

91.301, Organization of Programming Languages  
Spring 2006, Prof. Yanco

### Sample Quiz 1

Name: \_\_\_\_\_

There are 11 pages in this exam. You may refer to one sheet of handwritten notes, which must be turned in (they will be returned to you). You have 75 minutes to complete this exam. Please read through all of the questions before starting. If you can't do the first part of a problem, you can assume that it's been defined so you can use it in later parts of the problem.

<b>Problem</b>	<b>Possible Points</b>	<b>Your Score</b>
1	9	
2	12	
3	12	
4	8	
5	15	
6	10	
7	12	
8	16	
9	6	
Total	100	

**Problem 1 (9 points)**

What is the result of the evaluation of the final statement in each of the following groups of Scheme expressions? Write “error” if an error would result or “procedure” if a procedure would be returned. Assume that each group is evaluated separately.

```
(define a 1)
(define b 3)
(define c 5)

(let ((a 2)
      (b (+ a 5))
      (c b))
  (+ a b c))
```

```
(define (map lst op)
  (if (null? lst)
      nil
      (cons (op (car lst))
            (map (cdr lst) op))))

(map (lambda (x) (if (even? x) 1 0))
     (list 2 3 4 5 6 7))
```

```
(define (my-func f)
  (lambda (x y) (f (f x y) (f x y))))

((my-func *) 2 3)
```

**Problem 2 (12 points)**

a. Define applicative order evaluation

b. Define normal order evaluation

c. Given the following code,

```
(define (add a b)
  (display " plus ")
  (+ a b))

(add (begin (display " one ")
            1)
     (begin (display " two ")
            2))
```

Which of the following could be printed in Scheme? In normal-order Scheme?

	<b>Scheme</b>	<b>Normal-order Scheme</b>
plus one two		
plus two one		
one plus two		
two plus one		
one two plus		
two one plus		

**Problem 3 (12 points)**

Assume the following Scheme expressions are evaluated sequentially. Fill in the table below, using `eq?`, `eqv?` and `equal?` to compare the items in the first two columns. You may find it useful to draw the box and pointer diagrams for this problem, but you are not required to do so.

```
(define a (list 1 2 3))
```

```
(define b (list 1 2 3))
```

```
(define c (list 4))
```

```
(define d (cons 4 a))
```

```
(define e (append c b))
```

		<code>eq?</code>	<code>eqv?</code>	<code>equal?</code>
<code>a</code>	<code>b</code>			
<code>d</code>	<code>e</code>			
<code>(cdr d)</code>	<code>(cdr e)</code>			
<code>(cdr d)</code>	<code>a</code>			

**Problem 4 (8 points)**

For the two expressions below, write the sequence of cars and cdrs needed to get the number 3 out of each list. You may find it helpful to draw the box-and-pointer diagrams, but you will not be graded on them.

```
(define first-list (cons 1 (cons 2 (list 3 4 5))))
```

```
(define second-list (list (cons 1 2) (cons 3 4) 5))
```

**Problem 5 (15 points)**

Write a procedure that takes two ordered lists (low to high) and returns a list of the merged elements with no duplicates. For example,

```
(merge '(1 3 5 6) '(2 4 6 8 10 12))
```

will return the list

```
(1 2 3 4 5 6 8 10 12)
```

While you can assume that the lists are ordered, do not assume that the lists are equal lengths.

What is the order of growth of the merge procedure in terms of time and space?

Time: \_\_\_\_\_

Space: \_\_\_\_\_

What variable in the function is n dependent upon?

Does merge generate a recursive or an iterative process? \_\_\_\_\_

**Problem 6 (10 points)**

Write a function `apply-twice` that takes a function `f` as its argument and returns a function that takes one argument as input and returns the value that one would obtain if `f` were applied twice to that argument.

For example,

```
((apply-twice square) 2)
```

would return

```
16
```

and

```
((apply-twice (lambda (x) (+ x 2))) 5)
```

would return

```
9
```

**Problem 7 (12 points)**

Using the following procedure for dealing with trees,

```
(define (tree-manip tree init leaf first rest accum)
  (cond ((null? tree) init)
        ((not (pair? tree)) (leaf tree))
        (else (accum
                 (tree-manip (first tree) init leaf first rest accum)
                 (tree-manip (rest tree) init leaf first rest accum))))))
```

write calls to perform the operations described below. Suppose that we provide a test tree,

```
(define test-tree '(1 (2 (3 (4) 5) 6) 7))
```

Write the call to `tree-manip` that will compute the sum of the leaves of `test-tree`, resulting in a return value of 28.

```
(tree-manip _____
             _____
             _____
             _____
             _____
             _____
             _____)
```

Write the call to `tree-manip` that will flatten the `test-tree`, resulting in a return value of `(1 2 3 4 5 6 7)`.

```
(tree-manip _____
             _____
             _____
             _____
             _____
             _____)
```

**Problem 8 (16 points)**

Despite the dot-com bust, Louis Reasoner has decided that he would like to create a web site to help people track their collections. The back end of the web site will be programmed in Scheme. First, we need to create a means for storing information about an item in a person's collection: each item record will contain the name of the item, its estimated value, and its condition.

We define the constructor `make-item` as follows:

```
(define (make-item item-name value condition)
  (list (cons item-name value) condition))
```

Write the selectors `item-name`, `item-value`, and `item-condition`.

To store all of the items in a person's collection we will use the following constructor:

```
(define make-collection-list list)
```

Write the selectors `first-item` and `rest-items`.

**Problem 8 continued**

Write a procedure called `total-value` which takes a collection-list and returns the total value of the collection. Be sure to use the constructors and selectors that you helped define on the previous page.

What is the order of growth of the `total-value` procedure in terms of time and space?

Time: \_\_\_\_\_

Space: \_\_\_\_\_

What variable in the function is  $n$  dependent upon?

Does `total-value` generate a recursive or an iterative process? \_\_\_\_\_

**Problem 9 (6 points)**

Given the following definition of `cons`, define `car` and `cdr`.

```
(define (cons x y)
  (lambda (m)
    (cond ((eq? m 'car) x)
          ((eq? m 'cdr) y)
          (else (error "Unknown message - CONS" m)))))
```