

# ALGORITHMIQUE EFFECTIVE – PARTIEL 1

Mardi 15 novembre 2005, 16h00 – 19h00

## 1 Excess

In 1979, Dan Bricklin and Bob Frankston wrote VisiCalc, the first spreadsheet application. It became a huge success and, at that time, was the killer application for the Apple II computers. Today, spreadsheets are found on most desktop computers.

The idea behind spreadsheets is very simple, though powerful. A spreadsheet consists of a table where each cell contains either a number or a formula. A formula can compute an expression that depends on the values of other cells. Text and graphics can be added for presentation purposes.

You are to write a very simple spreadsheet application. Your program should accept several spreadsheets. Each cell of the spreadsheet contains either a numeric value (integers only) or a formula, which only support sums. After having computed the values of all formulas, your program should output the resulting spreadsheet where all formulas have been replaced by their value.

A1	B1	C1	D1	E1	F1	...
A2	B2	C2	D2	E2	F2	...
A3	B3	C3	D3	E3	F3	...
A4	B4	C4	D4	E4	F4	...
A5	B5	C5	D5	E5	F5	...
A6	B6	C6	D6	E6	F6	...
...	...	...	...	...	...	...

Figure 1: Naming of the top left cells

### Input

The first line of the input file contains the number of spreadsheets to follow. A spreadsheet starts with a line consisting of two integer numbers, separated by a space, giving the number of columns and rows. The following lines of the spreadsheet each contain a row. A row consists of the cells of that row, separated by a single space.

A cell consists either of a numeric integer value or of a formula. A formula starts with an equal sign (=). After that, one or more cell names follow, separated by plus signs (+). The value of such a formula is the sum of all values found in the referenced cells. These cells may again contain a formula. There are no spaces within a formula.

You may safely assume that there are no cyclic dependencies between cells. So each spreadsheet can be fully computed.

The name of a cell consists of one to three letters for the column followed by a number between 1 and 999 (including) for the row. The letters for the column form the following series: A, B, C, ..., Z, AA, AB, AC, ..., AZ, BA, ..., BZ, CA, ..., ZZ, AAA, AAB, ..., AAZ, ABA, ..., ABZ, ACA, ..., ZZZ. These letters correspond to the number from 1 to 18278. The top left cell has the name A1. See figure 1.

### Output

The output of your program should have the same format as the input, except that the number of spreadsheets and the number of columns and rows are not repeated. Furthermore, all formulas should be replaced by their value.

#### Sample Input

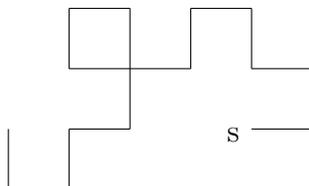
```
1
4 3
10 34 37 =A1+B1+C1
40 17 34 =A2+B2+C2
=A1+A2 =B1+B2 =C1+C2 =D1+D2
```

#### Sample Output

```
10 34 37 81
40 17 34 91
50 51 71 172
```

## 2 Origami

If a large sheet of paper is folded in half, then in half again, etc, with all the folds parallel, then opened up flat, there are a series of parallel creases, some pointing up and some down, dividing the paper into fractions of the original length. If the paper is only opened “half-way” up, so every crease forms a 90 degree angle, then (viewed end-on) it forms a “dragon curve”. For example, if four successive folds are made, then the following curve is seen (note that it does not cross itself, but two corners touch):



Write a program to draw the curve which appears after  $N$  folds. The exact specification of the curve is as follows.

The paper starts flat, with the “start edge” on the left, looking at it from above. The right half is folded over so it lies on top of the left half, then the right half of the new double sheet is folded on top of the left, to form a 4-thick sheet, and so on, for  $N$  folds. Then every fold is opened from a 180 degree bend to a 90 degree bend. Finally the bottom edge of the paper is viewed end-on to see the dragon curve.

From this view, the only unchanged part of the original paper is the piece containing the “start edge”, and this piece will be horizontal, with the “start edge” on the left. This uniquely defines the curve. In the above picture, the “start edge” is the left end of the rightmost bottom horizontal piece (marked ‘s’). Horizontal pieces are to be displayed with the underscore character “\_”, and vertical pieces with the “|” character.

### Input

Input will consist of a series of lines, each with a single number  $N$  ( $1 \leq N \leq 13$ ). The end of the input will be marked by a line containing a zero.

### Output

Output will consist of a series of dragon curves, one for each value of  $N$  in the input. Your picture must be shifted as far left, and as high as possible. Note that for large  $N$ , the picture will be greater than 80 characters wide, so it will look messy on the screen. The pattern for each different number of folds is terminated by a line containing a single ‘^’.

