Circular Array + Queue

Circular Array

A circular array is just an array used in a circular way: when you get to the end go back to the beginning. They are great for small queues where data might arrive in bursts or processing might have occasional pauses.

Modulo operator

Two parameters.

- `a` : This refers to the dividend value.
- `b` : This refers to the divisor value.

Basic concept of circular array

```
arr[i++ % arr.length] = foo;
```

`%` vs. `Math.floorMod()`

- `11 % 10 = -1 or 1`
- `Math.floorMod()` guarantees positive result but `%` does not.
  - more specifically, result has the same sign as the divisor `b`
- why is it important to return positive modulo?

Queue

- FIFO (*First-In-First-Out*)
- `void enqueue(T item);`
- `T dequeue();`

Circular arrays in context of Queue

```
[ ][ ][ ][ ][ ]  initial         front: 0, back: 0
[a][ ][ ][ ][ ]  enqueue(a)      front: 0, back: 1
[a][b][ ][ ][ ]  enqueue(b)      front: 0, back: 2
[a][b][c][ ][ ]  enqueue(c)      front: 0, back: 3
[a][b][c][d][ ]  enqueue(d)      front: 0, back: 4
[a][b][c][d][e]  enqueue(e)      front: 0, back: 0
[ ][b][c][d][e]  dequeue()       front: 1, back: 0
[ ][ ][c][d][e]  dequeue()       front: 2, back: 0
[ ][ ][ ][d][e]  dequeue()       front: 3, back: 0
[ ][ ][ ][ ][e]  dequeue()       front: 4, back: 0
[ ][ ][ ][ ][ ]  dequeue()       front: 4, back: 0
```

What's the formula?

```
// context:
//  front     : int  -> front index of circular array
//  manyItems : item -> number of items in the array
//  data      : T[]  -> array of type T

int getBackIndex() {
  return Math.floorMod(..., ...);
}

void enqueue(T item) {
  data[getBackIndex()] = item;
  manyItems++;
}
```

What's the formula? (cont.)

```
// context:
//  front     : int  -> front index of circular array
//  manyItems : item -> number of items in the array
//  data      : T[]  -> array of type T

int getBackIndex() {
  return Math.floorMod(front + manyItems, data.length);
}

void enqueue(T item) {
  data[getBackIndex()] = item;
  manyItems++;
}
```

Exercise

```
int nextIndex(i) {
  // TODO:
  // return ...;
}

T dequeue() {
  // TODO:
  // T result = ...;
  // front = ....;
  manyItems--;
  return result;
}
```

Exercise (cont.)

```
int nextIndex(i) {
  return Math.floorMod(i++, data.length);
}

T dequeue() {
  T result = data[front];
  front = nextIndex(front);
  manyItems--;
  return result;
}
```

Concerning homework #7

- We changed front and back
- how would `getBackIndex`, `enqueue`, `getNextIndex` and `dequeue` would change?

Concerning homework #7 (cont.)

```
int getBackIndex() {
  return Math.floorMod(front - manyItems, data.length);
}

int nextIndex(i) {
  return Math.floorMod(i--, data.length);
}

// enqueue and dequeue don't get affected
```

Lab assignment #7: