Last name, NetID

_____

CSC 372 Ruby Mid-term Exam
April 11, 2014

READ THIS FIRST

Read this page now but do not turn this page until you are told to do so. Go ahead and fill in your last name and NetID in the box above.

This is a 45-minute exam with a total of 100 points of regular questions and an extra credit section.

The last five minutes of the exam is a "seatbelts required" period, to avoid distractions for those who are still working. If you finish before the "seatbelts required" period starts, you may turn in your exam and leave. If not, you must stay quietly seated—no "packing up"— until time is up for all.

You are allowed no reference materials whatsoever.

If you have a question, raise your hand. We will come to you. DO NOT leave your seat.

If you have a question that can be safely resolved with a minor assumption, like the name of a function or the order of function arugments, state the assumption and proceed.

Feel free to use abbreviations.

Don't make a problem hard by assuming it needs to do more than is specifically mentioned in the write-up.

If you're stuck on a problem, please ask for a hint. Try to avoid leaving a problem completely blank—that's a sure zero.

It is better to put forth a solution that violates stated restrictions than to leave it blank—a solution with violations may still be worth partial credit.

When told to begin, double-check that your name is at the top of this page, and then **put your initials in the lower right hand corner of the top side of each sheet, checking to be sure you have all seven sheets.**

BE SURE to enter your NetID on the sign-out log when turning in your completed exam.

Problem 1: (21 points)

In this problem you are to write a Ruby program `tail.rb` that prints the last N lines of standard input, just like the UNIX `tail` command. Here's an example, after first demonstrating that the file `five` has five lines.

```
% cat five
one
two
three
four
five
% ruby tail.rb -2 < five
four
five
%
```

If the file has less than N lines, all lines are printed. Example:

```
% ruby tail.rb -10 < five       # prints all five lines
one
two
three
four
five
%
```

Exactly one command line argument should be specified and, with the leading dash, it should look like a negative integer, like "`-3`" or "`-100`". Otherwise, print "`Usage: tail -N`" and exit by calling `exit(1)`.

Examples of erroneous invocations:

```
% ruby tail.rb < five
Usage: tail -N

% ruby tail.rb 100 < five
Usage: tail -N
```

Behavior is undefined with the argument "`-0`". (In other words, ignore that case.)

Important: Make this simple by assuming you can hold the entire file in memory—don't imagine you need to use a circular queue or something like that!

There's plenty of space on the next page for your solution.

(space for solution for `tail.rb`)

Quick reminder of operation:

```
% ruby tail.rb -4 < five
two
three
four
five
% ruby tail.rb 4 < five
Usage: tail -N
%
```

Problem 2: (17 points)

In this problem you are to write a method `load_facts(file_name)` that reads Prolog facts from the specified file and returns a Ruby hash that holds a representation of the facts. Here's a sample input file, `fcl.pl`, shown with `cat`:

```
% cat fcl.pl
food(apple).
%food(broccoli).
food(lettuce).

color(sky,blue).
color(dirt,brown).
color(grass,green).
color(x).

thing(apple,red,yes).
thing(a,b).
```

Usage, with the hash contents shown **formatted by hand**, to make it easier to read.

```
>> h = load_facts("fcl.pl")
=> {
"food"=>[["apple"], ["lettuce"]],
"color"=>[["sky", "blue"], ["dirt", "brown"],
          ["grass", "green"], ["x"]],
"thing"=>[["apple", "red", "yes"], ["a", "b"]]
}
```

Each of the functors `food`, `color`, and `thing` are keys in the hash. The value associated with each key is an array of arrays. Each entry in the inner arrays represents the term(s) for one fact. **Because there are two one-term facts for `food`, `h["food"]` references an array containing two one-element arrays.**

Similarly, `color` has three two-term facts and one one-term fact.

Empty lines or lines that start with a `%` are ignored. All other lines are well-formed Prolog facts with at least one term. Lines contain no whitespace.

To read from a file, use `f = File.open("x")` to open a file named `"x"` and then `f.gets` to read from it. `f.gets` returns `nil` at end of file. `f.close` closes it.

There's plenty of space on the next page for your solution.

