CSc 372, Fall 2006
Assignment 5, Part 2
Due: Thursday, November 2 at 23:59:59 MST

Pair Programming

Subject to the rules described in part 1 of this assignment you may work in pairs on all problems on this assignment except for `speedup.rb` and `answers.txt`. You may switch pairs from problem to problem, pair on some problems and not others, or pair with the same partner on every problem.

Problem 2. (16 points) `xstring.rb`

Implement a hierarchy of three Ruby classes: `XString`, `ReplString`, and `MirrorString`.

`XString` serves as an abstract superclass of `ReplString` and `MirrorString`; it simply provides some methods that are used by `ReplString` and `MirrorString`.

`ReplString` represents strings that consist of zero or more replications of a specified string. Example:

```
>> s1 = ReplString.new("abc", 2)      => ReplString("abc",2)
```

A handful of operations are supported: `size`, `length`, `[n]`, `[n,m]`, `inspect` (used to show results in `irb`), `to_s`, and `each`. The semantics of `[n]` and `[n,m]` are the same as for Ruby's `String` class. Here are some examples:

```
>> s1.size             => 6

>> s1.to_s             => "abcabc"

>> s1.to_s.class       => String

>> s1[0]               => 97

>> s1[0,1]             => "a"

>> s1[2,4]             => "cabc"

>> s1[-5,2]            => "bc"

>> s1[-3,10]           => "abc"

>> s1[10]              => nil
```

A `ReplString` can represent a *very* long string:

```
>> s2 = ReplString.new("xy", 1_000_000_000_000)
=> ReplString("xy",1000000000000)

>> s2.length                => 2000000000000

>> s2[-1]                   => 121

>> s2[-2,2]                 => "xy"
>> s2[1_000_000_000]        => 120

>> s2[1_000_000_000,1]      => "x"

>> s2[1_000_000_001,1]      => "y"

>> s2[1_000_000_001,7]      => "yxyxyxy"
```

Some operations are impractical on very long strings. For example, `s2.to_s` would require a vast amount of memory.

A `MirrorString` represents a string concatenated with a reversed copy of itself. Examples:

```
>> s3 = MirrorString.new("1234")      => MirrorString("1234")

>> s3.size                            => 8

>> s3.to_s                            => "12344321"

>> s3[-1,1]                           => "1"

>> s3[2,4]                            => "3443"
```

A `ReplString` or a `MirrorString` can be made from a `ReplString` or a `MirrorString`. Here is a simple example, a string made of three replications of two replications of `"123"`:

```
>> s1 = ReplString.new(ReplString.new("123",2),3)
=> ReplString(ReplString("123",2),3)

>> s1.to_s      => "123123123123123123"
```

Below is a `ReplString` inside a `MirrorString` inside a `ReplString`. I've added a method called `commas` to `Integer` (a superclass of `Fixnum` and `Bignum`) that displays the values with commas added for readability. (See `/home/cs372/fall06/a5/commas.rb`.)

```
>> s4 = ReplString.new("0123456789", 1_000_000_000)
=> ReplString("0123456789",1000000000)

>> s4.size.commas                 => 10,000,000,000

>> s5 = MirrorString.new(s4)
=> MirrorString(ReplString("0123456789",1000000000))


>> s5[10_000_000_000-1,2]         => "99"

>> s5.size.commas                 => 20,000,000,000

>> s6 = ReplString.new(s5, 1000)   =>
ReplString(MirrorString(ReplString("0123456789",1000000000)),1000)

>> s6.size.commas                 => 20,000,000,000,000

>> s6[20_000_000_000-2,4]         => "1001"
```

The iterator `each` is available for both `ReplString` and `MirrorString`. It produces all the characters in turn, as one-character strings:

```
>> s1 = ReplString.new("abc",2)  => ReplString("abc",2)

>> s1.each {|x| puts x}
a
b
c
a
b
c
=> 0...6

>> MirrorString.new("abc").each { |x| puts x }
a
b
c
c
b
a
=> 0...6
```

