

## Format for Lab Write-Ups

At times during this semester, I will ask you to write up your laboratory exercises. This document provides some basic guidelines for laboratory writeups.

- File Types and Names
- Starting the File
- Sample Output
- Documenting Procedures
- Turning it in

### File Types and Names

The writeup should be typed and saved as a `.ss` file. For example, you might save laboratory writeup one as `smith.writeup1.ss`. Please make sure that you include a group member's name as part of the file name so that the graders and I can distinguish them.

### Starting the File

The writeup should begin with Scheme comments that give

- The names and mailboxes of all students in the group
- The course and professor
- The homework name or number
- The date
- The file name
- Where (if anywhere) it can be found
- Any citations you think are appropriate For example,

```
;;; Sarah Schemer '01, box 01-01
;;; Steven Schemer '04, box 00-08
;;; CSC151-01, Spring 2001, Samuel A. Rebelsky
;;; Laboratory Writeup 1: Conditionals, Strings, Recursion
;;; Friday, 23 February 2001
;;; File name: schemers.writeup.01.ss
;;; Available in MathLAN as
;;; /home/schemers/cs151/schemers.writeup.01.ss
;;; Available on the Web as
;;; http://www.cs.grinnell.edu/~schemers/cs151/schemers.writeup.01.ss
;;; Assistance from:
;;; Sam Rebelsky, our beloved but disheveled professor
;;; (Problems 1, 3, and 5)
;;; Anne Feltovich, our astounding class assistant
;;; (Problems 1 and 2)
```

```

;;; Harvey and Hermione Hacker, our somewhat confused classmates
;;; (Problem 2)
;;; Some student at MIT who posted the answer to problem 4.
;;; The URL is ...

```

You can often create your .ss file by starting with the log from a session (or from your definitions window), which you've saved as a text file. My teaching assistants and I will load the .ss file and execute it, comparing output as we go.

## Sample Output

You should include sample output for any test expressions in your program. That sample output and any comments you have should be preceded by semicolons. For example, if you were testing the `list` procedure, you might write:

```

(display "Testing list with no parameters")
(list)
; ()
; Hmm ... no parameters gives the empty list.

(display "Testing list with null as a parameter")
(list null)
; (()
; No, that's not the same thing. Why not? Perhaps I
; need to check lengths.

(length (list))
; 0
; Okay, nothing is in that list, so the length is 0.

(length (list null))
; 1
; Hmmm ...that's not the same as the empty list. Ah! I remember,
; Sam said you can have lists in lists. So this must be the list
; of the empty list, which means that there's one element.

```

Note that you do not need to include the observations in most cases, although I do think they help you remember why you were doing the tests and what confused you (or what they showed you).

Note also that I have not included the Scheme prompts (`>`) in my file. You should not include the prompts, either.

## Documenting Procedures

Whenever you write your own procedures, you should make sure to document them with the 6P's:

- The *procedure* (that is, its name);
- The *parameters* (their names, their expected types, their semantics);
- The *purpose* (what the procedure does);
- The *produced value* (what the result is);
- The *preconditions* (and whether or not you've bothered to verify them); and

- The *postconditions* (some of which you should express as formal as you can).

The preconditions represent requirements you have in order for your procedure to work. The postconditions represent guarantees about the result (and the state of the Scheme system after you're done). For more information, see the reading on preconditions and postconditions.

Here's an example of a procedure with the 6P's.

```

;;; Procedure:
;;; markup
;;; Parameters:
;;; tag, the role that some text plays
;;; text, some text to mark up
;;; Purpose:
;;; Generates some nice happy HTML for me.
;;; Produces:
;;; A string for HTML that represents the marked text.
;;; Preconditions:
;;; Both parameters are strings.
;;; The text is valid HTML.
;;; The tag is nonempty, contains only alphanumeric characters and
;;; corresponds to a valid HTML tag.
;;; Postconditions:
;;; You get some nice HTML.
;;; Does not affect the tag or the text.
;;; Examples:
;;; (markup "p" "hello")
;;; => "<p>hello</p>"
;;; (markup "strong" (string-append "Scheme " (markup "em" "rules")))
;;; => "<strong>Scheme <em>rules</em></strong>"
;;; Note:
;;; If we wanted to support multiple kinds of markup (e.g., LaTeX and RTF
;;; in addition to HTML), we could use a global setting. That is a
;;; task for the far-far future, though.
(define markup
  (lambda (tag text)
    (string-append "<" tag ">" text "</" tag ">")))

```

As this example suggests, you may also want to include some notes and some examples.

## Turning it in

Here's how to use the dropbox:

- Go to <http://blackboard.grinnell.edu>
- Select this course.
- When prompted to log in, use your normal account as the name Use a variant of your student ID number as password. Returning students use the last seven digits of student ID as password. New students use trailing non-zero digits. [Due to stupidity in the design of the system, you may be required to log in twice.]
- Click on "Tools"
- Click on "Digital Drop Box"

- Click on the "Browse" button to select a file.
- Type in something useful for the title, like "Homework 1 from Sarah and Stephen"
- Click on "Submit"
- Pray