Principles of Programming, Spring 2006
Practice 3

Daejun Park, Heejong Lee
Programming Research Lab.@SNU

March 23, 2006

1. Define a precedure `last-pair` that returns the list that contains only the last element of a given (nonempty) list.

   ```
   (last-pair (list 23 72 149 34))
   ;(34)
   ```

2. Define a procedure `reverse` that takes a list as argument and returns a list of the same elements in reverse order.

   ```
   (reverse (list 1 4 9 16 25))
   ;(25 16 9 4 1)
   ```

3. The procedures `+`, `*`, and `list` take arbitrary numbers of arguments. One way to define such procedures is to use `define` with *dotted-tail notation*. In a procedure definition, a parameter list that has a dot before the last parameter name indicates that, when the procedure is called, the initial parameters (if any) will have as values the initial arguments, as usual, but the final parameter's value will be a *list* of any remaining arguments. For instance, given the definition

   ```
   (define (f x y . z) <body>)
   ```

   the procedure f can be called with two or more arguments. If we evaluate

   ```
   (f 1 2 3 4 5 6)
   ```

   then in the body of `f`, `x` will be `1`, `y` will be `2`, and `z` will be the list `(3 4 5 6)`. Given the definition

   ```
   (define (g . w) <body>)
   ```

the procedure `g` can be called with zero or more arguments. If we evaluate

```
(g 1 2 3 4 5 6)
```

then in the body of `g`, `w` will be the list `(1 2 3 4 5 6)`.

Use this notation to write a procedure `same-parity` that takes one or more integers and returns a list of all the arguments that have the same even-odd parity as the first argument. For example,

```
(same-parity 1 2 3 4 5 6 7)
;(1 3 5 7)

(same-parity 2 3 4 5 6 7)
;(2 4 6)
```

4. The procedure `square-list` takes a list of numbers as argument and returns a list of the squares of those numbers.

```
(square-list (list 1 2 3 4))
;(1 4 9 16)
```

   Complete the procedure by filling in the missing expressions.

```
(define (square-list items)
  (if (null? items)
      nil
      (cons <??> <??>)))
```

5. We can make the procedure `square-list` easily by using the `map` procedure.

```
(define (square-list items))
  (map square items))
```

   Define a procedure `myMap` that acts like `map`.

```
(myMap square (list 1 2 3 4))
;(1 4 9 16)

(myMap abs (list -1 -2 -3 -4))
;(1 2 3 4)
```