CS3 MIDTERM 1 – WITHOUT RECURSION! Summer 2008

Read this page a[illegible] age yet.

Solution

Name:	
Instructional login (eg, cs3-ab):	
UCWISE login:	
Name of the person sitting to your **left**:	
Name of the person sitting to your **right**:	

(1 pt) Prob 0	
(11 pts) Prob 1	
(2 pts) Prob 2	
(12 pts) Prob 3	
(8 pts) Prob 4	
(3 pts) Prob 5a	
(3 pts) 5b	
(3 pts) Prob 6	
(8 pts) Prob 7	
Raw Total (out of 51) **Scaled Total (24)**	

You have 120 minutes to finish this test. Your exam should contain 8 problems (numbered 0-7) on 9 total pages.

This is an open-book test. You may consult any books, notes, or other paper-based inanimate objects available to you. Read the problems carefully. If you find it hard to understand a problem, ask us to explain it.

Restrict yourself to Scheme constructs covered in chapters 3-6 and the *Difference Between Dates* case study. You can always use helper procedures and procedures from other questions you've answered, unless the question specifically disallows it.

Please write your answers in the spaces provided in the test; if you need to use the back of a page make sure to clearly tell us so on the front of the page.

Partial credit will be awarded where we can, so do try to answer each question.

Good Luck!

Problem 0. (1 point)

Put your login name on the top of each page.
Also make sure you have provided the information requested on the first page.

Problem 1. Some Warms Ups (11 points)

Write the result of evaluating the Scheme expression that comes before the ➔. If the Scheme expression will result in an error, write *ERROR* in the blank and describe the error.

1	`(count (butlast '(cat dog)))` ➔ **1**
2	`(or 'false (equal? (+ 3 5) 8))` ➔ **false**
3	`(or (and (odd? 3) (odd? 2)) (and (< 2 3) (> 4 3)))` ➔ **#t**
4	`(sentence (word cat dog) 'elephant)` ➔ **Error – unbound variable**
5	`(+ '(square 2) '(square 3))` ➔ **Error – not a number**
6	`(first (last (first (last 'elephant))))` ➔ **t**
7	`(first (butfirst '(elephant zebra))` ➔ **zebra**
8	`(last (quote 'elephant))` ➔ **elephant**

9	`; assume square has been defined below.`
	`(define (square x)`
	`  (* x x))`
	
	`(square '(10))` ➔ **Error – not a number**
	
	`(square (square 4) (square 2))` ➔**Error - too many arguments**

10	`(member? 'ab '(abcde))` ➔ **#f**

Problem 2. Get Those Letters! (2 points)

For each part below, define the bolded procedure. When that procedure is applied to the arguments shown, it should produce the same output as the example. You can't use any quoted characters in your solution.

Part A: (1 point)

```
(procA '(A B C) '(D E F)) ➔  (B C D E AF)
```

```
(define (procA sent1 sent2)
     (sentence
          (butfirst sent1)
          (butlast sent2)
          (word (first sent1) (last sent2))))
```

Part B: (1 point)

```
(procB '(A B C) '(D E F)) ➔  BE
```

```
(define (procB sent1 sent2)
     (word
          (first (butfirst sent1))
          (first (butfirst sent2))))
```

Or

```
(define (second sent)
     (first (butfirst sent)))

(define (procB sent1 sent2)
     (word
          (second sent1)
          (second sent2)))
```

Problem 3. Getting Ready for the Half Marathon! (12 points)

Colleen is training for a half marathon and she wants to figure how long it will take her.

Write a procedure `half-marathon-time` that takes Colleen's time **to run one mile** as an argument and returns her time **to run 13.1 miles.**

The format of the time is a sentence in the order: hours minutes seconds.

For example

1 hour 2 minutes and 3 seconds would be represented as ➔ '(1 2 3).
3 hours 15 minutes and 18 seconds would be represented as ➔ '(3 15 18)
0 hours 10 minutes and 30 seconds would be represented as ➔ '(0 10 30)

Example call:

```
(half-marathon-time      '(0 6 0) ) ➔ '(1 18 36)
(half-marathon-time    '(0 11 15) ) ➔ '(2 27 22)
```

Hints:

- Use Helper Procedures!
- If you write a helper procedure include a comment about what it does.
- You can get partial credit for successful helper procedures.
- You might find `quotient` or `remainder` helpful.

