CSCE476/876 **Fall 2018**

Homework 2: Learning Lisp with the Adaptive Remote Agent (II)

Assigned on: Friday, August 31^{st}, 2018.

Due: Monday, September 10^{th}, 2018.

Quiz: Monday, September 10^{th}, 2018 (in class).

As you may have guessed, this homework is the continuation of the previous one. The goal is that you learn Lisp by experimenting with the Adaptive Remote Agent available on the Web, courtesy of Professor Dr. Gerhard Weber:

`http://art2.ph-freiburg.de/Lisp-Course`

You are requested to login into the agent and run the *last three lessons* by Monday, September 10^{th}, 2018. You will be graded as follows:

1. (25 points) A quiz will be given in class with questions taken from the tutorial, randomly.

2. (20 points) A short but substantive analysis in which you state what you learned from the web agent, how will you learned, what you found most helpful and what you found most difficult in your interactions with the agent. You need to be specific in your evaluation. Comments such as "a cool tool" are not be accepted.

   You analysis should be submitted in electronic from and as an ASCII file using the webhandin system.

3. (5 points) Exercises are to be completed either typed and submitted with the webhandin system or with pen and paper and submitted physically in class.

4. (40 points) Programming Assignment:
   For each of the problems, create a separate lisp file. Name them problem1.lisp, problem2.lisp, and so on. Store all of your work on a given problem in the same file. When required to define several functions, put them all in the same file.

---

1 Exercise (5 Points)

1. **Describe what results from the following function:** **(5 points)**

```
(defun foo(s)
  (cond ((null s) 1)
        ((atom s) 0)
        (t (max (+ (foo (first s)) 1)
                (foo (rest s))))))
```

2 Programming (40 Points)

1. **Exponentiate** **(5 points)**
   Write the function `(power n m)` that raises and number `n` to an integer power `m`. For example, `(power 3 2)` should return `9`.

2. **Even numbers** **(5 points)**
   Common Lisp has built-in functions that can be used to test whether a value is even or odd. These functions are called `evenp` and `oddp`. Both function take a single integer argument. Experiment with them to see what they do. Write a function `(all-even list)` that will take a list of integers and return a list containing only the even integers. For example

   ```
   (all-evenp '(1 2 3 4 5 6 7 8 9 10))
   ```

   should return `(2 4 6 8 10)`. This can easily be done by using a loop to iterate across the list and using the `evenp` function to decide whether or not to save the current element.

3. **The `cond` conditional** **(10 points)**
   Review the syntax of the `cond` conditional operator. You will use it in this problem to handle a three case situation. Write a function `(what-is n)` that will return `ATOM` if the argument is an atom, `LIST` if the argument is a list, or `NUMBER` if the argument is a number. For example, `(what-is -91)` will return `NUMBER` and `(whatis '(1 2 3))` will return `LIST`. Use `cond` to test for which value to return.

4. **Learn to use `reduce`** **(10 points)**
   Find an on-line manual of Lisp, such as:
   `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/cltl/cltl2.html`
   `http://www.franz.com/support/documentation/6.2/ansicl/ansicl.htm`
   and study the definition and use of the function `reduce`. This is a particularly elegant and powerful construct (instructor's favorite). Using `reduce`, write a very short function that takes a list of numbers and returns the value of their average.

5. **Member** **(10 points)**

Common Lisp has a built-in function called `member`, which is called with the syntax

```
(member element list)
```

and will return `nil` if the `element` is not found in the `list`. If, on the other hand, the element is found in the list, the function will return a portion of the list, starting with the first occurrence of the element. For example, `(member 'b '(a b c d))` will return `(B C D)`. Also, observe that `(member 'b '(a b c a b c))` returns `(B C A B C)`. Experiment with the function, to be certain that you understand what it does.

(a) Write a function `(my-member-cond element list)` that duplicates the functionality of the built-in `member` function. Implement the function using `cond` and a recursive call.

(b) Write a function `(my-member-do element list)` that duplicates the functionality of the built-in `member` function. Implement the function iteratively, using the `do` primitive (see page 117 in your Lisp textbook).