ECE 364

Sample Practical I

September 21, 2017

Name: ______________________________ ID: ee364_____

Passing this exam will satisfy course objectives CO1, CO2, CO3, and CO7

Instructions

- **During the exam, TAs can only answer questions clarifying instructions. They cannot help you debug your program, as is usually the case, or answer questions as they can during the regular lab.**
- You may use the command below to copy the necessary files:

  ```
  cp -r ~ee364/labfiles/Sample/* ./
  ```

- Remember to add and commit all files to SVN. **We will grade the version of the file that is in SVN!**
- Make sure you Python file compiles. **You will not receive any credit if your file does not compile.**
- Make sure your output from all functions match the given examples. **You will not receive points for a question whose output mismatches the expected result.**
- Make sure you are using Python 3.4 for your project. In PyCharm, go to:

  File Menu → Settings → Project Interpreter

  And make sure that Python 3.4 (`/usr/local/bin/python3.4`) is selected.
- During the lab, you can access the following resources:
  - The lecture notes in electronic form.
  - The Python Help Documentations in PDF format.
  - Your own previous labs & prelabs.

  You are not allowed to access any other resources.

Bash

Part I [25 pts]

For each of the parts below, write short blocks of code to achieve the given task. Use the provided template file `Bash_Part1.bash`. (**Note**: All other supporting files mentioned below are not provided to you. However, you may create them as necessary to test your code.)

a Count the number of .pdf files in a directory called `myDir`.

b Print out the n-th line of a file called `data.txt`, where n is a variable that stores an integer.

c Run the script `userinput.o`, which expects data from STDIN. However, you must provide data from a file `input.txt` instead of from the terminal.

d Compare the contents of two files `foo1.txt` and `foo2.txt`. If the files match, print `"Files are simliar."`. If not, print `"Files are different."`. Suppress all other output.

e Compile the file `windows8.c` with the gcc compiler and redirect both STDERR and STDOUT to the file `compile.out`.

Part II [25 pts]

Introduction

You have been provided the file `data.txt` that contains a list of people's names, and the times at which they are available during a regular 7AM-5PM day. Your task is to create a meeting scheduler, that uses this data to figure out the best time to set a meeting, i.e. the time at which maximum number of people are available.

The format of the input file is as follows:

```
NAME TIME_1,TIME_2,TIME_3,...,TIME_N
```

where each `TIME_N` is given in 24-hour format, and indicates a person's availability for that one hour. Note that the list of times may not necessarily appear in numerical order.

Implementation Details

Using the template file, `scheduler.bash`, write a script that satisfies the following requirements:

- Check that the number of arguments provided is exactly one. If not, print an error message as shown in the sample output, and exit with a return code of 1.
- Check that the input file exists and is readable. If not, print an error message as shown in the sample output, and exit with a return code of 2.
- Read the input file and tabulate the data in a form similar to the sample output. Each entry in the table should either be a 'Y' if the person is available at that time, or a '-' if he/she isn't available. This output should be saved to a file called `schedule.out`.
- **[Extra credit - 10 pts]**
  - Add a 'Total' row to `schedule.out`, in which each value indicates the total number of people available at that particular time. (See sample output)
  - Print a message to STDOUT of the format:
    `The maximum number of people are available at <TIME>.`
    where `<TIME>` (in 24-hour format) indicates the best choice of a meeting time for this group of people. If there are multiple times at which maximum people are available, use the time that is earlier in the day.

Sample Output

```
$ scheduler.bash data.txt
The maximum number of people are available at 15:00.


$ cat schedule.out
        07:00 08:00 09:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00
Michael Y     -     Y     -     -     -     -     Y     Y     -     -
Marcus  Y     -     -     Y     -     -     -     -     -     Y     Y
Mollie  -     -     Y     Y     Y     Y     -     -     -     -     -
Marlene -     -     -     -     -     -     -     -     Y     Y     Y
Mikayla -     Y     Y     Y     -     -     -     -     Y     Y     -
Madison -     -     -     -     -     -     -     -     Y     -     -
Maxwell Y     Y     Y     Y     Y     Y     Y     Y     Y     Y     Y
Mathew  -     Y     -     -     -     -     -     -     -     -     -
Morgan  -     Y     Y     Y     -     -     -     -     -     -     -
Maddie  -     -     -     -     -     -     -     -     Y     Y     Y

Total   3     4     5     5     2     2     1     2     6     5     4
```

Note: The file `schedule.out.gold` is provided for you to check your output. You may use the command:

```
$ diff -yb schedule.out schedule.out.gold
```

Python (50pt)

Use the Python file named `practical1.py`, and do all of your work in that file. This file should only consists of function blocks, and, optionally, a conditional main block, (i.e. `if __name__ == "__main__":`). Do not include any loose Python statements, but you can, however, write any number of additional utility/helper functions that you might need.

Part I

In the popular game "Sudoku," the objective is to populate a 9×9 grid with the digits 1 to 9 such that:

a The digits 1 to 9 are present once in every column.

b The digits 1 to 9 are present once in every row.

