

CS 380: Optional Lisp Assignment

Fall 2006

1. Describe what happens when the following expressions are evaluated:

- (a) `(+ (- 5 1) (+ 3 7))`
- (b) `(list 1 (+ 2 3))`
- (c) `(if (listp 1) (+ 1 2) (+ 3 4))`
- (d) `(list (and (listp 3) t) (+ 1 2))`

2. Give three distinct `cons` expressions that return `(a b c)`.

3. Using `car` and `cdr`, define a function to return the fourth element of a list.

4. Define a function that takes two arguments and returns the greater of the two.

5. What do the following functions do?

- (a)

```
(defun enigma(x)
  (and (not (null x))
        (or (null (car x))
            (enigma (cdr x)))))
```
- (b)

```
(defun mystery (x y)
  (if (null y)
      nil
      (if (eql (car y) x)
          0
          (let ((z (mystery x (cdr y))))
            (and z (+ z 1)))))))
```

6. Show the following lists in box notation¹:

- (a) `(a b (c d))`
- (b) `(a (b (c (d))))`
- (c) `(((a b) c) d)`
- (d) `(a (b . c) . d)`

7. Write a version of `union` that preserves the order of the elements in the original lists. For example:

```
[1]> (new-union '(a b c) '(b a d))
(A B C D)
```

8. Write a *recursive* function that takes an integer as input and returns the “reverse” of that integer. The function should not need to call any supplementary functions (*i.e.*, those that are not built into common lisp). Your function may use conversion to strings/lists, although it must return an integer. There is a way of doing this without using strings (*Hint*: think logarithms, exponentiation, and modulus). Here’s an example of like what the output should look:

```
[1]> (reverse-integer 12345)
54321
```

Many of these problems were adapted from *ANSI Common Lisp* by Paul Graham (ISBN 0-13-370875-6): the book that used to be required for this class.

¹Since you don’t necessarily own a book on Lisp (one such book *used* to be used for this class), you will only be responsible for knowing box notation if we cover it in class.