Univerza v Ljubljani

# Prvenstvo
# UNIVERZE V LJUBLJANI
# v
# programiranju

## 2002

## 3. krog

**Pokrovitelji**:


UNIVERZA V LJUBLJANI
Fakulteta za matematiko in fiziko

# Prvenstvo v programiranju 2002

3. krog 25. september 2002

---

# NAVODILA

**Prijava**: Prijavite se lahko le na en računalnik. Če bodo med tekmovanjem z njim slučajno težave, javite vodstvu tekmovanja. V primeru, da boste hkrati prijavljeni na dveh računalnikih, bo ekipa diskvalificirana.

Uporabniško ime je sestavljeno iz besede acm in številke računalnika.

Primer: uporabniško ime na računalniku št. 1 je: acm01

uporabniško ime na računalniku št. 12 je: acm12

Geslo je sporočeno pred začetkom tekmovanja.

Vaše domače področje na disku je `/home/acmXX` kjer je XX številka računalnika

**Priprava delovnega okolja**

Po uspešni prijavi morate odpreti dva terminala in program Netscape.

Terminal št.1: v njem boste pisali programe, prevajali programe in pošiljali programe sodniku.

Terminal št.2: v njem boste dobivali obvestila o pravilnosti rešitev in pošiljali morebitna vprašanja sodniku.

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

**Pisanje programov:**

Na voljo so urejevalniki besedil: *vi, joe, jed, mcedit, emacs*, *gedit in kedit*

Ime programov je točno določeno pri opisu nalog. Pazite na `return 0,` ki mora zaključiti vsak uspešno izveden C program.

**Prevajanje programov:**

Za prevajanje programov morate uporabljati naslednje prevajalnike:

```
C       (*.c)         gcc    primer:   gcc -o program program.c
C++     (*.cpp)       g++    primer:   g++ -o program program.cpp
Pascal (*.pas)        gpc    primer:   gpc -o program program.pas
```

**Pošiljanje programov:**

Za pošiljanje programov sodniku uporabljate ukaz `submit`.

Primer: `submit program.c`

**Sporočila o napakah – PE in WA**

Zaradi različnih razlogov je zelo težko razlikovati med `Presentation Error` in `Wrong Answer`. Ker preverjanje poteka več ali manj avtomatsko, program pogosto ne zna razlikovati med tem, ali ste le pozabili piko v odgovoru, ali pa ste napačno zračunali. Zato tudi `Wrong Answer` včasih lahko pomeni

le manjkajoč znak v odgovoru. Praviloma naj bi sporočilo `PE` dobili le v primeru napačno postavljenih presledkov ali ločil in praznih vrstic, ni pa to nujno!

**Komunikacija sodniki – ekipa:**

V terminalu 2 izvedete ukaz: `telnet svarun`

Za prijavo na računalnik svarun uporabite isto uporabniško ime in geslo, kot ste ga uporabili za prijavo na vaš računalnik. V tem terminalskem oknu se bodo izpisovala sodnikova sporočila vam. Če želite poslati sporočilo sodniku, morate v tem terminalu napisati ukaz:

```
write sodnik
```

in potem sporočilo, ki ga zaključite s kombinacijo tipk CTRL + D na začetku prazne vrstice. Prosimo, da **ne** pošiljate sporočil tipa `... Sedaj smo poslali program ...` in `... Naš program zagotovo dela, pa dobimo WA ...` Ekipa, ki bo pošiljala tovrstna sporočila, je lahko tudi diskvalificirana.

**Trenutni rezultati:**

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

Po zagonu programa greste na naslov:

http://svarun.fmf.uni-lj.si/index.html

Kadar želite videti trenutno stanje, morate narediti »reload«.

**Pritožbe:**

So možne le takoj po zaključku tekmovanja. Kasneje se rezultati lahko spremenijo le v primeru drastične sodniške napake.

