EECS 1012: LAB 09 – Code Breaker—Part 2: server-side (Nov 16–18, 2020)

A. IMPORTANT REMINDERS

1) Note that the deadline of remaining labs is on Wednesday night.
2) Each lab including the pre-lab mini quiz is about 2.0 % of your overall grade.
3) You must attend your own lab session (the one you are enrolled in). If you need to change your lab enrollment, you should go to the department. Instructors or TAs cannot change your enrollment. TAs are available via Zoom to help you. The attendance is optional, but is highly recommended. You can also have your work verified and graded during the lab sessions. Feel free to signal a TA for help if you stuck on any of the steps below. Yet, note that TAs would need to help other students too. In case you run out of time, the submission you make over eClass will be marked by the TAs after the lab ends (possibly not by the same TAs who assisted you during the lab session).
4) You can submit your lab work anytime before the deadline. We do not accept late submissions.
5) You must complete the pre-lab quiz posted on eClass no later than the first 15 minutes of your lab time.

B. IMPORTANT PRE-LAB WORKS YOU NEED TO DO BEFORE GOING TO THE LAB

1) We highly encourage you to develop alternative solutions to each of your 40+ problems in your Learning Kit in your spare time. In what language? Perhaps Java, as this is going to prepare you for your next course (EECS1022) a lot better and basically give you a huge advantage.
2) Download this lab's files and review them completely.
3) You should have a good understanding of JSON
   - If you still are not comfortable with JSON, we highly encourage you to revisit it here https://www.w3schools.com/js/js_json_intro.asp , and make sure you have a clear idea about what `stringify` and `parse` methods are for (https://www.w3schools.com/js/js_json_stringify.asp and https://www.w3schools.com/js/js_json_parse.asp).
4) Asynchronous JavaScript and XML (AJAX) is an important concept—in JavaScript—for sending requests to servers asynchronously. AJAX syntax is well simplified in jQuery's *get* and *post* methods. You may want to revisit these methods before doing this lab.
5) On the server's side, we use *node.js* (a JavaScript server environment, https://www.w3schools.com/nodejs/) together with one of its popular web frameworks, called *Express*. What are in lecture notes are sufficient for starting point of *Express*. But, if you want to learn more, visit here https://expressjs.com/en/5x/api.html#app and here https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction .
6) You need to install `Node.js` and `Express.js` on your computer before doing this lab. Instructions are in Steps 1 and 2 of `readme.txt`.

C. GOALS/OUTCOMES FOR LAB

1) To practise more computational thinking in a different programming language (preferably Java)
2) To become familiar with Node.js and Express.js
3) To complete the development of a fully-fledged web application

D. TASKS

1) TASK 1: Server-Side of the Code Breaker Game.

E. SUBMISSIONS

1) Manual verification by a TA (optional)

You may have one of the TAs verify your lab before submission. The TA will look at your various files in their progression. The TA may also ask you to make minor modifications to the lab to demonstrate your knowledge of the materials. The TA can then record your grade in the system.

2) eClass submission

Create a **folder** named "**Lab09**" and copy **all** your HTML and JS files; Once you are done, compress the folder and upload the zip (or tar) file to eClass.

F. FURTHER DETAILS

Task 1: In this task, you complete the **server-side**. What do you need to do in this task? The following 4 steps:

i) Revisit details of the requests that are sent from the client-side: one is in `initGameBoard()` and the other is in `processAttempt()` function. Open `code_breaker_client.js`, and make sure you understand lines 148-151 as well as lines 206-214.

ii) Also, revisit Slides 10-4 and 10-6. Note that when the request is responded by the server, a callback function is called automatically. We named the callback function *`response`*. The details of the response function were re-illustrated in Slides 10-8 to 10-10 and can be found in the `code_breaker_client.js` we already provided you with.

iii) Copy `code_breaker_serverV0.js` to a new file named `code_breaker_server.js` and follow the comments to complete the code. In particular, there are ten /*TODO…*/ comments that you should replace with your code. More detailed instructions can be found in the file. Here, we will provide you with some more hints on each of these concepts:

`/*TODO 1 …`
Note that the `req.query['data']` will return the part of the request that is specified by `data`, which is in JSON format. If we assign that to a variable name `z`, we can access components of that with `z['name']`, `z['action']`, `z['attempt_code']`, and `z['current_attempt_id']`. This is valid because in JavaScript, we can treat an object as an associative array.

