Homework #1: Queue<Anon>

Issued: Wednesday, January 14
Due: Monday, January 26

Purpose

This assignment asks you to understand the design of, and complete the implementation of, a general-purpose list module. The data structure is actually a double-ended doubly-linked queue of pointers to arbitrary (aka, anonymous) data objects.

Your module will be useful in future assignments, perhaps as a shared library.

Skeletal Code

A partial implementation of the module is in:

```
pub/hw1
```

You'll notice that its makefile needs the makefile in its parent directory.

The module's interface is repeated, below.

Requirements

1. Implement the given interface: You cannot change it.
2. The interface, as given, is mostly syntactic. We will discuss its semantics in lecture. Implement what we discuss: An excuse of "I didn't want to do it that way" is unacceptable.

3. Implement a double-ended doubly-linked list. Pairwise, operations on opposite ends are symmetric. Factor-out commonality, and only implement it once.
4. Develop and demonstrate a thorough test suite.
5. Use `valgrind` to show that your module has no memory leaks.
6. Provide good documentation and error messages.
7. Your submission will be evaluated on `onyx`.

Submission

Homework is due at 11:59PM, Mountain Time, on the day it is due. Late work is not accepted. To submit your solution to an assignment, login to a lab computer, change to the directory containing the files you want to submit, and execute:

`submit jbuffenb` *class assignment*

For example:

```
submit jbuffenb cs452 hw1
```

The `submit` program has a nice `man` page.

When you submit a program, include: the source code, sample input data, and its corresponding results.

Scores are posted in our `pub/scores` directory, as they become available. You will receive a code, by email, indicating your row in the score sheet. You are encouraged to check your scores to ensure they are recorded properly. If you feel that a grading mistake has been made, contact me as soon as possible.

Interface

```
4  // put: append onto an end, len++
5  // get: return from an end, len--
6  // ith: return by 0-base index, len unchanged
7  // rem: return by == comparing, len-- (iff found)

8  typedef void *Deq;
9  typedef void *Data;

10 extern Deq deq_new();
11 extern int deq_len(Deq q);

12 extern void deq_head_put(Deq q, Data d);
13 extern Data deq_head_get(Deq q);
14 extern Data deq_head_ith(Deq q, int i);
15 extern Data deq_head_rem(Deq q, Data d);

16 extern void deq_tail_put(Deq q, Data d);
17 extern Data deq_tail_get(Deq q);
18 extern Data deq_tail_ith(Deq q, int i);
19 extern Data deq_tail_rem(Deq q, Data d);

20 typedef char *Str;
21 typedef void (*DeqMapF)(Data d);
22 typedef Str  (*DeqStrF)(Data d);

23 extern void deq_map(Deq q, DeqMapF f); // foreach
24 extern void deq_del(Deq q, DeqMapF f); // free
25 extern Str  deq_str(Deq q, DeqStrF f); // toString
```