Lab 6 (2) – Revision

1. The cost of delivering an item to a country for a post office, is given by the following table:

AIRMAIL RATES[#^]		
Destination	**Weight-Step Up To** (max weight: 2kg)	**Letters, Printed Papers[*] and Packets/Packages**
Zone 1 Malaysia and Brunei	20g	$0.50
	50g	$0.70
	100g	$1.10
	per additional 100g	$1.10
Zone 2 Countries in Asia & The Pacific (except Australia, Japan & New Zealand)	1st 20g	$0.70
	per additional 10g	$0.25
Zone 3 Countries in the rest of the world, including Australia, Japan, New Zealand, Africa, The Americas, Europe & The Middle East	1st 20g	$1.30
	per additional 10g	$0.35

The cost is dependent on the weight of the item and the zone of the destination.

a) Write a program that reads the weight and the zone and outputs the cost of postage. Weight must be positive and non-zero and zone must be either 1, 2 or 3.

b) Modify part a) so that a user may repeatedly enter the weight of the item and the zone of the destination for as many items as he wishes to send. Assume that data entry is terminated when user enters an empty string when prompted for the weight of item.

2. a) An Armstrong number of N digits is an integer such that the sum of its digits to the power of N is equal to the number itself. For example, 153 = 1**3 + 5**3 + 3**3 and 1634=1**4 + 6**4 + 3**4 + 4**4 . Write a function `isArmstrongNumber(num)` which has an integer parameter num. The function returns `True` if num is an Armstrong number and `False` otherwise.

b) Write a main function that continually displays a sequence of dice values. The sequence starts with the first value. Subsequent value must be higher, then lower, then higher etc. until the sequence breaks. This is a sample display:

2
4
1
5
6 ← breaks the sequence

3. This is a multiple player high-low guessing game. At the start of the game session, the names of players playing the guessing are entered.

   Each player takes turn to guess a number generated by the program. The outcomes are revealed only after the players have guessed the number. For each game, track the winner(s) so that a summary of the winners for each game can be printed at the end of the game session.

   Example output:

```
Enter player's name or <Enter> to end: Tom
Enter player's name or <Enter> to end: Jane
Enter player's name or <Enter> to end: Alex
Enter player's name or <Enter> to end: <Enter>

Enter a guess between 1 and 100 inclusive.
Tom, make a guess: 50
Jane, make a guess: 75
Alex, make a guess: 25
Processing guess of Tom: 50 Too low.
Processing guess of Jane: 75 Too low.
Processing guess of Alex: 25 Too low.
Tom, make a guess: 80
Jane, make a guess: 85
Alex, make a guess: 90
Processing guess of Tom: 80 Too low.
Processing guess of Jane: 85 Too low.
Processing guess of Alex: 90 Too high
Tom, make a guess: 86
Jane, make a guess: 86
Alex, make a guess: 87
Processing guess of Tom: 86 Correct! in 3 tries
Processing guess of Jane: 86 Correct! in 3 tries
Processing guess of Alex: 87 Too high
Winners: Tom Jane
Continue game?(y/n): y
Enter a guess between 1 and 100 inclusive.
Tom, make a guess: 60
Jane, make a guess: 40
Alex, make a guess: 20
Processing guess of Tom: 60 Too high
Processing guess of Jane: 40 Too high
Processing guess of Alex: 20 Too high
Tom, make a guess: 10
Jane, make a guess: 13
Alex, make a guess: 15
Processing guess of Tom: 10 Too low.
Processing guess of Jane: 13 Too low.
Processing guess of Alex: 15 Correct! in 2 tries
Winner: Alex
Continue game?(y/n): n
Games Winners
Game 1 Winners: Tom Jane
Game 2 Winner: Alex
```

4. A soup restaurant requires a program to manage different types of soups. Each type of soup has a quantity of servings.

   The available soups are stored in a file `soups.txt`.

   a) Write a function initializeSoups() that reads the available soups from soups.txt. Store the soups as a dictionary in the following format:

```
soups={'C':['Clam Chowder', 50], 'M':['Mushroom', 45],
'T':['Tomato', 40], 'P':['Pumpkin', 50], 'O':['Oxtail',10]}
```

      The first letter of the soup is the key, and the value is a list with the description and quantity. Return the dictionary.

   b) Develop a program with the following menu that allows a user to choose one of the following options:

```
1. Purchase Soup
2. Replenish Soup
3. Display all soups
0. Exit
```

      Option 1 displays only available types of soups (servings > 0) and quantity available. The user can choose the type of soup (identified by the key) and enters the servings required. The servings left for each soup should not be negative. No validation required.

      Option 2 displays only types of soups (servings = 0) that requires replenishment . The user can enter the type of soup and quantity to replenish by. The final servings left for a soup that is replenished should not exceed 50.

      Option 3 displays all the soups and servings available.

soups.txt

```
Clam Chowder, 50
Mushroom, 45
Tomato, 40
Pumpkin, 50
Oxtail, 10
```