

C SCI 316 (Kong): Lisp Assignment 5

Solutions to these problems are to be submitted *no later than Thursday 10/27.**

See page 5 for instructions on how to submit.

SECTION 1 (Nonrecursive Preliminary Problems)

The three problems in this section (A – C) do not carry direct credit, but are intended to help you solve problems 1 – 3 in Part I of Section 2. Note that there *may* be questions on exam 1 or on the final exam that are of a similar nature to A – C.

Your solutions to problems A – C *must not be recursive*. You can test your solutions to these three problems on *venus* or *euclid*: Functions INDEX, MIN-FIRST, and SSORT with the stated properties are predefined for you when you start Lisp using `cl` on those machines.

- A. INDEX is a function that is already defined on euclid and venus. If *N is any positive integer and L is any list*, then (INDEX N L) returns the Nth element of L if N does not exceed the length of L; if N exceeds the length of the list L, then (INDEX N L) returns the symbol ERROR. For example, (INDEX 3 '(A S (A S) (A) D)) => (A S) (INDEX 6 '(A S (A S) (A) D)) => ERROR. Complete the following definition of a function MY-INDEX *without making further calls of* INDEX and without calling MY-INDEX recursively, in such a way that if N is any integer *greater than* 1 and L is any *nonempty* list then (MY-INDEX N L) is equal to (INDEX N L).

```
(defun my-index (N L)
  (let ((X (index (- N 1) (cdr L))))
    — ))
```

[Note: You should not have to call any functions.]

- B. MIN-FIRST is a function that is already defined on euclid and venus; if *L is any nonempty list of real numbers* then (MIN-FIRST L) is a list whose CAR is the minimum of the numbers in L and whose CDR is the list obtained when the first occurrence of that value is removed from L. (Thus (MIN-FIRST L) is "L with the first occurrence of its minimum value moved to the front".) For example, (MIN-FIRST '(0 3 1 1 0 3 5)) => (0 3 1 1 0 3 5)
(MIN-FIRST '(4 3 1 1 0 3 5 0 3 0 2)) => (0 4 3 1 1 3 5 0 3 0 2).

Complete the following definition of a function MY-MIN-FIRST *without making further calls of* MIN-FIRST and without calling MY-MIN-FIRST recursively, in such a way that if L is any list of *at least two* real numbers then (MY-MIN-FIRST L) is equal to (MIN-FIRST L).

```
(defun my-min-first (L)
  (let ((X (min-first (cdr L))))
    _____
    _____ ))
```

[There are two cases: (car L) may or may not be \leq (car X).]

*If you have difficulty with these problems, you are encouraged to come to see me in my office, either during my office hours or (if you cannot come at that time) by appointment. Questions about these problems that are e-mailed to me will not be answered before the submission deadline.

C. SSORT is a function that is already defined on euclid and venus; if *L* is any list of real numbers then (SSORT *L*) is a list of those numbers in ascending order. Complete the following definition of a function MY-SSORT *without making further calls of* SSORT and without calling MY-SSORT recursively, in such a way that if *L* is any *nonempty* list of real numbers then (MY-SSORT *L*) is equal to (SSORT *L*).

```
(defun my-ssort (L)
  (let* ((L1 (min-first L))
        (X  (ssort (cdr L1))))
    _____ ))
```

SECTION 2 (Main Problems)

Your solutions to the following 16 problems will count a total of 2% towards your grade if the grade is computed using rule A—the 10 problems in Part I will count 1%, and the 6 problems in Part II will also count 1%. It is expected that your solutions to problems 1 – 3 will be based on your solutions to A – C above. [On euclid and venus, when you load in your function definitions for problems 1 – 3, you will replace the predefined functions with the same names with your versions of those functions.]

