CS 296 — Deque Lab

1 Introduction and Learning Objectives

Your objective for this lab is simple: implement a doubly ended queue. A doubly ended queue is like a regular queue, but items can be added or removed from both ends. It is usually written *deque*, and pronounced like *deck*. Some people write it as *dequeue*, but that looks like the queue operation, so we mangle the word. We're in the engineering college; such things don't bother us.

So go to your repository and do a `git pull`. You will find the directory `deque-lab`, set up with the usual project files.

2 Given Code

2.1 The `deque` Namespace

At the beginning of `core.clj`, you will see these lines:

```
1 (ns deque.core)
2
3 (defrecord Deque [front back size])
```

We will implement this by creating a record of three fields: front, back, and size. They work just as they did in the functional queue presented in lecture.

3 Your Code

You will need to implement the following functions:

- `make-deque`
- `deque-size`
- `push-front`
- `push-back`
- `flip-front`
- `flip-back`
- `pop-front`
- `pop-back`
- `front`
- `back`

Note that the `push` and `pop` functions[1] return the resulting deques. If you want the element, use `front` and `back`. The `flip-front` function makes sure that the front list has data in it. If not, it reverses the back list and moves that to the front list. You saw this in lecture. The `flip-back` function does the reverse.

Also, I've included the `push-front` code, so you can see an example of idiomatic code. Notice how the `let` uses destructuring to make it easy to access the parts of the deque.

```
(defn push-front
  "Adds an element to the front of the deque."
  [dq elt]
  (let [{:keys [front back size]} dq]
    (Deque. (cons elt front) back (inc size))))
```

3.1 Test Cases

The file `test/deque/t_core.clj` contains testing code. The only test in it now is just an "autofail" test so it's clear who didn't even attempt the MP. Delete that one.

[1] By "push functions," I mean `push-front` and `push-back`, similarly for "pop functions." That may seem obvious to you, but if I don't put this here somebody will ask on Piazza.