

# Problem Set

of the first ACM Arab and African Regional Programming  
Contest

**(AARC'98)**

<i>Problem #</i>	<i>Problem Title</i>	<i>Suggested Solution</i>
1	<a href="#">Parkinson</a>	<a href="#">parkinson.c</a>
2	<a href="#">Ellie and The Factorial</a>	<a href="#">ellie.c</a>
3	<a href="#">Sofian Numbers</a>	<a href="#">sofian.c</a>
4	<a href="#">Mapping the Swaps</a>	<a href="#">map.c</a>
5	<a href="#">Find The Base</a>	<a href="#">base.c</a>
6	<a href="#">Lines and Points</a>	<a href="#">line.c</a>
7	<a href="#">Perfect Cubes</a>	<a href="#">cubes.c</a>

# Parkinson

Source file : parkinson.{c|cpp}

Input file: parkinson.in

Output file: parkinson.out

In medical diagnosis, a number of problems occur concerning the correct assignment of symptoms to a specific disease.

In a clinical study, some neurologists specialized in the Parkinson disease assigned a letter value (A,B,C..) for each one of 9 Parkinson symptoms (swinging arms, speech, independence ...) for different patients. Each patient record is a sequence of 9 letter values. '\$'s are included at various places in the records to make them easier to read, but have no other significance.

The sample input and expected output shown below illustrate many valid, and some invalid, record formats.

An algorithm is set to check the development of the disease for each patient based on the patient's record (The collection of the 9 values).

3 sets of 3 values represent some inter-related symptoms, and are represented as the 3 corners of a triangle. The first letter value is the first corner of the first triangle, the second value is the first corner of the second one ...

The length of each of triangle edge is then computed according to a pre-defined schema:

The unit step between A and B is 1

between B and C is 2

between C and D is 3

between D and E is 4

...

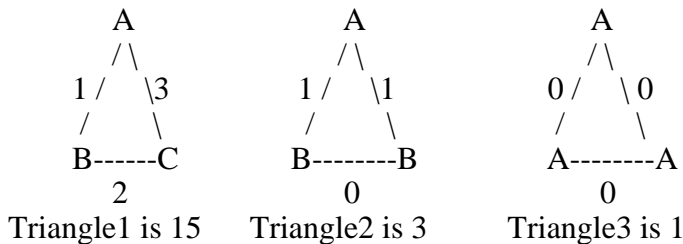
between A and G is 21

Each of the triangle's edge is then used to compute the value of the triangle.

Value of the triangle = (edge1)<sup>2</sup> + (edge2)<sup>2</sup> + (edge3)<sup>2</sup> + 1

Then, the 3 values of the symptom groups are then summed up. The patient is considered to have the Parkinson disease if the final value is a prime number. Otherwise, the patient is in good health.

To better understand the procedure, consider the correct patient's record A\$AA\$BB\$AC\$BA. First look at the symptom groups, and the calculations performed:



The final value of this record is 19, which is a prime number; thus showing that the patient has the Parkinson disease.

## Input file:

The sample input file contains a single data record per line, possibly preceded and/or followed by spaces. No line may contain more than 70 characters, but the data record might contain illegal characters, and more or fewer than 9 letter values. The input file is terminated by an end of file.

## Output file:

The output file should include the display of the data record and a statement of invalid record in case it is so. If the record is legal, the output should specify if the patient is in good health, or has the Parkinson disease.

### **Sample Input:**

```
ABCD
ABCDEFGH
A$$$BCD$HJUY$Q
AFW#GFDGFD
ABCD$EFG$$KP
AAABBACBA
```

### **Sample Output:**

```
The data sequence ABCD is invalid.
The data sequence ABCDEFGH is invalid.
The data sequence A$$$BCD$HJUY$Q signals that patient has Parkinson disease.
The data sequence AFW#GFDGFD is invalid.
The data sequence ABCD$EFG$$KP signals that patient is in good health.
The data sequence AAABBACBA signals that patient has Parkinson disease.
```

# Ellie and The Factorial!

Source file : `ellie.{c|cpp}`

Input file: `ellie.in`

Output file: `ellie.out`

In a software company, Mrs. Ellie has to schedule tasks on her ten machines.