(Question 3 – continued)

```
(define (convert-to-seconds time)
     (+
          (last time)
          (* 60 (first (butfirst time)))
          (* 3600 (first time))))


(define (convert-from-seconds seconds)
     (sentence
          (num-hours seconds)
          (num-minutes seconds)
          (num-seconds seconds)))

(define (num-hours seconds)
     (quotient seconds 3600))


(define (num-minutes seconds)
     (remainder (quotient seconds 60) 60))

(define (num-seconds seconds)
     (remainder seconds 60))

(define (half-marathon-time mile-time)
     (convert-from-seconds
          (*
               13.1
               (convert-to-seconds mile-time))))
```

Problem 4. We want a rhyme! (8 points)

Write a procedure `check-rhyme` that takes in two sentences and reports how well they rhyme. We have pretty low standards for rhyming.

- If the last two letters of each sentences are the same return 'good-rhyme
- If the last letter of each sentence is the same return 'bad-rhyme
- If the last two letters of each sentence don't match at all return 'no-rhyme

You can assume that `check-rhyme` will be called with two sentences, that each have at least one word.

```
(define (last-two sent)
      (word
            (last (butlast (last sent)))
            (last (last sent))))

(define (check-rhyme sent1 sent2)
      (cond
            ((equal? (last-two sent1) (last-two sent2)) 'good-rhyme)
            ((equal? (last sent1) (last sent2)) 'bad-rhyme)
            (else 'no-rhyme)))
```

Question 5: Mystery! (6 points)

```
(define (mystery a b c)
  (cond
    ((and (odd? a) (odd? b)) (not (odd? c)))
    ((or  (odd? a) (odd? b)) (odd? c))
    (else #f) ) )
```

Part A: (3 points)

Fill in the blanks below to create a call to mystery that returns the value shown.

(mystery 1 _____) ➔ #t

(mystery 1 1 2) ➔ #t
or
(mystery 1 2 1) ➔ #t

(mystery 2 _____) ➔ #t

(mystery 2 1 1) ➔ #t

(mystery _____ 1) ➔ #f

(mystery 2 2 1) ➔ #f

Part B: (3 points)

<u>In twelve words or less</u>, complete the comment that tells what the `mystery` function returns.

```
;; Given integers a, b, and c, mystery returns true if
```

...exactly two of the arguments are odd...

```
;; and returns false otherwise.
```

Question 6: (3 points)

Rewrite the following procedure without using `if` or `cond`

```
(define (tester a b)
        (if (equal? a b)
                (if  (odd? a)
                        #t
                        #f)
                #f))
```

```
(define (tester a b)
      (and
            (equal? a b)
            (odd? a)))
```

Question 7: PIZZA! PIZZA! (8 points)

At Pizza Hut the pizza cost depends upon the size of the pizza plus $1 for each topping.

- A small pizza costs $5
- A medium pizza costs $10
- A large pizza costs $15
- Additional toppings cost $1 each
- But, there is an extra cost if you want tofu on your pizza. If one of the toppings is tofu that topping costs $2

Write the procedure `pizza-cost` that takes in a sentence representing one pizza order and returns the cost in dollars.

The format of the pizza order will be a sentence with the size of the pizza followed by each of the requested toppings. Each topping will only appear once.

Example calls:

```
(pizza-cost '(medium olive mushrooms)) ➔  12
(pizza-cost '(small)) ➔  5
(pizza-cost '(small tofu)) ➔  7
(pizza-cost '(large garlic onion chicken sausage pepperoni tomato)) ➔  21
```

```
(define (pizza-cost order)
     (+
          (pizza-size-cost (first order))
          (topping-cost order)))

(define (pizza-size-cost pizza-size)
     (cond
          ((equal? pizza-size 'large) 15)
          ((equal? pizza-size 'medium) 10)
          (else 5))

(define (topping-cost order)
     (+
          (*
               (- (count order) 1) ;; count – 1 = total # of
          toppings
               1) ;; $1 is the cost per topping
          (if
               (member? 'tofu order)
               1  ;; extra dollar for tofu
               0)))
```