COSC 1437 (DL) - Fall 2016
Program Set #1

See 1437 Grading /Program Guide Sheet for directions and grading/submission information.

1. Flight times in the United States can be estimated using a formula based on the direction of travel and distance between airports. Regardless of the distance or direction, there is also a fixed amount of time for the take-off and landing cycles. Needed information:

- It takes `0.2` hours for takeoff and climb-out from the origin airport.
- It takes `0.4` hours for approach and landing at the destination airport.
- The aircraft ground speed after climb-out when traveling west is `480` miles/hour.
- The aircraft ground speed after climb-out when traveling east is `540` miles/hour.
- The aircraft ground speed after climb-out when traveling north or south is `510` miles/hour.

Given this data, the flight time (`FT`) can be estimated as:

```
FT = 0.20 + 0.40 + (distance / groundSpeed);
```

Write a C++ program that asks the user for the distance traveled (`0-6000`) expressed in whole miles and the flight direction as a character (`N, S, W, E`). For each flight entered output the approximate flight time to the nearest `1/100` of an hour in a time format. Check for case and valid inputs; use constants as needed. To pad numbers with leading zeros you need to insert a `0` into the output format-using the `setfill()` manipulator.

For example: To travel from Dallas/Fort Worth to Chicago O'Hare. The distance is `801` miles and the general direction of travel is north. The travel time can be computed as follows:

`0.20 + 0.40 + (801 / 510) = 0.6 + 1.57 = 2.17` (two decimal places)

Finally, convert decimal answer to time. Format (`HH:MM:SS`):

`2.17 = 02:10:12`

Output should look similar to below.

Sample Runs:

```
Enter the distance traveled: 1883
Enter the direction: E

The estimated flight time is: 4.09 or 04:05:24

Enter the distance traveled: 1100
Enter the direction: S

The estimated flight time is: 2.76 or 02:45:36
```

Name the program: `FlightTimesXX.cpp`, where `XX` are your initials.

2. *Rock, Paper, Scissors, Lizard, Spock!* is an extension of the classic rock, paper, scissors game between two players. This version of the game adds two weapons, Lizard and Spock. The user plays the computer. The player picks one of the five weapons and the computer's weapon is chosen randomly from the five. The outcome is determined based on those choices. It is a tie if the same weapon chosen by the user and computer. If they are different the outcome is determined as follows. The rationale for the outcome is in parenthesis.

- Scissors beats Paper (Scissors CUTS Paper)
- Paper beats Rock (Paper COVERS Rock)
- Rock beats Lizard (Rock CRUSHES Lizard)
- Lizard beats Spock! (Lizard POISONS Spock)
- Spock! beats Scissors (Spock SMASHES Scissors)
- Scissors beats Lizard (Scissors DECAPITATES Lizard)
- Lizard beats Paper (Lizard EATS Paper)
- Paper beats Spock! (Paper DISPROVES Spock)
- Spock! beats Rock (Spock VAPORIZES Rock)
- Rock beats Scissors (Rock CRUSHES Scissors)

Write a C++ program that implements *Rock, Paper, Scissors, Lizard, Spock!* The user will check for valid input a positive integer between 1 and 5. The user plays until either the user or the computer wins the best two of three games. Output should look similar to below.

Sample Run:

```
Welcome! The best 2 of 3 games wins!

Choose:
scissor (1)
rock (2)
paper (3)
lizard (4)
Spock! (5):  1

The computer is scissor. You are scissor too. -It is a draw

Choose:
scissor (1)
rock (2)
paper (3)
lizard (4)
Spock! (5):  4

The computer is rock. You are lizard. (Rock CRUSHES Lizard)- You loose

Choose:
scissor (1)
```

```
rock (2)
paper (3)
lizard (4)
Spock! (5):  3

The computer is Spock! You are paper. (Paper DISPROVES Spock)- You win

Choose:
scissor (1)
rock (2)
paper (3)
lizard (4)
Spock! (5):  2

The computer is lizard. You are rock. (Rock CRUSHES Lizard)- You win

Congratulations- You beat the computer: 2 games to 1
```

Name the program: `RPSLSXX.cpp`, where XX are your initials.

3. Carbon-14 dating is a common technique for measuring the approximate age of matter that was once living. The formula for measuring time base on Carbon-14 (C14) decay is:

```
t = (T / ln(2) * ln(1 + D / P))
```

where `T` is the half-life of Carbon-14 (`5730` years), `ln(2)` is natural logarithm of 2, `D` is the amount of carbon-14 fund in a given dead sample, and `P` is the amount of Carbon-14 in a living specimen today. Write a C++ program that asks the user input the value of `D/P` (such as `0.5`) for a given sample and outputs the approximate age of the sample in years. Use a named constant and a user-defined function in the problem. Output the result to two decimal places.

Sample Run:

```
This program calculates the age of a dead sample based on the ratio of
Carbon 14 in a dead sample.

Enter the ratio D/P: .05

Given a D/P ratio of 0.05 for C14, the age of the sample is 403.33 years.
```

Name the program: `CarbonDatingXX.cpp`, where XX are your initials.

4 (**). In traditional "long multiplication" we determine the product of two integers, x and y, by multiplying x and the individual digits of y, in turn, starting with the unit's digit. The results of these multiplications are arranged appropriately and added, yielding the completed product. The representation of these operations are usually done in a particular manner. For example, the multiplication of 123 by 95 is:

```
  123
   95
x ---
  615
1107
-----
11685
```

The numbers to be multiplied, x and y, are each displayed on a separate line, followed by a horizontal line. The results of multiplying each digit of y by x are then displayed on separate lines, followed by another horizontal line, and then the final product. Write a C++ program to perform a sequence of such multiplications, displaying the results in this traditional representation. Input will be two integers, x and y from the keyboard. Each integer will have no more than 10 digits.

For each pair of integers, perform the multiplication of x by y, displaying the results similar to example shown above. If y contains only a single significant digit, omit the second horizontal line and the sum (since it would be overkill). Display 0 digits only when they are significant. The number of hyphens (-) in the first horizontal line should be the same as the number of digits in the larger value of x and y. The number of hyphens in the second horizontal line, if it is produced, should be the same as the number of digits in the product of x and y. Output the letter x for the multiplication symbol in the answer. Check only for positive numbers (including 0). Finally, ask the user if he/she wishes to run the program again. No credit will be given for just giving the answer without showing the steps involved. Output should look similar to below.

Sample Runs:

```
Enter the first number: 4
Enter the second number: 7

    4
    7
 x --
   28

Run again (Y/N)? y

Enter the first number: 135
Enter the second number: 46
```

```
  135
   46
x ---
  810
 540
 ----
 6210


Run again (Y/N)? N
```

Name the program: `LongMultiplicationXX.cpp`, where `XX` are your initials.