CSC 372, Spring 2014
Assignment 4
Due: Wednesday, March 26 at 23:00

Use Ruby 1.9.3!

There was some confusion about Ruby versions on assignment 3 so to be clear, Ruby 1.9.3 is to be used for all Ruby assignments, including this one. On lectura, use "`rvm 1.9`" each time you login to set things for 1.9.3. You might get a blat about "`Warning! PATH is not properly set up...`" but if `ruby --version` shows 1.9.3, things should be fine. Here's what I see:

```
$ rvm 1.9
Warning! PATH is not properly set up
...lots more...
$ ruby --version
ruby 1.9.3p484 (2013-11-22 revision 43786) [x86_64-linux]
```

Use a symbolic link for easy access to the tester and data files

An easy way to access data files and the testing script is to put a symlink (symbolic link) to `/cs/www/classes/cs372/spring14/a4` in your assignment 4 directory. I describe a symbolic link as a Windows shortcut done right. Here's what to do:

```
$ cd ~/372/assn4
$ ln -s /cs/www/classes/cs372/spring14/a4 .
```

Then, take a look at what `ls` shows:

```
$ ls -l a4
lrwxrwxrwx 1 whm whm 33 Mar  8 23:00 a4 -> /cs/www/classes/cs372/spring14/a4
```

That lowercase "L" at the start of the line and the `->` after `a4` indicate a symbolic link. If you "`ls a4`" or "`cd a4`" you'll actually be operating on `/cs/www/classes/cs372/spring14/a4`.

Using "`.`" as the last argument for `ln` causes the link to have the same name as the target (`a4`) but you could name it something else. Omitting the last argument has the same effect.

With that link you could copy in the test cases for `pancakes` with

```
$ cp a4/pancakes.? .
```

or you could run the tester with

```
$ a4/tester
```

You don't need to create that symlink but this write-up and future write-ups will assume the presence of an `aN` symlink in the examples.

`tester` script

`a4/tester` is an automated testing script for assignment 4, but you need to be on lectura to run it. If you `cd` to the directory with your `a4` solutions and run it with no arguments it'll test all problems in turn. It'll start like this:

```
------------------------------------------------------------
|                                                          |
|                          .: nzip                         |
|                                                          |
------------------------------------------------------------


------------------------------------------------------------
|                                                          |
|                      Test Execution                      |
|                                                          |
------------------------------------------------------------


Test: 'ruby --version': PASSED

Test: 'echo 'load "nzip.rb"; nzip([1,2,3,4], %w{one two three four
five six}) { |x| printf("result: %s\n", x.inspect)};' | ruby':
PASSED

Test: 'echo 'load "nzip.rb"; nzip([1,2,3,4], %w{one two three four
five six}, [10]*10) { |x| printf("result: %s\n", x.inspect)};' |
ruby': PASSED
```

A line starting with `Test:` is printed for each test. Examples:

```
Test: 'ruby --version': PASSED

Test: 'echo 'load "mirror.rb"; mirror(1..3) { |v| puts v };' |
ruby': PASSED

Test: '/bin/echo -e '3+4*5' | ruby calc.rb': PASSED

Test: 'ruby switched.rb 1951 1958': PASSED
```

If you fail a test, a good first step is to copy the text between the outermost quotes onto the clipboard and then paste onto the command line.

If a test fails, the `diff` command is to used to show you the differences, but "diffs" can be hard to understand. Here's a failure, with line numbers added for reference:

```
1.  Test: 'ruby switched.rb 1948 1952': FAILED
2.  Differences (expected/actual):
3.  *** /cs/www/classes/cs372/spring14/a4/master/tester.out/switched.out.2
        2014-03-08 23:22:21.335914566 -0700
4.  --- tester.out/switched.out.2   2014-03-08 23:58:55.279287964 -0700
5.  ***************
6.  *** 1,2 ****
7.                 1948   1949   1950   1951   1952
8.  ! Lavern        1.13    Inf    Inf    n/a   0.86
9.  --- 1,3 ----
10.                1948   1949   1950   1951   1952
11. ! Lavern        1.13    Inf    Inf   n/a   0.86
12. !
```

Lines 3 and 4 name the files that are being diffed. The first is the file that contains the expected output, `a4/master/tester.out/switched.out.2`. The second contains the output produced by running `ruby switched.rb 1948 1952` in the current directory. You can look at both files with `cat`, `more`, editors, or any other tool. Those file names are preceded by `***` and `---` , which are used later, on lines 6 and 9, to identify blocks of text from those files.

