Lab Test 2 [25 marks]

Coding time: 1.5 hours

General Instructions:

1. You will perform the lab test on your personal laptop.

2. You are not allowed to communicate with anyone or access any network during the test. Disable the following connections on your laptop before the test begins: Wi-Fi, Bluetooth, and any other communication devices (e.g. 3G/4G modems).

4. You may refer to any file on your laptop during the test.

5. Make sure your code can generate exactly the same output as we show in the sample runs. You may be penalized for missing spaces, missing punctuation marks, misspelling, etc. in the output.

6. Do not hardcode. We will use different test cases to test and grade your solutions.

7. Follow standard Python coding conventions (e.g., naming functions and variables).

8. Python script file that cannot be executed will NOT be marked and hence you will be awarded 0 marks. You may wish to comment out the parts in your code which cause execution errors.

9. Include your name as author in the comments of all your submitted source files. For example, include the following block of comments at the beginning of each source file you need to submit.

```
# Name: Lum Ting Wong
# Email ID: lum.ting.wong.2019
```

Instructions on how to submit your solutions:

1. Before the test begins, you will be instructed to download the resource files of the test from eLearn. After downloading the resource files, you should disble the internet on your laptop.

2. You will be given a password to unzip the downloaded .zip file. After unzipping the file, **you should rename the folder to your email id.** For example, if your SMU email is lum.ting.wong.2019@sis.smu.edu.sg, you should rename the folder to lum.ting.wong.2019. You need to save your solutions to this folder for submission later. You may be penalized for not following our instructions.

3. When the test ends, you will be instructed to enable your laptop's internet and submit your solutions as a single zip file to eLearn.

Question 1 (Difficulty Level: *) **[5 marks]**

Inside `q1.py`, implement a function called `compute_product()`. The function takes in a list of integers as its parameter. The function **returns** the product of the odd integers in the list.

If the list doesn't contain any odd integer, the function returns 1.

Some examples are shown below:

- `compute_product( [1, 2, 3, 4] )` returns 3 (= 1 x 3), because the odd integers in the list are 1 and 3.
- `compute_product( [1, 3, 5] )` returns 15 (= 1 x 3 x 5), because the odd integers in the list are 1, 3 and 5.
- `compute_product( [2, 4, 6, 8, 111] )` returns 111, because 111 is the only odd integer in the list.
- `compute_product( [-2, -3, -5, -3, 100] )` returns -45 (= (-3) x (-5) x (-3)), because the odd integers in the list are -3, -5 and another -3.
- `compute_product( [4, 40, 400] )` returns 1, because the list doesn't contain any odd integer.
- `compute_product( [] )` returns 1, because the list doesn't contain any odd integer.

Question 2 (Difficulty Level: *) **[5 marks]**

In `q2.py`, implement a function call `get_longer_words()` that does the following. The function takes in a single parameter called `file_name` (of type `str`), which is the name of an existing text file. Each line of this file contains two words separated by the symbol `'&'`. The function reads the file, takes the longer word in each line of the file and returns them as a list of strings. If the two words in the same line are of the same length, then the first word should be included in the returned list.

Note:

- You can assume that each line of the file contains excactly a single `'&'` character, i.e., each line contains exactly two words.
- The two words separated by `'&'` can contain any character other than `'&'`.
- The words can be empty strings.

Some examples are below:

- If the file `q2_input_1.txt` looks like the following:

```
SIS&LKCSB
Python&Java
print&return
int&float
a&b
IS110&IS111
```

  Then `get_longer_words('q2_input_1.txt')` should return `['LKCSB', 'Python', 'return', 'float', 'a', 'IS110']`.

- If the file `q2_input_2.txt` looks like the following:

```
Grab&Uber
McDonald&KFC
Amazon&Lazada
tea&coffee
New York&San Francisco
&empty string
```

  Then `get_longer_words('q2_input_2.txt')` should return `['Grab', 'McDonald', 'Amazon', 'coffee', 'San Francisco', 'empty string']`.

Question 3 (Difficulty Level: **) **[5 marks]**

In `q3.py`, implement a function called `compute_total_price`. The function takes in two parameters:

- `price_dict` (type: `dict`): This is a dictionary where keys are strings and values are floats. Each (key, value) pair indicates the price of an item. E.g., if `price_dict` is `{"keyboard": 25.5, "mouse": 10.6}`, it means the price of a keyboard is $25.50 and the price of a mouse is $10.6.
- `item_list` (type: `list`): This is a list of tuples, where each tuple contains a string and an integer. The string is the name of an item and the integer is its quantity. E.g., if `item_list` is `[("keyboard", 1), ("mouse", 3)]`, it means there are 1 keyboard and 3 mice in this item list.

The function `compute_total_price()` **returns** the total price of the items in the item list based on the price information in the price dictionary.

Note:

- The same item may appear more than once in the item list.
- The quantity of an item in a tuple could be 0.
- If an item in the item list is not found in the price dictionary, treat the price of that item as 0.0.

