Instructions

As a pre-interview, we're asking all development candidates to send in a solution to a programming problem. Details on the programming problem are given below. We apologize for the imposition—if you've ever tried to fill a programming position, you know that resumes are not an ideal way to evaluate programming ability.

What to send us:

- An up-to-date copy of your resume.
- A short discussion (a paragraph or two) about the possible solutions you see to the problem, and why you chose the approach you did.
- Implementation of your solution in C/C++, preferably in a single source file.

We're not expecting you to spend more than a couple of hours on your solution. The interpretation of the Problem Statement below is part of the evaluation, and the recruiting team will not answer requests for details or clarifications. Please do your best with the information we have provided.

Problem Statement

The problem is to write a set of functions to manage a variable number of byte queues, each with variable length, in a small, fixed amount of memory.

You should provide implementations of the following four functions:

```
// Creates a FIFO byte queue, returning a handle to it.
Q * create_queue();

// Destroy an earlier created byte queue.
void destroy_queue(Q * q);

// Adds a new byte to a queue.
void enqueue_byte(Q * q, unsigned char b);

// Pops the next byte off the FIFO queue
unsigned char dequeue_byte(Q * q);
```

So, the output from the following set of calls:

```
Q * q0 = create_queue();
enqueue_byte(q0, 0);
enqueue_byte(q0, 1);
Q * q1 = create_queue();
enqueue_byte(q1, 3);
enqueue_byte(q0, 2);
enqueue_byte(q1, 4);
printf("%d", dequeue_byte(q0));
printf("%d\n", dequeue_byte(q0));
enqueue_byte(q0, 5);
enqueue_byte(q1, 6);
```

```
  printf("%d", dequeue_byte(q0));
  printf("%d\n", dequeue_byte(q0));
  destroy_queue(q0);
  printf("%d", dequeue_byte(q1));
  printf("%d", dequeue_byte(q1));
  printf("%d\n", dequeue_byte(q1));
  destroy_queue(q1);
```

should be:

```
  0 1
  2 5
  3 4 6
```

You can define the type Q to be whatever you want.

Your code is not allowed to call malloc(), new(), or other heap management routines. Instead, all storage (other than local variables in your functions) must be within a provided array:

```
  unsigned char data[2048];
```

Memory efficiency is important. On average while your system is running, there will be about 15 queues with an average of 80 or so bytes in each queue. Your functions may be asked to create a larger number of queues with less bytes in each. Your functions may be asked to create a smaller number of queues with more bytes in each.

Execution speed is important. Worst-case performance when adding and removing bytes is more important than average-case performance.

If you are unable to satisfy a request due to lack of memory, your code should call a provided failure function, which will not return (it can be just a stub):

```
  void on_out_of_memory();
```

If the caller makes an illegal request, like attempting to dequeue a byte from an empty queue, your code should call a provided failure function, which will not return:

```
  void on_illegal_operation();
```

There may be spikes in the number of queues allocated, or in the size of an individual queue. Your code should not assume a maximum number of bytes in a queue (other than that imposed by the total amount of memory available, of course!) You can assume that no more than 64 queues will be created at once.