

Homework 1

CSC 4101, Fall 2006

Due: 14 September 2006

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

2. (10 pts)

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

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

3. (10 pts)

Given the following Scheme definition:

```
(define x '(define (fib n)
            (if (<= n 1) n
                (+ (fib (- n 1)) (fib (- n 2))))))
```

(This defines not the Fibonacci function `fib`, but the variable `x`.)

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

- (a) `(fib n)`
- (b) `<=`
- (c) `(fib (- n 1))`
- (d) The third occurrence of `fib`.
- (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)))
     (+ 1 (foo (cdr x))))
    ((eq? x '()) (foo (car x)))
    ((eq? x '(x)) (foo x))
    (#t (+ (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 list with all occurrences, no matter how deep, of the given atom deleted. The returned list cannot contain anything in place of the deleted atoms.

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.