CS3L Summer 2011 Final Exam, Part 2

You may leave when finished; or, you must stop promptly at 12:20

Name: _____ Login: cs3-_____

First names of the people to your left and right, if any: Left: _____ Right: _____

1. If you have difficulty understanding exactly what a question is asking, **please ask us for clarification as soon as possible**.

2. Assume that your procedures will only be given inputs of the form described in the problem statement. You don't need to anticipate / check for inputs with the wrong number / type of arguments, etc. You must explicitly define any procedures that you use that aren't built into STk, even if you've previously defined them in lab.

3. On problems in which you fill in blanks in our code, **you may not add to or change the existing code, and what you fill into the blanks should be atoms or (at most) very short compound expressions.** You will probably get less credit or no credit if you try to cram in much longer answers, on the grounds that your solution is unnecessarily complex.

4. There are no restrictions on what you may use to solve coding problems, except that you may not use `set!`, `set-car!`, and `set-cdr!`, since we never studied them. (They would probably not be particularly helpful anyway on this exam.)

#	Question name	Possible	Your score
1	What Will Scheme Print?	9	
2	Miscellany	17	
3	Half Off	9	
4	Fourbearance	8	
5	Film No-R	8	
6	Treetotaling	8	
7	Telegraph Troubles	16	
	TOTAL	**75**	

Question 1: What Will Scheme Print?

Suppose that you type each of the following into STk. What will STk print in each case? (Treat the problems as independent, with STk starting in its default state each time.) If STk would print something like `#[closure arglist=...]`, just write **procedure**. If you would get a crash / error message (for any reason), just write **error**, unless the error would be a segfault or infinite hang due to "running forever", in which case just write **run-forever**.

```
STk> (keep (not odd?) '(1 23 456))          _____________________

STk> (cddr '(dance dance revolution))       _____________________

STk> ((lambda (x) (x)) (lambda () 'y))      _____________________

STk> (append (list 1 '()) (cons 2 '()))     _____________________

STk> (accumulate append '()
       '((one) ((two))))                    _____________________

STk> (vector-set!
       (vector-set! #(0 1) 0 1) 1 0)        _____________________

STk> (define (z) (z))
z
STk> (and (or #t (z))
          (not (z)))                        _____________________

STk> (define (peculiar a)
       (let ((b a))
         b)
       b)
peculiar
STk> (peculiar 'hmm)                        _____________________

STk> (define (new-even? n) (not (new-odd? n)))
new-even?
STk> (define (new-odd? n) (not (new-even? n)))
new-odd?
STk> (new-odd? 2011)                        _____________________
```

Question 2: Miscellany

I. *(4 pts)* Answers are either true or false for very good (not incidental, obscure, or sneaky) reasons.

True False 1. Each of the four built-in sentence HOFs (`every`, `keep`, `accumulate`, `repeated`) takes a procedure of one argument as its first argument.

True False 2. A list can be defined recursively as a pair in which the `cdr` is either the empty list or another list (as per this same definition).

True False 3. A vector and all of its entries (e.g., numbers, Booleans, other vectors) are guaranteed to all reside together in the same contiguous block of memory, with nothing else in between.

True False 4. The amount of running time needed to find the number of occurrences of a particular value in a vector (e.g., all the 2s in `'#(2 two #t 2 (2))` has (little to) no dependence on the length of that vector.

II. *(2 pts)* Circle any/all of the following that are examples of **selectors** for the indicated data types.

`filter` for sentences `item` for words `leaf?` for Trees `vector-set!` for vectors

III. *(4 pts)* For each of questions 1-4, choose a **single answer** from among the following. A given answer may be used once, more than once, or not at all.

A. `car` B. `'car` C. `(car)` D. `'(car)` E. none of A-D F. more than one of A-D

1. Typing which of these at the STk prompt is equivalent to typing `(quote car)`?

2. Which of these, if typed in at the STk prompt, would return a list of length 1?

3. Which of these would cause an error if typed in at the STk prompt?

4. Which of these, if typed in at the STk prompt, would cause Scheme to print:
`car`

Question 2, continued

IV. *(4 pts)* Here is a good implementation of `append`:

```
(define (new-append ls1 ls2)
  (if (null? ls1)
      ls2
      (cons (car ls1) (new-append (cdr ls1) ls2))))
```

Assume that the running time of `append` is directly linearly proportional to the number of calls to `cons`, `car` and `cdr`. (Also assume that `cons`, `car`, and `cdr` take the same amount of time for any inputs.) Circle one of the three options from each bolded group below.

