Homework 3: Data Abstraction and Numbers

Due:

All the directions from homework 2 apply to this homework. In particular, remember to hand in a printout for procedures, and either a calculation or a transcript showing your testing.

1 Section 3.2, Operations on Numbers

1. (0 points; suggested practice) Do problem 3.1. Check your answer on the computer.

2. (0 points; suggested practice) Do problems 3.2, 3.3, and 3.4.

   You can think of the n-tuples discussed in these exercises as the vectors of physics. (Especially if the n-tuple has 3 elements.) So `pairwise-sum` is vector addition, `dot-product` forms the scalar product of 2 vectors, and `mult-by-n` multiplies a vector by a scalar.

3. (6 points) In Physics, the laws of mechanics are summarized by the vector equation $\vec{F} = m\vec{a}$, where $\vec{F} = (F_x\ F_y\ F_z)$ is a 3-tuple, representing the force vector, m is the mass, and $\vec{a} = (a_x\ a_y\ a_z)$ is the acceleration vector. Write a procedure, `force`, that takes a number, `m`, a 3-tuple of numbers, `a`, and returns the 3-tuple of numbers $(m \times a_x \quad m \times a_y \quad m \times a_z)$ that represents the force vector. For example,

   ```
   (force 3.0 '(5.2 3.1 8.7)) ==> (15.6 9.3 26.1)
   (force 72 '(-8.2 -3.14 -32.0)) ==> (-590.4 -226.08 -2304.)
   ```

   Note that your answers will not be exact if you use inexact (floating point) numbers, but that is okay.

   By the way, Geordi La Forge (of Star Trek the next generation) wants you to be sure that he can use your program when they enter universes with different dimensions. Would your program work for 4-dimensional worlds? Or 8-dimensional universes? (That is, would it work for 4- or 8-tuples as well as for 3-tuples?) If it doesn't, fix it so it does.

4. (5 points) Do problem 3.5.

5. (7 points) Do either problem 3.6 or problem 3.7 in the text. You choose which one of these you want to do; only hand one of these in.

6. (8 points; basic) Define a procedure `list-tail` that takes a list, `ls`, and a nonnegative integer `num` and returns the last `num` top-level items in `ls`. If `num` is larger than the number of top-level items in `ls`, `list-tail` signals an error. To signal the error call the procedure `error` as follows.

   ```
   (error "length of " ls " is less than " num ".")
   ```

   Signalling an error does not return the message as a value, but merely prints it, and escapes to the Scheme system without returning any value. For example,

   ```
   (list-tail '(a b c d e f g) 4) ==> (d e f g)
   (list-tail '(1 2 3 4) 4) ==> (1 2 3 4)
   (list-tail '(strange and wonderful) 0) ==> ()
   (list-tail '(a b c) 4)  =signals=> Error: length of (a b c) is less than 4
   ```

The procedure **error**, from page 199 of the text, is automatically loaded into Chez Scheme on Project Vincent (if you did homework 0 correctly). If you are using PC Scheme, you may have to load that yourself.

7. (0 points; suggested practice) Do (one or more of) problems 3.10, 3.11, and 3.12. Check your answer on the computer.

8. (5 points; extra credit only) Do problem 3.13.

9. (extra credit) James Gleick's book *Chaos: Making a New Science* (Penguin Books, 1987) describes the emerging science of "chaos theory," which is largely concerned with nonlinear systems.

   An early inspiration for chaos theory comes from biology. (See pages 62-64 of Gleick's book.) Biologists use the logistic difference equation to model the growth of a population over time. For simplicity, one can represent a population, say of fish in a pond, as a fraction between 0 and 1. The fraction 0 represents extinction, while 1 represents the greatest conceivable population of fish that will fit in the pond. A simple linear function $L(x) = rx$ that gives the population for the next year in terms of the current year's population, x, is inadequate, since it does not take limited resources into account. Instead, some biologists in the 1950s started to use a non-linear equation: the logistic difference equation:

   $$P_r(x) = rx(1-x) \tag{1}$$

   This equation gives next year's population $P_r(x)$ in terms of this year's population x, a parameter r that represents the amount of "nonlinearity", and a factor $1 - x$ that keeps the population from getting too big.

   For example, if r is 2.7 and the population at year 0 is 0.02. The population at year 1 is $0.05292 = 2.7 \times 0.02 \times (1 - 0.02)$. The population at year 2 is about 0.1353. If you keep computing, you'll see that the population eventually reaches an equilibrium of about 0.6296.

   To describe the process more concisely, I'll introduce a bit of notation. The notation is for iterations of a function P. Let $P^{(0)}(x) = x$, and for all integers $k > 0$, let $P^{(k)}(x) = P(P^{(k-1)}(x))$. The claim above is that for sufficiently large k and $x > 0$, $P_{2.7}^{(k)}(x) \approx 0.6296$.

