UCF

School of computer Science

COT 4020: Programming Languages

Fall 2006

Homework 1 (LISP)

Due on Thursday 03/02/06 by 11:59pm

Instructions

You must implement the following 5 functions in Common Lisp.

For this homework you are restricted to only using the following Common Lisp commands.

- CAR
- CDR
- COND
- NULL
- CONS
- DEFUN
- LIST
- ATOM
- LISTP
- EQ
- EQL
- EQUAL
- and the arithmetic expressions and comparators.

Make sure all the functions are named as specified in the assignment bellow. Put them all in a file named with your first initial concatenated with your last name, with a .lisp extension. So if you name is John Smith, then you will turn in a file called `jsmith.lisp`.

You will lose a letter grade if you do not follow instructions.

Email this file to `cmillward@cs.ucf.edu` no later than Thursday, March 2 at 11:59pm. The assignment will be accepted late for up to a day at a loss of 30%. Then after 11:59pm on Friday 03/03/06, the assignment will no longer be accepted and you will receive no credit.

Plagiary will not be tolerated. Feel free to discuss amongst yourselves, but do your own work. If you are not getting it, please make arrangements to see the TA (Chris), he can help explain things. The point of the homework is to learn to think recursively in Lisp.

Functions to implement

1.- Write a function called `SIMPLE-LENGTH` that returns the number of items in the top level of a list.

```
(SIMPLE-LENGTH '(a b c)) returns 3
(SIMPLE-LENGTH '(a (b c))) returns 2
(SIMPLE-LENGTH '(a b c (a (b c)))) returns 4
```

2.- Write a function called `COMPLEX-LENGTH` that returns the numbers of atoms in a list no matter how deeply nested they are.

```
(COMPLEX-LENGTH '(a b c)) returns 3
(COMPLEX-LENGTH '(a (b c))) returns 3
(COMPLEX-LENGTH '(a b c (a (b c)))) returns 6
```

3.- Write a function called `SWAP` that accepts two atoms (A and B) and a list and replaces every instance of A with B in the list

```
(SWAP 'a 'b '(a b c)) returns (b b c)
(SWAP 'a 'b '(a b (a))) returns (b b (b))
```

4.- Write a function called `MAX-DEPTH` that returns the level of the most deeply nested list in the input list.

```
(MAX-DEPTH '(a b c)) returns 1
(MAX-DEPTH '(a (b) c)) returns 2
(MAX-DEPTH '(a b (((c))))) returns 4
(MAX-DEPTH '((a (b)) (c))) returns 3
```

5.- Write a function called `SIMPLE-MERGE` that accepts two presorted (low-to-high) lists and returns a list that is the sorted concatenation of the the two input lists. Be sure to handle repeated numbers.

```
(SIMPLE-MERGE '(1 4 6) '(2 4 5)) returns (1 2 4 4 5 6)
(SIMPLE-MERGE '(2 8 12) '(24)) returns (2 8 12 24)
(SIMPLE-MERGE '(6 9 10) '(2 7 14)) returns (2 6 7 9 10 14)
```