CSE413 Midterm

05 November 2007

Name _______ **Sample solution** __________ Student ID __________

Answer all questions; show your work. You may use:

1. The Scheme language definition.
2. One 8.5 * 11" piece of paper with handwritten notes

Other items, including laptop computers, calculators, cell phones, and other communications devices, are not allowed.

For all Scheme programming problems use recursion and functional operations. No loops; no assignment (`set!` and its relatives).

You may also assume that the data or arguments to all functions are as specified and you do not need to check for errors (e.g., don't add code to deal with non-numbers when the data is specified to be a list of integers, or nested lists if the data is supposed to be a simple list, but do handle these things if the problem specifies it).

Advice You have 50 minutes, **do the easy questions first**, and work quickly!

Total: 100 points.

Question	Max Points	Score
1	14	
2	10	
3	3	
4	12	
5	15	
6	15	
7	15	
8	16	
Total	100	

1. [14 pts.] Suppose we enter the following top-level definitions into a Scheme interpreter.

```
(define a 100)
(define b '(100 62))
(define c 47)
(define (apples a) (odd? a))
(define (pie a) (apples (- 3 a)))
(define book (lambda (a) (> a c)))
```

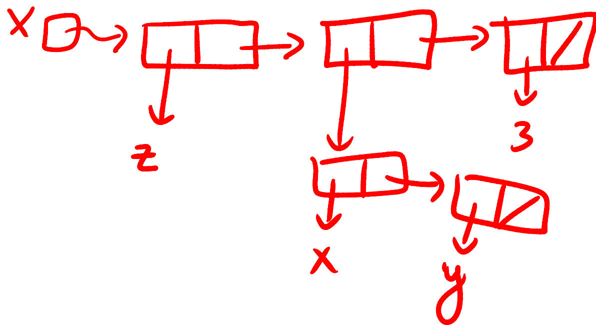
What is the value of each of the following expressions, given that the above definitions are in effect? If evaluating the expression produces an error, explain what is wrong.

```
a) (cons a b)      =>        (100 100 62)

b) (cons b b)      =>        ((100 62) 100 62)

c) (append a b)    =>

   append: expected argument of type <proper list>; given
   100

d) (book 13)       =>        #f

e) (apples 12)     =>        #f

f) (map apples b)        =>  (#f #f)

g) (map pie '(7 8 9))    =>  (#f #t #f)
```

2. [10 pts.] For each of the following sections, draw a boxes-'n-arrows diagrams as requested showing the final result after executing all of the expressions in that section. After each diagram, answer the questions about the printed representations of some of the values defined.

(a)
```
(define x '(z (x y) 3))
(define y (list x (car x)))
(define z (cdr (car y)))
(define w (cons 5 (cdr x)))
```

Draw the boxes and arrows diagram for `x`:

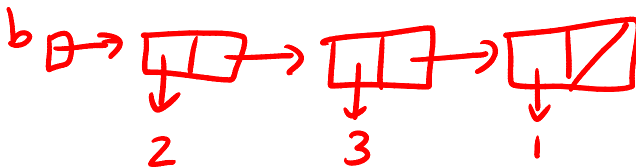


What are the printed values of `x`, `y`, `z`, and `w`?

x = (z (x y) 3)
y = ((z (x y) 3) z)
z = ((x y) 3)
w = (5 (x y) 3)

(b)
```
(define a '(1 2 3))
(define b (append (cdr a) (cons (car a) '())))
```

Draw the boxes and arrows diagram for `b`:



What is the printed values of `a` and `b`?

(1 2 3)
(2 3 1)

3. [3 pts.] Suppose we enter the following definitions:

```
(define x 100)
(define y 7)
```

What is the value of the following expression?

```
(let* ((x (+ x y))
       (y (- x y)))
     (- x y))
```

Answer : **7**

4. [12 pts.] Suppose we enter the following definitions:

```
(define k 5)
(define foo (lambda (n)
              (lambda (w) (+ k n w))))
```

Describe the value of each of the following expressions. If the value is something simple like an integer, just write it down. If the value is a procedure (function closure), describe the closure precisely: what is the procedure code, and what environment bindings are included in the closure? If something is wrong with the expression, describe the error.

(a) `(foo 3)`

A function closure. Code: (lambda (w) (+ k n w))
Bindings: n is bound to 3. (k is not bound to 5 at this time)

(b) `((foo 3) 7)`

15

(c) `((foo 3) k)`

13

(d) `(((foo 3) 7) 2)`

An error. ((foo 3) 7) returns the value 15. What is expected is a procedure that can be applied to the argument 2.

