Homework 3: Tracing, Data Abstraction, and Numbers

Due:

For problems that ask you to write procedures, hand in a printout of your code, and either a transcript showing how you tested each procedure with results, or an English argument that says why your code is correct or both. For other problems, you may handwrite a solution if you wish.

Transcripts should not be edited; it is better if you find problems with your code that you don't have time to fix than if we find them. English arguments should not be trivial (don't say "it looks right" or "I tested it and it works"), and it should not merely say what happens; instead they should attempt to convince the reader that your program works correctly and can't be wrong on any input. See Homework 2 for examples. See the *Course Specification* handout for details of how we grade homeworks like this.

1. (6 points; basic) Do exercise 2.23, parts a and b. Make a transcript, and then underline on the transcript the output that is printed and draw a circle around each part that is the return value of the expression.

2. (6 points; extra credit) Do exercise 2.24. You can handwrite this, but we advise trying it out on the computer.

3. (8 points; basic) Do exercise 2.25, hand in a printout and a transcript.

4. (2 points; extra credit) Do exercise 2.26.

5. (4 points; basic) Do exercise 3.1.

6. (6 points; basic) Define `pairwise-product` from exercise 3.2. (You don't need to do `pairwise-sum`).

7. (10 points) Do exercise 3.3; note that for maximum points on this exercise you should define `dot-product` twice and write a short English comparison of the two versions that are in the book.

8. (6 points; basic) Do exercise 3.4.

9. (8 points; extra credit only) Sam Smalltalk has noted that previous problems do multiplication of both n-tuples and numbers, but that he has to remember what procedure to use (and the order of arguments for `mult-by-n`). Sam finds this too prolix and wants you to write a single procedure procedure `mult` that can multiply numbers and n-tuples so he doesn't have to call 3 different procedures. For example,

   ```
   (mult 3 '(1 2 3))  ==>  (3 6 9)
   (mult '(1 2 3) 3)  ==>  (3 6 9)
   (mult '(4 5 6) '(6 7 8))  ==>  (24 35 48)
   (mult 6 8)  ==>  48
   ```

10. (5 points; extra credit only) Eva Lou Ator points out that the `mult` procedure of the previous problem could have been called `*` instead of `mult`. How can this be done, without using any parts of Scheme that we haven't seen yet? (You should do your solution on the computer to be sure it works.)

11. (6 points) Do exercise 3.5.

12. (8 points; basic) Do exercise 3.8. To signal an error call the procedure `error` on page 199 of the text as follows.

    ```
    (error "length of " ls " is less than " num ".")
    ```

    Technically this does not return the message in the example as the result, but merely prints it; that is what is meant by "signalling an error" in this problem.

13. (10 points) Do exercise 3.12.

14. (5 points; basic) Do exercise 3.14. Low-level solutions will lose points.

15. (6 points; basic) Do exercise 3.15.

16. (8 points; basic) Do exercise 3.17.

17. (extra credit) The $3x + 1$ problem concerns iterates of the following mathematical function:

$$T(n) = \begin{cases} (3n+1)/2, & \text{if } n \equiv 1 \pmod 2, \\ n/2, & \text{if } n \equiv 0 \pmod 2. \end{cases}$$

which takes odd numbers n to $(3n + 1)/2$ and even numbers n to $n/2$.

(a) (3 points; extra credit) Implement the function that computes T. For example,

```
(T 33)  ==>  50
(T 4)  ==>  2
```

(b) (6 points; extra credit) The iterates of T are defined as follows. Let $T^{(0)}(n) = n$, and for all integers $k > 0$, let $T^{(k)}(n) = T(T^{(k-1)}(n))$.
Define a function `T-iterate` that takes natural numbers `k` and `n` such that `(T-iterate` k n`)` returns $T^{(k)}(n)$ for $k \geq 0$. For example,

```
(T-iterate 0 7)  ==>  7
(T-iterate 5 7)  ==>  20
```

(c) (6 points; extra credit) The sequence of iterates $(n, T(n), T^{(2)}(n), \ldots)$ is called the trajectory of n. For example, the trajectory of 7 is:

$$7, 11, 17, 26, 13, 20, 10, 5, 8, 4, 2, 1, 2, 1, 2, 1, \ldots$$

Define a function `trajectory-head` so that `(trajectory-head n k)` returns a list of the first `k`+1 elements (starting from 0) of the trajectory of `n`. For example,

```
(trajectory-head 7 3)  ==>  (7 11 17 26)
```

(This problem illustrates one technique for dealing with potentially infinite data in a program.)

(d) (7 points; extra credit) The $3x+1$ conjecture states that for each $n > 0$, there is some k such that $T^{(k)}(n) = 1$. Define a function `trajectory-to-1` such that `(trajectory-to-1 n)` returns the first subsequence of the trajectory of `n` that includes 1, provided that `n` is not a counter-example to the conjecture. For example,

```
(trajectory-to-1 1)  ==>  (1)
(trajectory-to-1 7)  ==>  (7 11 17 26 13 20 10 5 8 4 2 1)
```

(Try using `trajectory-to-1` on 27 and 703.)