

# CS440: Assignment #1: Introduction to Common Lisp

Assigned Aug. 28th

**Due 12:30, Sept. 6th**

Solve the Common Lisp problems described in Section 1. Section 2 is a description of the procedure you must use to turn in your answers. You may discuss these problems with classmates, but your solutions must be your own. Understanding all problems will help you prepare for quizzes and tests.

## 1 Problems

- Using `car` and `cdr`, define a function named `4th` to return the fourth element of a list.
- Define a function that takes two arguments and returns the greater of the two.
- What could occur in place of the `x` in each of the following exchanges?

(a) USER(1): (car (x (cdr '(a (b c) d))))  
B

(b) USER(1): (x #'list 1 nil)  
(1)

- Write each of the following infix expressions in Lisp. Remember, you can use the `apropos` to look for functions with certain strings in their names. Check your answers with a calculator.

(a)  $45/9 - 3$

(b)  $-45 + 1$

(c)  $0.5(20 - 12/2)$

(d)  $3.14159/10/20$

(e)  $\arcsin(\sin(1.0))$

(f)  $\sqrt{25}$

(g)  $\log_2 8.0$

(h)  $\exp^{2\pi}$

- What is the Lisp expression that evaluates to this list:

```
[ * | * ]----->[ * | * ]---->NIL
  |                 |
  v                 v
[ * | * ]---->NIL  [ * | * ]---->[ * | * ]----->[ * | * ]---->NIL
  |                 |                 |                 |
  v                 v                 v                 v
  A                 B                 [ * | * ]---->[ * | * ]---->NIL  "what about me"
                                     |                 |
                                     v                 v
                                     BILL                32
```

- Write a function called `inches` that accepts feet and inches and returns the equivalent length in centimeters. If  $f$  is the first argument and  $i$  is the second, the formula is  $2.54(12f + i)$ . For example,

```
USER(1): (inches 0 1)
2.54
```

```
USER(1): (inches 6 (+ 2 1/2))      same as      (inches 6 2.5)
189.22999999999999
```

7. Write an interactive function using `dotimes` that returns a list of the first fifteen cubes (1, 8, 27, 64, ..., 3375).
8. Now write a recursive version of the above function.

9. Use `mapcar` to write a function that takes a list and creates a new list whose elements are lists obtained by repeating original elements. For example, if the old list was (X Y (Z W)), then the new list is ((X X) (Y Y) ((Z W)(Z W))).

10. Write an iterative function to return the inner product (dot product) of two vectors  $x$  and  $y$  implemented as lists. This is a single value computed as  $x_1y_1 + x_2y_2 + \dots + x_ny_n$ . Time its execution using the function `time` and arguments each having 10 components. For example:

```
USER(1): (time (dotimes (i 1000)
  (inner-product '(1 2 3 4 5 6 7 8 9 10) '(1 2 3 4 5 6 7 8 9 10))))
```

11. Now write a recursive version of the inner product function and time it given the same arguments. How do the two compare in execution time?

12. Use the Unix `grep` command to discover what functions are defined in the file `~/cs440/Russell-code/search/algorithms/simple.lisp`. All function definitions start as

```
(defun ...
```

You can use the Unix command `man grep` to learn more about `grep`. What is the first and last function defined in Russell's `search.lisp` file?

13. The following are faulty function definitions. Debug them so that they produce the example results.

(a) -----  

```
(defun myreverse (x)
  (labels ((revh (x result)
    (cond ((null x) nil)
          (t (revh (cdr x) (cons (list (car x)) result))))))
    (revh x nil)))

(myreverse '(1 2)) =>(2 1)
```

(b) -----  

```
(defun flip (switches)
  (mapcar \#' (lambda (switch)
    (if (equal switch 'on)
        (setf switch 'off))
        (if (equal switch 'off)
            (setf switch 'on)
            ;:else
            (setf switch switch))))
    switches))

(flip '(on off on)) => (OFF ON OFF)
(flip '(a b c on)) => (A B C OFF)
```

(c) -----  

```
(defun flush (this that)
  (cond ((null that) nil)
        ((equal this (car that)) (cdr that))
        (t (cons (car that) (flush this (cdr that))))))

(flush 1 '(5 4 1 2 43 1 2)) => (5 4 2 43 2)
```

```

(d) -----
(defun funny (x)
  (do ((f x (cdr x))
      (b (reverse x) (cdr x))
      (new1 nil (append (list (car f)) new1))
      (new2 nil (append (list (car b)) new2)))
      ((null f) (append new1 new2))))

(funny '(a b c d)) => (D C B A A B C D)
(e) -----
(defun up-to-first-num (arg)
  (do ((a arg (cdr a))
      (b nil))
      ((numberp (first a)) b)
      (setf b (append b (list (first a))))))

(up-to-first-num '(a b 1 2)) => (A B)

(up-to-first-num '(a b c)) => (A B C)

(up-to-first-num '(1)) => NIL
(f) -----
(defun to-percent (fractions)
  (let ((answer nil))
    (dolist (d fractions)
      (setf answer (append answer (percent (first d) (second d))))))

(defun percent (num denom)
  (* 100 (/ num denom)))

(to-percent '((50 100) (3 4))) => (50 75)

```

**(Section 2 is on the next page. Don't forget to read it!)**

## 2 How to Submit Assignment Solutions

Your solution to this assignment must be submitted in one Lisp file named `assignment1.lisp`. Somewhere in your Lisp file you must include the Lisp forms that run each of your answers. To do so, define the following `do-problem` function at the top of your file following some comments. Here is an example of how to start your `assignment1.lisp` file. It even includes the answer to the first problem, as a freebie. Also shown is a probably incorrect answer to problem 10 to demonstrate how to produce the answers to questions that require you to write a response.

```
;;; Chuck Anderson
;;; Assignment 1
;;; CS440, Fall 2001

(defun do-problem (n form)
  "Example: (do-problem '10a '(my-solution '(3 4)))"
  (format t "~&-----")
  (format t "~&Problem ~s: ~s => ~s" n form (eval form)))

(format t "~&Chuck Anderson: Assignment 1")

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Problem 1

(defun 4th (lst)
  (car (cdr (cdr (cdr lst)))))

(do-problem '1 '(4th '(a b c d e f)))

(do-problem 10 nil)
(format t "~&The dotimes version of inner-product runs 1 million
times faster than the recursive version.")
```

Your Lisp file will be run by simply doing

```
(load "yourfile.lisp")
```

Here is what you will see when you load the above example.

```
CL-USER(17): (load "yourfile.lisp")
; Loading yourfile.lisp
Chuck Anderson: Assignment 1
-----
Problem 1: (4TH '(A B C D E F)) => D
-----
Problem 10: NIL => NIL
The dotimes version of inner-product runs 1 million
times faster than the recursive version.
T
```

When you are happy with how it functions, you must electronically check in your Lisp file by doing

```
~cs440/bin/checkin assignment1.lisp
```

You may do this multiple times before the due date and time. Your last entry will be the one we grade. We will be reading your Lisp file, so you may include comments in your file as explanations of your answers.