CS 61A, Summer 2006 June 26, 2006 (and June 27, 2006)
LAB ASSIGNMENT 1A

Note: This lab is for Monday and for Tuesday (instead of meeting in discussion on Tuesday, you will meet again in lab). So don't panic if you can't finish it all on Monday.

PART I

1. Pick up an account form from your TA and login. The first thing to do is to change your password. A sample interaction is shown below. Make sure to logout of po after you successfully change your password! Be aware that it may take a few minutes before your new password is recognized by all the machines.

```
nova [1] ~ > ssh po
po [1] ~ > passwd
Enter old password:
Enter new password:
Re-enter new password:
Password successfully changed!
po [2] ~ > logout
nova [2] ~ >
```

2. Are you new to Unix and/or Emacs and/or newsgroups? Your TA should have a handout entitled *A Somewhat Quick Introduction to Using CS 61A Computing Resources*. Pick up a copy and read through it before doing anything else.

3. Set up the newsgroup. The *Somewhat Quick Introduction* handout will have instructions on how to do this. I suggest that you use pine as your news reader, but you are free to use whatever you want, if you already have and know how to use another news reader. **Do not use Google Groups to post on the newsgroup**; such posts won't show up in pine (i.e. I won't see it). The class newsgroup is `ucb.class.cs61a`, and Berkeley's news server is `news.berkeley.edu`.

4. Start the Emacs editor, either by typing `emacs` (or `emacs &`) in your main window or by selecting it from the right-mouse-button menu. (Your TA will show you how to do this.) From the `Help` menu, select the Emacs tutorial. You need not complete the entire tutorial at the first session, but you should do so eventually.

5. Start Scheme, either by typing `stk` in your main window or by typing meta-S in your Emacs window. Type each of the following expressions into Scheme, ending the line with the Enter (carriage return) key. **Think about the results!** Try to understand how Scheme interprets what you type.

```
3
(+ 2 3)
(+ 5 6 7 8)
(+)
(sqrt 16)
(+ (* 3 4) 5)
+
'+
'hello
'(+ 2 3)
'(good morning)
(first 274)
(butfirst 274)
```

```
(first 'hello)
(first hello)
(first (bf 'hello))
(+ (first 23) (last 45))
(define pi 3.14159)
pi
'pi
(+ pi 7)
(* pi pi)
(define (square x) (* x x))
(square 5)
(square (+ 2 3))
```

6. Use Emacs to create a file called `pigl.scm` in your directory containing the Pig Latin program shown below:

```
(define (pigl wd)
  (if (pl-done? wd)
      (word wd 'ay)
      (pigl (word (bf wd) (first wd)))))

(define (pl-done? wd)
  (vowel? (first wd)))

(define (vowel? letter)
  (member? letter '(a e i o u)))
```

Make sure you are editing a file whose name ends in `.scm`, so that Emacs will know to indent your code correctly!

7. Now run Scheme. You are going to create a transcript of a session using the file you just created, like this:

```
(transcript-on "lab1a")      ; This starts a transcript file with the name lab1a.
(load "pigl.scm")            ; This reads in the file you created earlier.
(pigl 'scheme)               ; Try out your program.
                             ; Feel free to try more test cases here!
(trace pigl)                 ; This is a debugging aid. Watch what happens
(pigl 'scheme)               ; when you run a traced procedure.
(transcript-off)
(exit)
```

8. Use `lpr` to print your transcript file. (If your transcript file is called `lab1a`, then type `lpr lab1a` at your shell.)

PART II

1. Predict what Scheme will print in response to each of these expressions. *Then* try it and make sure your answer was correct, or if not, that you understand why!

```
(define a 3)

(define b (+ a 1))

(+ a b (* a b))

(= a b)

(if (and (> b a) (< b (* a b)))
    b
    a)

(cond ((= a 4) 6)
      ((= b 4) (+ 6 7 a))
      (else 25))

(+ 2 (if (> b a) b a))

(* (cond ((> a b) a)
         ((< a b) b)
         (else -1))
   (+ a 1))

((if (< a b) + -) a b)
```

2. In the shell, type the command:

```
cp ~cs61a/lib/plural.scm .
```

Note the period at the end of the line! This will copy a file from the class library to your own directory. Then, using emacs to edit the file, modify the procedure so that it correctly handles cases like `(plural 'boy)`.

3. Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

4. Write a procedure `dupls-removed` that, given a sentence as input, returns the result of removing duplicate words from the sentence. It should work this way:

```
> (dupls-removed '(a b c a e d e b))
(c a d e b)
> (dupls-removed '(a b c))
(a b c)
> (dupls-removed '(a a a a b a a))
(b a)
```