Homework 8: Vectors

Due:

We continue suggesting some problems for additional practice. These have 0 points. You are not to hand these in to your TA. You should check your own work on them, and if you have questions, discuss them in recitation or with the staff, or with others in class.

1 Vectors (Section 9.2)

1. (0 points) Try the examples on pages 271 and 274 – 276 of the text. Load the file

   ```
   /home/cs227/lib/load-vectors.ss
   ```

   to get the code.

2. (2 points; extra credit only) Write a procedure `fussy-make-vector` that does the same thing as `make-vector`, but only works with two (2) arguments. For example,

   ```
   (view (fussy-make-vector 3 'x))
   ```

   prints `#(x x x)`, but `(fussy-make-vector 3)` is an error. That is, `(fussy-make-vector 3)` gives an error message. The exact error message is unspecified.

3. (3 points) Write a procedure `vector-stretch-fill` that is like `vector-stretch` (program 9.5), but takes an additional argument, `fill-value`. The `fill-value` is used as the initial value for elements of the result for which there is no corresponding element in the argument `vec`. For example, if `v` is `#(a b c)`, then `(view (vector-stretch-fill v 6 'new))` prints `#(a b c new new new)`.

4. (3 points; basic) Define a procedure `vec-` that takes two vectors as arguments and returns a vector. You may assume the arguments have the same length. If the argument vectors have values $\langle a_1, \ldots, a_n \rangle$ and $\langle b_1, \ldots, b_n \rangle$, then the result has value $\langle a_1 - b_1, \ldots, a_n - b_n \rangle$. For example,

   ```
   > (view (vec- (vector 1 2 3 4) (vector 0 1 2 3)))
   #(1 1 1 1)
   ```

5. (2 points; basic) The *weighted score* of a vector of data, $\langle d_1, \ldots, d_n \rangle$, with weights $\langle w_1, \ldots, w_n \rangle$ and possible scores $\langle p_1, \ldots, p_n \rangle$, is given by the formula

   $$\frac{w_1 \times d_1 + \ldots + w_n \times d_n}{w_1 \times p_1 + \ldots + w_n \times p_n}.$$

   Write a procedure `weighted-score` that takes a vector of numbers, `data`, and a vector of numbers between 0.0 and 1.0, `weights`, and a vector of numbers, `possible`, and returns the weighted score of the data. You can assume that all vectors have the same length, that not all of the weights are 0.0, and that the sum of the possible scores is greater than 0.0. For example, `(weighted-score (vector 90 100) (vector 0.3 0.7) (vector 100 100))` returns 0.97.

6. (4 points; extra credit only) Write a procedure interactive-weighted-score that takes a vector of numbers between 0.0 and 1.0, **weights**, and a vector of numbers, **possible**, and which: prompts for data scores, prints the weighted score of the data, then prompts, and prints in this way until "quit" is entered instead of a score, when it quits and prints "Bye." For example,

```
> (interactive-weighted-score (vector 0.3 0.7) (vector 100 100))
Data 1? 90
Data 2? 100
The weighted score of #(90 100) is: 0.97
Data 1? quit
Bye.
```

7. (3 points; extra credit only) Barbara Liskov thinks that the elements of a vector should be printed abstractly. She points out that if you print a vector of vectors using `view`, you'll see the internal representation printed. Solve her problem by writing a procedure `view-abs` that takes a procedure, `element-print`, and a vector, `vec`, as arguments, and prints `vec`, using `element-print` to print the elements. For example,

```
> (view-abs view (vector (vector 1 2) (vector 3 4)))
#(#(1 2)
#(3 4))
> (view-abs write (vector "printed" "with" "quotes"))
#("printed" "with" "quotes")
```

2 Representing Vectors (9.3)

8. (5 points) Do exercise 9.1.

9. (3 points) Do exercise 9.2.

10. (0 points; suggested practice only) Do exercise 9.3. You can ignore the last sentence.

11. (5 points; basic) Do exercise 9.4. You can handwrite your answer to the last question.

12. (0 points; suggested practice only) Do exercise 9.5.

13. (4 points) Do exercise 9.6. As an example,

```
> (view (vector-append (vector 1 2 3) (vector 4 5)))
#(1 2 3 4 5)
```

14. (5 points; extra credit only) The way we solved the Tower of Hanoi problem in Chapter 6, we only saw the moves, not what the moves did. Write a procedure `vector-tower-of-hanoi`, that is like `tower-of-hanoi`, except that instead of returning a list of moves, it returns a list of 3-element vectors. Each 3 element vector represents the state of the problem: how many disks are on the left, center, and right needles. For example, `#(3 2 1)` represents the state where there are three disks on the left needle, 2 on the middle, and one on the right. For example,

```
(vector-tower-of-hanoi 2) ==> (#(2 0 0) #(1 1 0) #(0 1 1) #(0 0 2))
```

15. (3 points; extra credit only) On the bottom of page 270, the text says that Program 9.3 is "very inefficient." Analyze the program and estimate its order (see p. 126). (Computer Scientists call this "time complexity.")

3 Matrices (9.4)

16. (3 points) Do exercise 9.8.

17. (4 points) Do exercise 9.9.

18. (0 points; suggested practice only) Do exercise 9.10.

19. (7 points) Do exercise 9.11. Ignore the directions in the last sentence.

20. (10 points) Do exercise 9.12.

21. (0 points; suggested practice only) Do exercise 9.13.

22. (6 points; extra credit only) Do exercise 9.14.

4 Chapter 10

We are not going to cover chapter 10 in class this semester. But there are many interesting problems in that chapter. These problems concern such basic computer algorithms as sorting and searching, and an introduction to relational databases (in section 10.4).

We will give you 2 extra credit points per problem you do correctly, for all problems in chapter 10. (Note that some may be much harder than others.) You should be sure that your solutions are exceptionally clear and clearly tested (to be easier on your TA).