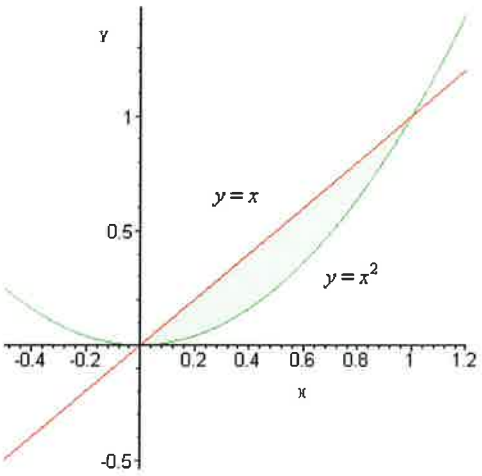
HARDING
UNIVERSITY

Computer Science Department
Student Programming Contest

February 21, 2006

Problem Set

Problem A: Quadratic Slice



The figure above shows the graphs of the two equations: $y = x$ and $y = x^2$. For this problem, write a program that will determine if a given point lies within the shaded region.

Input

Your program should prompt for and read the x-coordinate and y-coordinate for a point. Each coordinate is a floating-point number (may be negative, zero, or positive). For each point, report whether or not the corresponding point lies within the shaded region. Continue in this fashion until the point (0.0, 0.0) is entered. (See sample run below.) Points that lie on either curve (either $y = x$ or $y = x^2$) are NOT considered to lie within the region.

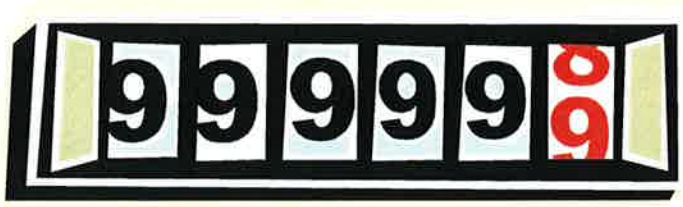
Output

For each point, your program should report either '`YES`' or '`NO`'.

Example Run

```
X-Coordinate? 0.2
Y-Coordiante? 0.1
YES
X-Coordinate? 0.4
Y-Coordinate? 0.2
YES
X-Coordinate? 0.6
Y-Coordiante? 0.2
NO
X-Coordinate? 0.354
Y-Coordinate? 0.354
NO
X-Coordinate? 0.0
Y-Coordinate? -0.16
NO
X-Coordinate? 0.0
Y-Coordinate? 0.0
```

Problem B: Strange Odometer



Problem Statement

The odometer on Jimmy's car is broken. Occasionally, after he comes to a complete stop and once he starts up again, the odometer will start counting backwards. Jimmy has noticed that the odometer will only switch which way it's counting (up or down) after he has come to a complete stop. Write a program that, when given the exact value of Jimmy's odometer at every stop will tell Jimmy how many miles he drove that day. The odometer will never go below zero or above 999999.

Input and Output

More than one test case will be entered and each records the odometer readings in Jimmy's car each time he stopped during one day. Each test case is entered on a separate line and each line of the input will contain at least two and at most ten different integers separated by spaces. The first integer represents the odometer reading at the beginning of the day, before he leaves his driveway. The following integers represent the odometer reading each time he stops for any reason. The end of the line is marked by a negative value.

After the last test case, the end of the input is indicated by a line that contains a single negative value..

A separate line of output should be written for each test case which should simply be the total distance Jimmy drove that day.

Sample Input

```
10 50 12 98 100 -1
97 45 12 -1
100 150 100 -1
-1
```

Sample Output

```
166
85
100
```

Problem C: Word Statistics

Problem Statistics

Jill and her friends are very competitive. Recently Jill and some of her friends have been assigned several research papers. They decide to have a contest to see who can write the most words and see who can have the largest word length average (that is how many letters on average are in their words). While most modern day word processors can do this, Jill doesn't own one. Instead she uses "`Notepad`" to write her papers. Consequently, she will need a program that can report these statistics. In finding the average word length, punctuation at the beginning or end of a word (periods, commas, colon, semicolon, quotation marks, etc…) does not count; however, punctuation within a word does (such as an apostrophe or hyphen). Punctuation characters are defined to be any character other than letters, digits, or whitespace.

Numbers are not included in the average or word count and there are no words that combine digits with letters. Round the average word length to the nearest integer.

Input and Output

All of Jill's papers have been combined into one text file named `word.txt`. (In order to test your program you will need to create your own version of this file.) This input file may contain more than one research paper, each paper may contain any number of lines, and each line may be of arbitrary length. Between each paper there will be a line that contains a single asterisk, '`*`'. After the last paper, the end of the input file will be marked with a line that contains a single exclamation point, '`!`'. Output should follow the example shown below. (Yes, I know, these examples are strange research papers.)

Sample Input File (`word.txt`)

```
"Hey there Bob, how are you doing? I hope well."
We're doing fine here; however, Jill's house is over-crowded.
See you soon!
*
He is a well-respected man and is 10,000 years old.
!
```

Sample Output

Paper 1:
Word Count: 22
Average Word Length: 4

Paper 2:
Word Count: 9
Average Word Length: 4

Problem D: Homework Schedule

Problem Statement

Steve always has too much homework to do every week. So in order for Steve to do his best he needs to know exactly what homework assignments he should do and how long he should work on them to get the highest possible average. All projects are evenly weighted and effort is always reflected in the grade (that is he would make a 50 on a paper that he spent half the time that was required to finish it). There will never be more than 10 projects in a week.