Doubling `ls1`'s length would **(not significantly change / roughly double / roughly quadruple)** the running time of `new-append`.

Doubling `ls2`'s length would **(not significantly change / roughly double / roughly quadruple)** the running time of `new-append`.

V. *(3 pts)* You have a perfectly balanced / "triangular-looking" binary search tree (BST) called `pyramid`, in which every branch node has either two branch nodes or two leaf nodes as children, and leaf nodes appear only at the deepest depth of the tree. `pyramid` has one node at depth 0 (the root), two nodes at depth 1, four nodes at depth 2, eight nodes at depth 3, and sixteen (leaf) nodes at depth 4. `pyramid` contains only numbers as data, and no numbers are duplicated.

Fill in the blanks in the expression below so that it will return the **largest number in `pyramid`**. You should not need to remember exactly how we implemented binary trees as lists; use `datum`, `left-child`, `right-child`, `leaf?`, `make-binary-tree`, etc. instead of their `car`, etc. equivalents.

```
(_________________

   ((repeated ___________________ ___________________)

    pyramid))
```

Question 3: Half Off

Suppose that you're trying to implement a procedure `left-half`, which is supposed to take a word and return its left half (i.e. the first *n*/2 characters, if the word is *n* characters long; **you may assume that *n* will always be even**.) But the procedure isn't working!

```
(define (left-half word)

  (let ((halfsize (/ (length word) 2)))

    (define (helper rest)

      (if (< (length rest) halfsize)

          '()

          (word (first rest) (left-half (bf rest)))))

    (helper word)))
```

1. You type `(trace left-half)` into STk, but the `trace` output isn't very useful. Then you type `(trace helper)` into STk, but you get an error message. Why? **Choose one.**

A. `helper` has been defined as a special form, and special forms can't be `trace`d.
B. In Scheme, there is no way to `trace` procedures defined inside of other procedures.
C. No binding for `helper` exists in the global environment.
D. Only one procedure may be `trace`d at a time; you must `(untrace left-half)` first.
E. `trace`s within local environments still happen, but the output is not displayed to the user.

2. Fix all bugs in the above procedure so that it will work properly. **You may only fix a bug by crossing out a small part of the code and replacing it with something else.** Do not change large segments of the code or add in new lines. There are no bugs pertaining to parentheses.

3. Does the recursive procedure `left-half` generate a **recursive** or **iterative** process?

Question 4: Fourbearance

In some Asian cultures, the number 4 is unlucky. Write a procedure `(no4 proc)` that takes a procedure `proc` **(that takes one argument)** and returns a new procedure that has the exact same behavior as `proc`, except that when it's given an input for which `proc` would normally return the number 4, the number 8 is returned instead. (`proc` will not necessarily always return a number.) You should not replace any other possible return values like `-4`, `44`, `4chan`, `four`, `(4)`, or `iv`... just the number `4` itself. Moreover, your implementation **must not call `proc` more than once** -- what if `proc` is `random`, or involves mutation, for example?

Examples:

```
STk> (no4 abs)
[#closure arglist...]

STk> ((no4 (lambda (x) 4)) 6)
8

STk> ((no4 word?) 'four)
#t
```

```
STk> (define new-square (no4 square))
new-square
STk> (new-square 2)
8
STk> (new-square 4)
16
```

2. Assuming that we've already typed `(define (square x) (* x x))`, how many **different** values could `(square ((no4 random) 10))` possibly return if you typed it in arbitrarily many times? (You don't need to enumerate these possibilities... just say how many possibilities there are.)

_____

Question 5: Film No-R

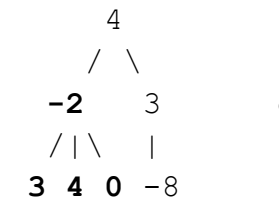
Ian worked in a movie theater once, and had to pick out the big plastic letters for the marquee (the movie title display on the outside of the theater). Sometimes the most common letters ran out. `film-no-r` *is a pun on "film noir", in case you were wondering...*

Write a procedure `film-no-r` that takes a list of movie titles (each of which is a sentence) and returns the same list, but with every instance of the letter r removed from every word.

```
STk> (film-no-r
       '((friends with benefits) (x-men first class) (harry potter)))
((fiends with benefits) (x-men fist class) (hay potte))
```

Question 6: Treetotaling

Write a procedure `(best-total t)` that takes a Tree `t` in which every datum is a number, and returns the maximum value attainable from adding up all of the numbers in some subtree of `t`. *Any* node (branch or leaf) of `t`, including the root node, counts as a subtree; a subtree is a node and *all* of that node's descendants. (So for a Tree with *n* nodes, there are *n* subtrees that could have the best total.) Use `datum`, `children`, `leaf?`, etc. instead of `car`, `cadr`, etc.; you may use these without defining them, and you shouldn't need to remember exactly how we implemented Trees. You may assume that the Tree will contain at least one node.