(space for solution for `load_facts`)

For reference, here are the facts and resulting hash again:

```
% cat fcl.pl
food(apple).
%food(broccoli).
food(lettuce).

color(sky,blue).
color(dirt,brown).
color(grass,green).
color(x).

thing(apple,red,yes).
thing(a,b).
```

```
>> h = load_facts("fcl.pl")
=> {
"food"=>[["apple"], ["lettuce"]],
"color"=>[["sky", "blue"], ["dirt", "brown"],
          ["grass", "green"], ["x"]],
"thing"=>[["apple", "red", "yes"], ["a", "b"]]
}
```

Problem 3: (15 points)

This problem is a partner to `load_facts`. You are to write a Ruby method `print_preds` that takes the hash produced by `load_facts` (the previous problem) and prints a list of predicate indicators.

Usage, using the facts from the previous problem:

```
>> h = load_facts("fcl.pl")
=> {
"food"=>[["apple"], ["lettuce"]],
"color"=>[["sky", "blue"], ["dirt", "brown"],
          ["grass", "green"], ["x"]],
"thing"=>[["apple", "red", "yes"], ["a", "b"]]
}

>> print_preds(h)
color/1
color/2
food/1
thing/2
thing/3
=> nil
```

Predicates are shown in alphabetical order, with a predicate indicator (*`functor/N`*) for each of the forms present. We can see that the hash `h` holds one-term facts for `food`, one- and two-term facts for `color`, and two- and three-term facts for `thing`.

Here are some possibly useful things:

`Hash#keys` returns an array of the keys in a hash.
`Array#sort` returns a copy of the array with the entries sorted.
`Array#min` and `Array#max` return the minimum and maximum values in an array.
`Enumerable#map` is the iterator analog for Haskell's `map` function.

Problem 4: (5 points)

Write a method `multstring(spec, s)` that behaves like this:

```
>> multstring("3,1,5", "x")
=> "xxx,x,xxxxx"

>> multstring("3,0,2,0", "Abc")
=> "AbcAbcAbc,,AbcAbc,"

>> multstring("10", "")
=> ""
```

The first argument, `spec`, is a comma-separated list of non-negative integers. The result is a string that consists of comma-separated replications of the second argument (`s`) corresponding to each integer in turn. Assume that `spec` is well-formed and contains no whitespace.

Problem 5: (4 points)

Write a method `addrev` such that after calling `addrev`, the unary minus operator applicatively reverses a string:

```
>> addrev
=> nil

>> x = -"testing"
=> "gnitset"

>> -x
=> "testing"

>> x
=> "gnitset"
```

Problem 6: (11 points) One point each unless otherwise indicated

(a) Write a regular expression that is equivalent to `/x+/` but that doesn't use the `+` operator.

(b) Write a regular expression that is equivlent to `/ax|bx|cx/` but that doesn't use the `|` operator.

(c) Describe in English the strings matched by `/a*b+c$/`

(d) (3 points) Write a regular expression that matches only strings that are binary constants like these:

```
"0b0"
"0B11111"
"0b01010101"
```

The first two characters are a "`0`" and an upper- or lower-case "`B`". One or more "`1`"s or "`0`"s follow.

Here are two non-matches: "`00b1`" (extra leading zero), "`0b12`" (trailing non-0/1).

(e) (5 points) Write a regular expression that matches a string if and only if it is a comma-separated sequence of integers **greater than or equal to 10**. Examples of matches:

```
"10, 11, 12"
"30,200,500"
"100"
```

One optional space may appear after each comma but that is the only place a space may appear. **Leading zeros are not permitted!**

Here are three non-matches: "`10, 020`" (has a leading zero), "`7,11`" (7 is less than 10), "`1x`" (has a trailing "`x`").

Problem 7: (7 points)

Write an iterator `take_while(a)` that calls its block with each element of the array `a` in turn. As long as the block returns true, array elements are accumulated. When the block first returns a non-true value or when the array elements are exhausted, an array of the accumulated elements is returned.

```
>> take_while([1,2,3,4,5]) { true }
=> [1, 2, 3, 4, 5]

>> take_while([1,2,3,4,5]) { false }
=> []

>> take_while([1,2,3,4,5]) { |e| e.odd? }
=> [1]

>> take_while([1,2,3,4,5,3,2]) { |e| e < 5 }
=> [1, 2, 3, 4]

>> take_while("eat peas all day".split) { |w| w =~ /e/ }
=> ["eat", "peas"]
```

Note: This is much like `Enumerable#take_while`, so you can't use that method in your solution! (It's like Haskell's `takeWhile`, too.)