Input and Output

There will be 1 or more test cases. The first line in a test case will contain 2 positive integers. The first integer indicates the number of hours Steve has available that week to spend on homework. The 2nd integer, N, is the number of assignments he has due that week ($1 \leq N \leq 10$). The next N lines will each contain the time required to complete each assignment.

After each test case, the next line will contain 2 more integers. If these two are positive integers, then what follows is another test case using the same format as described in the previous paragraph. However, if the 2 integers are both zero, the end of the input has been reached.

The output should report what is Steve's highest possible average for each week. All averages should be rounded and show two decimal places.

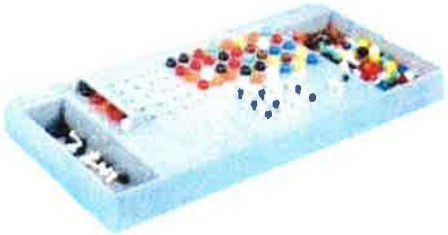
Example Input

```
10 5
1
12
3
6
8
20 3
15
15
15
0 0
```

Example Output

```
Week 1: 60.00
Week 2: 44.44
```

Problem E: Mastermind



Mastermind is a game that has been popular for many years. The game is played as follows: the "master" creates a special code that the player must guess. This code is formed using either colors or numbers from a given set. The player is then asked to guess the code. After each guess, the player is told two things: 1) How many items in the guessed code are in the correct location and 2) How many items in the guessed code are in the actual code but not in the correct location. The player must use this information to deduce the final code.

Example:
(Note: each '+' denotes a digit in the correct position, each '-' denotes a digit that is in the code but not in the correct position, and each '.' denotes a digit that is not in the code.)

Guess	Result	
1234	--..	
5678	+...	
1278	....	
3456	---.	
5555	....	
6666	++..	
3434	--..	
3466	+---	
3664	++--	
4663	++++	← This is the correct answer

Problem: given a series of guesses and results, you must determine the correct code.

Input: The input will contain 1 or more test cases stored in a text file named `mastermind.txt`. (For testing purposed you will need to create your own version of this text file.) For each test case, the first line will contain 2 numbers separated by a single space, m and n ($2 \leq m \leq 5$ and $1 \leq n \leq 20$), where m is the number of digits in the code and n is the number of guesses of the code that follow. Following this first line will be n pairs of lines that contain guess/result combinations. Each guess consists of m non-zero digits. Each result will be written as a string of m characters formatted in the following order:
1) one '+' for each digit in the correct position
2) one '-' for each digit in the code but not in the correct position
3) one '.' for each digit not in the code

The line after the last guess/result pair of lines will contain the m and n values for the next game. The end of the input will be indicated by m and $n = 0$.

(Note: there will always be enough information to solve the code.)

Output: For each test case, the output must be one line formatted as follows: "Game #g:", followed by one space and the correct code where g is the particular game number.

Example Input:

```
2 3
11
+.
22
+.
12
--
5 2
12345
.....
67876
.....
4 9
1234
--..
5678
+...
1278
....
3456
---.
5555
....
6666
++..
3434
--..
3466
+---
3664
++--
0 0
```

Example Output:

```
Game #1: 21
Game #2: 99999
Game #3: 4663
```

Problem F: LaZ Emulator

You are to write a machine emulator for the LaZ processor. The processor is entirely linear in form, lacking any form of branching or subroutines.

The LaZ processor has 6 registers labeled `A`, `B`, `C`, `D`, `E` and `H`. Each of these register may contain a single 32-bit integer (ie. equivalent to an `int` or `long` in C++). Each register is initialized with a default value of `0`.

The instruction set for the LaZ processor consists of the following 10 instructions (`<reg1>` and `<reg2>` may be is any one of the 6 registers, and `<lit>` is any literal integer):

LaZ instruction	Textual Description	Formula
`mov <reg1>, <lit>`	Copies a literal value to a register	`<reg1> = <lit>`
`mov <reg1>, <reg2>`	Copies a register to another register	`<reg1> = <reg2>`
`swp <reg1>, <reg2>`	Swaps two registers	`<reg1> <=> <reg2>`
`add <reg1>, <reg2>`	Adds a register to another register	`<reg1> += <reg2>`
`sub <reg1>, <reg2>`	Subtracts a register from a register	`<reg1> -= <reg2>`
`mul <reg1>, <reg2>`	Multiplies 2 registers together	`<reg1> *= <reg2>`
`div <reg1>, <reg2>`	Divides 2 registers	`<reg1> /= <reg2>`
`mod <reg1>, <reg2>`	Takes the modulus of the two registers	`<reg1> %= <reg2>`
`pnt <reg1>`	Prints out a register to the console	`cout << <reg1>`
`stop`	Stop execution	

The instructions will appear exactly as above, with a space between the instruction and the first register (except for the '`stop`' instruction); and a space between the comma and the second register, if any. All instructions will appear as above.

The division and modulus operator should report an error when trying to divide by 0; however, if this occurs, the values in the registers should be left unchanged.

Example input

```
mov a, 5
mov b, 6
add a, b
pnt a
mov c, 7
sub a, c
pnt a
div a, d
pnt a
stop
```

Example output

```
11
4
ILLEGAL DIVIDE BY 0
4
```