Line 6's `"*** 1,2 ****"` means that what follows are lines 1-2 from the expected output. Line 9's `"--- 1,3 ----"` means that what follows are lines 1-3 from the actual output.

The exclamation marks on lines 8, 11, and 12 indicate that those lines differ between the expected and actual output. Line 12 is in fact an extra blank line but `diff` classifies it as a difference, not an addition.

Here's another diff:

```
Differences (expected/actual):
*** /cs/www/classes/cs372/spring14/a4/master/tester.out/pancakes.out.1
2014-03-08 20:56:34.444119063 -0700
--- tester.out/pancakes.out.1   2014-03-09 00:17:51.082836671 -0700
***************
*** 1,7 ****
  Order: 1 3 1 / 1 2 3

  Even-width pancake.  Order ignored.
-
  Order: 51 49

  **************************************************
--- 1,6 ----
```

As before, `***` and `---` are used to mark blocks of text from the expected and actual output, respectively. The minus sign just after the line `Even-width pancake. ...` indicates that that line in the expected output is missing from the actual output. Let's look at the first six lines of the expected output,

```
$ head -6 /cs/www/classes/cs372/spring14/a4/master/tester.out/pancakes.out.1
Order: 1 3 1 / 1 2 3

Even-width pancake.  Order ignored.

Order: 51 49
```

and the first six lines of the actual output:

```
$ head -6 tester.out/pancakes.out.1
Order: 1 3 1 / 1 2 3

Even-width pancake.  Order ignored.
Order: 51 49

************************************************
```

We can see that the actual output is missing a blank line.

Now that we've been through those preliminaries, here are the problems.

Problem 1. (4 points) `nzip.rb`

Write a Ruby iterator `nzip(a1, a2, ..., aN)` that yields a series of arrays. The first array is `[a1[0], a2[0],...aN[0]]`, the second array is `[a1[1], a2[1],...aN[1]]`, etc. `nzip` produces values as long as all arguments have an element at a given position. In other words, the shortest array determines how many results `nzip` will yield. `nzip` returns the number of arrays that it produced. If called with no arguments, `nzip` returns `nil`.

There's a minor restriction on `nzip`: You can't use `Array#zip`!

```
>> a1 = [1,2,3,4]
=> [1, 2, 3, 4]

>> a2 = %w{one two three four five six}
=> ["one", "two", "three", "four", "five", "six"]

>> a3 = [10] * 10
=> [10, 10, 10, 10, 10, 10, 10, 10, 10, 10]

>> nzip(a1,a2) { |x| printf("result: %s\n", x.inspect) }
result: [1, "one"]
result: [2, "two"]
result: [3, "three"]
result: [4, "four"]
=> 4

>> nzip(a1,a2,a3) { |x| printf("result: %s\n", x.inspect) }
result: [1, "one", 10]
result: [2, "two", 10]
result: [3, "three", 10]
result: [4, "four", 10]
=> 4

>> nzip([10]) { |x| printf("result: %s\n", x.inspect) }
result: [10]
=> 1

>> nzip([ ], a2) { |x| printf("result: %s\n", x.inspect) }
=> 0
```

Remember that you can't use `Array#zip`!

Note that nzip is not a whole program. It is a freestanding method. A test program might look like this:

```
% cat nziptest.rb
load "nzip.rb"
nzip([1,2]) { puts "x" }

% ruby nziptest.rb
x
x
%
```

Problem 2. (4 points) `vrepl.rb`

Write a Ruby iterator `vrepl(a)` that produces an array consisting of varying numbers of repetitions of values in `a`. The number of repetitions for each element is determined by the result of the block when the iterator yields that element.

```
>> vrepl(%w{a b c}) { 2 }
=> ["a", "a", "b", "b", "c", "c"]

>> vrepl(%w{a b c}) { 0 }
=> []

>> vrepl((1..10).to_a) { |x| x % 2 == 0 ? 1 : 0 }
=> [2, 4, 6, 8, 10]

>> i = 0
=> 0

>> vrepl([7, [1], "4"]) { i += 1 }
=> [7, [1], [1], "4", "4", "4"]
```

If the block produces a negative value, zero repetitions are produced:

```
>> vrepl([7, 1, 4]) { -10 }
=> []
```

Like `nzip`, `vrepl` is not a program. It is a freestanding method.

