

Laboratory: Input and Output

Summary: In this laboratory, you will experiment with the use and application of some of Scheme's basic input and output procedures.

Procedures Covered: `read`, `write`, and `display`.

Contents

- Exercises
 - Exercise 0: Preparation
 - Exercise 1: Simple Input and Output
 - Exercise 2: Running from the Command Line
 - Exercise 3: Local Input Values
 - Exercise 4: Computing Roots of a Quadratic Equation
 - Exercise 5: Repetition
 - Exercise 6: Debugging

Exercises

Exercise 0: Preparation

a. Make sure that you understand what `read`, `write`, and `display` are supposed to do. You may find an old reading on the topic helpful.

b. Start DrScheme.

Exercise 1: Simple Input and Output

Consider the following sequence of Scheme commands:

```
(display "Please enter a value and I will square it: ")
(define val (read))
(define val-squared (* val val))
(display (string-append "The value of "
                        (number->string val)
                        " squared is "
                        (number->string val-squared)))
(newline)
```

a. What do you expect the code to do?

b. Verify your answer via experimentation.

Exercise 2: Running from the Command Line

- a. Save the above code in a file (e.g., `square.ss`).
- b. Open a terminal window.
- c. In that terminal window, type

```
mzscheme -r file.ss
```

- d. Reflect on what happened. Did you need to type any Scheme? Could someone else use step c without understanding the underlying Scheme?

Exercise 3: Local Input Values

Rewrite the code in exercise 1 to use `let` or `let*` (or both) rather than `define`.

Exercise 4: Computing Roots of a Quadratic Equation

- a. Write a Scheme program that reads in the three coefficients of a quadratic equation (the a , b , and c in $ax^2 + bx + c$) and prints out the roots of the equation. You should model this program on exercise 3.
- b. Save the program in a file and execute it from the command line.
- c. Reflect as in exercise 2d.

Exercise 5: Repetition

Update your answer to number 3 so that your program repeatedly asks for a value and computes its square root. You will probably need to

- a. Put the stuff (request for value, computation of a value, display of that value) in a procedure.
- b. Have that procedure recurse.
- c. Decide upon a base case to stop recursion. (I'd suggest that you stop when someone enters something other than a number.)

Exercise 6: Debugging

Consider the following updated version of the `assoc-all` procedure the we wrote together in class. It is updated in that we have added five lines that let us display what's going on as the procedure is called.

```
(define assoc-all
  (lambda (key database)
    (display "Searching for ")
    (display key)
    (display " in ")
    (display database)))
```

```

(newline)
; Nothing can be in the empty database, so return false.
(if (null? database)
    #f
    ; If we match the car of the database, look in the rest.
    (if (equal? key (car (car database)))
        ; If the key does not appear any more in the database
        ; make a list of just the one value we've seen
        (if (not (assoc-all key (cdr database)))
            (list (car database))
            ; Otherwise, attach the matched entry to
            ; the remaining matched entries
            (cons (car database) (assoc-all key (cdr database))))))
        ; We didn't match the car, so just look in the rest
        (assoc-all key (cdr database))))))

```

a. What do you expect to happen for each of the following calls? Verify your answer through experimentation.

i.

```
(assoc-all "Sam" null)
```

ii.

```
(assoc-all "Sam"
  (list (list "Sam" "A")
```

iii.

```
(assoc-all "Sam"
  (list (list "Rebelsky" "A")
```

iv.

```
(assoc-all "Sam"
  (list (list "Sam" "A")
        (list "Sam" "B")
        (list "Sam" "C")
        (list "Sam" "D")
        (list "Sam" "E"))))
```

iv.

```
(assoc-all "Sam"
  (list (list "Sam" "A")
        (list "John" "B")
        (list "Sam" "C")
        (list "Jack" "D")
        (list "Joe" "E"))))
```

b. Explain the results.