As she is a great fan of mathematics, she starts looking for a method to assign tasks and working days for each machine.

First, she tries to find out a relation between the number of tasks she has to schedule on a specific day and the corresponding day.

This is what she notices:

(Mrs. Ellie hates leap years and she never works that year, so assume 365 days)

Day Day Number Number of Tasks

1 January 1 1

3 January 3 1

4 January 4 2

5 January 5 3

6 January 6 3

...

...

27 January 27 29

...

...

24 March 83 125

...

...

1 August 213 406

...

...

3 November 307 632

...

...

31 December 365 779

Mrs. Ellie is clever enough to notice that the number of tasks for a specific day number  $n$  is SIMPLY the number of digits of  $n!$ . So this is what she decides:

She enumerates her machines from 0 to 9, then each day, she computes the factorial of  $n$  (the day number of that day). Since the number of digits of  $n!$  corresponds exactly to the number of tasks for that day, she has just to assign to a machine  $i$ ,  $T_i$  tasks where  $T_i$  is the frequency of the digit  $i$  in the number  $n!$ .

## Example:

Day Number 32(1st of February):

$32! = 263130836933693530167218012160000000$  (36 digits)

So, on the 1st of February, there will be 36 tasks and the machines-tasks schedule will be:

Machine 0 has 10 tasks

Machine 1 has 5 tasks

Machine 2 has 3 tasks

Machine 3 has 7 tasks

Machine 4 has no task

Machine 5 has 1 task

Machine 6 has 5 tasks

Machine 7 has 1 task

Machine 8 has 2 tasks

Machine 9 has 2 tasks

### **Program:**

Your program should read the day number from the input file, convert it to the corresponding day, then compute the machines-tasks schedule for that day as described in the sample output.

The input file is a set of day numbers(1-365) each on a different line and the end of file is a line containing the null number 0.

The output file should be a set of machines-tasks schedule for each day number n.

Each set is composed of a first line consisting of the day and the month corresponding to n, then 10 lines with the form of :

Machine  $i$  :  $T_i$  (one space between Machine and  $i$ ,  $i$  and :, : and  $T_i$ )

Where  $i$  goes from 0 to 9 and  $T_i$  is the frequency of the digit  $i$  in the number  $n!$

Each line should end with the character '\n'.

### **Sample Input File:**

```
8
32
364
0
```

### **Sample Output File:**

```
8 January
Machine 0 : 2
Machine 1 : 0
Machine 2 : 1
Machine 3 : 1
Machine 4 : 1
Machine 5 : 0
Machine 6 : 0
Machine 7 : 0
Machine 8 : 0
Machine 9 : 0
1 February
Machine 0 : 10
Machine 1 : 5
Machine 2 : 3
Machine 3 : 7
Machine 4 : 0
Machine 5 : 1
Machine 6 : 5
Machine 7 : 1
Machine 8 : 2
Machine 9 : 2
30 December
Machine 0 : 158
Machine 1 : 69
Machine 2 : 61
Machine 3 : 74
Machine 4 : 74
Machine 5 : 74
Machine 6 : 59
Machine 7 : 70
Machine 8 : 69
Machine 9 : 68
```



# Sofian Numbers

Source file: `sofian.{c|cpp}`

Input file: `sofian.in`

Output file: `sofian.out`

Sofia loves change. She gets bored easily especially when doing routine job. Being an accountant, she deals mostly (if not always) with calculations and numbers.

In the past, she worked with Arab numbers (1,2,..) but she wanted some change, so she decided to work in an Arab country where they use Indian numbers.

After a short time, she felt the need for novelty, so she opted for the Roman numbers.

After exhausting all these numbering systems, Sofia had a great idea, i. e : create her own numbering system (called Sofian numbers), so that she can change it whenever she feels the need for it!

The Sofian representation consists of using the first five letters of the alphabet to represent any given number between 1 and 100. A for 1, B for 5 , C for 10, D for 50 and E for 100.