Be sure to include `Enumerable` in `XString`, so that methods like `collect` and `all?` work:

```
>> MirrorString.new("abc").collect { |c| c }
=> ["a", "b", "c", "c", "b", "a"]

>> MirrorString.new(
              ReplString.new("a", 100000)).all? { |c| c == "a" }
=> true
```

Implementation Notes

One way to save some typing when doing manual testing is to use the names `MS` and `RS` instead of `MirrorString` and `ReplString`. Here's an example:

```
class MS < XString
     ...
end

MirrorString=MS
ReplString=RS

>> s = MS.new("abc")
=> MirrorString("abc")
```

Try to push as much of the code as possible up into `XString`. My implementations of `ReplString` and `MirrorString` have only four methods: a constructor, `size`, `inspect`, and `char_at(n)`. All of those methods are tiny—one or two short lines of code. When grading, tests will rely only on `ReplString` and `MirrorString`; `XString` will not be tested directly. (Note that none of the examples above do anything with `XString`.)

Note that one method must handle both `s[n]` and `s[n,m]`. Here's one way to do it:

```
def [](start, len = nil)
    ...
```

If `len` is `nil` assume the `s[n]` form is being used.

The code that adds `commas` to `Integer` is in `/home/cs372/fall06/a5/commas.rb`. If you're working on both Windows and `lectura`, here's a way to pick a pathname appropriately:

```
if ENV["SYSTEMROOT"]
    require 'c:/372/a5/commas.rb'
else
    require '/home/cs372/fall06/a5/commas.rb'
end
```

When done with this problem you might find it interesting to consider what's needed to make it work with arrays, too.

Problem 3. (10 points) `gf.rb`

Here's something I saw in a book:

```
class Fixnum
    def hours; self*60 end  # 60 minutes in an hour
end

>> 2.hours      => 120

>> 24.hours     => 1440
```

You are to write a Ruby method `gf(spec)` that dynamically adds a number of such methods to `Fixnum`, as directed by `spec`. Example:

```
gf("foot/feet=1,yard(s)=3,mile(s)=5280")
```

Using `Kernel#eval` method, this call to `gf` adds six methods to `Fixnum`: `foot`, `feet`, `yard`, `yards`, `mile`, `miles`. Respectively, on a pair-wise basis, those methods produce the `Fixnum` (which is `self`) multiplied by `1`, `3`, and `5280`.

```
>> load "gf.rb"   => true

>> gf("foot/feet=1,yard(s)=3,mile(s)=5280")      => true

>> 1.foot         => 1

>> 10.feet        => 10

>> 5.yards        => 15

>> 3.miles        => 15840

>> 8.mile         => 42240

>> 1.feet         => 1
```

It would perhaps be useful to detect mismatches like `8.mile` and `1.feet` and produce an error but that is not done.

In addition to the six methods mentioned above, three others are generated: `in_feet`, `in_yards`, and `in_miles`:

```
>> (30.feet+10.yards).in_yards  => 20.0

>> 10_000.feet.in_miles         => 1.89393939393939
```

Two more examples:

```
>> gf("second(s)=1,minute(s)=60,hour(s)=3600,day(s)=86400")=> true

>> (12.hours+30.minutes).in_days      => 0.520833333333333

>> gf("inch(es)=1,foot/feet=12")      => true

>> 18.inches.in_feet                  => 1.5

>> 1.foot                             => 12

>> 1.foot.in_inches                   => 12.0
```

Note that methods later generated by `gf` simply replace earlier methods of the same name. After the two calls `gf("foot/feet=1")` and `gf("foot/feet=12")`, `1.foot` is `12`.

An individual mapping must be in the form *`singular/plural=integer`* or `unit`*`(pluralSuffix)=integer`*. None of the parts may be empty. Mappings are separated by commas. Only lowercase letters are permitted in the names. No whitespace is allowed. If any part of a specification is invalid a message is printed and `false` is returned but the result is otherwise undefined. Here is an example of the output in the case of an error:

```
>> gf("foot/feet=1,yards=3")
bad spec: 'foot/feet=1,yards=3'
=> false
```

Note that the error is not pin-pointed—the specification as a whole is cited as being invalid.