   Does iterating the logistic difference equation always converge on an equilibrium population? Surprisingly, the answer is no! This is what you will investigate in this problem.

   (a) (1 point; extra credit) Write a procedure **logistic-diff** that takes a value for r and x and returns $P_r(x)$.

   Note, use inexact (floating point) numbers for this problem.

   (b) (2 points; extra credit) Add error checking to **logistic-diff**, that is, print an appropriate error message if the r and x arguments have values they shouldn't.

   (c) (4 points; extra credit) Write a procedure **iterate-logistic-diff** that takes r, k, and x, and returns the list

   $$(P_r^{(0)}(x) \;\; P_r^{(1)}(x) \;\; \ldots \;\; P_r^{(k)}(x)).$$

   Use your program to try larger and larger numbers for r. (Don't start with 100; start small, say around 2 and increase slowly so you can see what happens.) Can you see where the iterates no longer reach equilibrium? What do the iterates look like? What happens if you keep going higher?

   (d) (20 points; extra credit) Write a procedure that searches for the smallest r for which P_r does *not* reach equilibrium for a fixed x. Get an answer to 2 decimal places accuracy. (The value of x will be an input.) You'll have to develop some test for "reaching equilibrium". Be sure to watch out for infinite loops!

2 Section 3.3, Exact Arithmetic and Data Abstraction

In this section your code should be written at the appropriate level of detail. That is, unless you are implementing one of the basic procedures for rationals, your code should work for any representation of rationals. (This property is called representation independence.) If your code is too low-level, you will lose points (and you'll have a harder time writing it too).

You will find it useful to use `rprint` in testing your code. You can load the code from chapter 3 by typing `(load "/home/cs227/lib/ch3.ss")` to the Scheme interpreter. To test whether your procedures are written at the right level, you can load one or more of the files `rational4.ss` or `rational3.ss` in the directory `/home/cs227/lib`.

10. (0 points; suggested practice) Do problems 3.14 and 3.15. Check your answers the computer.

11. (5 points; basic) Do problem 3.16.

```
(rprint (rabs (make-ratl 2 3))) =prints=> 2/3
(rprint (rabs (make-ratl -7 3))) =prints=> 7/3
(rprint (rabs (make-ratl 9 -7))) =prints=> 9/7
```

12. (0 points; suggested practice) Write a procedure, `rational->inexact`, that takes a rational number `r` and returns a floating point number that approximates `r`. Note that adding `0.0` to an integer makes it a floating point number.

13. (8 points; basic) Do problem 3.17.

14. (7 points) The "golden ratio" is a number that is found in art, biology, and architecture. For example, the ratio of the breadth to the height of the Parthenon is the golden ratio. The golden ratio is the positive solution to the equation

$$x = 1 + \frac{1}{x}.$$

Substituting the right hand side for the x on the right hand side gives us the formula.

$$x = 1 + \frac{1}{1 + \frac{1}{x}}.$$

If we keep going we get that

$$x = 1 + \frac{1}{1 + \frac{1}{1+\frac{1}{x}}}.$$

If we keep this up forever, we get what mathematicians call a *continued fraction.*

$$x = 1 + \frac{1}{1 + \frac{1}{1+\frac{1}{1+\frac{1}{1+\cdots}}}}.$$

This problem calls for you to make a rational number approximation to the golden ratio. That is, you are to use the procedures for rational numbers to compute a rational. Specifically, you are to write a procedure, `golden-ratio`, that takes as it input a nonnegative integer, `k`, and returns a level `k` approximation to the golden ratio as a rational number. A level 0 approximation is defined as $1/1$; and for $k > 0$, a level k approximation is defined as $(1/1) + (1/z)$, where z is a level $k - 1$ approximation. For example,

```
(rprint (golden-ratio 0)) =prints=> 1/1
(rprint (golden-ratio 3)) =prints=> 5/3
(rprint (golden-ratio 11)) =prints=> 233/144
```

15. (3 points; extra credit) Adriana Pythagoras thinks the previous problem is silly. "Why, you can just solve the equation!" she says. Do that using the form of calculations we have used in class, to find the exact value of the golden ratio.

16. (5 points; extra credit) This problem is intentionally ill-specified. My video cassette recorder (VCR) has two speeds. Recording in "single play mode" (SP) a tape runs out in 2 hours. Recording in "extended play mode" (EP) a tape runs out in 6 hours. I often like to tape broadcasts of operas, and want to get the entire opera on one tape. My strategy is to program the VCR to record for as long as possible in single play mode, and then to switch to extended play mode for the rest of the opera. Write a procedure `recording-times` that takes the time of the broadcast, in minutes, and returns a list of the form, `((SP` *numS*`) (EP` *numE*`))` where *numS* is the number of minutes I should record using single play mode, and *numE* is the number of minutes I should record using extended play mode. Note that the sum of *numS* and *numE* should be the time of the broadcast. Note any additional assumptions you make in solving this problem.