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

Write a Ruby iterator `mirror(x)` that yields a "mirrored" sequence of values based on the values that `x.each` yields. Unlike `nzip` and `vrepl`, which operate on arrays, all that `mirror` requires is that `x` responds to the method `each`. (Duck typing!) The value returned by `mirror` is always `nil`.

```
>> mirror(1..3) { |v| puts v }
1
2
3
2
1
=> nil
```

```
>> mirror([1, "two", {a: "b"}, 3.0]) { |v| puts v }
1
two
{:a=>"b"}
3.0
{:a=>"b"}
two
1
=> nil

>> mirror({:a=>1, :b=>2, :c=>3}) {|x| p x}
[:a, 1]
[:b, 2]
[:c, 3]
[:b, 2]
[:a, 1]
=> nil

>> mirror([]) { |v| puts v }
=> nil
```

Problem 4. (12 points) `pancakes.rb`

Write a Ruby version of the Haskell pancake printer from assignment 2.

The Ruby version is a program that reads lines from standard input, one order per line, echoes the order, and then shows the pancakes.

Example:

```
$ cat pancakes.1
3 1 / 3 1 5
3 1 3
1 5/          1 1 1/11 3 15     /3 3 3     3/1
1
$ ruby pancakes.rb < pancakes.1
Order: 3 1 / 3 1 5

     ***
***   *
 *  *****

Order: 3 1 3

***
 *
***

Order: 1 5/          1 1 1/11 3 15     /3 3 3     3/1

                        ***
      *   ***********   ***
  *   *       ***       ***
***** * ************** *** *

Order: 1
```

```
*

$
```

A blank line is printed after the `Order:` line and again after the stacks.

Assume that input lines consist exclusively of integers, spaces, and slashes, which separate stacks. Assume that there is at least one stack. Assume all stacks have at least one pancake. Assume all widths are greater than zero. Assume the input is well-formed—you won't see something like `"1 / / 3"` or `"/ 3 /"`. Assume there are no empty lines in the input.

If an order specifies an even-width pancake, the message shown below is printed. Processing then continues with the next order in the input, if any.

```
$ ruby pancakes.rb < pancakes.2
Order: 1 3 1 / 1 2 3

Even-width pancake.  Order ignored.

Order: 51 49

**************************************************
 **************************************************

$
```

If you want to play "Beat the Teacher", it took me about 25 minutes to write `pancakes.rb`, sketching on paper included. If you care to, let me know how long it takes you. Think about it all you want to but start the clock the moment a tangible artifact is produced, like a mark on a piece of paper.

Problem 5. (15 points) `calc.rb`

Write in Ruby a simple four-function calculator that evaluates expressions composed of integer literals and variables, and supports the operations of addition, subtraction, multiplication, and division. All operators have equal precedence. Evaluation is done in a strict left to right order. Control-D exits the program. Here are examples of expressions involving integer literals:

```
$ ruby calc.rb
? 3+4
7
? 3*4+5
17
? 3+4*5          Note that the addition is done first because it is the leftmost operator.
35
? 1/2*3+4
4
? 5/3
1
? 143243243243242323*342343443234324
49038385111943393068867603094652
? ^D
$
```

Variables are created with assignments. Variables begin with a letter and are followed by zero or more letters or digits. Variables have a default value of zero. The result of an assignment is the value assigned.

```
$ ruby calc.rb
? x=7
7
? yval=10
10
? z
0
? x=x+yval+z
17
? yval=x+yval
27
? yval
27
? big=1111111111111111*1111111111111111
1234567901234567876543209876543 21
? big=big/big
1
```

An assignment always consists of a variable followed by an equals sign followed by an expression. An expression may not contain an assignment. For example, `x+y=0` is not valid.

Input lines will consist solely of letters, digits, and the five symbols `+*-/=`. Assume all expressions are well formed. You won't see something like `x==3` or `+10/5+`. If a string starts with a letter, it is a variable; you won't see something like `15x`. There is no negation; you won't see something like `x=-10`. Don't worry about division by zero. There will be no empty lines in the input.

Implementation note

Use `String#scan` with a *regular expression* to break up input lines. Here's an example of `scan`:

```
>> "x2=3*val+40-500".scan(/(\w+|\W+)/)
=> [["x2"], ["="], ["3"], ["*"], ["val"], ["+"], ["40"], ["-"],
["500"]]
```

As of press time we have yet to cover regular expressions. Simply use the argument to `scan` that's shown above. Think of it as a black box. To ensure you get it right, copy and paste from the PDF.