Here are more examples of errors:

```
gf("foot/feet=1,")           # trailing comma
gf("foot/feet=1.5")          # non-integer
gf("foot/=1")                # empty plural
gf("inch()=12")              # empty plural suffix
gf("foot/feet=1,Yard(s)=3") # capital letter
```

<u>This is NOT a restriction but to get more practice with regular expressions I recommend that your solution not use any string comparisons; use matches (=~) to break up the specification.</u> And, using regular expressions will probably increase the likelihood that you accept exactly what's valid.

I recommend that you use `eval` (slide 148) instead of `Module#define_method`, et al., to add the methods to `Fixnum`.

Keep in mind that you're writing a method, not a program. Helper methods are permitted.

Problem 4. (3 points) `speedup.rb`

Note: See below for an important restriction on this problem.

Mr. Barnes produced a very interesting solution for `minmax.rb` on assignment 3. Here's a version of it with some cosmetic changes:

```
begin
   lines = Hash.new([]); line_number = 1
   while line = gets do
      lines[line.chop.size] = lines[line.chop.size] + [line_number.to_s]
      line_number = line_number + 1
   end
   puts "Min length: " + lines.sort.first.first.to_s + " (" + (lines.sort.first.last * ", ") + ")"
   puts "Max length: " + lines.sort.last.first.to_s + " (" + (lines.sort.last.last * ", ") + ")"
rescue
   puts "Empty file"
end
```

I see this as an Original Thought but it's got a problem: it's very slow on large inputs. On `lectura` it takes over two minutes of CPU time to process `/usr/share/dict/words`. Your task on this problem is to speed it up while staying with Mr. Barnes basic approach—using a hash to accumulate the length of EVERY line. **To receive any points on this problem your revised version must process `/usr/share/dict/words` in less than 2.2 seconds of "user" CPU time**, as reported by `/usr/bin/time`, on `lectura`. Example:

```
% /usr/bin/time ruby speedup.rb < /usr/share/dict/words
Min length: 1 (9, 38, 40, 31876, 31878, 57049, 57051, 97101, 97103,
120349, 120351, 137529, 137530, 153807, 153809, 168908, 168909,
188220, 188221, 203658, 203660, 208077, 208079, 214569, 214571,
229273, 229277, 255473, 255475, 272147, 272151, 287461, 287463,
329584, 329586, 332857, 332859, 354559, 354561, 406354, 406355,
432266, 432268, 456211, 456213, 463230, 463232, 475253, 475254,
475885, 475887, 477707, 477708)
Max length: 45 (308762)
1.87user 0.05system 0:04.00elapsed 48%CPU (0avgtext+0avgdata
0maxresident)k
0inputs+0outputs (0major+1755minor)pagefaults 0swaps
```

In the output above the "user" CPU time—1.87 seconds—is underlined.

Restriction: There's a very simple fix that produces a dramatic speedup. It's so simple that it can be easily given away with about three words of advice. Therefore, **on this problem there is to be absolutely no communication about it with anyone except me or Poorna.** Think of this problem as a take-home test.

If you find a dramatic speedup but it doesn't take you low enough, say just down to three seconds, let me know. You may have found a solution that's close but a dead-end, ultimately.

Note that Mr. Barnes handles the empty file case by catching an exception—very nice. Our coverage of Ruby won't be including exception handling but if you're curious you can read about the topic on-line and/or in the text.

We did work through the solution for this problem in a recent Honors Section meeting. I'll be assigning them an interesting replacement problem but if you happened to hear about this problem from one of them, let me know.

Start with the code in `fall06/a5/speedup0.rb`.

For one point of extra credit, use `eval` to eliminate the repetitious code that's used for printing the results.

Problem 5. (10 points) `label.rb`

`Array#inspect`, which is used by `Kernel#p` and by `irb` does not accurately show when an array contains multiple references to the same array and/or to itself. Example:

```
>> a = []

>> b = [a,a]

>> p b
[[], []]
```

By simply examining the output of `p b` we can't tell whether `b` references two distinct arrays or has two references to the same array.