**Preklapljanje med angleško in slovensko tipkovnico:**

(Velja samo za okolje X windows)

Prednastavljena je slovenska tipkovnica. Za preklop na angleško tipkovnico v terminalu izvedete ukaz

```
setxkbmap us
```

za preklop nazaj na slovensko pa:

```
setxkbmap sl
```

Izbrana tipkovnica potem velja za celotno X okolje.

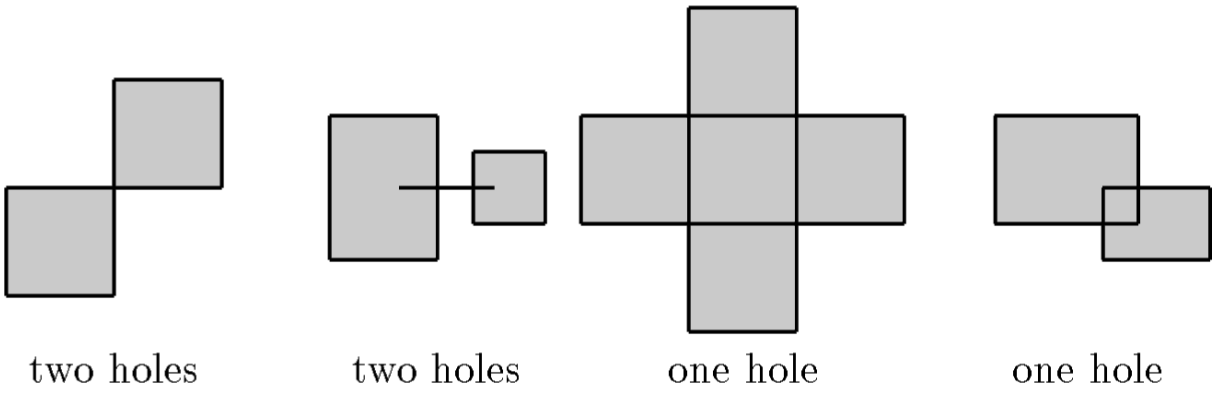
# Tin Cutter

Program TIN.C, TIN.CPP, TIN.PAS

In a Tin Cutting factory there is a machine for cutting parts from tin plates. It has an extraordinarily sharp knife able to make horizontal or vertical segment cuts in the tin plates. Each cutting process consists of a sequence of such cuts. Each segment cut is given by its endpoints that are always located inside the tin plate. During the cutting process some parts of tin plate can fall out and so some holes in the plate can emerge.

Factory management needs to predict the number of holes in the plate at the end of the given sequence of cuts. Write a program that answers this question. Single segment cuts are not considered to be holes.

Here there are examples of some situations that can arise after cutting:




## Input

The input file consists of blocks of lines. Each block except the last describes one cutting process. In the first line of the block there is a number N <= 100 indicating the number of segment cuts in the cutting process. These cuts are defined by the following N lines. The line defining one segment cut has the form $X_1$ $Y_1$ $X_2$ $Y_2$ where $X_1$ , $Y_1$ and $X_2$ , $Y_2$ are the co-ordinates of the end points of the segment cut. They are separated by one space. The co-ordinates are integers and always define horizontal or vertical segment (i.e. segment parallel with x or y axis). The last block consists of just one line containing 0.

## Output

The output file contains the lines corresponding to the blocks in the input file. Each such line contains the number of holes that remain in the tin plate after the execution of the corresponding cuts. There is no line in the output file corresponding to the last ``null'' block of the input file.