Hint: Use a `map` to flatten that array of arrays:

```
>> "x2=3*val+40-500".scan(/(\w+|\W+)/).map {|e| e[0] }
=> ["x2", "=", "3", "*", "val", "+", "40", "-", "500"]
```

or, just use `flatten`:

```
>> "x2=3*val+40-500".scan(/(\w+|\W+)/).flatten
=> ["x2", "=", "3", "*", "val", "+", "40", "-", "500"]
```

Using `to_i` to look for integers like you did on `xfield` won't be good enough for `calc` because an expression might include a 0. You can use `"0" <= c && c <= "9"` to see if `c` is digit, or look ahead a little in the material on regular expressions.

Problem 6. (15 points) `switched.rb`

The U.S. Social Security Administration makes available yearly counts of first names on birth certificates back to 1885. Over time, some names change from predominantly male to predominantly female or vice-versa. For this problem you are to create a Ruby program `switched.rb` to look for such changes.

`switched.rb` takes two command-line arguments: a starting year and an ending year. Here's a run:

```
% ruby switched.rb 1951 1958
             1951   1952   1953   1954   1955   1956   1957   1958
Dana         1.19   1.20   1.26   1.29   1.00   0.79   0.67   0.64
Jackie       1.40   1.29   1.14   1.13   1.11   0.94   0.72   0.57
Kelly        4.23   2.74   3.73   2.10   2.32   1.77   0.98   0.51
Kim          2.58   1.82   1.47   1.08   0.61   0.30   0.17   0.12
Rene         1.43   1.32   1.15   1.24   1.13   0.88   0.87   0.89
Stacy        1.06   0.81   0.62   0.47   0.44   0.36   0.29   0.21
Tracy        1.51   1.14   1.02   0.73   0.56   0.55   0.59   0.59
```

The 1.19 for Dana in 1951 indicates that in 1951 there were 1.19 times as many male babies named Dana as there were female babies named Dana. We can see that in `a4/yob/1951.txt`, which has the 1951 data.

```
$ grep Dana, a4/yob/1951.txt
Dana,F,1076
Dana,M,1277
```

By 1958 things had changed—there were only .64 males named Dana for every female named Dana:

```
$ grep Dana, a4/yob/1958.txt
Dana,F,2388
Dana,M,1531
```

Note that I'm using the `a4` symlink mentioned at the top of the write-up. There's a comma after "Dana" so that "Danae" doesn't turn up, too.

The data format of the `a4/yob/YEAR.txt` files is simple: lines of name, sex, and the associated count.

`switched.rb` reads the `a4/yob/`*`YEAR`*`.txt` files for all the years in the range specified by the command line arguments and looks for names for which the male/female quotient is > 1.0 in the first year and < 1.0 in the last year. For all the names it finds, it prints the male/female quotient for all the years from the first year through the last year. Names are printed in alphabetical order.

As a specific example, Dana is included in the list for 1951 through 1958 because males/females in 1951 was 1.19 (> 1.0) and males/females in 1958 was 0.64 (<1.0). The quotients for the middle years are not examined to decide whether to include a name; they are shown only to provide a more complete picture of the data between the endpoints.

There's a big shift for Kim from 1954 through 1957. I wonder if that perhaps because the actress Kim Novak had a breakout role in 1955's *Picnic*.

The values are being are being formatted using a `%7.2f` format with `printf`, demonstrated on the command line with `ruby -e:`

```
$ ruby -e 'printf("%7.2f\n", 1277.0/1076.0)'
  1.19
```

Names are left-justified in a 10-wide field using a `printf` format of `%-10s`.

IMPORTANT: To eliminate the less significant results, discard the *YEAR*`.txt` line for a name if the count is less than 100. However, that simple rule has two consequences:

1. If there are 100 or more males and less than 100 females, the quotient is `Inf`; but there's no need for special-case code; `printf` will handle it automatically.

2. There may be a name that's selected based on the first-year to last-year change, but in some middle years falls below 100 for both males and females. In that case, output "`n/a`".

Here's a range that illustrates both cases:

```
$ ruby switched.rb 1948 1952
             1948   1949   1950   1951   1952
Lavern       1.13    Inf    Inf    n/a   0.86
```

Do `grep Lavern, a4/yob/19{48,49,50,51,52}.txt` to see the data underlying the output above.

If no names meet the criteria, `switched` prints "`no names found`" and exits by calling `exit(1)`:

```
$ ruby switched.rb 2011 2012
no names found
```

It's interesting to combine `switched` with a `bash` `for`-loop that runs the program with a gradually shifting range. The output is not shown but here's the command, in case you'd like to try it:

```
for i in $(seq 1940 2002); do ruby switched.rb $i $(($i+9));
echo ====; done
```

Two obvious extensions to `switched` would be command-line options to adjust the 100-baby minimum and to look for female to male flips for a name. You might find those interesting to implement and experiment with, but neither are required.

