Andrew's Leap 2003
Introductory Programming

Project list, July 15

Here are the next round of projects. You will work in groups on one or more of these projects. My hope is that you will be able to complete them with minimal help from me.

1. Write a program that counts the number of occurrences of each letter of the alphabet in a string. For example, if the user were to input, "andrews leap", the program would tell the user that there are 2 As, 1 D, 2 Es, 1 L, 1 N, 1 P, 1 R, 1 S, and 1 W. For simplicity, the user's input should be in lowercase and without punctuation. You will find it helpful to keep the following tidbits in mind:

Use a method for counting the number of occurrences of a character in a string like the counter method we used yesterday for counting the number of occurrences of a number in an array. It should take a string and a character (instead of a double[] and a number) as its input, and it should return the number of occurrences of that character. Remember: we use brackets to specify array indices, but we use the `s.charAt(n)` method to find the *n*th character in the string `s`.

Use a `for` loop to go through the characters a to z. To advance a character one letter down the alphabet (like from a to b), you can use the `++` operator on the character. (Ultimately, characters only stand for numbers, so this is legal in Java. You can look at a table of characters and their corresponding numbers at <http://www.asciitable.com>. The "Char" column lists characters and the "Dec" column lists their corresponding decimal numbers. This system of characters and numbers is called "ASCII Code," pronounced "ASK-ee.")

Store your values for each character in an integer array of size 26, and report your values at the end of the program by using a `for` loop to print every value of the array. You may modify our `printArray()` method from yesterday for this purpose.

2. Write a statistics program. The program should allow your user to input a set of data (doubles), and your program should tell the user the arithmetic mean and geometric mean. First, ask the user how many numbers there are in the data set. Create an array of that size, and use a `for` loop to allow your user to input the numbers into the array.

Finding the arithmetic mean – Use a `for` loop to total the numbers in the array, and then divide by the number of elements.

Finding the geometric mean – The geometric mean of a set of numbers is the *n*th root of the product of the *n* numbers. For example, the geometric mean of 3, 3, and 24 is the cube root of 216. Use a `for` loop to find the product of all the numbers in the array, and use `Math.pow()` to take the *n*th root.

3. Write a program that accepts a year written as a four-digit Arabic (ordinary) numeral and outputs the year written in Roman numerals. The Roman numerals are: I for 1, V for 5, X for 10, L for 50, C for 100, D for 500, and M for 1000. Ignore the fact that some Roman numerals are sometimes created through subtraction (in other words, represent the number 4 as IIII, not as IV). (Modified from *Problem Solving with C++*, by Walter Savitch)

4. Write a program that asks the user for a value *n* and calculates the *n*th Fibonacci number. Use the following formula for the *n*th Fibonacci number:

If *n* is even:

$$F_n = {}_{n}C_0 + {}_{n-1}C_1 + \ldots + {}_{n/2}C_{n/2}$$

If *n* is odd:

$$F_n = {}_{n}C_0 + {}_{n-1}C_1 + \ldots + {}_{(n+1)/2}C_{(n-1)/2}$$

Examples: $F_8 = {}_{8}C_0 + {}_{7}C_1 + {}_{6}C_2 + {}_{5}C_3 + {}_{4}C_4$

$F_5 = {}_{5}C_0 + {}_{4}C_1 + {}_{3}C_2$

The notation ${}_{n}C_r$ ("*n* combination *r*" or "*n* choose *r*") denotes the number of ways that *r* objects can be chosen from a set of *n* objects. The formula for ${}_{n}C_r$ is:

${}_{n}C_r = n! / (r!(n-r)!)$

You will need to write a method that finds ${}_{n}C_r$ (its parameters should be two integers, *n* and *r*, and it should return an integer). This method will need to call another method that finds the factorial of a number (its parameter should be an integer, and it should also return an integer).

Finally, you will need to use a `while` loop to sum the series. In both the even and odd cases, the *n*-value always decreases by 1 for each subsequent term, and the *r*-value always increases by 1 for each subsequent term. The adding continues for as long as *n* is greater than or equal to *r* in both the even and odd cases, so your program will not have to use different algorithms for the different cases.