Sample Input:

```
4
0 1 1 1
1 1 1 0
1 0 0 0
0 0 0 1
2
0 1 2 1
1 2 1 0
0
```

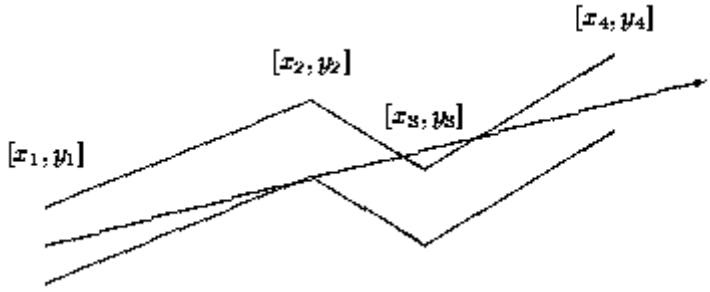
Sample Output:

```
1
0
```

# Pipe

Program PIPE.C, PIPE.CPP, PIPE.PAS

The GX Light Pipeline Company started to prepare bent pipes for the new transgalactic light pipeline. During the design phase of the new pipe shape the company ran into the problem of determining how far the light can reach inside each component of the pipe. Note that the material which the pipe is made from is not transparent and not light reflecting.




Each pipe component consists of many straight pipes connected tightly together. For the programming purposes, the company developed the description of each component as a sequence of points [$x1$; $y1$], [$x2$; $y2$], . . ., [$xn$; $yn$], where $x1 < x2 < \ldots xn$ . These are the upper points of the pipe contour. The bottom points of the pipe contour consist of points with $y$-coordinate decreased by 1. To each upper point [$xi$; $yi$] there is a corresponding bottom point [$xi$; ($yi$)-1] (see picture above). The company wants to find, for each pipe component, the point with maximal $x$-coordinate that the light will reach. The light is emitted by a segment source with endpoints [$x1$; ($y1$)-1] and [$x1$; $y1$] (endpoints are emitting light too). Assume that the light is not bent at the pipe bent points and the bent points do not stop the light beam.

## Input

The input file contains several blocks each describing one pipe component. Each block starts with the number of bent points $2 <= n <= 20$ on separate line. Each of the next $n$ lines contains a pair of real values $xi$, $yi$ separated by space. The last block is denoted with $n = 0$.

## Output

The output file contains lines corresponding to blocks in input file. To each block in the input file there is one line in the output file. Each such line contains either a real value, written with precision of two decimal places, or the message '`Through all the pipe.`'. The real value is the desired maximal $x$-coordinate of the point where the light can reach from the source for corresponding pipe component. If this value equals to $xn$, then the message '`Through all the pipe.`' will appear in the output file.

## Sample Input

```
4
0 1
2 2
4 1
6 4
6
0 1
2 -0.6
5 -4.45
7 -5.57
12 -10.8
17 -16.55
0
```

## Sample Output

```
4.67
Through all the pipe.
```

# Gizilch

Program GIZILCH.C, GIZILCH.CPP, GIZILCH.PAS

The game of gizilch has very simple rules. First 100 grapes are labeled, in nontoxic ink, with the numbers 1 to 100. Then, with a cry of "GIZILCH!", the referee fires the grapes up into the air with a giant gizilcher. The two players, who each start with a score of "1", race to eat the falling (or, shortly thereafter, fallen) grapes and, at the same time, multiply their scores by the numbers written on the grapes they eat. After a minute, the hungry squirrels are let loose to finish the remaining grapes, and each contestant reports his score, the product of the numbers on the grapes he's eaten. The unofficial winner is the player who announces the highest score.

Inevitably, though, disputes arise, and so the official winner is not determined until the disputes are resolved. The player who claims the lower score is entitled to challenge his opponent's score. The player with the lower score is presumed to have told the truth, because if he were to lie about his score, he would surely come up with a bigger better lie. The challenge is upheld if the player with the higher score has a score that cannot be achieved with grapes not eaten by the challenging player. So, if the challenge is successful, the player claiming the lower score wins. (There's an exception to this rule: player with lower score might do something really stupid, like claiming to have 121 points – in which case his challenge is certainly hopeless.)

So, for example, if one player claims 343 points and the other claims 49, then clearly the first player is lying; the only way to score 343 is by eating grapes labeled 7 and 49, and the only way to score 49 is by eating a grape labeled 49. Since each of two scores requires eating the grape labeled 49, the one claiming 343 points is presumed to be lying.