`/*TODO 2 …`
Once we have prepared a response in a string format, we can send the response back to the requester using method `send()` where we pass the string as a parameter to this method.

`/*TODO 3-8 …`
One of best hints on how to do these 6 TODOs is to see the single Slide 10-9, where the associative array response[] has been used in the client side to figure out what data the server has sent back in its response. Just recall that objects in JavaScript could be defined as:

```
{
   property1 : value1,
   property2 : value2,
   property3 : value3,
   …
}
```

`/*TODO 9 …`
If you look at the Port # that requests are sent to from the client side in the `code_breaker_client.js` file, this becomes trivial. Hence, just see line 12 of the client side.

`/*TODO 10 …`
Here, you just want to declare an empty array. The array will be populated by 5 unique id of marbles in the following lines.

iv) Finally, follow the instructions in `readme.txt` to run the server locally on your machine and play the game on your own device at any time.

Show your code-breaker code to your TA. (optional)

G. AFTER-LAB TASKS (THIS PART WILL NOT BE GRADED)

In order to review what you have learned in this lab as well as expanding your skills further, we recommend the following questions and extra practice:

1) This `CodeBreaker` project is a great source of learning. Read all the comments carefully and make sure you have a clear understanding of the code—line by line. Many of lines of this code are just advanced programming some of which somewhat above the expectations of a typical CS1. Yet, we are sure many of you have a great thirst to learn more. So, this code should serve you for that purpose.
2) An interesting way to enhance this project is as follows. Add a button “Hint”, such that every time the user clicks on it, the server reveals one of the five pegs, in any order you wish. The server should not reveal more than 3 pegs. In order to implement this component, you need to add a new action to your requests. So far, you had two actions: “generateCode” and “evaluate”. Let’s call the third one, “hintMe”.
3) Make sure you understand objects in JavaScript can be treated as *associative* arrays and vice versa. See the following example:

```
var student = { firstName : "Anna",
                lastName : "Smith",
                major : "CS",
                age : 23 };

var x = student.lastName;   // is the same as var x = student["lastName"];
```

   See this page for further information on arrays and objects: https://www.w3schools.com/js/js_arrays.asp

4) In order to go beyond EECS1012 in the sense of getting the the confidence that, once you learn computational thinking, the syntax of a language is easy to pick, we highly encourage you to provide an alternative solutions to all problems that you addressed in your Learning Kit project. You may want to do that in Java because not only Java is still a very popular language, it helps you be very well prepared for your next course, 1022. To help you with this transition, two examples have provided to you on the following page.

Please feel free to discuss any of these questions in the course forum or see the TAs and/or Instructors for help.

Example 1: devise a program to receive two integer numbers from user and output their division.
Solution in JavaScript. Function `divide2numbers()` is invoked when an event occurs, e.g. a button is clicked.

```
function divide2Numbers() {
    /* pre-conditions: first, second in Z, and second<>0 */
    /* post-conditions: first/second is outputted */

    /* notify the user to enter a number */
    /* this function inputs anything user enters, the input is assigned to first as a string */
    var first = prompt("Enter an integer number: ");

    /* this function converts first to Integer and assigns it to first again*/
    first = parseInt(first);  // in JavaScript the datatype can change on the fly

    /* notify the user to enter another number */
    /* this function inputs anything user enters, the input is assigned to second as a string */
    var second = prompt("Enter another integer number: ");

    /* this function converts second to Integer and assigns it to second again*/
    second = parseInt(second);  // in JavaScript the datatype can change on the fly

    /* verify whether second is zero or not */
    if (second != 0) {
        /* calculate the division of first over second and assign it to result */
        var result = first / second;

        /* output the result to the user */
        alert("The division result is: " + result);
    }
    else alert("cannot divide!");
}
```