Another problem is that if an array references itself, Ruby "punts":

```
>> a = []

>> a << a

>> p a
[[...]]
```

Write a method `label(a)` that produces a labeled representation of an array that references other such arrays and/or contains strings and/or integers. Here's what label shows for the first case above:

```
>> a = []; b = [a,a]

>> puts label(b)
a1:[a2:[],a2]
```

The outermost array is labeled with `a1`. Its first element is an empty array, labeled `a2`. The second element is a reference to that same empty array. Its contents are not shown, only the label `a2`. Here's another step, and the result:

```
>> c =[b,b]

>> puts label(c)
a1:[a2:[a3:[],a3],a2]
```

Note that the labels are not preserved across calls. The array that this call labels as `a3` was labeled as `a2` in the previous example. To explore relationships between the contents of `a`, `b`, and `c` we could wrap them in an array:

```
    >> puts label([a,b,c])
    a1:[a2:[],a3:[a2,a2],a4:[a3,a3]]
```

Another example:

```
>> a = [1,2,3]

>> a << [[[a,[a]]]]

>> a << a

>> puts label(a)
a1:[1,2,3,a2:[a3:[a4:[a1,a5:[a1]]]],a1]
```

One more example:

```
>> a = [1,2,3]

>> b = ["abc"]

>> 3.times { a << b; b << a }

>> puts label([a,b])
a1:[a2:[1,2,3,a3:["abc",a2,a2,a2],a3,a3],a3]
```

Keep in mind that your solution must be able to accommodate an arbitrarily complicated array. However, the only types you'll encounter are integers, strings, and arrays. You won't see something like a hash that contains arrays of hashes with arrays for both keys and values, for example.

This routine is a simplified version of the `Image` routine from the Icon library. I've looked around and asked around for something similar in Ruby. I haven't found anything yet but it may well exist. If you find such a routine, which would trivialize this problem, you may not use it. However, you may study it and then, based on what you've learned, create your own implementation.

Implementation notes

My solution is recursive. It starts like this:

```
def label(x, sequence = [0] , h = {})
    return x.inspect if !x.is_a? Array
```

Thirteen more lines follow. Some of those are one token or less.

Use `Array#object_id` to produce a unique integer for each array. Perhaps use that id as a key in a hash, with the value being a label, like `"a1"`.

You'll need to match my sequence of labels, which essentially requires you to traverse the structure in the same order I do. I use a depth-first traversal. Here's an example that illustrates that:

```
>> puts label([[[10]],[[21,22]]])
a1:[a2:[a3:[10]],a4:[a5:[21,22]]]
```

Problem 6. (8 points) `re.rb`

In this problem you are to write four methods. Each returns a regular expression that matches the specified strings (no more, no less) and, in some cases, also sets some groups.

Here is an example specification:

Write a method `oddnum_re` that produces a regular expression that matches strings that represent an odd number.

Here is the solution:

```
def oddnum_re
    /^-?\d*[13579]$/
end
```

Here are the four methods you are to write:

(a) Write a method `range_re` that produces a regular expression that matches strings that represent a Ruby range, like `1..10` and `-20...10`. The group `$1` is set to the first number, `$3` is set to the second number, and if `$2` is not empty, it indicates a three-dot range was matched.

(b) Write a method `sentence_re` that produces a regular expression that match sentences, as follows: Sentences must begin with a capital letter. Sentences are composed of one or more words. Words are separated by exactly one blank. The sentence must end with a period, question mark, exclamation mark, or one of the two strings `"!?"` and `"?!"`.

Two good sentences: `"I shall test this!"`, `"Xserwr AAA x."`

A bad sentence: `"it works!"` (Doesn't start with a capital.)

(c) Write a method `path_re` that produces a regular expression that matches UNIX paths and sets `$1` to the directory name, `$2` to the filename, minus extension, and `$5` to the extension, which is defined as everything in the filename to the right of the leftmost dot. If an element is not present, the group is set to the empty string.

