CSc 451, Spring 2003
Assignment 5
Due: Thursday, February 27 at 18:00

Problem 1. (10 points) `mcycle.icn`

For this problem you are to write a very simple macro processor. `mcycle` reads lines from standard input and writes lines to standard output. Here is a sample input file: (`mcycle.1`)

```
color=red,green,blue
what=pencil,crayon
   The <color> <what> made a <color> mark that the <color>
<what> erased.  Then the <what> became a <what>.
```

The input has two sections. The first is a series of "variables", consisting of a name, an equals sign, and one or more values, separated by commas. The second is a text section, a series of lines into which the values of the variables are to be cyclically substituted. Here is the result of running `mcycle`:

```
% mcycle < mcycle.1
   The red pencil made a green mark that the blue
crayon erased.  Then the pencil became a crayon.
%
```

Each value of a variable is substituted in turn, possibly cycling through the values of a variable many times.

The transition between the variable section and the text section is indicated by the absence of an equals sign on a line.

To keep things simple, you only need to be sure that your program works with two input files: `mcycle.1` and `mcycle.2`, both in the reference version directory.

Problem 2. (5 points) `day.icn`

Write a program `day` that accepts a date as a command line argument and prints the day of the week on which the date falls. There is a well-known algorithm for calculating the day of the week. Unfortunately, you can't find it. Instead, you'll have to rely on the venerable UNIX `cal` command. Your implementation of `day` will open a pipe to read from `cal` and process the data that `cal` produces. Here is a sample of `cal` output:

```
% cal 2 2003
   February 2003
 S  M Tu  W Th  F  S
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28
```

Here is `day` in action:

```
% day 2/13/2003
Thursday

% day 1/1/2000
Saturday

% day 11/31/2002
No such day
```

This problem is made more interesting by September of 1752:

```
% cal 9 1752
   September 1752
 S  M Tu  W Th  F  S
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30

% day 9/14/1752
Thursday
```

Aside from the names of the days of the week your program should contain no calendar-specific information such as the number of months in a year or the number of days in a given month. It should simply rely on `cal`.

You may assume that the command line argument is well-formed and that the month and year specified are acceptable to `cal`. You may assume that the day of the month is a positive integer.

Problem 3. (30 points) `mtimes.icn`

Write a program `mtimes` that reads a text file with data on theaters, movies, and showtimes, and for each movie of interest, displays a table of showtimes for the movie.
Here is a subset of the data: (in `mtimes.ex`)

```
El Con
Biker Boyz (PG-13) (12:25, 2:00, 5:25), 6:30, 8:00
Confessions of a Dangerous Mind (R) (4:20), 10:15
Darkness Falls (PG-13) (12:15, 2:15, 4:15), 6:15, 8:15, 10:15
The Lord of the Rings: The Two Towers (PG-13) (12:20, 4:10), 7:55

Gateway
Narc (R) (2:20, 4:40, 7:10, 9:30)
Drumline (PG-13) (1:50, 4:25, 7:10, 9:40)
Star Trek Nemesis (PG-13) (1:30, 4:10, 7:00, 9:35)

Park Place
Biker Boyz (PG-13) (12:05, 2:35, 5:05), 8:00
Confessions of a Dangerous Mind (R) (11:45, 2:25, 5:00), 7:45
Darkness Falls (PG-13) (11:35, 1:45, 3:55, 6:00), 8:15, 10:25
```

`mtimes` is run with one or more command line arguments, each a word that is in the title of one or more movies. Showtime information is displayed for each matching movie, if any. Information is displayed in alphabetical order by movie name. Examples:

```
% mtimes of

Confessions of a Dangerous Mind
         El Con      Park Place
11:45                    x
 2:25                    x
 4:20      x
 5:00                    x
 7:45                    x
10:15      x

The Lord of the Rings: The Two Towers
       El Con
12:20    x
 4:10    x
 7:55    x
% mtimes xyz
%
```

If run with no command line arguments, `mtimes` prints information for all movies.

A movie is listed iff a command line argument is a full word in the title. A movie is never listed more than once. Words are treated in a case-insensitive manner. Example:

```
% mtimes star boYz fall on Trek

Biker Boyz
         El Con      Park Place
12:05                    x
12:25      x
 2:00      x
 2:35                    x
 5:05                    x
 5:25      x
 6:30      x
 8:00      x             x

Star Trek Nemesis
       Gateway
 1:30     x
 4:10     x
 7:00     x
 9:35     x
```

There will be only one data file used for grading purposes: `/home/cs451/a5/mtimes.1`. It will not change. Your program should be able to handle all the data in that file but there's no need to contemplate "what-if"s with other data. One fact you should take advantage of is that no movie starts before 10:00am or at/after midnight.

For testing purposes you'll probably find it convenient to have the program read from standard input, so that you can easily test with subsets of the data, but for final submission "wire-in" the data file name, like this:

```
timefile := open("/home/cs451/a5/mtimes.1", "r")
```

In terms of ballpark size, I expect that good solutions will require 75-125 lines of code.

Use the built-in `map()` function to meet the case-insensitivity requirement.

For your convenience the reference version of `mtimes` accepts a `"-"` argument that directs the program to read the data from standard input instead of `mtimes.1`. Example:

```
mtimes - < mtimes.ex star
```

You do not need to implement that behavior in your versions.

In addition to the cases shown above, here are some invocations I'll test with:

```
mtimes
mtimes the
mtimes a and the
mtimes rings
mtimes 1 to 2
mtimes about that schmidt fisher
mtimes drum
mtimes drumline
mtimes hour 2
```

Be sure to use `diff` to compare your output with that of the reference version. Note that the columns are centered in a field that is four characters wider than the longest theater name, for a given movie.

Miscellaneous

Restriction: To encourage creative use of the facilities we have already discussed, YOU MAY NOT USE any of the string scanning facilities and related functions on ANY OF THESE PROBLEMS. Specifically, you may not use these functions: `any`, `bal`, `find`, `many`, `match`, `move`, `pos`, `tab`, and `upto`. With a little bit of thinking you'll find that `split` is enough. If you find yourself writing code that does character by character processing, such as looping through a string looking for a character, you're most likely making the problem too hard.

No comments or explanation of any sort need be included with your solutions.

You are specifically prohibited from directly copying any code, except that presented in class or otherwise provided by me. However, you may study discovered code, such as that found in a textbook—not the code of a classmate—to the point of understanding how it works and then with that knowledge, write your own version.

Deliverables

Use `turnin` with the tag `451_5` to submit your solutions for grading. The deliverables for this assignment are `mcycle.icn`, `days.icn`, and `mtimes.icn`,

Reference Versions

Reference versions of all programs can be found in `/home/cs451/a5`.