Homework 5

Due Sunday, March 13 by 10:00pm

Handout 9
CSCI 334: Spring 2022

Turn-In Instructions

Turn in your work using the Github repository assigned to you. The name of the Github repository will have the form `cs334lab5_<your user name>`. For example, my repository would be `cs334lab5_dbarowy`. You should have received an invite to commit in the repository in your email. If you did not receive this email, please contact me right away!

Honor Code

This is a partner lab. You may work with another classmate if you wish, and you may co-develop solutions. Remember: although you can work on code together, you must each independently write up and submit your solution. No code copying is allowed. Be sure to tell me who your partner is by committing a `collaborators.txt` file to your repository (2 points).

Answers to problems should be typed using the LaTeX starter template in your repository.

Lisp files should have a "`.lisp`" suffix. For example, the pair programming problem **Q1** should appear as the file `q1.lisp`. Your code should be documented using comments. Comment lines start with a "`;`".

Honor code: You may collaborate with your partner on all aspects of this assignment, including discussing the problems, work on paper, and writing code. You may also work with students in the class other than your partner, but you may only discuss the problems at a high level (e.g., what the problems mean or general approaches to problems that do not involve code).

This assignment is due on Sunday, March 13 by 10:00pm.

Reading

1. No new reading for this assignment. Refer to prior readings and your class notes.

Problems

Q1. (18 points) .... Mapping Functions

Write a function `censor-word` that takes a word as an argument and returns either the word or `XXXX` if the word is a "bad" word:

```
* (censor-word 'lisp)
lisp

* (censor-word 'party)
XXXX
```

To write `censor-word`, you may find the following Lisp expression helpful:

```
(member word list)
```

`member` returns the sublist that includes `word` if `word` is in `list`, otherwise it returns `nil`. For example,

```
(member 'barrel '(fun as a barrel of lemonade))
```

evaluates to `(BARREL OF LEMONADE)` because `barrel` is in the word list.

Finally, use `censor-word` to define another function called `censor` that replaces all the bad words in a sentence:

```
* (censor '(I need an extension because I got lost in the steam tunnels))
(I NEED AN XXXX BECAUSE I GOT LOST IN THE XXXX XXXX)

* (censor '(I like programming languages more than a party on Hoxsey))
(I LIKE PROGRAMMING LANGUAGES MORE THAN A XXXX ON XXXX)
```

The bad word list can be anything you want, but to make it easy for TAs to grade your assignment, I suggest that you make your code reproduce the examples above.

Operations that are carried out on every element of a structure are typically written using a function from the `MAP` family of Lisp functions. In some ways, mapping functions are the functional programming equivalent of a "for loop", and they are now found in main-stream languages like Python, Ruby, and even Java. Be sure to use a mapping function in your definition of `censor`.

Q2. (20 points) .... Working with Structured Data

This part works with the following database of students and grades:

```
;; Define a variable holding the data:
* (defvar grades '((Riley (90.0 33.3))
                   (Jessie (100.0 85.0 97.0))
                   (Quinn (70.0 100.0))))
```

First, write a function `lookup` that returns the grades for a specific student:

```
* (lookup 'Riley grades)

(90.0 33.3)
```

It should return nil if no one matches.

Now, write a function `averages` that returns the list of student average scores:

```
* (averages grades)

((RILEY 61.65) (JESSIE 94.0) (QUINN 85.0))
```

You may wish to write a helper function to process one student record (ie, write a function such that `(student-avg '(Riley (90.0 33.3)))` returns `(RILEY 61.65)`, and possibly another helper to sum up a list of numbers). As with `censor` in the previous part, the function `averages` function is most elegently expressing via a mapping operation (rather than recursion).

We will now sort the averages using one additional Lisp primitive: `sort`. Before doing that, we need a way to compare student averages. Write a method `compare-students` that takes two "student/average" lists and returns true if the first has a lower average and `nil` otherwise:

```
* (compare-students '(RILEY 61.65) '(JESSIE 94.0))
t

* (compare-students '(JESSIE 94.0) '(RILEY 61.65))
nil
```

To tie it all together, you should now be able to write:

```
(sort (averages grades) #'compare-students)
```

to obtain

```
((RILEY 61.65) (QUINN 85.0) (JESSIE 94.0))
```

Q3. (30 points) .... Partial and Total Functions

For each of the following function definitions, (i) give the graph of the function. Say whether this is a (ii) partial function or a total function on the integers. If the function is partial, say where the function is (iii) defined and where it is (iv) undefined.