`switched.rb` does no error handling whatsoever. Behavior is only defined in the case of being given two command line arguments in the range of 1885 to 2012, and the first must be less than the second.

Implementation notes for `switched`

Use `File.open` to produce a `File` object whose `gets` method can be used to read lines. Example:

```
$ cat fileio.rb
year = ARGV[0]

f = File.open("a4/yob/#{year}.txt")

count = 0
```

```
while line = f.gets
    count += 1
end

f.close

puts "read #{count} lines"

$ ruby fileio.rb 2001
read 30258 lines
```

Alternatively, you could use `f.readlines()` to produce an array of all the lines in the file with a single call or `f.each { ... }` to process each line with the associated block.

If you're working on lectura and you've created the recommended `a4` symlink, the path used above in `File.open` will work.

You can download http://www.cs.arizona.edu/classes/cs372/spring14/a4/yob.zip for testing on your own machine. In the same directory you have switched.rb, make a directory named `a4` and then unzip `yob.zip` in that directory to produce a structure compatible with the `File.open` above.

I intend this problem to be an exercise in using the `Hash` class. For those who want to figure it out the structure themselves I won't say anything about my approach here but I'll put out a hint on it. (Remind me if you need it before I get around to it.) My solution makes use of the `keys` and `select` methods of `Hash`, among other things.

Problem 7. Extra Credit `observations.txt`

Submit a plain text file named `observations.txt` with...

(a) (1 point extra credit) An estimate of how long it took you to complete this assignment. To facilitate programmatic extraction of the hours from all submissions have an estimate of hours on a line by itself, more or less like one of these:

```
Hours: 10
Hours: 12-15.5
Hours: 15+
```

Other comments about the assignment are welcome, too. Was it too long, too hard, too detailed? Speak up! I appreciate all feedback, favorable or not.

(b) (1-3 points extra credit) Cite an interesting course-related observation (or observations) that you made while working on the assignment. The observation should have at least a little bit of depth. Think of me saying "Good!" as one point, "Interesting!" as two points, and "Wow!" as three points. I'm looking for quality, not quantity.

Turning in your work

Use the D2L Dropbox named `a4` to **submit a single zip file named `a4.zip` that contains all your work**. If you submit more than one `a4.zip`, we'll grade your final submission. Here's the full list of deliverables:

```
nzip.rb
```

`vrep.rb`
`mirror.rb`
`pancakes.rb`
`calc.rb`
`switched.rb`
`observations.txt` (for extra credit)

DO NOT SUBMIT INDIVIDUAL FILES—submit a file named `a4.zip` that contains each of the above files.

Note that all characters in the file names are lowercase.

Have all the deliverables in the uppermost level of the zip. It's ok if your zip includes other files, too.

Miscellaneous

Restrictions not withstanding, you can use any elements of Ruby that you desire, but the assignment is written with the intention that it can be completed easily using only the material presented on slides 1-153.

The output of your solutions should exactly match the output shown in this write-up. The data files used in the examples can be found in http://www.cs.arizona.edu/classes/cs372/spring14/a4

Point values of problems correspond directly to assignment points in the syllabus. For example, a 10-point problem on this assignment corresponds to 1% of your final grade in the course.

Feel free to use comments to document your code as you see fit, but note that no comments are required, and no points will be awarded for documentation itself. (In other words, no part of your score will be based on documentation.) A # is comment to end of line, unless in a string literal or regular expression. There's no analog to `/* ... */` in Java and `{- ... -}` in Haskell but you can comment out multiple lines by making them an *embedded document*—lines bracketed with `=begin`/`=end` starting in column 1. RPL 2.1.1 has more on comments.

Remember that late assignments are not accepted and that there are no late days; but if circumstances beyond your control interfere with your work on this assignment, there may be grounds for an extension. See the syllabus for details.

My estimate is that it will take a typical CS junior from 7 to 9 hours to complete this assignment.

Keep in mind the point value of each problem; don't invest an inordinate amount of time in a problem or become incredibly frustrated before you ask for a hint or help. Remember that the purpose of the assignments is to build understanding of the course material by applying it to solve problems. If you reach the five-hour mark, regardless of whether you have specific questions, it's probably time to touch base with us. Give us a chance to speed you up! **Our goal is that everybody gets 100% on this assignment AND gets it done in an amount of time that is reasonable for them.**

I hate to have to mention it but keep in mind that cheaters don't get a second chance. 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.)