Homework 1 — prerequisites — assigned 21 January — due 25 January

1.1 Scheme: basic concepts (10pts)

Give the value which Scheme would return if you entered the following expressions. If it would produce an error, state what the nature of the error is. If it evaluates to #<procedure>, say so.

- (apply * (map eval '((+ 1 2) 4)))
- (let ((a 0) (b 1)) (let ((b a) (a b)) (cons b a)))
- (equal? (cons () ()) (list ()))
- (apply (lambda args '(1 2 3)) '(4 5 6))
- (apply null? ())
- ((lambda args args) 1 2 3 4)
- (map (lambda (x y) (x y)) '((lambda (x) x) (lambda (y) y)) '(x y))
- (apply cons (list 1) (list 2))
- ((lambda (x) x) (lambda (x) x))
- (apply (lambda (x) x) ((lambda (x) x) (list (lambda (x) x))))

1.2 Scheme: basic concepts (10pts)

The function *iterate* takes a function, f, of one argument as its first argument, and an integer, $n \geq 0$, as its second argument. It returns f composed with itself *n* times:

$$\underbrace{f(f(...))}_{n \text{ times}}$$

The function *rotate* takes a list, *ls*, and a positive integer, *n*, as arguments. It returns a list with all elements of *ls* rotated to the right *n* times. For example, *(rotate '(1 2 3 4) 1)* should return *(2 3 4 1)* and *(rotate '(1 2 3 4) 2)* should return *(3 4 1 2)*. Write *rotate* using *iterate* and *iterate* using *compose*. Finally, write *compose*.

1.3 Scheme: program understanding (10pts)

Consider the following functions:

```
(define foo
  (lambda (key ls)
    (if (null? ls)
        #f
        (let ((x (car ls)))
          (if (eq? key (car x))
              x
              (foo key (cdr ls)))))))
(define bar
  (lambda (n)
    (letrec
      ((loop
         (lambda (x acc)
           (if (= x 0)
               acc
               (loop (- x 1) (cons (- x 1) acc))))))
        (loop n ()))))

(define mystery
  (lambda (ls n)
    (cdr (foo n (map cons (bar (length ls)) ls)))))
```

Describe in your own words what *foo* and *bar* do. Give an example input and output for each. Finally, rewrite *mystery* without using *foo* or *bar*.

1.4 Scheme: higher-order functions (10pts)

Consider the function:

$$g(n) = n^n = \prod_{i=1}^{n} n = \underbrace{n \cdot n \cdot \ldots \cdot n}_{n \text{ times}}$$

Now consider the function:

$$f(m) = \sum_{n=1}^{m} g(n) = 1 + 2 \cdot 2 + 3 \cdot 3 \cdot 3 + \ldots + \underbrace{n \cdot n \cdot \ldots \cdot n}_{n \text{ times}}$$

Write f and g without using explicit recursion (and without using *expt*), *i.e.*, using *apply*, *map*, *filter*, *iterate*, and *iota*. Hint: Divide and conquer. Use g in your solution to f.

1.5 Java: using a data abstraction (30pts)

The following is a specification of a data abstraction for handling sets of integers:

```
public class IntSet
{
  // OVERVIEW: IntSets are mutable, unbounded sets of integers.
  // A typical IntSet is {x_1, ..., x_n}.

  // constructors
  public IntSet ()
    // EFFECTS: Initializes this to be empty.

  // methods
  public void insert (int x)
    // MODIFIES: this
    // EFFECTS: Adds x to the elements of this , i.e., this_post = this ∪ { x }.

  public void remove (int x)
    // MODIFIES: this
    // EFFECTS: Removes x from this , i.e., this_post = this \ { x }.

  public boolean isIn (int x)
    // EFFECTS: If x is in this returns true else returns false.

  public int size ()
    // EFFECTS: Returns the cardinality of this.

  public int choose () throws EmptyException
    // EFFECTS: If this is empty, throws EmptyException else
    //    returns an arbitrary element of this.
}
```

In this specification, `this` refers to the state of the receiver object *before* the method is invoked, and `this_post` refers to the state of the receiver object *after* the method has been executed. When a method does not modify anything, the specification omits the MODIFIES clause.

Write a class `Prime` that uses the `IntSet` abstraction as follows.

The class `Prime` has a static method `makeBigger`. The method `makeBigger` accepts as argument an `IntSet` that must contain prime numbers alone, and it returns a distinct new `IntSet` that contains all the elements that the argument `IntSet` contains and one additional element, an `int` which is a prime number greater than any of the numbers in the argument `IntSet`.

You may assume that one such `int` exists—in other words, you may assume that there is a prime number greater than any in the original set, but not greater than `Integer.MAX_VALUE`. You may not assume anything else.

Write a specification of the method `makeBigger` carefully, in the style of the specification of `IntSet`, and then implement it. If you wish you may write auxiliary methods in class `Prime`. The entire class `Prime` and any auxiliary methods you write should also be accompanied by specifications. Remember to use appropriate access control modifiers (`public`, `private`).

Write a class `Tester` that will invoke `makeBigger` on various test examples.

Note: if you are not familiar with Java, you may do this exercise in another object-oriented programming language from the following list: C++, Eiffel, Smalltalk, Objective C.

1.6 C programming (30pts)

Write a program in C with the same functionality as the Java program of the preceding exercise.

There must be three `.c` files, `IntSet.c`, `Prime.c`, and `Tester.c`, as well as corresponding `.h` files. You may write additional files as needed.

Make sure that you carefully document how you are representing the object-oriented Java code in the procedural style of C; in particular, how object state is represented and how method invocations are represented.

How to turn in

Turn in your code by running

~clint/handin your-file

on a regular UNM CS machine. You should use whatever filename is appropriate in place of your-file.

Include the following statement with your submission, signed and dated:
I pledge my honor that in the preparation of this assignment I have complied with the University of New Mexico Board of Regents' Policy Manual, including Section 4.8, Academic Dishonesty.