Program in a functional style, *without using SETF / SETQ or iterative constructs such as DO*. Indent code appropriately (see Winston & Horn for examples of good indentation). Use LET / LET* or appropriate helping functions to avoid evaluating computationally expensive expressions more than once. Any helping functions you define for use in your solutions must be defined in the .lsp file that you submit for grading.

When writing a function, decide what the *valid* values for each argument will be. The statement of each problem specifies that certain argument values *must* be valid. For example, in problem 5 any list of real numbers *in ascending order* (including the empty list) *must* be a valid value for each of the arguments; other argument values *need not* be valid.

If *f* computes (*f y* ...) from (*f (smaller y)* ...) for certain values of *y*, then you should check that this *never* happens for any value of *y* which is valid (in the sense of the previous paragraph) for that argument but which satisfies one of the following conditions:

BC-1: (*smaller y*) is *undefined*.

BC-2: (*smaller y*) is *defined but is not a valid value for that argument of f*.

BC-3: (*smaller y*) is a *valid value for that argument of f but is no smaller in size than y* [e.g., the case *y* = NIL when (*smaller y*) is (cdr *y*)].

PART I [Solutions to problems 1 – 10 will count 1% towards your grade if the grade is computed using rule A.]

1. Define a recursive function INDEX with the properties stated in problem A. Note that the first argument of INDEX may be 1, and that the second argument may be NIL.
2. Define a recursive function MIN-FIRST with the properties stated in problem B. Note that the argument of MIN-FIRST may be a list of length 1.
3. Define a recursive function SSORT with the properties stated in problem C. In the definition of SSORT you may call SSORT itself, MIN-FIRST, CONS, CAR, CDR and ENDP, but you should not call any other function.

4. Use the function PARTITION from Lisp Assignment 4 to complete the following definition of a recursive function QSORT such that if *L is a list of real numbers* then (QSORT L) is a list of those numbers in ascending order.

```
(DEFUN QSORT (L)
  (COND ((ENDP L) ()))
        (T      (LET ((PL (PARTITION L (CAR L))))
                   (COND ((ENDP (CAR PL)) ... ) ; (CAR L) is the least element of L
                           (T      ... ))))))
```

5. Define a Lisp function MERGE-LISTS such that if each of L1 and L2 is a list of real numbers in ascending order then (MERGE-LISTS L1 L2) returns the list of numbers in ascending order that is obtained by merging L1 and L2. Your definition must ***not*** call any sorting function.

Examples: (MERGE-LISTS '(2 4 5 5 7 8 9) '(3 4 6 9 9)) => (2 3 4 4 5 5 6 7 8 9 9 9)
 (MERGE-LISTS '(1 2 3) '(4 5 6 7)) => (1 2 3 4 5 6 7)
 (MERGE-LISTS '(3 4 5 6 7) '(0 1 2 3)) => (0 1 2 3 3 4 5 6 7)

Hint: Consider the 4 cases $L1 = ()$, $L2 = ()$, $(< (car L1) (car L2))$, and $(>= (car L1) (car L2))$.

6. Use the function SPLIT-LIST from Lisp Assignment 4 and MERGE-LISTS to define a recursive Lisp function MSORT such that if L is a list of real numbers then (MSORT L) is a list consisting of the elements of L in ascending order. In the definition of MSORT you may call SPLIT-LIST, MSORT itself, MERGE-LISTS, CAR, CDR, CADR and ENDP, but you should not call any other function. Be sure to use LET or LET*, so that MSORT only calls SPLIT-LIST once.

Hint: Does a list of length 1 satisfy condition **BC-3** (see page 2) for one of your function's recursive calls?

In problems 7 – 10, assume the argument is a list.