To represent other values, a combination of these symbols is used. For example,

1	A	10	C	20	CC
2	AA	11	CA	30	CCC
3	AAA	12	CAA	40	AD
4	AB	13	CAAA	50	D
5	B	14	CAB	60	DA
6	BA	15	CB	70	DAA
7	BAA	16	CBA	80	DAAA
8	BAAA	17	CBAA	90	CE
9	AC	18	CBAAA	100	E
		19	CAC		

(Hint: Sofia was inspired by roman numbers).

You will help Sofia in her daily work by providing a solution that reads from a file an arithmetic expression (one at a time), evaluates it and provides the result.

Sofia is not interested in the result but in the number of A's, B's, C's, D's and E's that are needed to represent numbers from 1 to result. An example is:

Result = 2 so we need to represent 1 and 2 so we only need 2 A's .

Result = 20, we need to represent 1, 2,...,20 so we need 28 A's, 10 B's, 14 C's, 0 D's and 0 E's .

## Input:

The first line in the file holds the number of  $n$  expressions.

The following lines each contain one arithmetic expression of the type:

operand op operand such that op is \* or + or – and operand is an integer ranging between 1

and 100.

**Ouput:**

For each arithmetic expression in the file, the output file should contain one line containing the result (in decimal form) followed by the number of characters of each type required.

Assume that the result is always between 1 and 100.

**Example Input:**

4  
1 + 0  
2\*1  
30 - 10  
90 + 9

**Expected Output:**

1: 1 A, 0 B, 0 C, 0 D, 0 E  
2: 3 A, 0 B, 0 C, 0 D, 0 E  
20: 28 A, 10 B, 14 C, 0 D, 0 E  
99: 140 A, 50 B, 150 C, 50 D, 10 E

# Mapping the Swaps

Source file : `map.{c|cpp}`

Input file: `map.in`

Output file: `map.out`

Sorting an array can be done by swapping certain pairs of adjacent entries in the array. This is the fundamental technique used in the well-known bubble sort.

If we list the identities of the pairs to be swapped, in the sequence they are to be swapped, we obtain what might be called a swap map.

For example, suppose we wish to sort the array A whose elements are 3, 2, and 1 in that order.

If the subscripts for this array are 1, 2, and 3, sorting the array can be accomplished by swapping A2 and A3, then swapping A1 and A2, and finally swapping A2 and A3.

If a pair is identified in a swap map by indicating the subscript of the first element of the pair to be swapped, then this sorting process would be characterized with the swap map 2 1 2.

It is instructive to note that there may be many ways in which swapping of adjacent array entries can be used to sort an array.

The previous array, containing 3 2 1, could also be sorted by swapping A1 and A2, then swapping A2 and A3, and finally swapping A1 and A2 again.

The swap map that describes this sorting sequence is 1 2 1.

For a given array, how many different swap maps exist?

A little thought will show that there are an infinite number of swap maps, since sequential swapping of an arbitrary pair of elements will not change the order of the elements.

Thus the swap map 1 1 1 2 1 will also leave our array elements in ascending order. But how many swap maps of minimum size will place a given array in order?

That is the question you are to answer in this problem.

The input data will contain an arbitrary number of test cases, followed by a single 0.

Each test case will have a integer n that gives the size of an array, and will be followed by the n integer values in the array.

For each test case, print a message similar to those shown in the sample output below. In no test case will n be larger than 5.

## Sample Input:

```
2 9 7
2 12 50
3 3 2 1
3 9 1 5
0
```

## Expected Output:

There are 1 swap maps for input data set 1.

There are 0 swap maps for input data set 2.

There are 2 swap maps for input data set 3.

There are 1 swap maps for input data set 4.

# Find The Base

**Source file :** base.{c|cpp}

**Input file:** base.in

**Output file:** base.out

Numbers have different representations depending on the bases on which they are expressed. For example, in base 3, number 12 is written as 110 but in base 8 it is written as 14.

For this problem your program will be presented with a sequence of pairs of integers. Let's call the members of a pair X and Y.

What your program is to do is determine the smallest base for X and the smallest base for Y (likely different from that for X) so that X and Y represent the same value.

Consider, for example, the integers 12 and 5. Certainly these are not equal if base 10 is used for each. But suppose 12 was a base 3 number and 5 was a base 6 number?

