Homework IV

Policies

- This homework is due Monday, 11/03/2014, 00:01 am.
- Late hand-ins will be accepted until the same day, 08:00 am. They will be penalized with 10% of the maximum achievable score.
- If you have any questions, you can use the Piazza forum for our class: https://piazza.com/uci/fall2014/141/home - please make sure to select the correct folder. Do not use Piazza to post any partial solutions.
- Include your name and your student id on all pages and in all source code files.
- The submission process is described below.

Introduction

- For writing these LISP functions, you can use a list environment such as *Steel Bank Common Lisp (SBCL)*, which is also preinstalled on the lab computers.
- You may use other development environments, but you need to make sure that your submission works with SBCL.
- In writing the following LISP functions, you may use only the primitive functions listed below.
- You may also write and use auxiliary functions, but you must submit these along with your solution.
- In later exercises, it may be helpful to reuse functions that you have written before.
- Some functions require subfunctions.
- You MUST use recursion instead of looping.
- DO NOT assume the lists are SIMPLE (i.e., every element is an atom) unless it is explicitly specified in the problem statement.
- EFFICIENCY NOTE: Do not flatten the list before processing.
- Primitive functions: defun, cond, cons, car, cdr, null, eq, listp, atom, symbolp, +, -, <, and >.
- **Include comments detailing how your functions work.**

Exercise 1 (2 Points)

Define a function `contains` that takes two parameters, a symbol A and a list of symbols L, and returns t only if L contains A as one of its elements. Examples:

```
(contains 'a nil)              --> nil
(contains 'b '(a b c))         --> t
(contains 'b '(a (((b))) c))   --> t
(contains 'd '(a b c))         --> nil
```

Exercise 2 (2 Points)

Define the function `insert-q` which takes an object O and a list L and returns the list L with O added to the end. Examples:

```
(insert-q 'a nil)        --> (a)
(insert-q 'd '(a b c))   --> (a b c d)
(insert-q '(a) '(b c))   --> (b c (a))
```

Exercise 3 (3 Points)

Define the function 'sub-pattern' that takes two SIMPLE lists pat and str and returns the sublist of str if pat is a substring of str. Otherwise it returns nil. Examples:

```
(sub-pattern '(a b s) '(c d b a s))           --> nil
(sub-pattern '(c a c) '(b a j a c a c t u s)) --> (c a c t u
s)
(sub-pattern nil '(a n y l i s t))            --> nil
(sub-pattern '(l i s p) nil)                  --> nil
```

Exercise 4 (2 Points)

Define the function 'mapping' that takes 2 arguments – a SIMPLE list L, and an integer value val. Every element of the list L is a list of two atoms - key and object. The function returns a list of objects whose key is less than val. Examples:

```
(mapping '((35 kim) (67 clinton) (45 emma))  40) --> (kim)
(mapping '((24 a) (15 b) (56 c) (19 d)) 26) --> (a b d)
(mapping '((90 a) (80 b) (70 c))  40) --> nil
```

Exercise 5 (2 Points)

Define the function 'first-atom' that takes a list L and returns the first atom of L. Examples:

```
(first-atom '((2 (1) 4) 6))   --> 2
(first-atom '((((s)) o )))    --> s
(first-atom '(1 (((2)) 3 4))) --> 1
```

Exercise 6 (2 Points)

Define the function 'rest-list' that takes a list L and returns a list that removes the first atom. Examples:

```
(rest-list '(1 2 3 4 5))     --> (2 3 4 5)
(rest-list '((1) 2 3))       --> (2 3)
(rest-list '(((a b) c) d e)) --> (((b) c) d e)
```

Exercise 7 (2 Points)

Define a function `find-all` that takes a symbol A and a list L and finds and returns the first symbol following each occurrence of A in L, or nil if A does not occur in L. Note that A may occur nested within L, possibly as the last element of a sublist. You may assume that there is always a symbol occurring afterwards. Examples:

```
(find-all 'a nil)                 --> nil
(find-all 'a '(b a c a e))        --> (c e)
(find-all 'a '(b d c e))          --> nil
(find-all 'a '(b (a a) c))        --> (a c)
(find-all 'a '((b a) ((c a b)))) --> (c b)
```

Exercise 8 (2 Points)

Define the function 'occ' that takes a list L and a symbol A and counts the occurance of symbol A in L. Examples:

```
(occ '((a c) c e) 'c)  --> 2
(occ '(((s) o ) d) 'f) --> 0
```

Exercise 9 (3 Points)

Define the function 'total-reverse' that takes a list L and returns the reverse of L including all sublists. Examples:

```
(total-reverse '(1 2 3 4 5))          --> (5 4 3 2 1)
(total-reverse '((1 2 3) 4 ((5 6)))) --> (((6 5)) 4 (3 2 1))
```

To be most efficient, you may internally define a stack and push each atom of L to that stack whenever you pop it from L.

Submission

Please submit your answers to the EEE Dropbox for this homework (e.g. COMPSCI 141 HW 4). You can find instructions on how to do this here:
https://eee.uci.edu/help/dropbox/students/#studentsubmit

Submit a single *.lisp file for this homework.