Philadelphia Classic

The Dining Philosophers
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
`dp-exec@googlegroups.com`
`http://penndp.blogspot.com`

09 February 2008

Penn
UNIVERSITY *of* PENNSYLVANIA

1. Sudoku Puzzle Validator

The *Sudoku* craze has swept the nation. *Sudoku* puzzles regularly appear in magazines, newspapers and video games. According to Wikipedia: "*Sudoku* is a logic-based number placement puzzle. The objective is to fill a 9×9 grid so that each column, each row, and each of the nine 3×3 boxes (also called blocks or regions) contains the digits from 1 to 9, only one time each (that is, exclusively)."

For a *Sudoku* board to be valid, the digits 1 to 9 must each appear **exactly once** in each row, column and 3×3 square in the *Sudoku* board. The following are examples of valid and invalid boards:

Valid Board

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

Invalid Board

9	8	7	6	5	4	3	2	1
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

For this problem, you must be able to compute whether or not a given board is valid. Your program should accept 9 "rows" of input, with each row being separated from the next by a space. After the 9 rows have been input and the enter key pressed, your program should determine if the board is valid. If the board is valid, your program should print **VALID**. If the board is not valid, your program should print **NOT VALID**.

Examples

Input: `123456789 456789123 789123456 234567891 567891234 891234567 345678912 678912345 912345678`
Output: `VALID`

Input: `987654321 456789123 789123456 234567891 567891234 891234567 345678912 678912345 912345678`
Output: `NOT VALID`

2. Text Message Encryption

What would you do if you wanted to text a message to one of your friends that could only be understood by them? You might consider using a form of encryption. In cryptography, there are several forms of encryption including symmetric, asymmetric and public key encryption.

One of the oldest and most basic methods of encryption is called a **substitution cipher** since each character in an unencrypted message (called *plaintext*) can be substituted for another. The resulting encrypted message is called *ciphertext*. An example of this would be the following: to encode a plaintext message, the 26 letters of the alphabet could be transposed to define a *key* that maps (or translates) the message. A plaintext message of `hello world` using an encryption key of `a=b, b=c, c=d, ..., z=a` would encode to give the ciphertext message of `ifmmp xpsme`. To unencrypt, the key would be applied in reverse.

You must build a message encryption / decryption utility that accepts a single letter to designate the function (either E for Encrypt or D for Decrypt) followed by a colon (:) and a message that will be encrypted or decrypted. You must use the encryption key **a=f, b=g, . . . , z=e** for your program. After the enter key is pressed, your program should either encrypt or decrypt the given message and print it out to the screen. Assume that all messages will only be in lower case without any punctuation. Spaces should not be encoded / decoded.

Examples

Input: `E:this is a secret message`
Output: `ymnx nx f xjhwjy rjxxflj`

Input: `E:java is cool`
Output: `ofaf nx httq`

Input: `D:z hfsy wjfi ymnx`
Output: `u cant read this`

Input: `D:umnqfijqumnf hqfxxnh`
Output: `philadelphia classic`

3. Greatest Common Divisor

The greatest common divisor (GCD) of two integers is the largest integer such that it divides both of them without a remainder. For example, the GCD of 21 and 35 is 7, as no larger integer divdes both 21 and 35 without remainder. If the GCD of two numbers is 1, they are called coprime. Your job is to write a program that finds the GCD of two numbers. Do not assume they are both positive. The GCD of anything with 0 is just itself.

You will receive a string of integers as input. If they are coprime, print COPRIME.

Examples

Input: `21 35`
Output: `7`

Input: `83 22`
Output: `COPRIME`

Input: `0 100`
Output: `0`

4. Polynomial Multiplication

As you may know, you can multply two polynomials together, just like you can integers. You do this by multplying together every term in the first polynomial with every term in the second polynomial and grouping like powers of variables. For example:

$$(1+x)*(2+x+x^2) = 1(2+x+x^2)+x(2+x+x^2) = (2+x+x^2)+(2x+x^2+x^3) = 2+2x+2x^2+x^3$$

Your task is to write a program that can do this.

You will be given input in the form of the coefficients of the polynomial separated by spaces with the two polynomials separated by a semicolon. So $(1+2x+3x^2)*(1+2x)$ would be represented as 1 2 3; 1 2. Your output should consist of the coefficients of the result.

Examples

Input: `1 1; 2 1 1`
Output: `2 2 2 1`

Input: `1 2 3 4; 2 3`
Output: `2 7 12 17 12`

5. Dueling Histograms: Battle of the Sexes

Do men or women live longer? Who scored higher on that last test? Who is really smarter? These kinds of questions often invite heated arguments about men and women. We won't comment on that debate, but we will introduce you to a graphing tool that can assist in argument.

This type of information is often presented with a double-sided histogram. The data is split on some kind of demographic (gender, for example) and a histogram (bar graph) is drawn for each set of the data. The two categories are then aligned so that it is easy to compare differences in the groups. For example, this kind of chart can be used to show scores on an exam from two different groups.

