CS 61A Week 3

Topic: Recursion and iteration

Reading: Abelson & Sussman, Section 1.2 through 1.2.4 (pages 31–47)

Homework:

Note: Programming project 1 will also be due next week.

1. Abelson & Sussman, exercises 1.16, 1.35, 1.37, 1.38

2. A "perfect number" is defined as a number equal to the sum of all its factors less than itself. For example, the first perfect number is 6, because its factors are 1, 2, 3, and 6, and 1+2+3=6. The second perfect number is 28, because 1+2+4+7+14=28. What is the third perfect number? Write a procedure `(next-perf n)` that tests numbers starting with `n` and continuing with `n+1`, `n+2`, etc. until a perfect number is found. Then you can evaluate `(next-perf 29)` to solve the problem. Hint: you'll need a `sum-of-factors` subprocedure.

[Note: If you run this program when the system is heavily loaded, it may take half an hour to compute the answer! Try tracing helper procedures to make sure your program is on track, or start by computing `(next-perf 1)` and see if you get 6.]

3. Explain the effect of interchanging the order in which the base cases in the `cc` procedure on page 41 of Abelson and Sussman are checked. That is, describe completely the set of arguments for which the original `cc` procedure would return a different value or behave differently from a `cc` procedure coded as given below, and explain how the returned values would differ.

```
(define (cc amount kinds-of-coins)
  (cond
    ((or (< amount 0) (= kinds-of-coins 0)) 0)
    ((= amount 0) 1)
    (else ... ) ) )        ; as in the original version
```

4. Give an algebraic formula relating the values of the parameters `b, n, counter,` and `product` of the `expt` and `exp-iter` procedures given near the top of page 45 of Abelson and Sussman. (The kind of answer we're looking for is "the sum of b, n, and counter times product is always equal to 37.")

Continued on next page.

Week 3 continued...

Extra for experts:

1. The partitions of a positive integer are the different ways to break the integer into pieces. The number 5 has seven partitions:

```
5          (one piece)
4, 1          (two pieces)
3, 2          (two pieces)
3, 1, 1          (three pieces)
2, 2, 1          (three pieces)
2, 1, 1, 1          (four pieces)
1, 1, 1, 1, 1          (five pieces)
```

The order of the pieces doesn't matter, so the partition 2, 3 is the same as the partition 3, 2 and thus isn't counted twice. 0 has one partition.

Write a procedure `number-of-partitions` that computes the number of partitions of its nonnegative integer argument.

2. Compare the `number-of-partitions` procedure with the `count-change` procedure by completing the following statement:

`Counting partitions is like making change, where the coins are ...`

3. (Much harder!) Now write it to generate an iterative process; every recursive call must be a tail call.

Unix feature of the week: `mkdir`, `cd`, `pwd`, `ls`

Emacs feature of the week: `C-M-f`, `C-M-b`, `C-M-n`, `C-M-p` (move around Scheme code)