IDIA 618.185

Spring 2012

Homework 2

Please address each question with a PHP based solution. If the question requires input you may test your code using either the PHP console commands or a basic HTML form for the input. If you use the HTML form please make sure to include the HTML as part of your submission.

1. **Question One**

Read a nonnegative integer N (range 0 to 20) and print the powers of 2 from 1 to 2N. You do not need to validate that the integer is in range.

**Sample Input/Output**

Enter an integer: 6

2^0 = 1

2^1 = 2

2^2 = 4

2^3 = 8

2^4 = 16

2^5 = 32

2^6 = 64

1. **Question Two**

The CodeWars judges and scorekeepers need help printing the team names above each column on our score sheet. But we want them to look nice, so they need to be printed vertically. All we have to use is a list of team names. Please write us a program that will take a list of team names, and print them vertically.

**Program Input**

Your program will prompt for a team name. Continue prompting for names until you receive “END”. Unlike the real CodeWars, this one will be limited to 16 teams maximum.

**Program Output**

Your program will output to the screen each team’s name, in the order it was entered, arranged vertically such that it is read from top to bottom. There should be two spaces between each name as printed.

**Sample Input/Output**

Enter team name: Aardvarks

Enter team name: Emus

Enter team name: Road Runners

Enter team name: Coyotes

Enter team name: END

A E R C

a m o o

r u a y

d s d o

v t

a R e

r u s

k n

s n

e

r

s

1. **Question Three**

The combination of n things taken m at a time is a value often used in mathematics and business. With this program you will read input values, n and m, and output the number of different combinations of m things that can be formed from a group containing n members. This can be calculated by computing the factorial of n and dividing it by the factorial of m and then further dividing the result by the factorial of (n – m). The value, n-factorial is computed by multiplying n by n – 1, then multiplying the result by n – 2, and so forth, until the final result is multiplied by 1. n-factorial is sometimes written as n! Thus, 5-factorial (5!) is calculated in the following fashion: 5 x (5 – 1) x (5 – 2) x (5 – 3) x (5 – 4) = 5 x 4 x 3 x 2 x 1 = 120. 3! is calculated similarly as 3 x (3 – 1) x (3 – 2) = 3 x 2 x 1 = 6. And (5 – 3)!, otherwise known as 2!, is calculated as 2 x (2 – 1) = 2 x 1 = 2. Thus, the number of different combinations of 5 things taken 3 at a time is calculated as follows:

n! / [m! x (n – m)!]

= 5! / [3! x (5 – 3)!]

= 120 / [6 x2]

= 120 / 12

= 10.

Your program must calculate this value for any values of n and m. You must output the result as an integer.

**Sample Input/Output**

Enter n and m: 10 5

252

1. **Question Four**

You’re working for a growing e-commerce web site, which has become a popular target for thieves. The thieves gain access to customers’ accounts by guessing passwords, which are all too often trivial (such as “secret”, “password”, and “1234”). If your customers used better passwords, your company would have less trouble with fraudulent purchases.

You’ve been tasked with creating a password analyzer, which will inform a customer about the relative strength of their choice of password. A “strong” password is one that is hard to guess – either by sheer length, or by using a combination of letters, numbers and symbols. For your assignment, a strong password has all of the following characteristics:

* Is at least 8 characters long (example: “spookyfish”)
* Includes both upper and lower case letters (example: “sPookyFISH”)
* Includes letters and at least one number or symbol (examples: “sPookyFISH3” or “$PookyFI3H”)

A “good” password has two of these characteristics, an “acceptable” one has only one. A password that doesn’t meet any of these would be considered “weak”. Write a program that will analyze a given password and output the strength rating of the chosen password.

**Program Input**

Prompt for the password to be analyzed. You should allow for a maximum password length of 30 characters. No spaces are allowed.

**Program Output**

Your program will output to the screen the relative strength of the password typed, using the characteristics listed above.

**Sample Input/Output**

Enter your password: lizard

This password is WEAK

Enter your password: aardvark

This password is ACCEPTABLE

Enter your password: Aardvark

This password is GOOD

Enter your password: Aardvark77

This password is STRONG

1. **Question Five**

Write a program to create a number spiral of size n (n rows by n columns), where n is any odd number between 1 and 19. The spiral starts in the middle with number zero, moves outward to the right and spirals counter-clockwise.

**Program Input**

Prompt for the size of the number spiral. If the number is outside of the range of 1-19, or is not an odd number, your program should output an error message and exit.

**Program Output**

The program must output to the screen the number spiral, with the numbers aligned vertically with the least significant digits. Ensure that your largest spiral will still fit without the text wrapping on an 80 column screen.

**Sample Input/Output**

Enter the size of the number spiral: 3

4 3 2

5 0 1

6 7 8

Enter the size of the number spiral: 5

16 15 14 13 12

17 4 3 2 11

18 5 0 1 10

19 6 7 8 9

20 21 22 23 24

Enter the size of the number spiral: 7

36 35 34 33 32 31 30

37 16 15 14 13 12 29

38 17 4 3 2 11 28

39 18 5 0 1 10 27

40 19 6 7 8 9 26

41 20 21 22 23 24 25

42 43 44 45 46 47 48

1. **Extra Credit**

A musical interval is defined as the distance between one note on a scale and another note on the same scale. We can represent notes on a piano keyboard:

Beneath the keyboard graphic are the names of the notes, above the keyboard graphic are indications of *half steps* and *whole steps*. This keyboard graphic shows only some of the available keys. Notes drop to lower tones as you move to the left and raise to higher tones as you move to the right. The notes repeat themselves, A through G, with additional notes – called *sharps* (black keys) – in between the other notes (white keys). You move a half step when you move from one key to the immediately adjacent key, and you move a whole step when you move from one key to a key that is two keys away. Also note, for example, that while E and F (or B and C) are adjacent white keys, F and G (or G and A, or C and D, or A and B) are not.

This problem works with major scales. A major scale is that series of eight notes (called an octave) which begins and ends with the same note name and progresses from one note to the next via the following progression: 1) Initial note (e.g. A); 2) Whole step higher (B); 3) Whole step (C#); 4) Half-step (D); 5) Whole step (E); 6) Whole step (F#); 7) Whole step (G#); 8) Half step (A, again). You can begin with any key and follow this pattern to derive any of the major scales. Any other major scale can be derived by following this same pattern, beginning with a base note. For example: Initial note: G#; whole step: A#; whole step: C; half step: C#; whole step: D#; whole step: F; whole step: G; half step: G#.

Now, a specific *interval* represents the space between one note and a referenced note on the same scale, counting from the initial note as one (1) and moving along the scale to the named ordinal. Thus, a *second* interval up along the A major scale from B is C# (B=1, C#=2). A *seventh* interval down along the A major scale from G# is A (move left to go down, G#=1, F#=2, E=3, D=4, C#=5, B=6, A=7).

Your program will read a single scale name and a series of intervals (maximum of 15). The scale name can be a single letter (e.g. A) or a letter plus a sharp sign (e.g. A#). The program must walk the input intervals and output each note described along the input scale, starting with the input note and continuing to the final note. Begin each interval at the note reached by moving the previous interval. Intervals to the right are represented in the input by a plus (+) sign. Intervals to the left are represented by a minus (-) sign.

**Sample Input/Output**

Enter intervals: A+2+5-6+9+10-11+3+3

A B F# A B D A C# E

Enter intervals: G#-3+6-13+15+2-7+2-5+6-2+7

G# F C# F F G G# A# D# C A# G#

Enter intervals: B-3+2-7

B G# A# B