University of Puerto Rico Mayagüez
Department of Electrical and Computer Engineering

ICOM 4036: Programming Languages
Programming Assignment 2[1]
DUE Friday November 5, 2004

Integral Determinant

Write C++ and FORTRAN versions of the program described below and compare their wall clock runtimes. Determine which version is faster. In a separate README file argue why one version of the program is faster and carefully justify your answer. You should submit a zip file with three files inside: the C++ source, the FORTRAN source and the README file.

Write a program to find the determinant of an integral square matrix. Note that the determinant of a square matrix can be defined recursively as follows: the determinant of a 1x1 matrix $M = (m_{1,1})$ is just the value $|M| = m_{1,1}$; further, the determinant of an n x n matrix is:

$$\left|M\right| = \sum_{i=1}^{n} (-1)^{i+1} \cdot \left|M_{1,j}\right| \cdot m_{ij}$$

Here the notation $|M_{1,i}|$ is the *(n – 1) x (n – 1)* matrix by removing the first row and the i^{th} column of the original n x n matrix *M*.

A straightforward recursive method of calculating the determinant of an n x n matrix will end up with *n*! multiplications, a very time-consuming algorithm. To give you a feeling about this, note that 15! = 1,307,674,368,000. To reduce the time complexity, there are two ways of modifying the original matrix for easier computation.

1. Exchanging two columns (or rows) of a matrix will change the sign of the determinant; for example
$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = - \begin{vmatrix} 2 & 1 \\ 4 & 3 \end{vmatrix}$$
2. Multiplying one column by any constant, and add them to another column will not change the value of the determinant; for example:
$$\begin{vmatrix} 2 & 1 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = - \begin{vmatrix} 2 & 1 & 3+2\times 2 \\ 4 & 5 & 6+4\times 2 \\ 7 & 8 & 9+7\times 2 \end{vmatrix}$$

[1] Programming assignment originally appeared in http://acm.uva.es

Using the above methods, you shall be able to write a program for computing the determinants of matrices, even for a size like 30 x 30, very efficiently. Below is an example to show how this can be done:

$$\begin{vmatrix} 2 & 3 & 5 \\ 1 & 6 & 7 \\ 4 & 8 & 9 \end{vmatrix} = \begin{vmatrix} 2 & 3-2 & 5-2\times 2 \\ 1 & 6-1 & 7-1\times 2 \\ 4 & 8-4 & 9-4\times 2 \end{vmatrix} = \begin{vmatrix} 2 & 1 & 1 \\ 1 & 5 & 5 \\ 4 & 4 & 1 \end{vmatrix} = - \begin{vmatrix} 1 & 1 & 2 \\ 5 & 5 & 1 \\ 1 & 4 & 4 \end{vmatrix}$$

$$= - \begin{vmatrix} 1 & 1-1 & 2-1\times 2 \\ 4 & 5-5 & 1-5\times 2 \\ 1 & 4-1 & 4-1\times 2 \end{vmatrix} = - \begin{vmatrix} 1 & 0 & 0 \\ 5 & 0 & -9 \\ 1 & 3 & 2 \end{vmatrix} = - \begin{vmatrix} 0 & -9 \\ 3 & 2 \end{vmatrix} = -27$$

Note that the answer shall be an *integer*. That is, all the operations needed are just integer operations; by reducing to floating numbers would result in the round-off errors, which will be considered as the **wrong** answer. Do not worry about the problem of integral overflows problem. You can assume that the given data set will not cause the integer overflow problem. What is emphasized here is the required integer precision. Anyway use of floating numbers is not forbidden.

Input

Several sets of integral matrices each one represented as a list of integers. Within each set, the first integer (in a single line) represents the size of the matrix, *n*, which can be as large as 30, indication an n x n matrix. After *n*, there will be *n* lines representing the *n* rows of the matrix; each line (row) contains exactly *n* integers. Thus, there are n^2 integers for the particular matrix. These matrices will occur repeatedly in the input as the pattern described above. An integer *n* = `0` (zero) signifies the end of input.

Output

For each matrix of the input, calculate its (integral) determinant and output it in a line. Output a single star (`*`) to signify the end of outputs.

Sample Input

```
2
5 2
3 4
3
2 3 5
1 6 7
4 8 9
0
```

Sample Output

```
14
-27
*
```