More Recursion Problems

Here are a second set of recursion problems to do at your leisure. Once you solve a Question, you can use its results for the other Questions. Also, you’re free to use helper functions to help manage the complexity of a problem. Make sure to double check to see that your answers are right.

**Question 1 (Easy)** When we have a hand of cards, we often “rotate” the cards — we move the first card to the last, or vice versa. We can imagine a hypothetical **rotate-left-by-1** that would behave like this:

```lisp
> (rotate-left-by-1 '(h7 sq d4 c4 da))
(sq d4 c4 da h7)
```

```lisp
> (rotate-left-by-1 '(sq d4 c4 da h7))
(d4 c4 da h7 sq)
```

Write a **rotate-left-by-n** function that, given a hand of cards and a number \(n\), rotates our hand \(n\) times. For example:

```lisp
> (rotate-left-by-n '(h7 sq d4 c4 da) 2)
(d4 c4 da h7 sq)
> (rotate-left-by-n '(h7 sq d4 c4 da) 5)
(h7 sq d4 c4 da)
```

By the way, rotating the empty hand should return back an empty hand, since there’s nothing to rotate.

**Question 2 (Medium)** A word is considered **elfish** if it contains the letters: e, l, and f in it, in any order. For example, we would say that the following words are elfish: “whiteleaf”, “tasteful”, “unfriendly”, and “waffles”, because they each contain those letters.

- Write a predicate function called **elfish?** that, given a word, tells us if that word is elfish or not. The solution to this is not necessarily recursive — it’s a warmup.
- Write a more generalized predicate function called **x-ish?** that, given two words, returns true if all the letters of the first word are contained in the second. For example:

```lisp
> (x-ish? 'left 'rightfullness)
#t
> (x-ish? 'entwife 'waterproofing)
#t
> (x-ish? 'elf 'shelf)
#t
> (x-ish? 'left 'shelf)
#f
```

- Finally, write a function called **keep-leftish** that takes a sentence and returns all the leftish words in that sentence. For example:

```lisp
> (keep-leftish '(this stressful time on the twelfth felt strangely uneventful))
(stressful twelfth felt uneventful)
```

**Question 3 (Hard)** Now that we’ve played enough games with cards, we turn our attention to a more serious and enterprising business: piloting broken spacecraft.

Imagine a ship flying around in a 2d-grid playing field. We can pinpoint a ship by knowing its position — represented in \(x, y\) coordinates — and its direction — north, south, east, or west. Our derelict ship will only accept two commands: left or thrust, which will turn our spaceship left by 90 degrees or move it forward by one space, respectively.
Write a function called `make-it-so` that, given a ship’s initial position and direction, as well as a sequence of commands, gives back to us a sentence of the ship’s final position and direction after the ship executes our commands. For example:

> `(make-it-so ' (1 1 west) '())
(1 1 west)
> `(make-it-so ' (5 5 east) ' (thrust thrust))
(7 5 east)
> `(make-it-so ' (0 0 north) ' (thrust left left left thrust left thrust))
(1 2 north)
> `(make-it-so ' (-1 0 east) ' (thrust left thrust left thrust left thrust))
(-1 0 south)

You may find the following two definitions helpful:

```lisp
(define directions '(north west south east north)) ;; Having north twice is intentional
(define (direction-to-offsets d)
  (cond ((equal? d 'north) '(0 1))
        ((equal? d 'east) '(1 0))
        ((equal? d 'south) '(0 -1))
        ((equal? d 'west) '(-1 0))))
```

as well as the `member` function described in your book.