Problem 8: (12 points)

(a) Who invented Ruby? (1 point)

(b) Consider a Ruby method `last(x)` that returns the last element of a string or array. Why would `last` be awkward to write and use in Java? (2 points)

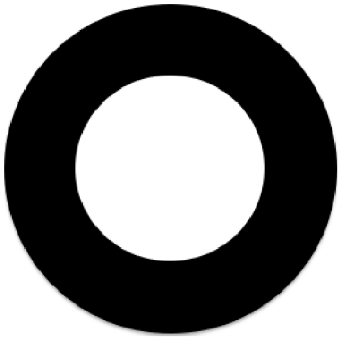
(c) What's the effect of adding "`include Enumerable`" to a class definition? (2 points)

(d) Cite one way in which Ruby's `if-then-else` is like Haskell's `if-then-else` and one way in which it is different. (2 points)

(e) A Java newcomer to Ruby might think that `private` and `attr_reader` are keywords but they aren't. What are they? (2 points)

(f) What's the essential difference between dynamic typing and static typing? (3 points)
Hint: If your answer contains `/(interprete|compile)[rd]?/` there's a fair chance it'll be wrong.

Problem 9: (8 points)



Recall the `Shape`/`Circle`/`Rectangle` inheritance hierarchy from the slides. `Shape`'s constructor requires a single argument: a string that serves as a label for the shape. `Shape#label` returns the label.

In this problem you are to create a subclass of `Shape` named `Ring`. An instance of `Ring` represents the region between two concentric circles (the black area in the drawing to the right). Along with a label that is passed to `Shape`'s constructor, `Ring`'s constructor takes the radii of the two circles but they may specified be in either order. If the two circles have the same radius, the area of the ring is zero.

Your implementation of `Ring` should have five methods: a constructor, `area`, `inner`, `outer`, and `inspect`. The methods `area`, `inner`, and `outer` return the area of the ring, the inner radius and the outer radius, respectively. `inspect` displays the inner (smaller) radius followed by the outer radius.

Assume both radii are non-negative. `Math::PI` is π.

Examples:

```
>> r1 = Ring.new("a",3.2,1.9)
=> Ring a (1.9-3.2)       # This line displays the result of inspect. The inner
                          # radius, 1.9, is shown first.

>> r1.area
=> 20.82875929330033

>> r1.inner
=> 1.9

>> r1.outer
=> 3.2

>> Ring.new("x",3,3).area
=> 0.0
```

Write your solution to the right or on the next page.

(space for solution for `Ring`)

Examples (repeated):

```
>> r1 = Ring.new("a",3.2,1.9)
=> Ring a (1.9-3.2)        # This line displays the result of inspect. The inner
                           # radius, 1.9, is shown first.

>> r1.area
=> 20.82875929330033

>> r1.inner
=> 1.9

>> r1.outer
=> 3.2

>> Ring.new("x",3,3).area
=> 0.0
```

Extra Credit Section (½ point each unless otherwise noted)

(a) What does whm consider to be the all-time greatest Original Thought ever put forth by one of his 372 students?

(b) Based on the Prolog we've covered, is Prolog statically typed or dynamically typed? (And why?)

(c) What thing in Ruby is closest to a Prolog atom?

(d) Which is the oldest of Java, Haskell, and Ruby?

(e) The term "mixin" came from a business that sold what to college students?

(f) Regular expressions are Type _____ languages in the _____ hierarchy of languages.

(g) Predict the median score on this exam. (The median is the "middle" value in a range of values.)

(h) If you only remember one thing about Ruby, what will it be? (Ok to be funny!)

(i) What's the stupidest thing in Ruby? Can't be the same as (h) or (j). (1 point)

(j) What's the best thing in Ruby? Can't be the same as (h) or (i). (1 point)

(k) Implement `attr_reader`. (1 point)