CSE 341 - Programming Languages
Midterm - Autumn 2006 - Answer Key

Open book and notes. No laptop computers, PDAs, or similar devices. (Calculators are OK, although you won't need one.) Please answer the problems on the exam paper — if you need extra space use the back of a page.

80 points total

1. (15 points) Suppose that the following Scheme functions have been defined.

```
;; this works the same as the built-in
;; Scheme append function for 2 arguments
(define (append x y)
  (if (null? x)
      y
      (cons (car x) (append (cdr x)  y))))

(define (mystery x y)
  (lambda (z) (+ x y z)))
```

What is the value of each of the following Scheme expressions?

(a)
```
(let ((f (mystery 2 3)))
  (map f '(10 20 30 40)))
```

```
(15 25 35 45)
```

(b)
```
(let ((x 1)
      (y 5))
   (let ((x (+ y 2))
         (y (* x 3)))
     (+ x y)))
```

```
10
```

(c)
```
(let* ((x '(1 2 3))
       (y '(10 11 12 13))
       (z (append x y)))
  (set-car! x 'frog)
  (set-car! y 'toad)
  (set! x '(100 101))
  (set! y '(200 201))
  z)
```

```
(1 2 3 toad 11 12 13)
```

2. (15 points) Write a Scheme function `count` that takes two arguments, a symbol `s` and a value `y`, and counts how many occurrences there are of `s` in `y`. You can assume that `s` is a symbol. (Hint: the `pair?` predicate can be used to test whether something is a non-empty list.) For example:

```
(count 'c '(a b c b c))  =>  2
(count 'c 'c)  =>  1
(count 'c 42)  =>  0
(count 'c '(a ((b c)) (c) (d e f c)))  =>  3
```

```
(define (count s y)
  (cond ((eq? s y) 1)
        ((pair? y) (+ (count s (car y)) (count s (cdr y))))
        (else 0)))
```

3. (21 points) Suppose the following Miranda script has been read in:

```
cube x = x*x*x

average x y = (x+y)/2

composeall [] x = x
composeall (f:fs) x = f (composeall fs x)

binarytree * ::= Leaf * | Node * (binarytree *) (binarytree *)

|| map a function over a binary tree
treemap f (Leaf x) = Leaf (f x)
treemap f (Node n left right) = Node (f n)
                                     (treemap f left)
                                     (treemap f right)

|| define a few trees
t1 = Leaf 10
t2 = Node 7 (Leaf 2) (Node 4 (Leaf 1) (Leaf 5))
bigtree x = Node x (Leaf x) (bigtree x)
```

What is the *value* of each of the following expressions? (If there is a type error, say so.)

(a) `cube "squid"`
`type error`

(b) `composeall [(*2), cube, (+1)] 3`
`128`

(c) `treemap cube t2`
`Node 343 (Leaf 8) (Node 64 (Leaf 1) (Leaf 125))`

What is the *type* of each of the following expressions? Some of them may give type errors — if so, say that.

(a) `average`
`num->num->num`

(b) `composeall`
`[*->*]->*->*`

(c) `treemap`
`(*->**)->binarytree *->binarytree **`

(d) `bigtree`
`*->binarytree *`

4. (15 points) Consider the definition for `tree` in Question 3.

- Write a function `treemax` that finds the maximum value of all the elements in a tree.
- What is the type of `treemax`? (Give the most general possible type.)

```
treemax :: binarytree *->*
treemax (Leaf n) = n
treemax (Node n left right) = max [n, treemax left, treemax right]
```

5. (14 points) Tacky but easy-to-grade true/false questions!

(a) Scheme is statically typed. **False.**

(b) Scheme is type safe. **True.**

(c) A Miranda program is statically typed if the programmer includes a type declaration for all functions; otherwise it is dynamically typed. **False.**

(d) A parameter to a Miranda function is evaluated exactly 0 or 1 times. **True.**

(e) When writing Scheme macros, a good programming practice is to give unusual names (such as `zzzz_temp`) to any local variables in the macro, to avoid accidentally interfering with (for example) a variable named `temp` in the user's code. **False.**

(f) In the Scheme metacircular interpreter, `and` and `or` needed to be defined as new special forms or as derived expressions, rather than as new primitive procedures. However, `not` and `xor` can just be added as new primitive procedures. **True.**

(g) An advantage of making a function tail-recursive is that it can be compiled in a way that doesn't use stack space for every recursive invocation of the function. **True.**