CS 61A, Summer 2006 June 26, 2006
HOMEWORK ASSIGNMENT 1A
DUE WEDNESDAY JULY 5, 2006 AT 11:59 PM

People who've taken CS 3: Don't use the CS 3 higher-order procedures such as `every` in these problems; use recursion.

Recall that in lecture we defined a **word** to be a quoted string of alphanumerics (plus possibly some other characters like question marks or exclamation marks, but not spaces), and a **sentence** as a flat sequence of words (words separated by spaces and delimited by parentheses). Some examples of the procedures which deal with words and sentences that you may use in this assignment are: `sentence` (or `se`), `word`, `first`, `butfirst` (or `bf`), `butlast` (or `bl`), `empty?`, `equal?`, `member?`. (There may be others, but I can't think of them off the top of my head right now.)

1. Do exercise 1.6, page 25. This is an essay question. If you had trouble understanding the square root program in the book, explain instead what will happen if you use `new-if` instead of `if` in the `pigl` Pig Latin procedure.

2. Write a procedure `squares` that takes a sentence of numbers as its argument and returns a sentence of the squares of the numbers:

```
> (squares '(2 3 4 5))
(4 9 16 25)
```

3. Write a procedure `switch` that takes a sentence as its argument and returns a sentence in which every instance of the words `I` or `me` is replaced by `you`, while every instance of `you` is replaced by `me` except at the beginning of the sentence, where it's replaced by `I`. (Don't worry about capitalization of letters.) Example:

```
> (switch '(You told me that I should wake you up))
(i told you that you should wake me up)
```

4. Write a predicate `ordered?` that takes a sentence of numbers as its argument and returns a true value if the numbers are in ascending order, or a false value otherwise.

5. Write a procedure `ends-e` that takes a sentence as its argument and returns a sentence containing only those words of the argument whose last letter is E:

```
> (ends-e '(please put the salami above the blue elephant))
(please the above the blue)
```

6. Most versions of Lisp provide `and` and `or` procedures like the ones on page 19. In principle there is no reason why these can't be ordinary procedures, but some versions of Lisp make them special forms. Suppose, for example, we evaluate

```
(or (= x 0) (= y 0) (= z 0))
```

If `or` is an ordinary procedure, all three argument expressions will be evaluated before `or` is invoked. But if the variable `x` has the value `0`, we know that the entire expression has to be true regardless of the values of `y` and `z`. A Lisp interpreter in which `or` is a special form can evaluate the arguments one by one until either a true one is found or it runs out of arguments.

Your mission is to devise a test that will tell you whether Scheme's `and` and `or` are special forms or ordinary functions. This is a somewhat tricky problem, but it'll get you thinking about the evaluation process more deeply than you otherwise might.

Why might it be advantageous for an interpreter to treat `or` as a special form and evaluate its arguments one at a time? Can you think of reasons why it might be advantageous to treat `or` as an ordinary function?