

CS280
Lab-4: Data Abstraction
March 7, 2005

In this lab, we are going to develop a small rational arithmetic package; the process is very similar to developing the floating point arithmetic package that you will be doing for the first part of the project. **Therefore doing this lab is VERY important!**

First copy the following files into your own directory, so that you will be able to work with them.

```
~carberry/rationalcons  
~carberry/rationalexp  
~carberry/rationalprocall
```

File `rationalcons` contains the constructors that we discussed in class for representing rational numbers as pairs. File `rationalprocall` contains some procedures for computing with rational numbers; in particular, it contains the procedures that we discussed in class for adding rational numbers and printing them as fractions.

Enter scheme and load files `rationalcons` and `rationalprocall` that you have copied to your directory. Now do the following operations and examine the results carefully — make certain that you understand what is displayed and why.

```
(define r1 (make-rat 2 5))  
(define r2 (make-rat 3 4))  
r1  
r2  
(print-rat r1)  
(print-rat r2)  
(+rat r1 r2)  
(print-rat (+rat r1 r2))
```

You are going to do two things in lab:

1. Extend the rational arithmetic package given in file `rationalprocall` to include two additional procedures. The first, `reciprocal`, will take a rational number as argument and return its reciprocal as a rational number. The second, `square` will take a rational number as argument and return its square as a rational number. The procedures are outlined for you in file `rationalprocall`.
 - Complete these two procedures. Remember that the procedures for computing with rational numbers in file `rationalprocall` should not care what the underlying representation of rational numbers is.

- Test your two new procedures to make sure that they work. You can use the following tests:

```
(define r1 (make-rat 2 5))
(define r2 (make-rat 3 4))
(print-rat (reciprocal r1))
(print-rat (reciprocal r2))
(print-rat (square r1))
(print-rat (square r2))
```

- Notice that you have made no changes to file *rationalcons* that contains your constructors — or at least you shouldn't have. You have separated your rational arithmetic package from the implementation/representation and are therefore practicing data abstraction.
2. You have decided that for some reason you do not want to represent rational numbers as pairs, but instead as an integer whose value is

$$2^n 3^d$$

where n and d are the numerator and denominator of your rational number. Design the new constructors and selectors: they will still be called `make-rat`, `numer`, and `denom`, but now `(make-rat 4 1)` will return the integer 48 as the representation of the rational number whose numerator is 4 and denominator is 1, since $2^4 3^1$ is 48. Similarly, `numer` will be given an integer such as 48 and will have to return the numerator of the rational number that it represents. You can do this by counting how many times 2 exactly divides the integer passed to `numer` as its argument. Similarly, `denom` can compute the denominator of the rational number represented by its argument by counting how many times the argument is exactly divisible by 3.

- Implement these constructors and selectors by completing the procedures outlined for you in file `rationalexpr`.
- Now load files `rationalexpr` and `rationalprocall` into scheme. Run the same test cases that were given earlier (don't change them at all) and note the results — make sure that you understand what is printed and why.
- Notice that you have changed the implementation/representation of rational numbers by using the constructors and selectors that you defined in file `rationalexpr` but that this change in how rational numbers are represented has had no affect on your procedures in file `rationalprocall` for computing with rational numbers — at least they shouldn't have.

Copy the code in your files `rationalexpr` and `rationalprocall` into the lab submission system and submit it for Lab-4.

The first part of the project will be conceptually quite similar. You will be developing a package of procedures for computing with floating point numbers and several different representation packages, and you will demonstrate that your package of procedures for operating on floating point numbers works equally well with any of the different representation packages — ie., your procedures for computing with floating point numbers do not rely on a specific underlying representation.