Worksheet 3

Problem Solving for Computer Science

In this labscript, the basic objectives are:

1. To use the class `Queue`, which implements the queue abstract data structure
2. To complete functions that can cyclically permute the values in an array using a queue
3. To apply this cyclic permutation technique to implement a simple encryption scheme: the Caesar Cipher

The second objective will also be important for your Sudoku assignment.

Setting up

Go to Worksheet 3 resources and download `lab3.zip`. As with previous worksheets, we will only need to work with `lab3.js` inside the folder `lab3`.

1. Open `lab3.js` in your chosen IDE
2. Open your command line interface and change your directory to `lab3` by using `cd` followed by a space and then dragging and dropping the folder to the CLI

We will be using exactly the same methods for testing code as in previous labs. That is, first printing things to the console and then running `npm test`.

The Queue Constructor

In the file `lab3.js`, you will see the following constructor:

```
function Queue() {
  this.arr = [];
  this.head = function() {
    return this.arr[0];
  };
  this.dequeue = function() {
    if (this.arr.length == 0) {
      return "Queue underflow!";
    } else {
      return this.arr.shift();
    }
  };
  this.enqueue = function(o) {
    this.arr.push(o);
  };
  this.isEmpty = function() {
      return this.arr.length == 0;
  };
}
```

Every time this constructor is called using `new Queue()`, a queue object will be created, which will at first be empty. Then we can use the method `enqueue` to add elements to the queue. Consider the following code:

```
var queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
```

This will enqueue the numbers 1 and 2 to the queue, with the number 1 being stored in the head, and 2 being stored in the tail.

Visualising a queue

At the moment, the queue object is a bit difficult to visualise. To resolve this, there is a function called `visQueue` in `lab3.js` that can be used to visualise a queue. The best way to see this is trying it yourself with the following code:

```
var queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(3);
console.log(visQueue(queue));
```

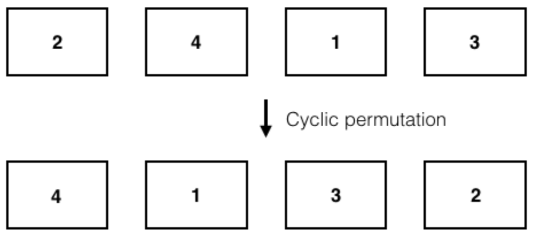
You should see the following printed in the terminal:

```
head --> 1 2 3 <-- tail
```

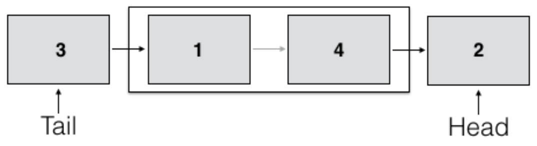
Here we have the head and the tail of the queue being pointed out to us. Hopefully this will be useful to you.

Using a queue for cyclic permutations

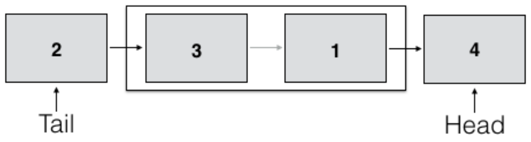
Queues are a useful tool for solving problems. One particular problem is that of cyclically permuting values in an array: a cyclic permutation of an array by one element to the left shifts all values in an array one place to the left with the value at the first element becoming the value at the rightmost element. The following picture depicts this:



A queue can be used to cyclically permute these values. First, the values in the array are enqueued one at a time to a queue from left to right to give the following:



Then the value at the head is enqueued to the queue, with the queue then dequeued. After this process, the queue should look like this:



Then we push the value at the head to a new array, dequeue, push the next value, and so on, until all values are stored in an array. We will implement this process in the next three tasks.

Task One: Enqueueing values from an array to a queue

Complete the function called `arrToQueue`, which takes an array as its argument and returns a queue. You should traverse the array from left to right, enqueuing the values stored in the array. You should be using the `enqueue()` method for the `queue` object.

To test your function use the following code:

```
var arr = [2, 4, 1, 3];
var queue = arrToQueue(arr);
console.log(visQueue(queue));
```

You should see the following printed in the terminal:

```
head --> 2 4 1 3 <-- tail
```

Task Two: Cyclically permuting elements in a queue

In this task you should complete the function called `permuteQueue`, which takes a queue and a number `k` as arguments. The function should return the argument `queue` but with the elements cyclically permuted `k` places to the left.

To do this, you need to repeat the following process `k` times: enqueue the value stored at the head of the queue, and then dequeue the queue. If `k = 0` you could just return the queue without any cyclic permutations.