For example, take the function $\texttt{f(x)} = \texttt{if x} > \texttt{0 then x} + \texttt{2 else x/0}$

The graph of this function is the set of ordered pairs $\{\langle x, x+2\rangle \mid x > 0\}$. The function is partial. It is defined on all integers greater than 0 and undefined on integers less than or equal to 0.

Functions:

(a) $\texttt{f(x)} = \texttt{if x} < \texttt{10 then 0 else f(x} - \texttt{2)}$
(b) $\texttt{f(x)} = \texttt{if x} + \texttt{3} > \texttt{3 then x} + \texttt{4 else x/0}$
(c) $\texttt{f(x)} = \texttt{if sin(x)} > \texttt{0 then 1 else f(x} + \pi\texttt{)}$

Q4. (20 points) .... Detecting Errors

Evaluation of a Lisp expression can either terminate normally (and return a value), terminate abnormally with an error, or run forever. Some examples of expressions that terminate with an error are `(/ 3 0)`, division by 0; `(car 'a)`, taking the `car` of an atom; and `(+ 3 "a")`, adding a string to a number. The Lisp system detects these errors, terminates evaluation, and prints a message to the screen. Suppose that you work at a software company that builds software using Impure Lisp. Your boss wants to handle errors in Lisp programs without terminating the entire computation, but doesn't know how.

(a) You boss asks you to implement a Lisp construct `(error E)` that detects whether an expression `E` will cause an error. More precisely, your boss wants evaluation of `(error E)` to

i. halt with value `t` if evaluation of `E` would terminate in error, and
ii. halt with value `nil` otherwise.

Explain why it is not possible to implement the `(error E)` construct. If you find it helpful to write code to answer this question, please do.

(b) After you finish explaining why `(error E)` is impossible, your boss proposes an alternative. Instead, your boss wants you to implement a Lisp construct `(guarded E)` that either executes `E` and returns its value, or if `E` halts with an error, returns 0 without performing any side effects. This could be used to try to evaluate `E`, and if an error occurs, to use 0 instead. For example,

```
(+ (guarded E) E2)
```

will have the value of `E2` if evaluation of `E` halts in error, and the value of `E + E2` otherwise. Observe that unlike `(error E)`, evaluation of `(guarded E)` does not need to halt if evaluation of `E` does not halt.

i. How might you implement the `guarded` construct?
ii. What difficulties might you encounter? Keep in mind that your company uses Impure Lisp.

Q5. (10 points) .... Garbage Collection

A garbage collection algorithm performs automatic cleanup of unused memory in a program. Modern programming language runtimes routinely perform garbage collection in order to dramatically simplify memory management. Garbage has the following definition.

> At a given point i in the execution of a program P, a memory location m is garbage if continued execution of P from i **will not** access location m again.

Nonetheless, garbage collection using the above definition of garbage is not computable. Instead, languages solve a simpler problem by using a slightly different definition of garbage:

> At a given point i in the execution of a program P, a memory location m is definitely garbage if continued execution of P from i **cannot** access location m again.

McCarthy's "mark sweep" algorithm uses this latter definition, because it only reclaims memory that is impossible to re-read.

Prove that garbage collection using the first definition is not computable. You should prove this fact using the "reductio ad absurdum" proof technique. Specifically, your proof should employ a reduction of another non-computable function to garbage collection. For example, you may rely on the fact that we know that the halting problem is not computable.

Assume that you have the following `isGarbage` function available in the standard library of the programming language of your choice.

```
boolean isGarbage(String p, String m, int i)
```

Calling `isGarbage` with the source code for program text `p`, variable name `m`, and line number `i` has the following behavior.

`isGarbage(p, m, i)`	returns `true` if `m` is garbage at line `i` of program `p`.
`isGarbage(p, m, i)`	returns `false` otherwise.

You may assume that `isGarbage` always halts. You may also assume that `p` is "simple" code that does not contain class or function definitions.

Q6. (1 bonus point) .... Optional: Feedback

How hard was this assignment on a scale of 1 to 5? (where 1 is easy and 5 is difficult).

Do you have any additional comments or feedback that you would like me to know?

Please supply your answer as a `feedback.txt` file.