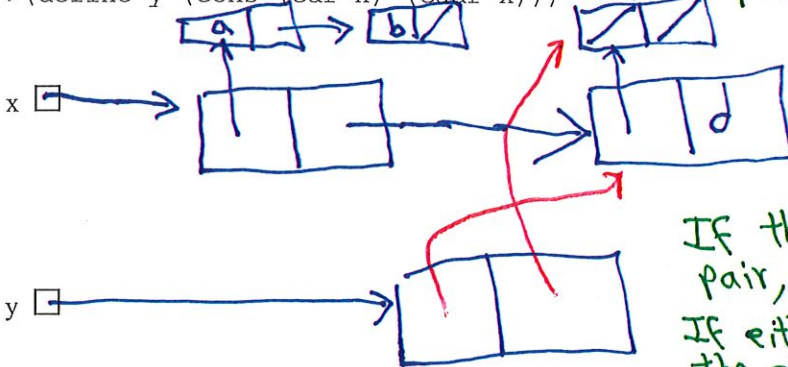
Other: vector, vector-set!, vector-ref

Syntax:

lambda, including (lambda x ...) and (lambda (x y . z) ...) define, if, cond, and, or, let, let*, letrec, named let, begin, set!, (no mutation is allowed in code that you write unless a problem specifically says that you can).

1. (2 points for x, and 4 for y) (a) Draw a box-and-pointer diagram that shows Scheme's internal representations of x, and y, assuming that the code is executed in the order given.

>(define x '((a b) (()) . d))
>(define y (cons (cdr x) (cadr x)))



0 if not mostly correct.
1 if mostly correct.
2 if completely correct.

If they create more than one new pair, -2.
If either the car of the new pair or the cdr of the new pair points to the wrong thing, -1 each.

(3 points) If we then have Scheme evaluate y, what will be printed? ((()) . d) (())

Full credit for correct answer.

2 points for incorrect answer that matches the student's y diagram

1 point for answer that almost matches y diagram.

2. (6 points) In the homework, you wrote `curry2`. Now write `uncurry2`. `uncurry2` takes as its argument a curried procedure that takes its two arguments one at a time, then `uncurry2` returns a procedure that takes both arguments at once. Your parenthesis placement is critical here.

Examples: Things in bold are what the user enters.

```
> (define make-adder
  (lambda (n)
    (lambda (m)
      (+ m n))))
> (define normal-add (uncurry2 make-adder))
> (normal-add 4 5)
9
> (let* ([curry2 (lambda (f)
                   (lambda (x)
                     (lambda (y)
                       (f x y)))]
         [plus-c (curry2 +)]
         [plus-unc (uncurry2 plus-c)])
  (plus-unc 4 5))
9
```

Grading: All or nothing. Of course the arguments can have different names, and line breaks can be different, but the parentheses must be exactly correct.

```
(define uncurry2
  (lambda (curried-f)
    (lambda (x y)
      ((curried-f x) y))))
```

3. (14 points) The code below begins with my `snlist-recur` code, then the execution of seven expressions that use that procedure. For each of those seven expressions, show the output. For each individual part, credit will be given only if your answer is precisely correct (which includes that the parentheses are precisely correct).

[Hint: The seven answers are all different]

```
(define snlist-recur
  (lambda (seed item-proc list-proc)
    (letrec ([helper
              (lambda (ls)
                (if (null? ls)
                    seed
                    (let ([c (car ls)])
                      (if (or (pair? c) (null? c))
                          (list-proc (helper c) (helper (cdr ls)))
                          (item-proc c (helper (cdr ls)))))))]
      helper)))
```

2 points or none for each part. The parentheses placement must be exactly correct.

<code>((snlist-recur '() cons cons) '(a (b)))</code>	_____ <code>(a (b))</code> _____
<code>((snlist-recur '() cons list) '(a (b)))</code>	_____ <code>(a (b) ())</code> _____
<code>((snlist-recur '() cons append) '(a (b)))</code>	_____ <code>(a b)</code> _____
<code>((snlist-recur '() list cons) '(a (b)))</code>	_____ <code>(a ((b ())))</code> _____
<code>((snlist-recur '() list list) '(a (b)))</code>	_____ <code>(a ((b ()) ()))</code> _____
<code>((snlist-recur '() list append) '(a (b)))</code>	_____ <code>(a (b ()))</code> _____
<code>((snlist-recur '() append list) '(a (b)))</code>	_____ <code>This one causes an error</code> _____

4. (4 points) For each of the following two procedures, give a better name for it, a name that describes what it really does. In this example, the ? in the name does **not** mean that the procedure is a predicate.

(define ? (lambda (lon) (apply + lon))) Better name for ?: list-sum

(define ?? (lambda (lon) (map + lon))) Better name for ?? list-copy

The names must be something like these. For the second one, give one point if the student says “identity” or something like that.

5. (7 points) Consider the following definition of a Scheme procedure:

```
(define whatsit
  (lambda (n m)
    (let outer ([n n])
      (if (zero? n)
          '()
          (cons (let inner ([m m])
                  (if (zero? m)
                      '()
                      (cons m (inner (- m 1))))))
                (outer (- n 1)))))))
```

Show the output from evaluating each of the following expressions (be careful about parentheses in your answers):

Points	Expression	Output
3	(whatsit 2 3)	((3 2 1) (3 2 1))
1	(reverse (whatsit 2 3))	((3 2 1) (3 2 1))
3	(cons (whatsit 1 3) (whatsit 3 1))	(((3 2 1)) (1) (1) (1))

First part: -1 for 3 groups of 2 instead of 2 groups of 2.

- 1 for counting up instead of down or for stopping the lists at 0 instead of wrong
- 1 for too many or too few parens
- 1 for not having repeating numbers

Second part: -1 if it is not the (single level) reverse of the student’s previous answer.

Third part:

- 1 if it’s not consistent with how they answered the first part,
- 2 if they did something that is really list or append (or something else) instead of cons.

6. (bonus – 3 points). What is a better name for the procedure that is defined below? list-sum

```
(define ???
  (lambda (lon)
    (apply apply (list + lon)))))
```

Answer must be something like list-sum. If answer is incorrect, just cross it out here, and do not write anything in the table on page 1.

If it is correct, write 3 there.