CS3 Fall 2005 Midterm 1 Review Solutions

Quickies

1. `(word? 25)` → `#t`
2. `(first (bf ''#t))` → `#t`
3. `(first (first (bf ''#t)))` → `ERROR`
4. `(member? 'abc 'xabcyz)` → `ERROR`
5. `(and 1 3 5)` → `5`
6. `(or 4 (/ 5 0))` → `4`

Question 1

Rewrite the following procedure using a `cond` instead of the `if`.

```
(define (sign number)
   (if (< number 0)
       'zero
       'positive))
```

Solution:

```
(define (sign number)
   (cond ((< number 0) ‘zero)
         (else ‘positive) ) )
```

Question 2

Given:
```
(define (mystery n)
        (if (> n 5)
            #f
            #t) )
```

1. Can mystery rewritten to be the following? Explain why or why not.

```
(define (mystery n)
   (or (and (> n 5) #f) #t))
```

Solution:

Try `(mystery 10)`. This test call is supposed to return `#f`, but the rewritten procedure would have returned `#t` instead because or returns the first true value it evaluates.

2. Rewrite this procedure so that the body is just one line (and I don't mean put all of `if` into one line).

Solution:

```
(define (mystery n)
   (not (> n 5)) )
```

Question 3

This is similar to the `sum-in-interval` procedure you had written in lab. The procedure `sum-two-intvl` sums together two intervals instead of one. For instance, `(sum-two-intvl '(1 4 7 9))` would return 34 as a result of summing from 1 to 4 and from 7 to 9, which is basically `(+ 1 2 3 4 7 8 9)`.

Solution:

```
(define (sum-two-intvl intvl)
```

```
(cond ((empty? intvl) 0)
      ((> (first intvl) (first (bf intvl)))
       (sum-two-intvl (bf (bf intvl))))
      (else (+ (first intvl)
               (sum-two-intvl (se (+ 1 (first intvl))
                                  (bf intvl))))) ) ) )
```

Try writing this using accumulating recursion!

Question 4

1. Describe what `mystery` does. What is its domain, and what is its range.
2. Rewrite the following procedure using `if`/`cond`'s. Please be as concise as possible, ie. eliminate unnecessary code.

```
(define (mystery y)
   (and #t
        (or (and (empty? y) 'done)
            (or (and (= (count y) 1) y))
            (or (and (not (equal? (first y) (first (bf y))))
                     (se (first y) (mystery (bf y))) )
                (mystery (bf y)) ) )
   ) )
```

Solution:

```
(define (mystery y)
  (cond ((empty? y) 'done)
        ((= (count y) 1) y)
        ((not (equal? (first y) (first (bf y))))
         (se (first y) (mystery (bf y))))
        (else (mystery (bf y)))))
```

3. Provide argument(s) to `mystery` that can exhaust all cases. What is the minimum number of different arguments needed, and what are they?

Solution: `(mystery '(a b b)) and (mystery '())`

4. Would `mystery` still work the same way if the bolded **`and`** was changed to "`or`," and `#t` was changed to `#f`? Why or why not? (ie. `(`**`or`** `#f (or (and (empty? ...)` `))`)

Solution: `mystery` would still work the same way because both "`and #t`" and "`or #f`" have no influence on the return value of `mystery`. In both case, scheme would continue to evaluate the second operand.

5. Would `mystery` still work the same way if the order of arguments to the bolded **`and`** were switched? Why or why not? (ie. `(`**`and`** `(or (and (empty? ...)  #t)` `)`)

Solution: No. `mystery` would return `#t` for arguments of count <= 1. It would generate an error for arguments of count > 1. This is because the recursive calls to mystery would pass in the boolean value `#t` to the sentence procedure, which can only take words as its arguments.

Question 5

If the following procedures were defined:

```
(define (mystery x)
   (cond ((and (< x 1000) (> x 100)) 100)
         ((and (< x 100) (> x 10)) 10)
         ((and (< x 10) (> x 0)) 0)))

(define (bar x)
   (+ x (mystery x)))

(define (foo z)
   (if z
       (se z 'is 'true)
       (se z 'is 'false)))
```

What numerical argument value can generate an error, if possible, for the function `bar`?

`(bar 100)` would return an error: `+: okay not a number.`

What do these return: `(and (< 8 3) (mystery 100))`, `(and (mystery 100) (< 8 3))`

`(and (< 8 3) (mystery 100))` → `#f`

`(and (mystery 100) (< 8 3))` → `#f`

Would the calls `(foo 38)` and `(foo 10)` generate an error? If so, where and why?

Nope.

Question 6

Assume that you have loaded a dessert database package into your Scheme session. This package gives your procedures to work with a new data object – a dessert recipe. Some of these procedures include

- **`name`** – takes a single dessert recipe, and returns the name of the dessert as a single word
- **`serving`** – takes a dessert recipe, and returns the serving size (ie. how many people this batch can serve)
- **`chocolicious?`** – takes a dessert recipe, and returns true if the dessert contains chocolates and false otherwise
- **`contains-ingredient?`** – takes an ingredient and a dessert recipe, and returns true if the dessert recipe uses the given ingredient and false otherwise
- **`contains-nuts`** – takes a sentence of dessert recipes, and returns a sentence of dessert recipes that uses nuts as its main ingredient

1. Write a procedure `dessert-ingredient`, which takes an ingredient and a sentence of dessert recipes, and returns a sentence listing the names of desserts that use the given ingredient.

Solution:

```
(define (dessert-ingredient ingredient dessert-sent)
    (cond ((empty? dessert-sent) '())
          ((contains-ingredient? ingredient (first dessert-sent))
           (se (name (first dessert-sent))
               (dessert-ingredient ingredient (bf dessert-sent))) )
          (else (dessert-ingredient ingredient (bf dessert-sent))) ) )
```

2. Write the procedure `picky-eaters`, which takes a sentence of dessert recipes, and returns a sentence of dessert-recipes that use both chocolate and nuts, and can serve at least 10 people.

Solution:

```
(define (picky-eaters dessert-sent)
    (cond ((empty? dessert-sent '())
          ((and (chocolicious? (first (contains-nuts dessert-sent)))
                (>= (serving (first dessert-sent)) 10) )
           (se (first dessert-sent) (picky-eaters (bf dessert-sent))) )
          (else (picky-eaters (bf dessert-sent)) ) )  )
```

Solution2:

```
(define (picky-eaters dessert-sent)
    (picky-eaters-helper (contains-nuts dessert-sent) ) )

(define (picky-eaters-helper nuts-sent)
    (cond ((empty? nuts-sent '())
          ((and (chocolicious? (first nuts-sent))
                (>= (serving (first nuts-sent)) 10) )
           (se (first nuts-sent) (picky-eaters (bf nuts-sent))) )
          (else (picky-eaters (bf nuts-sent)) ) )  )
```

3. The host wants to make all the chocolate desserts he knows, and divide them among his guests. He wants to figure out how many servings each guest would end up with. Help him by writing the procedure `serve-chocolate`, which takes the number of guests and a sentence of dessert recipes, and returns how many chocolicious dessert servings each guest can have.

Solution:

```
(define (serve-chocolate guests dessert-sent)
    (/ (total-choco-serving dessert-sent) guests) )

(define (total-choco-serving dessert-sent)
    (cond ((empty? dessert-sent) 0)
          ((chocolicious? (first dessert-sent))
           (+ (serve (first dessert-sent))
              (total-choco-serving (bf dessert-sent))) )
          (else (total-choco-serving (bf (dessert-sent))) ) ) )
```