5. [15 pts.] Write a Scheme function `remove-odd` that takes a list of integers as an argument and returns the same list with all of the odd numbers removed. Examples:

```
(remove-odd '(-10 6 5 0 5 -3 21 68)) => (-10 6 0 68)
(remove-odd '(100)) => (100)
(remove-odd '()) => ()
```

```
(define (remove-odd alist)
  (cond ((null? alist) '())
        ((odd? (car alist)) (remove-odd (cdr alist)))
        (else (cons (car alist)
                    (remove-odd (cdr alist))))))
```

6. [15 pts.] Write a Scheme function `findmax` that takes a list of integers as an argument and returns the largest value in the list. It should return false if given the empty list. You may find it useful to write a helper function. Examples:

```
(findmax '(-10 6 25 0 105 -3 21 68)) => 105
(findmax '()) => #f
(findmax '(10 600 25 -5 -3 21 77)) => 600
```

```
(define (findmax-helper alist max)
  (cond ((null? alist) max)   ; reached end of list
        ((> (car alist) max) ; found new max
         (findmax-helper (cdr alist) (car alist)))
        ; car is no bigger than current max
        (else (findmax-helper (cdr alist) max))))

(define (findmax alist)
  (if (null? alist) #f
      (findmax-helper (cdr alist) (car alist))))
```

7. [15 pts.] Write a Scheme function `count` that takes two arguments, an integer and list of integers, and counts how many occurrences there are of the integer in the list. The list could be the null list or a *nested list*. If it is a nested list you must recursively count the number of occurrences in all sub-lists. (Hint: the pair? predicate can be used to test whether something is a non-empty list.) Use = to test for equality of integers. For example:

```
(count 4 '(6 8 4 3 45 4))   => 2
(count 5 '((5 (5 4) 4) (3))) => 2
(count 4 '()) => 0
(count 4 '(3 ((43 4)) (4) (6 9 10 4))) => 3
```

```
(define (count num alist)
  (cond ((null? alist) 0)     ; alist is a null list
        ((pair? (car alist)) ; car is a non-null list
         (+ (count num (car alist))
            (count num (cdr alist))))
        (else                 ; car is an integer
         (if (= num (car alist))
             (+ 1 (count num (cdr alist)))
             (count num (cdr alist))))))
```

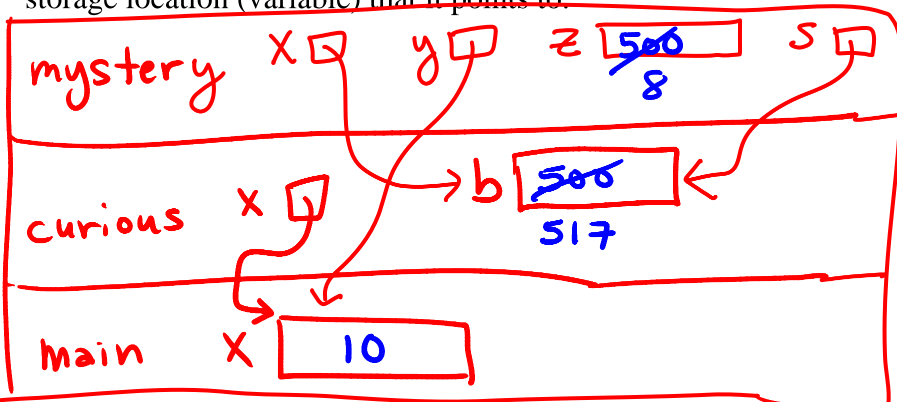
Shorter:

```
(define (count num alist)
  (cond ((eqv? num alist) 1) ; alist is the value
        ((pair? alist) ; alist is a non-null list
          (+ (count num (car alist))
               (count num (cdr alist))))
        (else 0)))
```

8. [16 pts.] Consider the following C program (which compiles and runs without errors):

```
#include <stdio.h>

void mystery(int* x, int* y, int z) {
  int *s = x;
  *x = z + 17;
  z = *y - 2;
  /* (a) draw picture at this point */
  printf("In mystery: *x=%d, *y=%d, z=%d, *s=%d\n", *x, *y, z, *s);
}

void curious(int* x) {
  int b;
  b = 500;
  printf("In curious, before mystery: *x=%d, b=%d\n", *x, b);
  mystery(&b, x, b);
  printf("In curious, after mystery: *x=%d, b=%d\n", *x, b);
}

int main() {
  int x;
  x = 10;
  curious(&x);
  printf("In main: x=%d\n", x);
  return 0;
}
```

(a) Draw a diagram with boxes for each active function showing the situation right when execution reaches the `printf` statement in function `mystery`. Be sure to show the values of all of the variables in each active function. If a variable is a pointer to another variable, indicate its value by drawing an arrow between the variable name and the storage location (variable) that it points to.



(b) What output is produced when this program is executed? If an address is printed out, please make up an address and label the appropriate box in your picture above with that address. (Don't worry about writing out the exact details of each message, but we should be able to tell which values you are printing where.)

In curious, before mystery: *x=10, b=500
In mystery: *x=517, *y=10, z=8, *s=517
In curious, after mystery: *x=10, b=517
In main: x=10