7. Do exercise 10.4(a) on page 418 of Sethi, but use Common Lisp instead of Scheme. Name your function REMOVE-ADJ-DUPL.

Hint for problems 8 and 9 One way to do these two problems is to give definitions of the following form:

```
(defun function-name (L)
  (cond ((endp L) ... ) ; L is ()
        ((or (endp (cdr L)) (not (equal (car L) (cadr L)))) ... ) ; L is (x) or (x y ... )
        ((or (endp (cddr L)) (not (equal (car L) (caddr L)))) ... ) ; L is (x x) or (x x y ... )
        (t ... ))) ; L is (x x x ... )
```

8. Do exercise 10.4(b) on the same page in Common Lisp. Name your function UNREPEATED-ELTS.
9. Do exercise 10.4(c) on the same page in Common Lisp. Name your function REPEATED-ELTS.
10. Do exercise 10.4(d) on the same page in Common Lisp. Name your function COUNT-REPETITIONS.

[End of Part I]

PART II [Solutions to problems 11 – 16 will count 1% towards your grade if the grade is computed using rule A.]

11. **[Exercise 8 on p. 141 of Wilensky]** Write a recursive function SUBSET that takes two arguments: a function and a list. SUBSET should apply the function to each element of the list, and return a list of all the elements of the argument list for which the function application returns a true (i.e., non-NIL) value. **Example:** (subset #'numberp '(a b 2 c d 3 e f)) => (2 3)

12. **[Exercise 7 on p. 141 of Wilensky]** Write (i) a recursive function OUR-SOME and (ii) a recursive function OUR-EVERY each of which takes two arguments: a function and a list. OUR-SOME should apply the function to successive elements of the list *until* the function returns a true (i.e., non-NIL) value, at which point it should return the rest of the list (including the element for which the function just returned a true value).^{*} If the function returns NIL when applied to each of the arguments, OUR-SOME should return NIL. OUR-EVERY should apply the function to successive elements of the list *until* the function returns NIL, at which point it should return NIL. If the function returns a true value when applied to each of the arguments, then OUR-EVERY should return T. **Examples:**

(our-some #'numberp '(A B 2 C D)) => (2 C D) (our-some #'numberp '(A B C D)) => NIL

(our-every #'symbolp '(A B 2 C D)) => NIL (our-every #'symbolp '(A B C D)) => T

^{*}Note that the built-in Lisp function SOME behaves differently from OUR-SOME in this case: SOME returns the non-NIL value that was just returned by the function.

13. Modify your solutions to problem 7 of Lisp Assignment 4 and problem 4 above to produce a solution to exercise 10.6b on p. 419 of Sethi. Name your function QSORT1. (Since PARTITION is used in problem 4, you must use some other name for your modified version of that function.)

Examples:
(QSORT1 #'> '(2 8 9 5 1 7 3)) => (9 8 7 5 3 2 1)
(QSORT1 #'< '(2 8 9 5 1 7 3)) => (1 2 3 5 7 8 9)
(QSORT1 (LAMBDA (L1 L2) (< (LENGTH L1) (LENGTH L2)))
'((X) (A D X E G K) (1 2 Q R S) NIL (S D F)))
=> (NIL (X) (S D F) (1 2 Q R S) (A D X E G K))

14. Do exercise 10.12a on p. 420 of Sethi in Common Lisp. **Example:**

(FOO #'- '(1 2 3 4 5)) => ((-1 2 3 4 5) (1 -2 3 4 5) (1 2 -3 4 5) (1 2 3 -4 5) (1 2 3 4 -5))

15. First, reread the subsection "Recursion Can Be Efficient" on pp. 75 – 8 of Winston and Horn. Then solve the following problems.

(a) Do exercise 10.5 on p. 419 of Sethi in Common Lisp. Name your functions TR-ADD, TR-MUL and TR-FAC. **Examples:**

(TR-ADD '(1 2 3 4 5) 0) => 15 (TR-MUL '(1 2 3 4 5) 1) => 120 (TR-FAC 5 1) => 120

Note that your definitions of TR-ADD, TR-MUL, and TR-FAC must be tail recursive.