Some examples are below:

- If `price_dict` is `{"keyboard": 25.5, "mouse": 10.6}` and `item_list` is `[("keyboard", 1), ("mouse", 3)]`, then `compute_total_price(price_dict, item_list)` returns 57.3 (because 25.5 x 1 + 10.6 x 3 = 57.3).

- If `price_dict` is `{"A": 2.5, "B": 4.0, "C": 9.0}` and `item_list` is `[("A", 2), ("C", 10)]`, then `compute_total_price(price_dict, item_list)` returns 95.0 (because 2.5 x 2 + 9.0 x 10 = 95.0).

- If `price_dict` is `{"a": 1.0, "b": 2.0, "c": 3.0}` and `item_list` is `[("a", 0), ("b", 1), ("c", 2), ("b", 2)]`, then `compute_total_price(price_dict, item_list)` returns 12.0 (because 1.0 x 0 + 2.0 x 1 + 3.0 x 2 + 2.0 x 2 = 12.0).

- If `price_dict` is `{"a": 1.0, "b": 2.0}` and `item_list` is `[("b", 1), ("c", 2)]`, then `compute_total_price(price_dict, item_list)` returns 2.0. Note that here the price of item "c" cannot be found in the dictionary. Therefore, its price is treated as 0.

- If `price_dict` is `{"a": 1.0, "b": 2.0}` and `item_list` is `[]`, then `compute_total_price(price_dict, item_list)` returns 0.0 (because there is no item in the item list).

Question 4 [10 marks]

Part (a) (Difficulty Level: **)

In `q4a.py`, implement a function called `store_family_relations()` that does the following. The function takes in the following parameter:

- `family_file` (type: `str`): This is the name of a file that stores the information of several families.

  Each line of the file shows the members of a family with a father, a mother and all their children. The gender of each child is also given.

  Each line of the file is of the following format:

  `(father,mother):(child_1,gender);(child_2,gender);…`

  For example, the following line shows a family where **Jason** and **Fenny** are the father and the mother, and there are three children in the family: two boys named **Jerry** and **Jeremy**, and a girl named **Jenny**.

  `(Jason,Fenny):(Jerry,M);(Jeremy,M);(Jenny,F)`

  The file contains multiple lines, each representing a different family.

  You can make the following assumptions:

  - Nobody's name contains any space.
  - In each line, father's name always appears before mother's name.
  - Each family has at least one child.
  - Gender is always indicated by either `M` (for male) or `F` (for female).
  - Each line is always well formatted as shown in the example above.
  - There is no space in the file.

The function `store_family_relations()` **returns** a ***dictionary*** to store all the ***parent-child relations*** found in the file. Specifically, each key in the dictionary is a tuple `(A, B)`, where `A` and `B` are two people's names. The value associated with `(A, B)` describes how `A` is related to `B`. For example, if `A` is `B`'s father, then the value for `(A, B)` is `'father'`.

The relation is always one of the following: `'father'`, `'mother'`, `'son'`, `'daughter'`. The dictionary needs to include ***all*** pairs of people who have such relations.

For example, if `family_file` looks like the following:

```
(Adam,Bella):(Cindy,F);(Darren,M)
(Eric,Fiona):(George,M)
```

then the returned dictionary should include exactly the following (key, value) pairs:

```
('Adam', 'Cindy') : 'father'
('Adam', 'Darren') : 'father'
('Bella', 'Cindy') : 'mother'
('Bella', 'Darren') : 'mother'
('Cindy', 'Adam') : 'daughter'
```

```
('Cindy', 'Bella') : 'daughter'
('Darren', 'Adam') : 'son'
('Darren', 'Bella') : 'son'
('Eric', 'George') : 'father'
('Fiona', 'George'):'mother'
('George', 'Eric'):'son'
('George', 'Fiona'):'son'
```

Note: The (key, value) pairs are stored in the dictionary **in any order**.

Part (b) (Difficulty Level: *)**

In `q4b.py`, implement a function called `get_relation_through_link()`. This function takes in the following parameters:

- `family_dict` (type: `dict`): A dictionary that stores the parent-child relations between people. This is a dictionary that the function `store_family_relations()` in `q4a.py` returns.

- `link` (type: `list`): A list of ***different*** people's names where each two adjacent people in the list are related through some parent-child relations in `family_dict`.

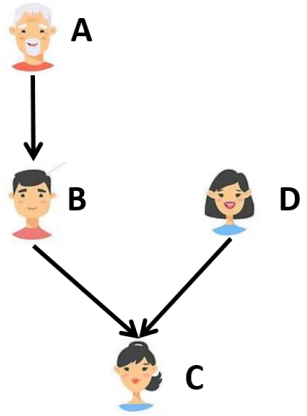
  For example, if `link` is `['Adam', 'Cindy', 'Joseph']`, then we can assume that `Adam` and `Cindy` are directly related as parent-child, i.e., we can find the key `('Adam', 'Cindy')` in `family_dict`. Similary, `Cindy` and `Joseph` are also directly related.

