Project #2 Instructions

For the second project,

1. Pick one of the problem below.
2. Solve it. Make sure you test your program thoroughly, making relevant test cases, and comment your code.
3. Share the code with me in any way you like: email, box, on paper, by mail, etc. Sharing only the source code and not the whole project is fine, but remember to write your name, the date, *and the name of the problem you are solving* in a delimited comment at the beginning of the source code, and make sure I have access to your project before **03/22, 11:59 PM**.
4. As always, partial feedback is possible, but make sure the code you are sending was written by you and by you only.

Report to the listings given with each problem for examples of execution (where the user input is underlined, and hitting carriage return is represented by ↵), and Lab 17 for some guidance.

Problem 1 *Daily Pay*

Write a program that asks the user for a number of days, and calculates how much money they would earn over that many days if their salary were one penny (one dollar cent) the first day, two pennies the second day, four pennies the third day, and so on, doubling each day. Your program should display a nicely formatted table showing the salary for each day, as well as the total pay for the number of days specified by the user. The output should be displayed in dollar amounts, and aligned (it is fine if for extremely large value, e.g., after ± 50 days or so, the display is "broken").

Once you implemented the previous functionalities, add input validation, so that a user entering non-numerical values or negative values would be asked to enter another value.

An example of execution could be:

```
How many days did you work?
Too Many ↵
How many days did you work?
-4 ↵
How many days did you work?
14 ↵
|       Day      |      Daily Pay        |       Earned So Far   |
-----------------------------------------------------------------
|         1      |          $0.01        |               $0.01   |
|         2      |          $0.02        |               $0.03   |
|         3      |          $0.04        |               $0.07   |
|         4      |          $0.08        |               $0.15   |
|         5      |          $0.16        |               $0.31   |
|         6      |          $0.32        |               $0.63   |
|         7      |          $0.64        |               $1.27   |
|         8      |          $1.28        |               $2.55   |
|         9      |          $2.56        |               $5.11   |
|        10      |          $5.12        |              $10.23   |
|        11      |         $10.24        |              $20.47   |
|        12      |         $20.48        |              $40.95   |
|        13      |         $40.96        |              $81.91   |
|        14      |         $81.92        |             $163.83   |
Press any key to continue . . .
```

Problem 2 *Newspaper Machine*

In this problem, you will implement a newspaper vending machine that accepts only coins. A newspaper costs \$0.65. The vending machine accepts only **N**ickels (\$0.05), **D**imes (\$0.10) and **Q**uarters (\$0.25). Your program should ask the user to enter a coin, and as long as the amount entered so far is less that the price of the newspaper, your program should loop and ask the user to enter another coin. The total entered so far should be displayed after each coin has been inserted. Once the user entered \$0.65 or more, you should display a message saying that the newspaper was delivered, and return the change if there is any.

Once you implemented the previous functionalities, add two features:

- The possibility for the user to type in lower-case or upper-case letters.
- The possibility at every step for the user not to insert any new coin, and to get his/her money back.

An example of execution could be:

```
Please insert a coin (Nickel, Dime, Quarter) or exit (with e).
n ↵
Your total so far is $0.05.
Please insert a coin (Nickel, Dime, Quarter) or exit (with e).
N ↵
Your total so far is $0.10.
Please insert a coin (Nickel, Dime, Quarter) or exit (with e).
D ↵
Your total so far is $0.20.
Please insert a coin (Nickel, Dime, Quarter) or exit (with e).
d ↵
Your total so far is $0.30.
Please insert a coin (Nickel, Dime, Quarter) or exit (with e).
Q ↵
Your total so far is $0.55.
Please insert a coin (Nickel, Dime, Quarter) or exit (with e).
p ↵
Sorry, I didn't get that.
Your total so far is $0.55.
Please insert a coin (Nickel, Dime, Quarter) or exit (with e).
q ↵
Your total so far is $0.80.
Here is your newspaper.
Here is your change: $0.15.
Press any key to continue . . .
```

Problem 3 *Rock, paper, scissors*

Write a program that plays **r**ock, **p**aper, **s**cissors. In this game, the computer and the player secretly chose between rock, paper and scissors, and reveal their choice: rock beats scissors ; scissors beats paper ; paper beats rock ; and all the remaining cases are ties. The player enters a single character P, R or S (or Q to quit), and the program should display what the computer played, and the number of wins, losses, and ties.

Once you implemented the previous functionalities, add the possibility for the user to type in lower-case or upper-case letters.

You will need to use the `Random` class to generate random numbers. Observe and re-use the following code to complete your program:

```
Random myRandomObject = new Random();     // Instantiate the Random class.
int a, i = 0;
while (i <= 100)
{
        a = myRandomObject.Next(1, 11); // Generate a random number between 1 and
        ↪  10, and assign it to a.
        Console.Write(a + " ");         // Display the value of a.
        i++;                            // Increment the counter.
}
```

An example of execution could be:

```
Welcome.
Enter R, P, S, or Q to quit.
S ↵
The computer played paper.
ties:  0, wins:  1, loses: 0
Enter R, P, S, or Q to quit.
p ↵
The computer played rock.
ties:  0, wins:  2, loses: 0
Enter R, P, S, or Q to quit.
R ↵
The computer played rock.
ties:  1, wins:  2, loses: 0
Enter R, P, S, or Q to quit.
q ↵
Bie!
Press any key to continue . . .
```

Problem 4 *Grade calculator*

You may remember that your grade for this class will be computed as follows:

Evaluation	Number	Points	Percents
Quizzes	5	20	10%
Projects	3	20	10%
In-class Tests	2	100	40%
Final Exam	1	100	40%

Write a program that asks the user for their grades obtained *so far* (it is possible that some quizzes, projects, tests are still to be taken, or that the final did not happened yet) and compute their average *so far*.

Your program can either ask the user for the number of quizzes, projects, test and exam taken so far, and then ask for the values, or use a sentinel value (as shown in the example below) to know when to "stop". Note that if the quiz number n, for $n < 5$, was not taken yet, then you should not ask the grade for quiz $n + 1$, and similarly for projects and in-class tests. It is fine to assume that the user will only enter "correct" numerical values and not to perform any user-input validation, but your program should be flexible enough so that changing the number of quizzes, for instance, would require to change only a variable or a couple of values.

An example of execution could be:

```
For all the questions below, enter
        - your grade, or
        - "0" if you missed the evaluation, or
        - "-1" if that evaluation did not happened yet.

What was your grade for quiz 1?
15 ↵
What was your grade for quiz 2?
18 ↵
What was your grade for quiz 3?
-1 ↵
Your average for the quizzes is 82.50%.
What was your grade for project 1?
13 ↵
What was your grade for project 2?
7 ↵
What was your grade for project 3?
-1 ↵
Your average for the projects is 50.00%.
What was your grade for in-class test 1?
78 ↵
What was your grade for in-class test 2?
-1 ↵
Your average for the tests is 78.00%.
What was your grade for the final?
-1 ↵
Your average so far is 74.08%.
Press any key to continue . . .
```