### Sample input

```
2
4
1
0
```

### Sample output

```
|_
^-|
^

  |_  |_  |_
  -|  -|  -|
|_|
^
-|
^
```

### 3 Change, please

New Zealand currency consists of \$100, \$50, \$20, \$10, and \$5 notes and \$2, \$1, 50c, 20c, 10c and 5c coins. Write a program that will determine, for any given amount, in how many ways that amount may be made up. Changing the order of listing does not increase the count. Thus 20c may be made up in 4 ways:  $1 \times 20c$ ,  $2 \times 10c$ ,  $10c + 2 \times 5c$ , and  $4 \times 5c$ .

#### Input

Input will consist of a series of real numbers no greater than \$300.00 each on a separate line. Each amount will be valid, that is will be a multiple of 5c. The file will be terminated by a line containing zero (0.00).

#### Output

Output will consist of a line for each of the amounts in the input, each line consisting of the amount of money (with two decimal places and right justified in a field of width 6), followed by the number of ways in which that amount may be made up, right justified in a field of width 17.

#### Sample input

```
0.20
2.00
0.00
```

#### Sample output

```
0.20          4
2.00        293
```

*(On pourra utiliser des entiers sur 64 bits (type `int64_t`) en incluant l'entête `stdint.h`)*

## 4 Sooner or letter

Cryptanalysis is the process of breaking someone else's cryptographic writing. This sometimes involves some kind of statistical analysis of a passage of (encrypted) text. Your task is to write a program which performs a simple analysis of a given text.

### Input

The first line of input contains a single positive decimal integer  $n$ . This is the number of lines which follow in the input. The next  $n$  lines will contain zero or more characters (possibly including whitespace). This is the text which must be analyzed.

### Output

Each line of output contains a single uppercase letter, followed by a single space, then followed by a positive decimal integer. The integer indicates how many times the corresponding letter appears in the input text. Upper and lower case letters in the input are to be considered the same. No other characters must be counted. The output must be sorted in descending count order; that is, the most frequent letter is on the first output line, and the last line of output indicates the least frequent letter. If two letters have the same frequency, then the letter which comes first in the alphabet must appear first in the output. If a letter does not appear in the text, then that letter must not appear in the output.

### Sample Input

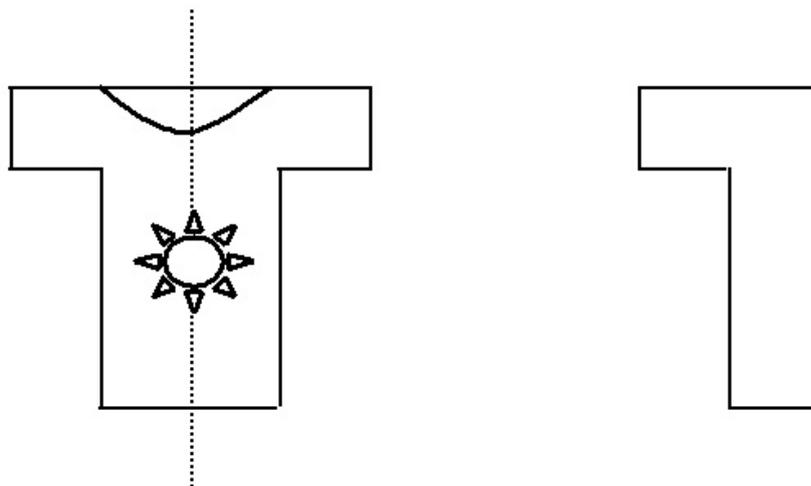
```
3
This is a test.
Count me 1 2 3 4 5.
Wow!!!! Is this question easy?
```

### Sample Output

```
S 7
T 6
I 5
E 4
O 3
A 2
H 2
N 2
U 2
W 2
C 1
M 1
Q 1
Y 1
```

## 5 Y-shirts?

I have a T-Shirt. When I don't wear it any more, I fold it up. For most of the time, I can find a line so that the parts of the T-shirt on both sides are symmetrical. Then, I can fold it along that line. But sadly, I cannot find such a line for some strange (really really strange, see sample input!) T-shirts.



In the example above, I can fold the T-shirt along the dash line, then, I got the figure on the right. Could you tell me if I can succeed?

### Input

The first line of the input contains a single integer  $t$  ( $t \leq 20$ ) indicating the number of test cases. Each test case begins with a line containing a single integer  $n$  ( $3 \leq n \leq 100$ ) indicating the number of points of the polygon. In the next  $n$  lines each contain a pair of integers  $(x_i, y_i)$ , indicating the position of the points. The points are given in the counter-clockwise order. The T-Shirt is valid, i.e not self-crossing. But the T-Shirt may be not convex.

### Output

For each test case, output a line corresponding the answer. Answer 'Yes' if the T-Shirt can be folded, 'No' otherwise.

### Sample Input

```
2
3
0 0
5 0
1 1
8
1 0
2 0
2 1
-2 1
-2 0
-1 0
-1 -3
1 -3
```

### Sample Output

```
No
Yes
```