Practice tasks

**Title:**Developing CLI tool to work with file system

**Purpose:**

The purpose of this task is to get some hands-on experience with fs module and build a simple CLI program based on Node.js. As a result of this home task, you will solidify your knowledge of Node.js modules system, dive dipper into fs module and asynchronous code workflow.

\*Note the usage of NPM packages is forbidden. And you should use only asynchronous version of functions that are exposed from Node.js core modules.

**Prerequisites:**

Before we start please make sure you have Node.js current LTS version installed on your machine. We advise you to use NVM (Node Version Manager), but you can still just install the Node.js directly. You can use IDE of your choice.

**Tasks:**

**Main Task 1: Implementing the CLI.**

**Estimated time: 1h - 2h**

Create an index.js file that will play an entry point into your CLI program. Here you should parse the arguments passed to you and then pass command arguments to the function that will implement a command. In scope of this task, you will develop the following commands.

**rimraf**

Recursively removes a folder and all its contents.

Create a file rimraf.js that will export a function that implements needed behavior. The function should accept path to the folder that should be removed and the callback function.

function rimraf (path, callback){};

module.exports = rimraf;

In the index.js file parse the arguments and pass –path argument value to the function. In case the argument was not provided throw an exception.

Below you can find examples of the expected usage.

1. This one works okay - node ./index.js rimraf –path ./folder
2. This one fails with an error, because argument –path was not provided, node ./index.js rimraf

**rename**

Renames a folder or a file.

Create a file rename.js that will export a function that implements expected behavior. The function should accept three arguments from, to and the callback function. From argument will point to the folder/file that should be renamed and to argument will tell the new name.

function rename (from, to, callback){};

module.exports = rename;

In the index.js file parse the arguments and pass them to the rename function. In case arguments –from and –to was not passed, throw an exception.

Below listed off examples of the expected usage for your CLI.

1. This one works okay - node ./index.js rename –from ./folder/test.txt --to guide.txt
2. This one fails with an error, because argument –from was not provided – node ./index.js  --to guide.txt

Apart from that add validation for command name if incorrect command name specified throw an exception.

**Assessment checklist:**

1. Validate that no synchronous version of functions is used.
2. Validate that passing a rimraf command without –path argument causes an exception.
3. Validate that rimraf command deletes a folder and all files inside without an error.
4. Validate that passing a rename command with incorrect arguments causes an exception.
5. Validate that rename command renames a folder or a file passed as –from argument to value passed as –to.
6. Validate that tests are passing.
7. Validate that if incorrect command name specified CLI thrown an exception.

**Main Task 2: Implementing analyze function**

**Estimated time: 2h - 4h**

The analyze function should gather information about a folder and its content. The function should return an object that will have the following format.

{

totalFiles: number; // Contains total count of files in the folder and all its sub-folders.

totalSubFolders: number; // Contains total count of subfolders in the folder and all its sub-folders.

fileTypesInformation: FileTypeInformation[]; // Contains an array of FileTypeInformation objects that are described below.

}

The FileTypeInformation object type has the following format.

{

fileExtension: string; // Contains file extension. This should be unique in the whole fileTypesInformation array.

fileCount: number; // Contains total count of files with such extension in the folder and all its sub-folders.

}

Create a file analyze.js that will export a function that implements expected behavior. The callback function passed to the analyze function should be called with the result object that has the format described in the beginning of the task. The first parameter of the callback will be an error object in case any occurred, the second parameter will be result. Keep in mind the node.js callback pattern.

**Assessment checklist:**

1. Check that unit tests passing.
2. Validate that no synchronous version of functions is used.