For example, for the Tree to the left, best-total should return 5. (This is the result of adding up all of the values in the bolded subtree. All of the other subtrees produce smaller totals: 4, -5, 3, 4, 0, -8.

You must add *all* of the numbers in some particular subtree; you couldn't take, e.g, just the 4 and the 3 below it, or the 4 and the -2 and 3 immediately below it, because neither of those would form a full subtree (they would omit some descendants).

Question 7: Telegraph Troubles

In Morse code, each letter in the English alphabet is represented by a series of short sounds (written as dots, but we'll use asterisks because of the special meaning of dots in Scheme) and/or long sounds (written as dashes). Here are all of the English letters that can be represented with at most two dots and/or dashes. **In these problems, we will only deal with these six letters.**

A	E	I	M	N	T
`*-`	`*`	`**`	`--`	`-*`	`-`

1. *(2 points)* Suppose that the association list `morse` is already defined: `(define morse '((* e) (- t) (** i) (*- a) (-* n) (-- m)))`

Fill in the blanks to complete the procedure `(from-morse ml)`, which takes a word `ml` (a single Morse letter, which will always be either `*`, `-`, `**`, `*-`, `-*`, or `--`), and returns the corresponding English letter (E, T, I, A, N, or M, respectively). For example, `(from-morse '-*)` should return `n`.

```
(define (from-morse ml)

  (________________ (________________ ml morse)))
```

To transmit a Morse word, you transmit each Morse letter, leaving a pause (written as a space) between each letter and the next. For example, the word TEATIME would normally be: **`- * *- - ** -- * .`** However, you're receiving messages from an inexperienced Morse code operator who hasn't realized that he needs to put spaces between the letters. For example, he would transmit TEATIME as: **`-**--**--*`** , and since you don't know where the spaces should have gone, this could also be reasonably interpreted as NANETTE: **`-* *- -* * - - *`** or a number of other words that may not actually be English words.

2. *(10 points)* Fill in the blanks to complete the memoized version of `(parse-morse mw)`, which takes a word `mw` consisting only of dots (asterisks) and dashes and returns a sentence of all of the possible translations of `mw` into English letters, given that spaces could be anywhere in `mw`. (The order in which these translations are returned is unimportant.) Example:

```
STk> (parse-morse '-*-*)
(tete ten tae nte nn)
```

Question 7, continued

Reminder: For fill-in-the-blank questions, do not alter the existing code outside of the blanks, and everything you fill into a blank should be rather short, or you're probably overcomplicating things (and may not get credit!)

```
(define (parse-morse w)

 (let ((memovect (make-vector (+ (count w) 1) _____________)))
  (define (helper rest)

   (if (_____________ rest)

       _____________

       (let ((lookup (vector-ref memovect (count rest))))

        (if lookup

            _____________
            (let ((answer
             (_____________
              (every (lambda (x)
               (word (from-morse (first rest)) x))
                     (helper (bf rest)))
              (if (< (count rest) 2)

                  _____________
                  (every (lambda (x)
                   (word (from-morse (word (first rest)

                                                 _____________))
                         x))

                   (helper _____________))))))

             (vector-set! memovect _____________ _____________)
             answer)))))
  (helper w)))
```

Question 7, continued

3. *(2 points)* Our `morse` association list makes it convenient to go from a Morse letter to an English letter, but what if we want to be able to translate in the other direction? Write a procedure `(ass-backwards al)` that takes an association list `al` (any list, not necessarily Morse-related) and returns a new association list in which every two-item sublist has been flipped:

```
STk> (ass-backwards morse)
((e *) (t -) (i **) (a *-) (n -*) (m --))

(define (ass-backwards al)

  (_________________

    (lambda (sublist) (__________________ (cadr sublist)
                                          (car sublist)))
    al))
```

4. *(2 points)* Give a mathematical expression/description for the number of items `n` in the list returned by `(parse-morse mw)` when the number of characters in `mw` is `c`. Your answer should depend on `c`. Make sure your expression/description is specific enough. (Hint: this is *very* similar to at least two problems that you've seen before in lab...)

n = _____