

Homework 1

CSC 4101, Fall 2009

Due: 10 September 2009

1. (10 pts)

Draw binary tree diagrams for each of the following Scheme expressions:

- (a) `'(a (b c) d e)`
- (b) `'((((a b) (c d)) e))`
- (c) `'(a (b (c (d (e f)))) ())`
- (d) `'(((()) ((()))`
- (e) `'(((a b) c) d (((e f) g)))`

2. (10 pts)

Show the result of evaluating each of the following Scheme expressions:

- (a) `(cdr '((a (b)) (c)))`
- (b) `(cons '(a) '((b) (c)))`
- (c) `(car (car (cdr '(a ((b (c)))))))`
- (d) `(quote (car '(a ((b) c))))`
- (e) `(car (quote (cons 'a '((b) c))))`

3. (10 pts)

Given the following Scheme definition:

```
(define x '(define (fac n)
            (if (= n 0) 1
                (* n (fac (- n 1))))))
```

(This defines not the factorial function `fac`, but the variable `x`.)

Write Scheme expressions in terms of `x` that would have the effect of extracting the following expressions:

- (a) `(fac n)`
- (b) `0`
- (c) `(- n 1)`
- (d) The second occurrence of `fac`.
- (e) The last occurrence of `n`.

E.g., the expression `(car x)` would extract `define`.

4. (10 pts)

Consider the following Scheme function `foo`:

```
(define (foo x)
  (cond
    ((null? x) 0)
    ((not (list? (car x)))
     (cond
       ((eq? x '()) (foo (car x)))
       (else (+ 1 (foo (cdr x))))))
    (else (+ (foo (car x)) (foo (cdr x))))))
```

(a) (8 pts)

Explain what the function computes and how. Don't just restate the function definition in English — explain the algorithm. Hint: not all the code in this function does something useful.

(b) (2 pts)

Show the result of executing the expression

```
(foo '((a (b c) d) ((d) e) f) g))
```

5. (10 pts)

Write a recursive Scheme function with two parameters, an atom and a list, that returns the number of occurrences, no matter how deep, of the given atom.

An atom is anything that's not a list, e.g., numbers or symbols. For comparing atoms for equality use `eqv?`. The pointer equality test `eq?` is not guaranteed to work for numbers, `=` only works for numbers. A precise description of these functions can be found in the section 'Equivalence predicates' in the *Revised(5) Report on the Algorithmic Language Scheme*, which is available off the course web page.

Except for the first question, you can check your answers in Scheme48 (or even in Emacs), but I strongly recommend you try them by hand first.