(b) Wilson's Theorem in number theory states that an integer $n > 1$ is prime if and only if $(n - 1)! \bmod n = n - 1$. Use *this theorem* and TR-FAC to write a predicate SLOW-PRIMEP such that if $n > 1$ is an integer then (SLOW-PRIMEP n) returns T or NIL according to whether or not n is prime. (You may of course use the built-in function MOD.) Test your predicate SLOW-PRIMEP by using it to find the least prime number greater than 20,000.

Important: To prevent stack overflow, enter (compile 'tr-fac) at clisp's > prompt before calling SLOW-PRIMEP with large arguments. [When a **tail recursive** function is compiled by clisp, the optimization referred to in footnote 2 on p. 78 of Winston and Horn is performed.]

16. **[Based on exercise 8 on p. 111 of Wilensky]** A matrix can be represented as a nonempty list of nonempty lists in which the i^{th} element of the j^{th} list is the element in the i^{th} column and j^{th} row of the matrix. For example, `((A B) (C D))` would represent a 2×2 matrix whose first row contains the elements A and B, and whose second row contains the elements C and D. Write three functions `TRANSPOSE1`, `TRANSPOSE2`, and `TRANSPOSE3`, each of which takes a single argument `M`; if `M` is a nonempty list of nonempty lists which represents a matrix in the above-mentioned way, then each of the three functions should return the list of lists which represents the transpose of that matrix. The three functions should work as follows:

- (a) `(transpose1 M)` computes its result from `(transpose1 (cdr M))` and `(car M)` using `mapcar #'cons`. **[Hint:** Take the set of valid values of `M` to be the set of lists of one or more nonempty lists that all have the same length. Then the **BC-2** base case is `(endp (cdr M))`, when the result to be returned is `(mapcar #'list (car M))`.]
- (b) `(transpose2 M)` computes its result from `(transpose2 (mapcar #'cdr M))` and `(mapcar #'car M)`. **[Hint:** Again, take the set of valid values of `M` to be the set of lists of one or more nonempty lists that all have the same length. This time, the **BC-2** base case is `(endp (cadr M))`, when the result is `(list (mapcar #'car M))`.]
- (c) `(transpose3 M)` obtains its result as follows: `(apply #'mapcar (cons #'??? M))` or, equivalently, `(apply #'mapcar #'??? M)`, where `???` is the name of an appropriate function.

Example (here n may be 1, 2 or 3):

`(TRANSPOSE n '((1 2 3 4) (5 6 7 8) (9 10 11 12))) => ((1 5 9) (2 6 10) (3 7 11) (4 8 12))`

How to Submit Your Solutions for Grading

You may work with up to two other students on these problems, but each student must write up his or her solutions individually—*no two students should submit identical files*. Put the function definitions you wrote for problems 1 – 16 in a single file named

your last name in lowercase-5.lsp

in your home directory on euclid. This file must include definitions of any helping functions that are used. If you are working with others, include the name(s) of your partners in comment(s) at the beginning of the file. Functions that are incorrectly named may receive no credit (e.g., if your solution to problem 5 is named `MERGE-LIST` instead of `MERGE-LISTS`, you may get no credit for that problem). If `clisp` cannot `LOAD` your file without error then you may well receive no credit for your submission, even if the only error is a single missing parenthesis!

Within the file, your solution to each problem should be preceded by a comment of the following form, where `N` is the problem number: `; Solution to Problem N`

Your solutions should appear *in the same order as the problems*. If you cannot solve a problem, put a comment of the form `; No Solution to Problem N Submitted` where a solution to that problem would have appeared.

Leave your final version of *your last name in lowercase-5.lsp* in your home directory on euclid *no later than midnight on the due date*.

Do NOT open the submitted file in an editor on euclid after the due date, unless you are resubmitting a corrected version of your solutions as a *late* submission. **[Note:** If euclid unexpectedly goes down after 6 p.m. on the due date, the deadline will *not* be extended. Try to submit no later than noon that day, and much sooner if possible.]