CSCI 4150 Introduction to Artificial Intelligence, Fall 1999
Assignment 2: out Tuesday September 7, due Tuesday September 14

The required parts of this assignment are to be turned in electronically to the directory `/dept/cs/ai/submit2`. See the course home page for more details.

The only addition to the set of procedures you should know are `map` and `apply`. Again, you should not be using the "more advanced" features of Scheme, in particular `set!` and its variants and iterative control forms.

For the questions 4, 5, and 6, you will write some of the support code for the 8-puzzle which you will use in Assignment 3.

READING: Read Chapters 1 and 2 of Russell and Norvig. I consider sections 1.2 and 1.3 interesting but optional. Regarding Chapter 2 — we're not going overboard on agents, but you should read this to understand the framework of the rest of the text.

1. (optional) Write procedures using `map` and/or `apply` that do the following:
   (a) take the square root of every number in a list
   (b) adds a constant (passed as an argument) to every number in a list
   (c) given a list of points, returns the point closest to the origin. For example:
   ```
   (closest-point '((0 4) (3 2) (-1 1))) ==> (-1 1)
   ```
2. (10 points) Write a procedure `(mag x)` that returns the magnitude of a vector `x` represented as a list of numbers. The vector `x` may be of any dimension. For example:
   ```
   (mag '(3 4))            ==>   5
   (mag '(4 -2 8 7 11))    ==>  15.937377450509228
   ```
3. (15 points) Write the following procedures:
   (a) `(positions lst e)` that returns a list of numbers corresponding to the position of every occurrence of element `e` in the list `lst`. For example:
   ```
   (positions '(1 3 5 3 3 7 2) 3)  ==>  (1 3 4)
   ```
   The order of the elements in the returned list is not important.
   (b) Using the above `positions` function, write a nonrecursive function `(positions-list lst elements)` using `map` and/or `apply` which returns a single list of positions of every occurrence of an element of the list `elements` in the list `lst`. For example:
   ```
   (define test-list '(a b c e b c f k l m o c k f a c))
   (positions-list test-list '(b c k))  ==> (4 1 15 11 5 2 12 7)
   ```
   The order of the elements in the returned list is not important.
4. (10 points) Write a procedure `(ep-distance i j)` that computes the Manhattan distance from cell `i` to cell `j`. Recall that Manhattan distance is the rectilinear distance from one cell to another. Assume the cells are numbered as follows:

```
 0 | 1 | 2
---+---+---
 3 | 4 | 5
---+---+---
 6 | 7 | 8
```

For example, the Manhattan distance from cell 0 to 7 is 3 (i.e. 1 right and 2 down).

5. (10 points) Write a function `(swap lst i j)` that returns a new list based on `lst` except that elements `i` and `j` are swapped. Assume that the first element of a list is element 0. For example:

```
(swap '(1 2 3 4 5 6 7 8 space) 5 8) ==> (1 2 3 4 5 space 7 8 6)
```

6. (25 points) Write a procedure `(ep-children s)` that generates the successors to a state of the 8-puzzle problem.

   For example:

```
 1 | 2 | 3          1 | 2 | 3          1 | 2 | 3
---+---+---        ---+---+---        ---+---+---
 4 | 5 | 6   ==>    4 | 5 |      and   4 | 5 | 6
---+---+---        ---+---+---        ---+---+---
 7 | 8 |            7 | 8 | 6          7 |   | 8
```

```
(ep-children '(1 2 3 4 5 6 7 8 space))
  ==> ((1 2 3 4 5 space 7 8 6) (1 2 3 4 5 6 7 space 8))
```

   You may find your `swap` procedure from the previous problem useful here.

7. (5 points) This problem is for those of you who enjoy a little more challenging problem or just have some extra time on your hands. Do this problem last because it's harder than the rest and is only worth 5 points!

   Write a procedure `(permute x)` which takes a list `x` and produces a list of all the permutations of `x`. For example:

```
(permute '(a b c)) ==> ((a b c) (a c b) (b a c) (b c a) (c a b) (c b a))
```

   The order is not important so long as you generate all permutations.