CSE324 Spr21 Lisp programming Assignment Due: ***Midnight of Thursday, April 15th.***

Write the following functions in *GNU CLISP 2.49*, running on our CS computer lab machines. All implemented functions MUST be recursive and the most efficient solution/implementation.

All functions inputs must be validated and error messages should be prompted for unexpected input error (i.e., illegal inputs).

1) insert-atom-list(x L): a function that takes a list L and an item ***x*** (atom/list) and returns L after the inserting ***x*** at the ***end*** if x is a ***list*** or at the *beginning* if x is an *atom*, inside L. *(10 pts)*

Example: **(insert-atom-list '4 '(1 2 3))** should return **(4 1 2 3)**.
(insert-atom-list '(4) '(1 2 3)) should return **(1 2 3 (4))**.
(insert-atom-list '**(4) ())** should return **((4))**.
(insert-atom-list '**4 ())** should return **(4)**.

2) insert-special(*x y* z L): a function that takes an item ***x*** to be inserted between first occurrence of ***y*** immediately followed by ***z*** in the list L, then return L with x inserted between y and z. If there is no y followed by z in L, returns L. *(15 pts)*

Example: **(insert-special '4 '3 '5 '(1 2 3 5 6 7 8 9 10 11))**
should answer --> **(1 2 3 4 5 6 7 8 9 10 11)**
(**insert-special** '**4** '**3** '**5** '(5 6 7 8 9 10))
should answer --> (5 6 7 8 9 10)
(**insert-special** '**4** '**3** '**5** '()) should answer --> nil

3) remove-duplicate-item(L): a function that takes a list L and removes any duplicated items (i.e., replaces any two occurrences of the same item by a single occurrence only), then it returns the L without any duplicate items. *(15 pts)*

Example: *(**remove-duplicate-item** '(1 2 **3 3** 4 5 6 6 7 8 **9 9**))
Should return > (1 2 **3** 4 5 6 7 8 **9**)

4) remove-lists (L): a function that takes a list L and outputs a list of only the atoms with L, i.e., removing any lists from L. *(15 pts)*

Example: **(remove-lists '(a (2) (3 4) d 1 (9 8) x (c)))** should return **(a d 1 x).**

5) count-match-items (L1 L2): a function that counts the number of spatially matching **atoms** in the two input lists L1 & L2. *(20 pts)*

Example: **(count-match-items '((a b) (c 4) s (t) y) '((a b) (c 4) s t s))**

should return **3**. . .. … . .. …

6) filter(P L): a function that takes as input: a predicate **P** and a list L and returns a list of only those elements of **L** that satisfy **P**. *(10 pts)*

Example: **(filter 'minusp '(2 –3 7 –1 –6 4 8))**

should return **(-3 –1 -6).**

Also, **(filter 'listp '(a (b c) d (e f g)))** should return **((b c) (e f g)).**

7) my-reverse(L): a function that returns it input list L in a reverse order. *(15 pts)*

Example: **(my-reverse '(1 2 3 4 5))** should return **(5 4 3 2 1)**

Extra Credit: (20 pts)

8) z-following-x(L x z): a function that returns YES for the first occurrence of item **x** followed immediately by item **z** in the non-empty list **L**.

Examples: **(z-following-x '(1 2 3 5 6 7 8 9 10 11) '3 '5)**

should return **YES**

(z-following-x '(1 2 3 5 6 7 8 9 10 11) '9 '10)

should return **YES**

(z-following-x '(1 2 3 5 6 7 8 9 10 11) '8 '10)

should return **NO**

(z-following-x '() '8 '10)

should return **NO**

More examples:

(z-following-x '(6 (1 2) (3 4) 1 2 3 5) '1 '2)→ YES

(z-following-x '(6 (1 2) (3 4) 2 3 5) '1 '2)→ NO

(z-following-x '(6 (1 2) (3 4) 2 3 5) '(1 2) '(3 4)) → YES

(z-following-x '(6 9 5 (1 2) (3 4) 2 3 5) '(1 2) '(3 4)) → YES

(z-following-x '(6 (1 2) (3 6) 1 2 3 4) '(1 2) '(3 4)) → NO

Warning!!!!!!

The usage of *built-in* functions such as **reverse** (to implement reverse-list()), also **remove** (to implement remove-from), and so forth, is **NOT** accepted. All functions should be implemented using **basic built-in** functions not part of the core required function to be implemented in this assignment, such as **car**, **cdr**, **cons**, **append**, list, …,**last, cond, etc.**

How to run lisp *interpreter*? At the prompt, type: **lisp** then hit return

How to *compile* and use lisp? To compile lisp file "example.lisp", containing some functions (the file extension **MUST** be **.lisp**) **from within the lisp interpreter**, write:

* (**compile-file** "example.lisp") which produces a new file: example.x86f

Then to load and execute any function in "example.lisp", **at the "lisp" interpreter level**:

* (**load** "example.x86f")

From now on, at the system level, you can use any function that you defined and successfully compiled in "example.lisp". This saves you time of worrying about the matching parenthesis (and other silly mistakes) at the hard to correct system level.

Hints:

- In case of missing up at the command level (e.g., mistyping a command), you will be sent to the debugger. Type **abort** to exit the debugger to the top lisp level.

- To exit the lisp interpreter write **(quit)**.

- Try to minimize writing code at the command interpreter prompt as much as you can, instead write most definitions in the lisp file (<file-name>.lisp) then compile/load it. Anything you defined inside the file will be accessible.

To print an error message or results, a word or a string, you may follow this example:

```
(cond ((null x) '(msg: error: undefined-symbole) )  ;; string error msg
          ( (null L)      'NO  )  ;; print NO
   ***continue coding****
( t   ....   rest of the code ....        )
```

Remember any text between a “;” and the end of line is considered comments.

Submission:

Your project must be submitted as a plain text file attachment on the Canvas. The filename must read: “yourname_hw4.txt”

Example: John Ashcroft would use the file name:

jashcroft_hw4.txt”

Warning: If you fail to submit according to the above criteria, your project may not be accepted. If any of the above requirements are not understood, contact the TA for this course.

Note: All work must be your own and failure to do so will result in a zero for the report. All rules in the Academic Honesty Policy strictly apply.