Lab 1 – Instructions

This lab is a little different from the others in that it has two parts: the first one is about working through a little bit of online material to learn the basics of Clojure, the language we will be using for our programming labs.

The second is about using that knowledge to work on a small programming assignment, which is about completing a small toolbox for finite relations and functions in Clojure.

Note that with all programming assignments in this course, efficiency is of no concern to us. The goal is to produce a solution that expresses the algorithm in a reasonable manner and computes the correct result.

The first step, however, is to install Clojure.

You will find some of the following material also on the course page for Clojure:
http://cs.lth.se/edaa40/clojure/

0. Installing Clojure

We will use Leiningen, a distribution of Clojure, throughout the course. You can find it here:

https://leiningen.org/

1. Introduction to Clojure

Clojure is a Lisp-like language built on top of the Java Virtual Machine. In this course, we will use it for the programming assignments because of its relative simplicity and good support for the kinds of data structures we are interested in here.

If you have never used Clojure, I recommend working through the very nice tutorial by Kyle Kingsbury, which you find here:

https://aphyr.com/tags/Clojure-from-the-ground-up

If you want to prepare using this tutorial, you should work through the following parts:

1. Welcome. All of this.
   https://aphyr.com/posts/301-clojure-from-the-ground-up-welcome
2. Basic types. All of this, too.
   https://aphyr.com/posts/302-clojure-from-the-ground-up-basic-types
3. Functions. Except "How does type work", we won't need this in this course.
   https://aphyr.com/posts/303-clojure-from-the-ground-up-functions

Go to the course page for links to other material, including an **extremely useful** cheat sheet for Clojure.

2. A small toolbox for relations and functions

This lab is about a small toolbox for (finite) relations and functions in Clojure. We will represent a relation, and therefore also a function, as a (Clojure) set of vectors of length 2. So for example, the relation $\{(1,1),(2,4),(3,9)\}$ is represented as the Clojure structure

```
#{ [1 1] [2 4] [3 9] }
```

Using this representation, the task will be to implement a few Clojure functions working with relations and functions. These are the steps:

1. Download the skeleton project and unpack it.

2. cd into its top-level directory, `edaa40lab1`. Start the Leiningen REPL there.

3. Have a look at the file

```
edaa40lab1/src/edaa40/lab1.clj
```

It contains the source code of the toolbox, with some functions commented out. I have removed their bodies and replaced them with a comment listing some functions you might find useful implementing it (either Clojure functions, or from this toolbox), and sometimes a hint intended to guide you in the right direction.

Your task is to implement all the functions that have been commented out. Do not change the name or the parameter list, just add the function body. Every commented-out function is preceded by a **declare** statement. Its purpose is to keep the compiler calm and allow you to load the package in spite of the fact that some functions initially remain undefined. If you try to call one of the declared-but-not-defined functions, you will get an error. After you have uncommented and implemented the function, you can remove the declare or just leave it there.

Right after each of the commented-out function skeletons are one or more **test?** statements. Their purpose is to help you check whether you are on the right track. Once you have an implementation, uncomment these tests, reload the package from the REPL (see below), and if the test passes (you will see something printed out on the REPL console) you might just have done it right. (As you can see, the tests are hardly exhaustive.)

From the Leiningen REPL you can load (and reload) the package using

```
(use 'edaa40.lab1 :reload)
```

while you work on the code and try it out.

Tip 1: It makes sense to look at the other code in the file. Some functions are pretty similar to the ones your are asked to implement.

Tip 2: Many commented-out functions include a comment like ";; uses ...", listing a few functions you may find useful implementing the function you are asked to implement. Note these are suggestions; it's definitely possible to realize the function without them. But if you get stuck, looking at the documentation for the suggested functions might help you find a solution.

Tip 3: It is a **really good idea** to implement one function at a time, uncomment its tests, and then reload the package to see how you are doing.

4. Once you are done, all functions are implemented, all the tests during loading pass, exit the REPL (using exit or CTRL-D). Without leaving the edaa40lab1 directory, type

```
lein test
```

This runs a few tests on your code. If you want to peek under the hood and see how the tests are done, check out the test harness in

```
edaa40lab1/test/edaa40/test/lab1.clj
```

If all goes well, have your work looked at by the person supervising the lab. Congratulations!

3. (optional) Using the toolbox

Try to use the toolbox to work on some of the problems from the exercises, at least those that are concerned with finite sets and relations.