The function returns ***the family relation between*** the ***first*** person in `link` and the ***last*** person in `link`. The function makes use of a file called `'relation_mapping.txt'` to derive the indirect relations.

For example, suppose `family_dict` contains the following parent-child relations.

```
('A', 'B') : 'father'
('B', 'A') : 'son'
('B', 'C') : 'father'
('C', 'B') : 'daughter'
('D', 'C') : 'mother'
('C', 'D') : 'daughter'
```

These relations also illustrated in the picture below. In the picture, an arrow always goes from a parent to his/her child.



- If `link` is `['A', 'B', 'C']`, then the function should return `'grandfather'`. This is because after looking up the relation between `A` and `B` and the relation between `B` and `C` in `family_dict`, we can find that `A` is `B`'s father and `B` is `C`'s father. We can then refer to the file `'relation_mapping.txt'`, which contains a mapping that says if X is Y's father and Y is Z's father, then X is Z's grandfather. (More explanation on the file `'relation_mapping.txt'` will be given below.) Therefore `A` is `C`'s grandfather.
- If `link` is `['C', 'B', 'A']`, then the function should return `'granddaughter'` (because `C` is `B`'s daughter and `B` is `A`'s son).
- If `link` is `['B', 'C', 'D']`, then the function should return `'husband'`. (`B` is `C`'s father and `C` is `D`'s daughter, and `D` is a different person from `B`. Therefore, `D` must be `C`'s mother. Hence `B` is `D`'s husband.)

- If `link` is `['D', 'C', 'B', 'A']`, then the function should return `'daughter-in-law'`. (Because `D` is `C`'s mother and `C` is `B`'s daughter, `D` must be `B`'s wife. And since `B` is `A`'s son, `D` must be `A`'s daughter-in-law.)

You can make the following assumptions:

- The list `link` always gives a sequence of people where adjacent people are related through a parent-child relation.
- The list `link` contains **at least 2 elements** and **no more than 4 elements**.
- All names in `link` appear only once in `link`.

To help you derive indirect family relations, you are given a file called `relation_mapping.txt`. Open the file to observe what is stored inside the file.

Each line of the file maps two relations to a single relation. Take a look at the following line from `relation_mapping.txt`:

```
(father,mother):grandfather
```

The line above states that if `X` is the father of `Y` and `Y` is the mother of `Z`, then `X` is the grandfather of `Z`.

In general, each line looks like the following:

```
(rel1,rel2):rel3
```

Suppose we also have the following parent-child relationships in `family_dict`:

```
('X', 'Y') : rel1
('Y', 'Z') : rel2
```

Then the line in the file states that if `X` is `Y`'s `rel1` and `Y` is `Z`'s `rel2`, then `X` is `Z`'s `rel3`. **Importantly**, please note that here we assume that `Z` is not already related to `X` through some parent-child relation.

Another example is the following line in the file:

```
(grandfather,son):father-in-law
```

The line above states that if `X` is `Y`'s grandfather and `Y` is `Z`'s son, then `X` is `Z`'s father-in-law. Please note that here because we assume `Z` is not related to `X` through some parent-child relation, `X` cannot be `Z`'s father.

Please note the following:

- For each line "`(rel1,rel2):rel3`" in `relation_mapping.txt`, the second relation `rel2` is always a direct parent-child relation, i.e., `rel2` is always one of the following: `father`, `mother`, `son`, `daughter`.
- You can assume that `relation_mapping.txt` contains ALL the possible relation mappings.

You should make use of the file `relation_mapping.txt` in your implementation of `get_relation_through_link()`.

Part (c) (Difficulty Level: **)**

In `q4c.py`, implement a function called `get_relation()`. This function takes in the following parameters:

- `family_dict` (type: `dict`): A dictionary that stores the parent-child relations between people. This is a dictionary that the function `store_family_relations()` in `q4a.py` returns.
- `p1` (type: `str`): A person's name.
- `p2` (type: `str`): another person's name.

The function returns ***the family relation between*** `p1` and `p2`.

You can assume that `p1` is always related to `p2` through no more than 3 hops of parent-child relations found in `family_dict`.

For example, suppose `family_dict` contains the following parent-child relations:

```
('A', 'B') : 'father'
('B', 'A') : 'son'
('B', 'C') : 'father'
('C', 'B') : 'daughter'
('D', 'C') : 'mother'
('C', 'D') : 'daughter'
```

- If `p1` is `'A'` and `p2` is `'C'`, then the function should return `'grandfather'`.
- If `p1` is `'C'` and `p2` is `'A'`, then the function should return `'granddaughter'`.
- If `p1` is `'B'` and `p2` is `'D'`, then the function should return `'husband'`.
- If `p1` is `'D'` and `p2` is `'A'`, then the function should return `'daughter-in-law'`.

Similar to part (b), you have access to the file `relation_mapping.txt` to help you with your implementation of `get_relation()`. You are also encouraged to call the function `get_relation_through_link()` from part (b) to help you with the implementation.