Your task is to write a program that will generate the two-sided histogram for two sets of data. You must take a series of numbers from 0 to 99 and output a formatted histogram of scores. The number of Xs next to a value is the number of scores that fell in that range. For example, two Xs next to 50 means that there were two scores between 50 and 59 (inclusive). You must generate a graph with a single set of range labels but two sets of Xs, one for each set of data. See the following examples below for output format.

You will receive as input two lists of numbers, separated by a semicolon. The first list is the left side of the chart, and the second list is the right side. Your program must output a chart of the data with spacing such that the numeric labels line up. You should still print rows even if they will be empty. You do not need to match our formatting exactly, but your results should be consistent and make sense. (*Hint:* Consider each row of Xs as having constant length)

Examples are on the following page.

Examples

Input: `34 23 27 93 97 92 87 86 55 43 42 43 47 49 51 99; 31 20 28 94 99 92 80 81 50 42 44 46 41 48 57 90`
Output:

```
 XXXX 9 XXXX
   XX 8 XX
      7
      6
   XX 5 XX
XXXXX 4 XXXXX
    X 3 X
   XX 2 XX
      1
```

Input: `15 25 96 27; 35 39 46 80`
Output:

```
 X 9
   8 X
   7
   6
   5
   4 X
   3 XX
XX 2
 X 1
```

6. Prime Numbers

A prime number is defined as a number with exactly two factors: 1 and itself (thus, 1 is not a prime number as it does not have two factors). 3, for example, is a prime number. 2 is the only even prime number. 6 is not a prime number: it is *composite*. Given a lower bound and an upper bound as input, your task is to return every prime number in between. (If the bounds are prime, do not include them, and you can assume that there will be at least one prime number to return.)

Examples

Input: `2 9`
Output: `3 5 7`

Input: `13 27`
Output: `17 19 23`

7. “The Force Is Strong With This One”

Researchers in the Jedi Archive on Coruscant have recently discovered a mystical connection between a Jedi’s strength with the Force and how much “distance” there is between his name and that of Anakin Skywalker, the most powerful Jedi who has ever lived (recall that his midichlorian count was over 20,000). They measure this distance by comparing the each of the letters of a Jedi’s last name with the respective letters of the name “Skywalker.”

If a pair of letters is identical, then the distance is 0; otherwise, the distance is equal to the number of letters apart they are in the alphabet (i.e., ”A” has a distance of 25 from ”Z,” and ”B” has a distance of 2 from ”D”). The distance between a blank character and any letter is 0, so you should only compare letters up to the end of the shorter name.

For some reason, it seems that differences in letters occurring early in the comparison are more significant than letters occurring later on. The function to calculate the distance between the two names is thus weighted to reflect this and is as follows:

Distance = (distance between 1st letters)
+ 0.875 * (distance between 2nd letters)
+ 0.875 * 0.875 * (distance between 3rd letters)
+ 0.875 * 0.875 * 0.875 * (distance between 4th letters)
+ . . .

Your program should take a Jedi surname as an input, evaluate the distance between it and “Skywalker,” and return the result. The Jedi Council is counting on these results to identify potential Jedi for recruitment, and the researchers are getting tired of doing this all by hand, so get going!

Examples

Input: `Yoda`
Output: `40.31640625`

Input: `Windu`
Output: `38.6240234375`

8. Reverse Polish Notation

(1+2)*(3+4). This is how we are used to writing expressions, and it is called *infix notation* because the operator is inside of the operands–that is, in 1+2, the + operator is between the operands 1 and 2. This is not the only way to represent expressions, however. There is also *postfix notation*, also known as *Reverse Polish Notation* (RPN), in which the operator comes *after* the operands. Thus, 1+2 becomes 1 2 +. Interestingly, RPN never needs parentheses, allowing us to write the equation above as 1 2 + 3 4 + *.

You must write a program which accepts an array of string inputs–each string will be either an integer or an operators–processes it according to the rules of RPN, and outputs the calculated result. The only operators you have to deal with are +, -, *, and ˆ–addition, subtraction, multiplication, and exponentiation–and all numbers will be integers. You may find the Integer.parseInt(String) method useful here. We won't do anything tricky–the test cases will be much like the example above–so you can rely on that structure of input.

To do this problem, each time your program iterates through the input array from the beginning it should identify the first operator and then operate on the two operands before it. For example, on input 1 2 + 3 4 + *, when you reach the first + you add 1 and 2 to get 3 and create the new string 3 3 4 + *. Repeating the procedure on this string yields 3 7 * and finally 21, the answer.

Keep in mind that an operator may not always have two numbers in front of it. The * above had a + in front of it because earlier operations needed to be resolved. Your program should operate only when there are two numbers immediately preceding the operator in question. (*Hint:* Think of some data structures you may have studied in class.)

Examples

Input: `1 2 + 2 3 * ^`
Output: `729`

Input: `1 3 4 2 3 5 6 1 3 ^ ^ * - * + + -`
Output: `48`