<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>CMPS 260: Module 2 Programming Assignment</title>

<style>\* { font-family: monospace; }</style>

<script>

// NOTE: You must implement the data structures using the no prototype approach.

// This is what the book uses, so you can copy it.

// See also: https://it.pointpark.edu/tutorials/no-prototype-vs-prototype/

// NOTE: Please review the following links regularly:

// https://it.pointpark.edu/tutorials/arrays-vs-objects/

// https://it.pointpark.edu/tutorials/no-prototype-vs-prototype/

// https://it.pointpark.edu/tutorials/implementation-vs-interface/

//new comment

//--------------------------//

// The stack data structure //

//--------------------------//

console.log("The stack data structure");

// 1. Implement the stack data structure described in the book. Instead of

// using 'let items = [];' use 'var items = []'. Note that this

// implementation does not use the prototype (see project).

function Stack() {

var items = [];

this.push(element) {

this.items.push(element);

}

this.pop() {

return this.items.pop();

}

}

// 2. Write a simple test program that shows your stack works.

var stack = new Stack();

stack.push(5);

stack.push(6);

stack.push(10);

var stackItem = stack.pop();

console.log(stackItem);

//----------------------------------//

// ECMAScript 6 and the Stack class //

//----------------------------------//

console.log("ECMAScript 6 and the Stack class");

// Skip.

//-------------------------------//

// Solving problems using stacks //

//-------------------------------//

console.log("Solving problems using stacks");

// 1. Use the prompt to ask for a decimal number. Then write a loop that divides

// the number by two in every iteration and prints the remainder (0 or 1).

// For example, if the number is 5 the remainder is 1 and the number in the

// next iteration should be 2 (5/2 rounded down).

var num = prompt("Please enter a decimal number such as 10.4.");

var result;

while (count > 2) {

num

}

// 2. The algorithm in (1) can be used to convert a decimal number to a binary

// number but there is one issue. What is the problem?

// 3. Solve the problem in (2) using a stack.

//--------------------------//

// The queue data structure //

//--------------------------//

console.log("The queue data structure");

// 1. Describe the difference between a stack and a queue. Give one example

// where a stack is appropriate and one example where a queue is

// appropriate (that has not yet been discussed in class).

//------------------//

// Creating a queue //

//------------------//

console.log("Creating a queue");

// 1. Implement the queue data structure described in the book. Instead of

// using 'let items = [];' use 'var items = []'. Note that this

// implementation does not use the prototype (see project).

function Queue() {

// properties and methods go here

}

// 2. Write a simple test program that shows your queue works.

//--------------------//

// The priority queue //

//--------------------//

console.log("The priority queue");

// The following class is used below.

function QueueElement(element, priority) {

this.element = element;

this.priority = priority;

}

// 1. Finish the implementation below for the priority queue. Note that this

// implementation does not use the prototype.

function PriorityQueue() {

var items = [];

this.enqueue = function(element, priority) {

// we need to create a queue element that stores the priority as well

var queueElement = new QueueElement(element, priority);

// insert into priority queue (difficult part)

// note that this should be a min priority queue, meaning a lower value

// gets priority over a higher value

// ...

}

// this is a helper function that is useful for printing

this.print = function() {

for (var i = 0; i < items.length; i++) {

console.log(`${items[i].element} - ${items[i].priority}`);

}

};

// add the other methods as well

// ...

}

// 2. Write a simple test that makes sure the priority queue works as expected.

//---------------------------------//

// The circular queue - Hot Potato //

//---------------------------------//

console.log("The circular queue - Hot Potato");

// 1. Finish the implementation for the hot potato game. nameList contains the

// names of the participants, and num is how many times the potato is passed

// before a participant is ejected from the game (the one that holds the

// potato). Note that this is effectively a simulation of the game.

function hotPotato(nameList, num) {

// create a queue to store the names

// ...

// add the names to the queue

// ...

// while there is more that one name in the queue

// dequeue and enqueue a name num times

// ...

// then dequeue one more time, which is the participant that is ejected

// ...

// /while (end of while loop)

// the person that remains in the queue wins

// ...

}

// 2. Test the code in (1).

</script>

</head>

<body>

See console!

</body>

</html>