CS121 - Computer Science II
Lab Assignment #5
Fall 2011

The purpose of this exercise is to implement a queue class using an array-based circular queue.

The **Producer-Consumer** model is very common in computer applications. One program (or thread within a program) is the *producer* - it is responsible for creating objects (characters, lines of text, messages, etc.) and placing them in a queue. The *consumer* is a second program (or thread) which removes the objects from the queue and processes them. The advantage of such a scheme is that the objects can be produced faster than they can consumed, at least for a short time - the queue acts as a "buffer" which stores the unprocessed objects during bursts of production, and allows the consumer to "catch up" during periods of slow production.

A very common example of this process that you probably use every day is the shell's *ring buffer* for characters. The characters you type are placed in a circular queue (the "ring buffer") by that part of the shell that performs input (the producer), and are removed from the queue by the program that tokenizes the input (splits the input into groups of chars separated by white space) and interprets the results (the consumer). Most of the time, the consumer is able to keep up with the producer; however, if the shell is processing a command that takes a long time (such as a long directory listing or compile), the user can type chars that are placed in the queue, which are not processed until the shell finishes the previous command.

Two problems might occur with the producer consumer model. It is possible for the queue to fill up if the producer creates objects faster than they can be consumed for an extended period of time - hopefully the queue is sized such that this situation rarely occurs, as there is often nothing that can be done except to ignore the items produced when the queue is full; some information might be lost. The second problem is more common - the consumer tries to take items from an empty queue. This situation is less traumatic - the consumer usually just waits for more items to be produced.

For this assignment, you are to implement a circular queue class, and demonstrate its operation by writing producer and consumer functions. (Normally such a system would involve writing separate programs, or separate threads, for the producer and consumer. Since this is beyond the scope of the course, we will write two functions instead.) The items to be queued are `chars`.

Input should come from standard input, and will consist of the following commands:

- `P chars` - Produce the following characters. That is, the producer should place the characters into the queue.
- `C n` - Consume the number of characters specified. The consumer should remove the specified number of characters from the queue and output them.
- `D` - display the queue contents
- `N` - Print out the number of items currently in the queue
- `Q` - quit

If there is no room in the queue for all the characters being produced, the producer should enqueue the characters that will fit, ignore the remaining, and print a message on the screen that some of the characters were lost. If the consumer is asked to dequeue more characters than are in the buffer, it should dequeue all the characters in the queue and print them. The next time a `C` command is issued, first the number of characters that were requested in the previous `C` commands that could not be fulfilled should be dequeued and printed, followed by those in the new request.

Normally, we want the queue to be large enough so that the probability of queue overflow is very small. However, for testing purposes a small queue is desirable, so the proper full-queue behavior can be tested.

For this exercise, you should create a class for your queue implementation. The class should include a constructor that accepts the size of queue to be created; your `main` program should input the size of queue to be created, then use the constructor to create the queue. All queue operations must be performed with `public` methods - the queue data itself should be `private`. The queue must be implemented as an array.

An example session is shown below. Note that the queue size has been set to 10 in this example (comments to the right are not part of the input):

```
$> ./a.out
Enter size of queue: 10

P testing                                  // put chars into queue
N                                          // inquire how many chars in in queue
There are 7 items in the queue
C 4                                        // consume 4 of the chars in queue
test
D                                          // display current contents of queue
ing
C 3
ing
P long message                             // attempt to add too many chars
Error-queue full, some chars will be lost
C 10
long messa
P new msg
C 10                                       // attempt to consume more chars than in queue
new msg
P test
P 123
N                                          // check to see how many chars are in the queue
7
C 3
tes                                        // remaining chars from last request
t12                                        // chars from this request
Q
```