$12_{\text{base } 3} = 1 \times 3 + 2 \times 1$ , or  $5_{\text{base } 10}$ , and certainly 5 in any base is equal to 5 base 10. So 12 and 5 can be equal, if you select the right bases for each of them!

## Input:

On each line of the input data there will be a pair of integers, X and Y, separated by one or more blanks; leading and trailing blanks may also appear on each line, and are to be ignored.

The bases associated with X and Y will be between 1 and 36 (inclusive), and as noted above, need not be the same for X and Y.

In representing these numbers the digits 0 through 9 have their usual decimal interpretations.

The uppercase alphabetic characters A through Z represent digits with values 10 through 35, respectively.

The last line of the input will contain nothing but zero or more blanks (and an end of line, of course), and represents the end of the data.

There will be no incorrectly formatted data in the input.

## Output:

For each pair of integers in the input display a message similar to those shown in the examples shown below.

Of course if the two integers cannot be equal regardless of the assumed base for each, then print an appropriate message; a suitable illustration is given in the examples.

## Sample Input:

```
12 5
10 A
12 34
123 456
1 2
10 2
```

## Sample Output:

```
12 (base 3) = 5 (base 6)
10 (base 10) = A (base 11)
12 (base 17) = 34 (base 5)
123 is not equal to 456 in any base 2..36
1 is not equal to 2 in any base 2..36
10 (base 2) = 2 (base 3)
```

# Lines and Points

**Source file:** line.{c|cpp}

**File Input:** line.in

**File Output:** line.out

Given a set of points in 2D space, each with integer  $x$  and  $y$  coordinates, find the largest number  $K$  of points lying exactly on the same line.

## Input:

The input file contains an unknown number of data points sets. Every set is preceded by an integer  $n$  specifying the number of points in that set. If  $n$  is 0, then there are no more sets.

Each line in the file stores the  $x$  coordinate and the  $y$  coordinate (in this order) separated by a space.

## Output:

The output file contains the number  $K$  for each set in a separate line.

## Sample Input File:

```
15
7 122
8 139
9 156
10 173
11 190
2 4
12 207
13 224
14 241
5 10
15 258
16 275
17 292
3 6
18 309
0
```

## Output File:

```
12
```

# Perfect Cubes

Source file : cubes.{c|cpp}

Input file: (none)

Output file: cubes.out

For hundreds of years Fermat's Last Theorem, which stated simply that for  $n > 2$  there exist no integers  $a, b, c > 1$  such that  $a^n = b^n + c^n$ , has remained elusively unproven. (A recent proof is believed to be correct, though it is still undergoing scrutiny.)

It is possible, however, to find integers greater than 1 that satisfy the "perfect cube" equation  $a^d = b^c + c^d + d^c$  (e.g. a quick calculation will show that the equation  $12^3 = 6^2 + 8^2 + 10^2$  is indeed true). This problem requires that you write a program to find all sets of numbers  $\{a, b, c, d\}$  which satisfy this equation for  $a \leq 100$ .

The output should be listed as shown below, one perfect cube per line, in non-decreasing order of  $a$  (i.e. the lines should be sorted by their  $a$  values).

The values of  $b, c,$  and  $d$  should also be listed in non-decreasing order on the line itself.

There do exist several values of  $a$  which can be produced from multiple distinct sets of  $b, c,$  and  $d$  triples. In these cases, the triples with the smaller  $b$  values should be listed first.

Note that the programmer will need to be concerned with an efficient implementation.

The official time limit for this problem is 2 minutes, and it is indeed possible to write a solution to this problem which executes in under 2 minutes on a 33 MHz 80386 machine.

Due to the distributed nature of the contest in this region, judges have been instructed to make the official time limit at their site the greater of 2 minutes or twice the time taken by the judge's solution on the machine being used to judge this problem.

The first part of the **output** is shown here:

Cube = 6, Triple = (3,4,5)

Cube = 12, Triple = (6,8,10)

Cube = 18, Triple = (2,12,16)

Cube = 18, Triple = (9,12,15)

Cube = 19, Triple = (3,10,18)

Cube = 20, Triple = (7,14,17)

Cube = 24, Triple = (12,16,20)