To test your function use the following code:

```
var arr = [2, 4, 1, 3];
var queue = arrToQueue(arr);
console.log(visQueue(permuteQueue(queue, 1)));
```

You should see the following printed in the terminal:

```
head --> 4 1 3 2 <-- tail
```

Task Three: Pushing values from a queue to an array

In this task you need to complete the function `queueToArr`, which takes a queue as an argument. This function should return an array storing the values of the queue. In particular, it should push the value stored at the head to an array, dequeue the queue, and push the next value, and so on, until the queue is empty. Try using a while loop. That is, while the queue is not empty push the value at the head to an array.

To test your function use the following code:

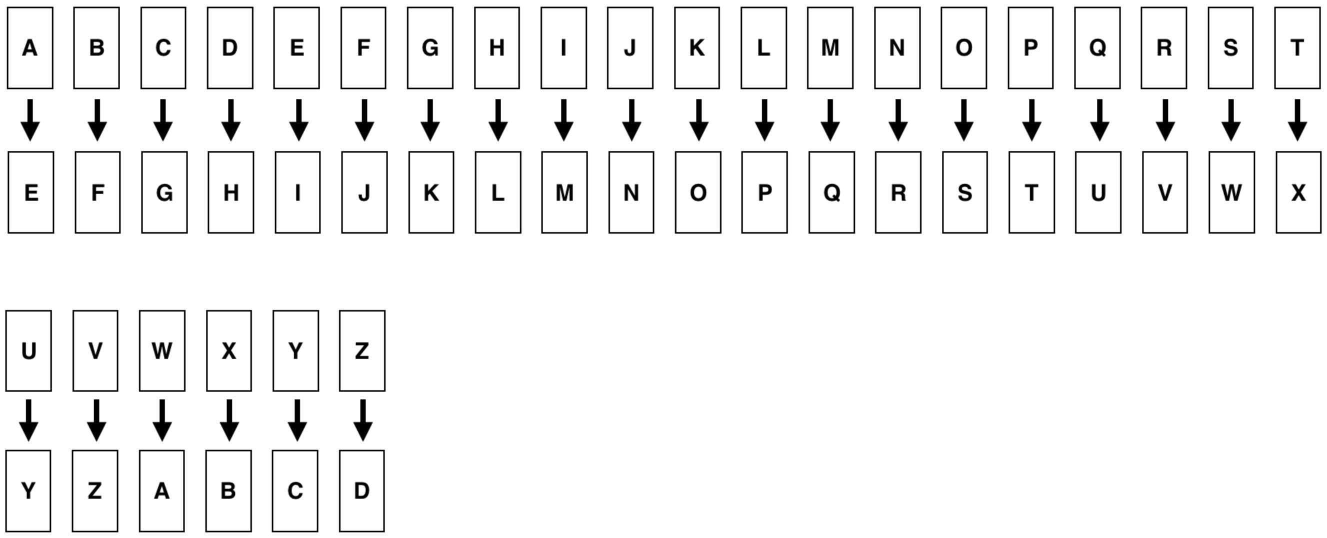
```
var arr = [2, 4, 1, 3];
var queue = arrToQueue(arr);
var arr1 = queueToArr(queue);
console.log(arr1);
```

You should see the following printed in the terminal:

```
[2, 4, 1, 3]
```

Background for the Advanced Task: The Caesar Cipher

A large part of cybersecurity is the about the encryption of data: converting data into something that looks like nonsense to someone who isn't a designated receiver. One of the oldest encryption schemes is the *Caesar Cipher*, named after Julius Caesar. The idea of this scheme is to take the alphabet and cyclically permute the letters by a certain number of elements. For example, here is a cyclic permutation of the alphabet by 4 letters to the left.


A B C D E F G H I J K L M N O P Q R S T
E F G H I J K L M N O P Q R S T U V W X
U V W X Y Z
Y Z A B C D

For example, if Caesar wanted to send the message "`hello`", it would be encrypted into "`lipps`".

Advanced Task: Encrypting messages

In `lab3.js` you will see a function called `encrypt` which takes a string called `letter` and a number `k` as arguments. It uses your completed functions from task one to three to encrypt a single letter into another letter. The encryption is done by the Caesar Cipher permuting the letters in the alphabet `k` spaces to the left.

In this task you need to complete the function `encryptMessage`, which takes as argument a string `message` and a number `k`. In the function `encryptMessage` you should loop over the characters in `message`, and apply `encrypt` to each character and add (concatenate) it to `string`. The argument `k` in `encryptMessage` should also be the argument `k` to `encrypt`.

HINT: use `charAt(i)` to find the character at index `i` in a string.

To test your function use the following code:

```
var message = "hello"
console.log(encryptMessage(message, 4));
```

You should see the following printed in the terminal:

```
"lipps"
```