Examples:

```
path: '/home/cs372/fall06/tester/Test.rb'
dir = '/home/cs372/fall06/tester/', file = 'Test', ext = 'rb'

path: '/etc/passwd'
dir = '/etc/', file = 'passwd', ext = ''

path: './../.../x'
dir = './../.../', file = 'x', ext = ''
```

(d) Write a method `calc_re` that matches valid input lines for `calc.rb`, from assignment 4.

The above is skimpy on examples but you'll find plenty in `.../fall06/a5/re.1`. That's an input file for `fall06/a5/re1.rb`. It includes the `oddnum_re` example mentioned above.

The deliverable, `re.rb`, should consist of four methods: `range_re`, `sentence_re`, `path_re`, and `calc_re`.

Problem 7. (3 points extra credit) `X.java`

Create a Java class `X` such that this code,

```
X x = new X();
for (Object o: x)
    System.out.println(o);
```

Produces this output:

```
x
```

As an alternative, and for five points instead of three points, use a generic:

```
X<Y> x = new X<Y>();
for (Y y: x)
    System.out.println(y);
```

Problem 8. (5 points; 1 point each) `answers.txt`

Create a text file named `answers.txt` with answers to the following questions. **DO NOT submit a Word document, PDF, rich text file, etc.**—I want plain text.

(a) As used in Ruby and other OO languages, what is the origin of the term "mixin"? (The answer won't be found in the slides; you'll have to do a little web research.)

(b) Here is a regular expression problem that has a problem:

> *Write a regular expression that matches a list of ranges like these: `"1-3"`, `"3-5,1-10"`. It should NOT match a range where the first number exceeds the second, like `"99-1"` or `"1-5,5-1"`.*

What's the problem with that problem?

(c) Create a class named `C1` and supply an `attr_accessor` for `x` Load the class in `irb` and confirm that `C1.new.x = 1` works. Change the `attr_accessor` for `x` to `attr_reader`. Without exiting `irb`, load the class again. You should find that although you've switched `x` to read-only, `C1.new.x = 1` still works. Why is that the case?

(d) Slide 24 says that some objects make a copy of a string when the string is added to it. Is `Array` such a class? Show a clipping from an `irb` session that supports your answer.

(e) In Ruby, both `yield x,y,z` and `yield [x,y,z]` call the block with three arguments. How can the block be called with one argument, the array `[x,y,z]`?

(f) (1 point extra credit) Estimate how long it took you to complete this assignment. Other comments about the assignment are welcome, too—I appreciate all feedback, favorable or not.

(g) (1 point extra credit) Cite an interesting course-related observation that you made while working on the assignment. The observation should have a little depth. For example, something like "I learned that regular expressions are very useful." wouldn't be worth a point.

Deliverables

The deliverables for this assignment are these files: `mtimes.rb`, `xstring.rb`, `gf.rb`, `speedup.rb`, `label.rb`, `re.rb`, `answers.txt`, and, for extra credit, `X.java`.

That list can be found on `lectura` in `/home/cs372/fall06/a5/delivs`.

Use the `turnin` tag `372_5` to submit your solutions.

Corrections and FAQs, late submissions, `turnin`, retests, etc.

Refer to the write-up for the first assignment for details on these topics and similar ones.

Miscellaneous

A tester script is in place: `fall06/a5/tester`. One of the things that the script does not check is meeting the time limit on `speedup.rb`; you'll need to do that by hand.

Solutions will be scored only on correctness.

Keep in mind the point value of each problem; don't invest an inordinate amount of time in a problem before you ask for help with it. Remember that the purpose of the assignments is to build understanding of the course material by applying it to solve problems. Seek the help of Poorna and me as needed to meet your time budget.

Other than `speedup.rb`, no problems on this assignment have any restrictions.

If you wish, you may incorporate code from lectures in your solutions.

Feel free to use comments to document your code as you wish but note that no comments, not even your name, are required.

I hate to have to mention it but keep in mind that I don't give cheaters a second chance to waste everybody's time. If you give your code to somebody else and they turn it in, you'll both likely fail the class, and more. (See the syllabus for the details.)