Eric Roberts and Jerry Cain
CS 106J

Handout #43S
May 24, 2017

Solution to Section #7

Portions of this handout by Eric Roberts

1. Maps

```
import "file";

/*
 * This main function makes a roadmap given a roads.txt file.
 * It then checks whether pairs of cities are reached in two steps
 * to test the implementation of isReachableInTwo.
 */
function main() {
    const filename = "roads.txt";
    var roadmap = {};
    populateRoadmap(roadmap, filename);
    if (isReachableInTwo("Los Angeles", "San Diego", roadmap))
console.log("pass");
    if (isReachableInTwo("San Francisco", "Reno", roadmap))
console.log("pass");
    if (!isReachableInTwo("San Francisco", "Salt Lake City",
roadmap)) console.log("pass");
}

/*
 * This function takes a roadmap object and a filename as inputs.
 * It parses the file and populates the roadmap so that it
 * represents what cities each city is connected to.
 */
function populateRoadmap(roadmap, filename) {
    var lines = File.readLines(filename);
    var delimiter = " - ";
    for (var i = 0; i < lines.length; i++) {
        var line = lines[i];
        var cutoff = line.indexOf(delimiter);
        var cityOne = line.substring(0, cutoff);
        var cityTwo = line.substring(cutoff + delimiter.length);
        if (roadmap[cityOne] === undefined) {
            roadmap[cityOne] = [];
        }
        if (roadmap[cityTwo] === undefined) {
            roadmap[cityTwo] = [];
        }
        roadmap[cityOne].push(cityTwo);
        roadmap[cityTwo].push(cityOne);
    }
    //do we need to return roadmap here? why or why not?
    //answer: no, roadmap is an object, thus passed by reference.
}

/*
 * This function takes two cities and a roadmap and returns
 * a boolean of whether the two cities can be reached within
 * two roads.
 */
```

```
function isReachableInTwo(currentCity, destinationCity, roadmap) {
    if (currentCity === destinationCity) return true;
    var firstLevelNeighbors = roadmap[currentCity];
    for (var i = 0; i < firstLevelNeighbors.length; i++) {
        var firstNeighbor = firstLevelNeighbors[i];
        if (firstNeighbor === destinationCity) return true;
        var secondLevelNeighbors = roadmap[firstNeighbor];
        for (var j = 0; j < secondLevelNeighbors.length; j++) {
            var secondNeighbor = secondLevelNeighbors[j];
            if (secondNeighbor === destinationCity) {
                return true;
            }
        }
    }
    return false;
}
```

2. Search

```
import "file";

/*
 * Makes an index of the web from all the files provided.
 * Then allows the user to enter in a search query, and if
 * a match is found from the web it will be returned to the user.
 */
function main() {
    var filenames = ["TheCatAndTheHat.txt",
"HowAreGummyWormsMade.txt", "EricRobertsWiki.txt",
"JerryCainCS110Homepage.txt", "IntroPhotography.txt"];
    var indexedWeb = IndexedWeb(filenames);

    var callback = function(query) {
        return indexedWeb.search(query);
    };
    console.requestInput("Enter your search query: ", callback);
}

/*
 * The IndexedWeb class takes an array of filenames, finds the most
 * frequent word in each file to use as the file's tag, and stores
 * a map from tag to files. It returns an object containing the
functionality
 * to search the IndexedWeb given search queries.
 */
function IndexedWeb(filenames) {
    var web_index = {};
    for (var i = 0; i < filenames.length; i++) {
        var tag = findMostFrequentWord(filenames[i]);
        console.log("filename: " + filename);
        console.log("tag: " + tag);
        web_index[tag] = filenames[i];
    }

    function findMostFrequentWord(filename) {
        words = File.read(filename).split(" ");
        var word_counts = {};
        for (var word in words) {
```

```
            if (word_counts[word] === undefined) {
                word_counts[word] = 0;
            }
            word_counts[word]++;
        }
        var largest_key = undefined;
        var largest_value = undefined;
        for (var key in word_counts) {
            if (largest_key === undefined) {
                largest_key = key;
                largest_value = word_counts[key];
            } else {
                if (word_counts[key] > largest_value) {
                    largest_key = key;
                    largest_value = word_counts[key];
                }
            }
        }
        return largest_key;
    }

    return {
        search: function(search_query) {
            return web_index[search_query];
        }
    };
}
```

3. Potions

```
import "file";

/*
 * Makes a potion collection from the potionsFile.
 * Then, allows the user to enter the name of a potion.
 * The program outputs the ingredients for that potion.
 */
function main() {
    const potionsFile = "potions.txt";
    var potionCollection = PotionCollection(potionsFile);
    var callback = function(name) {
        var potion = potionCollection.getPotion(name);
        if (potion !== null) {
            var ingredients = potion.getIngredients();
            for (var i = 0; i < ingredients.length; i++) {
                console.log(ingredients[i]);
            }
        }
        console.requestInput("Enter name of the potion: ",callback);
    };
    console.requestInput("Enter name of the potion: ", callback);
}

/*
 * The Potion class takes the name of the potion and
 * returns an object that contains the functions to get
 * the names and ingredients of the potion, as well as
```

```
 * to add an ingredient.
 */
function Potion(name) {
    var ingredientList = [];
    return {
        getName: function() {
            return name;
        },
        getIngredients: function() {
            return ingredientList;
        },
        addIngredient: function(ingredient) {
            ingredientList.push(ingredient);
        }
    };
}

/*
 * The PotionCollection class takes a filename of potions.
 * It adds all the potions in the file into a PotionCollection
 * and returns an object that contains the functions to get
 * a potion or get all potion names.
 */
function PotionCollection(filename) {
    //open file
    var lines = File.readLines(filename);
    if (lines === undefined) return null;
    var collection = {};

    //add potions from file
    var line_num = 0;
    while (line_num < lines.length) {
        var name = lines[line_num];
        line_num++;
        var potion = Potion(name);
        while (true) {
            var line = lines[line_num];
            line_num++;
            if (line_num >= lines.length || line === "") break;
            potion.addIngredient(line);
        }
        collection[name] = potion;
    }

    return {
        getPotion: function(name) {
            if (collection[name] === undefined) return null;
            else return collection[name];
        },
        getPotionNames: function(name) {
            var keys = [];
            for (var key in collection) {
                keys.push(key);
            }
            return keys;
            //ignore the order in which they appeared in file.
        }
    };
}
```