Spring, 1992 Name: ______________________

CS 342 — Principles of Programming Languages

Smalltalk Test (Problem Solving, Version 1)

See the separate directions page for explicit directions on what to hand in, how to hand it in, and for help with terminals.
An ASCII form of this test is found in the file

`/home/cs342/public/tests/testf.txt`

To get the code examples included in this test, execute the following command from your home directory:

`cp /home/cs342/public/tests/testf.smt inputs.smt`

1. (15 points) Write a global function (not in any class) `equal` that takes two objects of class `Set` and returns 1 (true) if they have the same elements, and 0 (false) otherwise. The elements of the two set arguments should remain unchanged. You may assume that the sets only contain symbols or numbers. You may also assume that the arguments are not aliased.

   For example, consider the following session, where only the "interesting" results printed by the interpreter are shown. (That is, outputs of <userval> and undefined outputs are omitted.) Note also that the order of outputs from `printColl` applied to a set is undefined, so the order of your results may differ.

```
-> (set myset (mkSet))
-> (printColl myset)
0

-> (set aset (mkSet))
-> (equal myset aset)
1

-> (add: aset #fun)
-> (equal myset aset)
0

-> (printColl aset)
fun
0

-> (equal aset myset)
0

-> (add: myset #fun)
-> (add: myset #is)
-> (add: myset #smalltalk)
-> (add: aset #fun)
-> (add: aset #smalltalk)
-> (equal myset aset)
0

-> (equal aset myset)
0
```

```
-> (add: aset #is)
-> (equal myset aset)
1
```

2. (20 points) Define a class `Record` as a subclass of some existing class in the textbook's collection hierarchy (see section 7.1.5). (That is, you may *not* make `Record` a subclass of `Object`.) Objects of the class `Record` can be thought of as similar to Pascal Records. Besides the usual `mkRecord` global function you should define methods to support the following protocol:

   - `at:`, which takes an additional argument of a symbol, returns the value of the field named by that symbol in the record itself. You may assume that this method is only called with a symbol that is a defined field name for the object.
   - `at:put:`, which takes two additional arguments, a symbol naming a field and a value, mutates the record's state so that the value of the named field is the given value. You may assume that this method is only called with a symbol that is a defined field name for the object.
   - `equal:`, which takes an additional argument, which can be assumed to be a record with the same field names, such that all the values of the fields in either record are symbols or integers. It returns 1 (true) if the corresponding fields are equal, and 0 (false) otherwise.

   Records do not have to support the `first` and `next` protocol of collections, nor do they have to support the `add:` protocol of collections.

   The `mkRecord` global function takes two lists of the same length: a list of field names (symbols) and a list of values; it returns a new record that has the given field names as its values.

   For example, consider the following session, where only the "interesting" results printed by the interpreter are shown. (That is, most outputs of <userval> and undefined outputs are omitted.)

```
-> (set fields (mkList))
-> (add: fields #xcoord)
-> (add: fields #y-coord)
-> (set values (mkList))
-> (add: values 3)
-> (add: values 4)
-> (set myrec (mkRecord fields values))
<userval>

-> (at: myrec #xcoord)
3

-> (at: myrec #y-coord)
4

-> (set fields2 (mkList))
-> (add: fields2 #y-coord)
-> (add: fields2 #xcoord)
-> (set values2 (mkList))
-> (add: values2 4)
-> (add: values2 3)
-> (set myrec2 (mkRecord fields values))

-> (at: myrec2 #xcoord)
3

-> (equal: myrec myrec2)
1

-> (at:put: myrec2 #xcoord 7)
-> (at: myrec2 #xcoord)
7
```

```
-> (equal: myrec2 myrec)
0
```