SI 413 Fall 2012: Homework 4

Your name:

Due: Friday, 14 September, before class

Instructions: Review the course honor policy for written homeworks.

This cover sheet must be the front page of what you hand in. Fill out the left column in the table to the right after we go over each problem in class, according to the rubric below.

This rubric is also on the website, in more detail, under “Other Stuff”→“Grading Rubrics”.

Make sure all problems are submitted IN ORDER.

- **5**: Solution is completely correct, concisely presented, and neatly written.
- **4**: The solution is mostly correct, but one or two minor details were missed, or the presentation could be more concise.
- **3**: The main idea is correct, but there are some significant mistakes. The presentation is somewhat sloppy or confused.
- **2**: A complete effort was made, but the result is mostly incorrect. There may be some basic misunderstandings of the topic or the problem.
- **1**: The beginning of an attempt was made, but the work is clearly incomplete.
- **0**: Not submitted.

Problem	Self-assessment	Final assessment
1		
2		
3		
4		
5		
6		

Comments or suggestions about this homework:

Comments or suggestions about the course so far:

Citations (other students, websites, . . .):

Use a separate sheet of paper for your answers! Many of these exercises are programming exercises, but you do not need to submit them electronically. Everything should be submitted in one packet, all printed out for me to see.

1 Reverse order print

Write a function `(print-reverse L)` that takes a list `L` and prints its elements in reverse, one per line, and then returns `(void)`.

For example, calling `(print-reverse '(1 2 3))` should print

```
3
2
1
```

2 Pretty-print inches

Write a function `(print-height inches)` that takes a number of inches and prints the feet and inches nicely. For example, calling `(print-height 70)` should cause the following to be printed:

```
5 feet 10 inches
```

Once this works, make it nicer so that, for example `(print-height 73)` prints

```
6 feet 1 inch
```

(notice not inches) and `(print-height 8)` just prints

```
8 inches
```

and any other cases which seem sensible to you.

3 Pushups

A certain sports team scores points in varying increments (2, 3, 6, 7), and after each score, certain fans must perform a number of push-ups corresponding to the total score at that time.

Write a function `(pushups points)` that takes the points from the most recent score, and returns the total amount of push-ups that must be performed at that time. (You will have to use mutation!)

For example, if we start with `(pushups 3)`, the returned value is 3. But if the next call is `(pushups 7)`, the returned value is 10, since that is the total score at that point.

4 Total Pushups

Using your `pushups` function from the last exercise, write a function `(total-pushups L)` that takes a list of scores `L` and returns the total number of pushups performed.

There are a number of ways you could write this, but I want you to do it by using your `pushups` function from the last exercise, along with **map** and **apply**.

For example, `(total-pushups '(3 7))` should return 13, but `(total-pushups '(7 3))` should return 17.

5 Memoized Efficiency Counter

In class we showed two versions of the Fibonacci function:

```
(define (fib n)
  (if (<= n 1)
      n
      (+ (fib (- n 1))
         (fib (- n 2)))))
```

and the memoized version:

```
(define fib-hash (make-hash))

(define (fib-memo n)
  (cond [(not (hash-has-key? fib-hash n))
         (hash-set!
          fib-hash
          n
          (if (<= n 1)
              n
              (+ (fib-memo (- n 1))
                 (fib-memo (- n 2)))))])
  (hash-ref fib-hash n))
```

I want you to count how many times each of these functions is called recursively in order to compute the 30'th Fibonacci number. Create two global variables `fib-count` and `fib-memo-count`, each initialized to 0, and an instruction in each of the `fib` function and the `fib-memo` function to increment this counter (increase by 1) every time the function is called.

Tell me how many times `fib` is called recursively to compute `(fib 30)`, and how many times `fib-memo` is called recursively to compute `(fib-memo 30)`.

6 Tail Recursion

Write new definitions for the following two functions from Homework 2 so that they are tail-recursive:

a) Write a recursive function `split-digits` that takes a number `n` and returns a list with the digits of `n`, in reverse.

b) Write a function called `count-down` which takes a positive integer `n` and produces a list with the integers from `n` down to `1`, in that order.