Section Handout 6

Based on handouts by Jerry Cain, Julie Zelenski, and Eric Roberts

Problem One: Double-Ended Queues

An important data structure that we do not provide as part of the CS106B collections classes is the *double-ended queue*, often shortened to *deque* (pronounced "deck," as in a deck of cards). A deque is similar to a queue, except that elements can be added or removed from both ends of the deque.

Here is one possible interface for a **Deque** class:

```
class Deque {
public:
    Deque();
    ~Deque();

    /* Adds a value to the front or the back of the deque. */
    void pushFront(int value);
    void pushBack(int value);

    /* Returns and removes the first or last element of the deque. */
    int popFront();
    int popBack();
};
```

One efficient way of implementing a deque is as a doubly-linked list. The deque stores pointers to the head and the tail of the list to support fast access to both ends.

Design the **private** section of the **Deque** class, then implement the above member functions using a doubly-linked list.

Problem Two: Quicksort Revisited

Recall that the quicksort algorithm for sorting a list of values works as follows. If the list of values has exactly 0 or 1 elements in it, it must be sorted. Otherwise:

- Choose a ***pivot element***, typically the first element of the list.
- Split the elements in the list into three groups – elements less than the pivot, elements equal to the pivot, and elements greater than the pivot.
- Recursively sort the first and last of these groups.
- Concatenate the three lists together.

Suppose that you have a linked list cell defined as

```
struct Cell {
    int value;
    Cell* next;
};
```

Write a function

```
void quicksort(Cell*& list);
```

That accepts as input a pointer to the first cell in a linked list, then uses merge sort to sort the linked list into ascending order. The function should change the pointer it receives as an argument so that it points to the first cell of the new linked list.