

Laboratory: Files

Summary: In this laboratory, we explore file creation, input, and output in Scheme.

Contents:

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Reading Values
 - Exercise 2: Summing Values
 - Exercise 3: File Length
 - Exercise 4: Missing Files
 - Exercise 5: Reading All Lines
 - Exercise 5: Displaying Files
 - Exercise 7: Creating Files
 - Exercise 8: Finding Divisors
 - Exercise 9: Reusing Output Files
 - Exercise 10: Counting Characters

Exercises

Exercise 0: Preparation

- a. Scan through this lab to determine what kinds of tasks you'll need to complete.
- b. You should also scan through the reading on files in Scheme.
- c. Start DrScheme.

Exercise 1: Reading Values

As you may recall from the reading, I have prepared two simple files for simple exploration of input and output, `/home/rebelsky/Web/Courses/CS151/2006F/Examples/hi.dat`, and `/home/rebelsky/Web/Courses/CS151/2006F/Examples/sample.dat`.

- a. Using a sequence of commands in the interaction window, read all the characters from `hi.dat`. For example,

```

> (define source
  (open-input-file "/home/rebelsky/Web/Courses/CS151/2006F/Examples/hi.dat"))
> (read-char source)
#\H
...
> (read-char source)
#<eof>
> (close-input-port source)

```

- b. Using a sequence of commands in the interaction window, read all the characters from `sample.dat`.
- c. Using a sequence of commands in the interaction window, read all the values from `sample.dat` (use `read` rather than `read-char`).
- d. Using a sequence of commands in the interaction window, read all the values from `hi.dat`.

Exercise 2: Summing Values

The file `/home/rebelsky/Web/Courses/CS151/2006F/Examples/numbers.dat` contains five hundred and twenty-eight natural numbers.

- a. Use `sum-of-file` from the reading to determine their sum.
- b. How would you quickly determine if your attempt to sum those numbers was correct?

Note: That file was copied from a similar file produced by Mr. Stone.

Exercise 3: File Length

Using `sum-of-file` (and its helpers) as a *pattern*, write a Scheme procedure `file-size` that takes as argument a string that names a file and returns the number of characters in that file (that is, the number of times that `read-char` can be called to read a character from the file without returning the end-of-file object).

Exercise 4: Missing Files

Find out what happens if `sum-of-file` or `file-size` is given a string that does not name any existing file.

Exercise 5: Reading All Lines

Write a procedure, `read-lines`, that takes the name of a file as a parameter and returns a list of all the lines of the file (with each line represented as a string). You can certainly use the `read-line` procedure from the reading as a helper for `read-lines`. You might use `sum-of-file` as a pattern for `read-lines`.

Exercise 5: Displaying Files

Write a procedure, `display-file`, that takes the name of a file as a parameter and displays the contents of the file with each line preceded by its line number. You may find it useful to call upon the previous procedure to do the reading.

Exercise 7: Creating Files

a. In the interactions pane, write a series of expressions that will create a file, `my-info`, with the following lines (substituting your own name and major).

```
Name: last, first
Major: major-or-undeclared
```

b. Write a Scheme procedure, `(dump-info file-name last-name first-name major)` that, given four strings as parameters, writes the following to the file named by `file-name`,

```
Name: last-name, first-name
Major: major
```

Exercise 8: Finding Divisors

Use the `store-divisors` procedure from the reading to draw up a list of the divisors of 120, storing them in a file named `divisors-of-120`. Examine the file afterwards and confirm that the answer is correct.

By the way, don't give this procedure an extremely large number as argument -- it's much too slow. There are more efficient ways to find divisors!

Exercise 9: Reusing Output Files

The Scheme standard says that if you try to open an output port to a file that already exists, “the effect is unspecified”, i.e., anything might happen. Hence, designers of a particular implementation of Scheme are free to do what they choose.

a. Find out through experimentation what DrScheme does in this situation.

b. Find out using DrScheme's Help Desk what DrScheme allows programmers to do in this situation. (The help desk document is entitled *Opening file ports*.)

Exercise 10: Counting Characters

Write a Scheme procedure that takes as arguments two file names (an input file and an output file), counts the number of occurrences of each vowel in the input file, and writes the result to the output file. Note that the output file should have the following form (with numbers in place of the number signs):

a: ###
e: ###
i: ###
o: ###
u: ###

If you have time or inclination, extend your procedure to deal with all 127 ASCII characters.

Copyright © 2006 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.