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/testm.txt`

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

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

1. (15 points) Write a global function (not in any class) `difference` that takes two objects of class `Set` and returns a new `Set` object such that `(difference s1 s2)` is a set that contains just the elements that are in `s1` but not in `s2`. The elements of `s1` and `s2` should remain unchanged. You may assume that `s1` and `s2` only contain symbols or numbers.

   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))
-> (set res1 (difference myset aset))
-> (printColl res1)
0

-> (add: aset #fun)
-> (set res2 (difference myset aset))
-> (printColl res2)
0

-> (printColl aset)
fun
0

-> (set res3 (difference aset myset))
-> (printColl res3)
fun
0

-> (printColl aset)
fun
0

-> (add: myset #fun)
-> (add: myset #is)
-> (add: myset #smalltalk)
```

```
-> (add: aset #fun)
-> (add: aset #smalltalk)
-> (set res4 (difference myset aset))
-> (printColl res4)
is
0

-> (set res5 (difference aset myset))
-> (printColl res5)
0
```

2. (20 points) Define a class `String` as a subclass of some existing class in the textbook's collection hierarchy (see section 7.1.5). (That is, you may *not* make `String` a subclass of `Object`.) Objects of the class `String` can be thought of as a collection of symbols, where each symbol is expected to be one character long. The protocol supported by `String` objects should be that of a `SequenceableCollection`, with the result of the `firstKey` method always returning 1; however, you do not have to support the method `add:`. In addition a `String` object should understand the method `equal:`, which compares self to another `String` and returns true if they have the same elements in the same order. (You may assume that the additional argument passed to `equal:` is not the same object as the receiver.) The global function `mkString` should take 2 arguments, the length of the new string and a symbol to use for each of the elements, and return a string containing that many copies of the symbol.

   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 es (mkString 0 #c))
<userval>

-> (isEmpty es)
1

-> (set it (mkString 2 #x))
<userval>

-> (isEmpty it)
0

-> (firstKey it)
1

-> (printColl it)
x
x
0

-> (at: it 1)
x

-> (at:put: it 2 #y)
-> (at: it 2)
y

-> (printColl it)
x
y
0

-> (set s2 (mkString 2 #x))
<userval>

-> (equal: it s2)
0

-> (at:put: s2 1 #y)
-> (equal: it s2)
```

```
0

-> (at:put: s2 1 #x)
-> (at:put: s2 2 #y)
-> (equal: s2 it)
1

-> (equal: es (mkString 0 #a))
1
```