Below is an example of a typical starting puzzle grid and its solution grid.

		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7								8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

4	8	3	9	2	1	6	5	7
9	6	7	3	4	5	8	2	1
2	5	1	8	7	6	4	9	3
5	4	8	1	3	2	9	7	6
7	2	9	5	6	4	1	3	8
1	3	6	7	9	8	2	4	5
3	7	2	6	8	9	5	1	4
8	1	4	2	5	3	7	6	9
6	9	5	4	1	7	3	8	2

The examples shown above are stored in the files `"Sample01.sud"` and `"Sample01Solution.sud"`. Take a look at these files and familiarize yourself with them, as you will be required to read and write files with that format.

In this part, we will only focus on two Sudoku cases: a puzzle that has an empty row, *or* one that has an empty column. Figures 1 and 2 below show some puzzles and their solutions.

1	9	8	7		4	2	6	5
5	6	4	1		2	3	7	8
2	7	3	8		5	9	1	4
3	1	5	4		7	6	8	9
8	4	9	3		6	7	2	1
6	2	7	9		8	5	4	3
7	3	6	5		1	8	9	2
4	5	2	6		9	1	3	7
9	8	1	2		3	4	5	6

1	9	8	7	3	4	2	6	5
5	6	4	1	9	2	3	7	8
2	7	3	8	6	5	9	1	4
3	1	5	4	2	7	6	8	9
8	4	9	3	5	6	7	2	1
6	2	7	9	1	8	5	4	3
7	3	6	5	4	1	8	9	2
4	5	2	6	8	9	1	3	7
9	8	1	2	7	3	4	5	6

Figure 1: An example of a Sudoku puzzle with a missing "column" along with its solution.

1	3	6	5	7	4	9	8	2
5	9	8	6	2	1	3	4	7
7	2	4	3	9	8	6	1	5
9	7	5	1	3	6	8	2	4
2	8	3	7	4	9	1	5	6
4	6	1	2	8	5	7	9	3
3	4	7	8	1	2	5	6	9
8	5	9	4	6	7	2	3	1

1	3	6	5	7	4	9	8	2
5	9	8	6	2	1	3	4	7
7	2	4	3	9	8	6	1	5
9	7	5	1	3	6	8	2	4
2	8	3	7	4	9	1	5	6
4	6	1	2	8	5	7	9	3
6	1	2	9	5	3	4	7	8
3	4	7	8	1	2	5	6	9
8	5	9	4	6	7	2	3	1

Figure 2: An example of a Sudoku puzzle with a missing "row" along with its solution.

Requirement

1. **[6pt]** Write a function called `isValidOutput(fileName)` that takes in a file name containing a completed Sudoku puzzle, and verifies whether the solution is valid or not, i.e. whether the objectives mentioned above are satisfied or not. If the solution is valid, return `True`, otherwise return `False`. The files `"valid.sud"`, `"invalid1.sud"` and `"invalid2.sud"` can be used as a reference.
2. **[2pt]** Write a function called `isColumnPuzzle(fileName)` that takes in a file name containing an open Sudoku puzzle, and checks whether it has a missing column and returns `True`, otherwise returns `False`.
3. **[17pt]** Write a function called `solvePuzzle(sourceFileName, targetFileName)` that reads the Sudoku puzzle from the source file, (which can be missing either a column or a row,) solves it, then stores the solution in the target file using the same format of the source. No returns are expected from this function. The files `"open1.sud"` and its solution `"solved1.sud"`, as well as `"open2.sud"` and its solution `"solved2.sud"` should be checked to validate your solution.

Part II

A group of people are living together and they share the same phone service. Each individual is assigned a different extension to be billed separately. The file `"People.txt"` contains the names of and extensions of these people. You are given the log file `"ActivityList.txt"` containing a list of outgoing calls from the house, where each entry contains the caller phone number (the extension is the last four digits), the destination number and the duration of each call.

1. **[10pt]** Write a function called `getCallersOf(phoneNumber)` that takes a phone number and returns a "sorted" list of names who called that number. If no one called that number, return an empty list. For example:

```
>>> getCallersOf("707-825-5871")
['Brooks, Carol', 'Butler, Julia', 'Carter, Sarah', 'Edwards, Rachel', 'James, Randy',
'Miller, Aaron', 'Nelson, Louise', 'Rogers, Elizabeth', 'Washington, Annie']
```

2. **[15pt]** Write a function called `getCallActivity()` that returns a `{string:tuple}` dictionary, where the dictionary key is the name, and the dictionary value is the `(int, string)` tuple. The first element is the number of calls made by that person, and the second element is the total duration of all the calls made by the person formatted as `hh:mm:ss`. For example:

```
>>> callMap = getCallActivity()
>>> callMap["Edwards, Rachel"]
(56, '13:54:46')
```

Note: To get the right duration format, use the format string: `"{0:02d}:{1:02d}:{2:02d}"` as in:

```
>>> "{0:02d}:{1:02d}:{2:02d}".format(9, 2, 7)
'09:02:07'
```