On the other hand, if one player claims 162 points and the other claims 81, it is possible for both to be telling the truth (e.g. one eats grapes 2, 3 and 27, while the other eats grape 81), so the challenge would not be upheld.

Unfortunately, anyone who is willing to referee a game of gizilch is likely to have himself consumed so many grapes (in a liquid form) that he or she could not reasonably be expected to perform the intricate calculations that refereeing requires. Hence the need for you, sober programmer, to provide a software solution.

## *Input*

Pairs of unequal, positive numbers, with each pair on a single line, that are claimed scores from a game of gizilch.

## *Output*

Numbers, one to a line, that are the winning scores, assuming that the player with the lower score always challenges the outcome.

### Sample Input

```
343 49
143 121
```

```
610 3599
62 36
```

Sample Output

```
49
143
610
62
```

# String Matching

Program STRING.C, STRING.CPP, STRING.PAS

It's easy to tell if two words are identical - just check the letters. But how do you tell if two words are almost identical? And how close is “almost”?

There are lots of techniques for approximate word matching. One is to determine the best substring match, which is the number of common letters when the words are compared letter-by-letter. The key to this approach is that the words can overlap in any way. For example, consider the words CAPILLARY and MARSUPIAL. One way to compare them is to overlay them:

```
CAPILLARY
MARSUPIAL
```

There is only one common letter (A). Better is the following overlay:

```
CAPILLARY
MARSUPIAL
```

with two common letters (A and R), but the best is:

```
CAPILLARY
MARSUPIAL
```

Which has three common letters (P, I and L).

The approximation measure appx(word1, word2) for two words is given by:

$$\frac{\text{common letters} * 2}{\text{length(word1)} + \text{length(word2)}}$$

Thus, for this example, appx(CAPILLARY, MARSUPIAL) = 6 / (9 + 9) = 1/3. Obviously, for any word W appx(W, W) = 1, which is a nice property, while words with no common letters have an appx value of 0.

## Input

The input for your program will be a series of words, two per line, until the end-of-file flag of -1. Using the above technique, you are to calculate appx() for the pair of words on the line and print the result. The words will all be uppercase.

## Output

Print the value for appx() for each pair as a reduced fraction. Fractions reducing to zero or one should have no denominator.

### Sample Input

```
CAR CART
TURKEY CHICKEN
MONEY POVERTY
ROUGH PESKY
A A
-1
```

### Sample Output

```
appx(CAR,CART) = 6/7
appx(TURKEY,CHICKEN) = 4/13
appx(MONEY,POVERTY) = 1/3
appx(ROUGH,PESKY) = 0
appx(A,A) = 1
```

# Flip Sort

Program FLIP.C, FLIP.CPP, FLIP.PAS

Sorting in computer science is an important part. Almost every problem can be solved effeciently if sorted data are found. There are some excellent sorting algorithm which has already acheived the lower bound nlgn. In this problem we will also discuss about a new sorting approach. In this approach only one operation ( Flip ) is available and that is you can exchange two adjacent terms. If you think a while, you will see that it is always possible to sort a set of numbers in this way.

## *The Problem*

A set of integers will be given. Now using the above approach we want to sort the numbers in ascending order. You have to find out the minimum number of flips required. Such as to sort "1 2 3" we need no flip operation whether to sort "2 3 1" we need at least 2 flip operations.

## *The Input*

The input will start with a positive integer N (N<=1000). In next few lines there will be N integers. Input will be terminated with a zero.

## *The Output*

For each data set print "Minimum exchange operations : M" where M is the minimum flip operations required to perform sorting. Use a seperate line for each case.

### Sample Input

```
3
1 2 3
3
2 3 1
0
```

### Sample Output

```
Minimum exchange operations : 0
Minimum exchange operations : 2
```