CISC 3130
Exercise 2

Working with Array-Based Lists

In this exercise:

- you will work with the book's data structures implementations
- you will refresh (and perhaps deepen) your C++ object-oriented knowledge
- you will refresh your ability to write and use makefiles

Introduction

In this exercise, we'll work with the book's code for several of the data structures discussed in this chapter. If you haven't already, you can download the book's C++ code here. By the way—the book's C++ classes are, unfortunately, NOT good examples of well-written C++ code!

Measuring some (More) Performance

Using the book's `ArrayDeque` class, write some code—similar to code in Exercise 1—that creates an `ArrayDeque` of `int`, adds 5000 elements at the front, removes 2500 elements from the rear, adds 5000 elements at the rear, and removes 7500 elements from the front. Measure the time it takes to execute that set of operations.

Note that your code will depend on at least two classes—`ArrayDeque` as well as the `array` class that `ArrayDeque` depends on. It will save a lot of time to write a quick makefile. Remember to use a variable like

```
CXXFLAGS = -Wall -std=C++11
```

so that you can invoke `g++` as `g++ $(CXXFLAGS)`.

Improve the Implementation

Now, write a class `SmartArrayDeque` that is derived from `ArrayDeque`. This class should have the following characteristics:

- It should have convenience methods `addFront()`, `removeFront()`, `addRear()`, and `removeRear()`.
- Instead of copying elements with `for` loops, it should use `std::copy()` wherever possible.

Modify your main program from above to time the same operations using a `SmartArrayDeque` object. You'll very likely need to modify your makefile to accommodate your `SmartArrayDeque` definition.

Hopefully your improved implementation is actually faster. What happens if you double the number of elements your code works with? Do the time savings also double?