CSc 372, Fall 2006
Assignment 4
Due: Thursday, October 12 at 23:59:59

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`.

```
>> 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
```

Note that `nzip` is NOT A 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 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`. 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
ab
3.0
ab
two
1
=> nil

>> mirror("abc") { |v| puts v }
abc
=> nil
```

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

Problem 4. (10 points) `pancakes.rb`

Write a Ruby version of the ML 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. (Use `puts`.)

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 35 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.

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

Write in Ruby a simple four-function calculator that evaluates expressions composed of integer literals and variables with 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=11111111111111111*11111111111111111
123456790123456787654320987654321
? 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`. 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"]]
```

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.

If `c` is a character value, perhaps the result of `c = s[0]`, use the expression

```
c >= ?0 and c <= ?9
```

to test whether `c` is a digit.

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

Instructors are always concerned with the progress students are making on programming assignments. One way to judge progress is to see what has been turned in. For this problem you are to write a program that reads a list of files to be turned in for an assignment and, for a given directory tree maintained by the `turnin` program, produces a report in the form of a matrix that shows which students have submitted which files.

A `turnin` tree based on the first assignment in this class and populated with empty files is in `/home/cs372/fall06/a4/ti/a1`. One way to view such a hierarchy is with "`ls -R`":

```
% ls -R /home/cs372/fall06/a4/ti/a1
/home/cs372/fall06/a4/ti/a1:
carlos  howard  martin  rcm

/home/cs372/fall06/a4/ti/a1/carlos:
answers.txt  ftypes.sml  paired.sml  street.sml

/home/cs372/fall06/a4/ti/a1/howard:
to_b.sml
```

```
/home/cs372/fall06/a4/ti/a1/martin:
answers.txt  ftypes.sml  street.sml

/home/cs372/fall06/a4/ti/a1/rcm:
answers.txt  paired.sml
%
```

As can be seen, each student has their own directory named according to their login id; submissions by that student are in that directory. A student's directory will not be present unless at least one file has been submitted. Students sometimes submit files that are not one of the deliverables.

`status` is invoked with a single argument that is the name of a configuration file. The first line of the configuration file is the name of the root of the `turnin` hierarchy; following lines specify the files to be turned in for the assignment. Here is a configuration file for the assignment:

```
% cat /home/cs372/fall06/a4/a1.cfg
/home/cs372/fall06/a4/ti/a1
ftypes.sml
to_b.sml
cpfx.sml
paired.sml
street.sml
editstr.sml
answers.txt
```

For the `a1` hierarchy shown above, `status` shows this:

```
% ruby status.rb /home/cs372/fall06/a4/a1.cfg
Name   ftypes  to_b  cpfx  paired  street  editstr  answers
carlos   x                x       x                x
howard          x
martin   x                        x                x
rcm                       x                        x
```

The output of `status` is in a precise format. The header line starts with "`Name`" left-justified in a field that is exactly as wide as the longest login id or 4, whichever is larger. Following it are the problem names, which are the file names minus the suffix. Problem names are shown in the same order as in the configuration file and are centered in a field of with `W+2` where `W` is the width of the problem name.

Students are shown in alphabetical order by login id. If a particular solution is present, an "`x`" is shown in the appropriate column. The "`x`" is centered in a field of width `W+2`, where `W` is the width of the problem name. (Use `String.center` to do the centering.)

Assume that all file names are of the form *`name.suffix`*. You won't see things like `.sml`, `x.`, or `abc`.

Any files present in a student's directory other than expected ones should be ignored.

Don't worry about trailing whitespace in your output—we'll ignore it when testing.

Formatting is a very important part of this problem. Be sure to use `tester` to check your output.

Login ids can be any width greater than zero. File names have a minimum width of three (`x.y`, for example) but no maximum. You'll need to handle any combination of long and short names. In some cases the output may wrap around. Don't worry about that—maybe the user has a very wide screen.

Here's another example:

```
% cat /home/cs372/fall06/a4/a2.cfg
/home/cs372/fall06/a4/ti/a2
y.x
graphical_user_interface_generator.rb
ParseXML.icn
z.v3rev4B

% ruby status.rb /home/cs372/fall06/a4/a2.cfg
Name           y  graphical_user_interface_generator  ParseXML  z
fed                              x
hatpc                                                    x
jerry_bob_2006                                                 x
MasterHacker   x
nate
peter          x                 x                       x
wnj                              x                       x      x
z              x                 x                       x      x
```

Note that there is a line for `nate` but no solutions are shown. That may mean that `nate` has submitted a file that is misnamed. (We'll let `nate` worry about that.) Login ids are ordered in a case-insensitive way—that's why `MasterHacker` appears in the middle.

Assume that a single configuration file is always specified and that the file exists. Assume the configuration file is well-formed, has no empty lines, specifies a directory that exists, and that at least one problem is specified.

If no students have yet submitted a file, i.e., the tree is empty, `status` should simply print "`No submissions`" and exit.

At press time, two `turnin` trees exist in `/home/cs372/fall06/a4/ti`: `a1` and `a2`. More will likely appear. The config files used above are in `fall06/a4`.

Implementation notes

To open a file, use `File.open`. `open` returns an IO object. Use `gets` to read from the file. Like this:

```
cfio = File.open(config_file)
dir = cfio.gets.chomp
```

Remember that you can assume that the config file exists.

To get an array of the entries in a directory, use `Dir.entries`:

```
>>  Dir.entries("/home/cs372/fall06/a4/ti/a1")
=> [".", "..", "martin", "rcm", "carlos", "howard"]
```

Discard "`.`" and "`..`". Assume the remaining entries are student directories.

Problem 7. (1 point extra credit) `extra.txt`

Estimate how long it took you to complete this assignment. Other comments about the assignment are welcome, too—I appreciate all feedback, good or bad.

Deliverables

The deliverables for this assignment are these files: `nzip.rb`, `vrepl.rb`, `mirror.rb`, `pancakes.rb`, `calc.rb`, and `status.rb`. That list can be found on `lectura` in `/home/cs372/fall06/a4/delivs`.

Use the `turnin` tag `372_4` 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

We'll provide a `tester` script that includes all the examples in this write-up and possibly more. It should be in place by noon on Saturday, October 6. Let us know if it appears to be overdue.

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.

There are no restrictions on any problems in this assignment. 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 1-108, keeping in mind the "canned" regular expression provided for `calc.rb`.

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.)