Solution in Java: you can type and run this program in any IDE. We recommend you to install/use `Eclipse`. Another good choice out there is `IntelliJ`.

```
import java.util.Scanner;

public class divide2Numbers {
    public static void main(String[] args) {
        /* pre-conditions: first, second in Z, and second<>0 */
        /* post-conditions: first/second is outputted */

        /* prepare to read something from standard input */
        Scanner scan = new Scanner(System.in);

        /* notify the user to enter a number */
        System.out.print("Enter an integer number: ");

        /* this method inputs an integer, the input is assigned to first */
        int first = scan.nextInt();

        /* notify the user to enter another number */
        System.out.print("Enter another integer number: ");

        /* this method inputs an integer, the input is assigned to second */
        int second = scan.nextInt();

        /* closing the Scanner, now that we no longer need it */
        scan.close();

        /* verify whether second is zero or not */
        if (second != 0) {
            /* calculate the division of first over second and assign it to result */
            double result = 1.0 * first / second;  // by the trick of multiplying first by 1.0, we convert that to real

            /* output the result to the user */
            System.out.println("The division result is: " + result);
        }
        else System.out.println("cannot divide!");
    }
}
```

Example 2: devise a program to receive five integer numbers, store them in an array, and then determine how many of them are prime and also have a digit 7.

Solution in JavaScript. Function `howmanyPrimeAndHas7()` is invoked when an event occurs, e.g. a button is clicked.

```
function howmanyPrimeAndHas7() {
    /* pre-conditions: a[i] in Z, i=0..4 */
    /* post-conditions: the number of prime numbers in a[i] that have a digit 7 is outputted */

    var a = []; // in JavaScript, the size of an array can change at runtime
    alert("In this program, you should enter 5 numbers first!")
    for (var i = 0; i < 5; i++) {
        a[i] = prompt("enter number " + (i + 1) + " : ");
        a[i] = parseInt(a[i]); // in JavaScript the datatype can change at runtime
    }

    var counter = 0;
    for (var i = 0; i < 5; i++) {
        if (isPrime(a[i]) && has7(a[i])) {
            counter++;
        }
    }

    alert("The count of numbers that are both prime and have a digit 7 is " + counter);
}

function isPrime(x) {
    /* pre-conditions: x in N */
    /* post-conditions: return true if x is prime, otherwise return false */

    var flag = true;
    var divisor = 2;
    while (flag && divisor <= Math.sqrt(x))
        if (x % divisor != 0) divisor++;
        else flag = false;
    if (x >= 2) return flag;
    else return false;
}

function has7(y) {
    /* pre-conditions: y in N */
    /* post-conditions: return true if y has a digit 7, otherwise return false */

    var flag = false;
    while (y > 0 && !flag) {
        if (y % 10 == 7) flag = true;
        y /= 10;
    }
    return flag;
}
```

Solution in Java: you can type and run this program in any IDE. We recommend you to install/use `Eclipse`. Another good choice out there is `IntelliJ`.

```java
import java.util.Scanner;

public class howmanyPrimeAndHas7 {
    public static void main(String[] args) {

        /* pre-conditions: a[i] in Z, i=0..4 */
        /* post-conditions: the number of prime numbers in a[i] that have a digit 7 is outputted */

        /* prepare to read something from standard input */
        Scanner scan = new Scanner(System.in);

        int[] a = new int[5]; // in Java, we should define the the array size at compile time
        System.out.println("In this program, you should enter 5 numbers first!");
        for (var i = 0; i < 5; i++) {
            System.out.print("enter number " + (i + 1) + " : ");

            /* this method inputs an integer, the input is assigned to first */
            a[i] = scan.nextInt();
        }

        int counter = 0;
        for (int i = 0; i < 5; i++) {
            if (isPrime(a[i]) && has7(a[i])) {
                counter++;
            }
        }

        System.out.println("The count of numbers that are both prime and have a digit 7 is " + counter);
    }

    public static boolean isPrime(int x) {
        /* pre-conditions: x in N */
        /* post-conditions: return true if x is prime, otherwise return false */

        var flag = true;
        var divisor = 2;
        while (flag && divisor <= Math.sqrt(x))
            if (x % divisor != 0) divisor++;
            else flag = false;
        if (x >= 2) return flag;
        else return false;
    }

    public static boolean has7(int y) {
      /* pre-conditions: y in N */
      /* post-conditions: return true if y has a digit 7, otherwise return false */

      boolean flag = false;
        while (y > 0 && !flag) {
            if (y % 10 == 7) flag = true;
            y /= 10;
        }
        return flag;
    }

}
```