1991 ACM Scholastic Programming Contest
European Regional Final

Problem A : JOKER

INPUT FILE : **JOKER.IN**
OUTPUT FILE : **JOKER.OUT**
SOURCE CODE : **JOKER.PAS**

Write a program that is able to compute the amount of money you have won with a given winning Joker number. You are given the winning Joker number and a list of numbers that should be checked to find out how much they have won. All numbers contain 7 digits. When two or more successive digits correspond to the winning Joker number and if they are in the same position as in the winning Joker number, you win 80 BF or more. The amount you win depends on the number of succesive digits which correspond with the Joker number, according to the following table:

No. digits	Money (BF)
0	0
1	0
2	80
3	800
4	8000
5	80000
6	800000
7	8000000

For example: the winning Joker number is 7254780; the number to check is 4253900. Only 2,5 and 0 are digits which correspond with the winning Joker number (4 is not in the right position). Since only 2 and 5 are successive digits, you win 80 BF.

Thie input file contains the winning Joker number on it's first line, and then a list of numbers to check, each on separate lines. The output file contains the number to check, followed by two blanks and the amount of money the number won.
The input file is error-free.

Continue ....

.... Continued

Example:

JOKER.IN:	JOKER.OUT:	
1234567	1253467	160
1253467	0123456	0
0123456	3234578	8000
3234578	1244569	880
1244569		

1991 ACM Scholastic Programming Contest
European Regional Final

Problem B : CALC

INPUT FILE : **CALC1.IN, CALC2.IN, ... , CALC10.IN**
OUTPUT FILE : **CALC.OUT**
SOURCE CODE : **CALC.PAS**

The main purpose of this highly ingenious CALC-program you are about to write, is (general consternation!) to calculate. You will have to produce some results, based on not terribly complicated mathematical formulae, contained in a file. The file does not provide you with ready-to-use numbers, but with numbers written out as ASCII-words instead, e.g. `one + two`. Unfortunately, I dropped my floppy-disk and the characters got mixed up. What I am actually trying to say is that the file is in code !

Your mission, should you choose to accept it (this line is stolen from the American TV-series Mission Impossible, but do not let this upset you !), is to give me the results of those formulae. Should you, or any of your crew be killed during this action, the Government will disavow any responsibility (Mission Impossible again).

Let us get down to business now. All the I/O the program has to do is to :

- read its input from 10 test files CALC1.IN to CALC10.IN
- write the result of the formulae to the file CALC.OUT (one line per input file)
- write the average of the 10 results to the file CALC.OUT (as the 11th. line)

All calculations are internal operations on the set of natural numbers (positive integers, including 0 i.e. 0, 1, 2,...). 'Internal' means that the result of the operation is a member of the set of natural numbers.

The 10 results and the final result are guaranteed to be perfectly normal, 15-bit non-negative values, but during calculation, anything can happen! As soon as something nasty happens, (15-bit overflow, non-internal operation, etc...) the result for the current input file is set to zero.

The 10 input files are all ordinary ASCII-files, containing a text, built from the following set of words, separated by space-characters (everything in upper case) :

Continue ....

.... Continued

```
ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN,EIGHT,NINE,ZERO,+,-,*,/,=
```

These words are grouped together to form phrases like :

```
ONE + TWO THREE / THREE =
```

meaning:

$$1 + 23 / 3 =$$

This is an example of the complicated mathematics you will encounter, only natural numbers, addition, subtraction, multiplication and division, internal on the natural numbers.

Calculations are simply carried out from left to right, so the result of this phrase is 8. You will produce a result as soon as you encounter an equality-sign (so don't read the input-file further than the first equality-sign you encounter, because you may find nonsense beyond that), or, as stated in the previous section, a zero when something goes wrong. Then, you process the next file.

Having processed the 10 files, you also show the average over the 10 files. Again, this calculation follows the rules of the previous section.

To allow recognition of these phrases, you will have to decode the input files first. The encoding scheme is fairly simple. All you have to do is to reorder the characters in the input file, because only the characters of the original set of words were used. You start decoding each file by making a histogram over the entire file, i.e. you count the number of occurences of each character in the file. The one occuring most will be remapped onto the space-character. The second one on the letter 'O', then 'N', 'E', 'T', 'W', 'H', 'R', 'F', 'U', 'I', 'V', 'S', 'X', 'G', 'Z', '+', '-', '*', '/', '='.

Good luck ! Luck is not a factor (or should not be !)

1991 ACM Scholastic Programming Contest
European Regional Final

Problem C : BITMAP

INPUT FILE : **BITMAP.IN**
OUTPUT FILE : **BITMAP.OUT**
SOURCE CODE : **BITMAP.PAS**

A video digitiser produces binary images of pictures containing some simple geometric shapes. The only shapes that the image contains are triangles and rectangles (solid shapes, not their outlines), but they can be rotated over random angles (so they need not and will not all have their edges parallel to the axes). For a good understanding : a rectangle has four corners of 90∞ ! Your task will be to identify all objects and return the bitmap containing only the triangles.

Since this may not be as simple a task as it seems, we will provide you with a number of clues:

* find a connected area (i.e.an area of set pixels that are all liked together)
* find it's 'bounding box', i.e. the rectangle (sides parallel to the axes) which contains the whole area
* dependent upon the maxima (reached upon the sides of the bounding box) you should be able to determine whether the shape is a triangle or a rectangle
* every occuring 'corner'-pixel has 5 unlit pixels among it's 8 nearest neighbours

We stress again that you are guaranteed that all shapes are either triangles or rectangles, so you need not check this ; this implies for example that every occuring line will be straight. However, since we're talking about a bitmap, this line may be more of a 'staircase' !!

The input file will be a textfile. The first line will contain the X- and Y-dimension of the bitmap X and Y respectively (<100), separated by blanks. There will be Y other lines, each containing X characters from the set ['0','1'], with '1' meaning that the pixel is lit.

We also guarantee that no object will touch the border of the bitmap (so the border will contain only 0's), that no two objects will touch or overlap, and that the minimum dimension of a triangle will be 3 pixels and of a rectangle 4 pixels.

Continue ....

.... Continued

The output file should be of the same format as the input file and contain only the triangles that were in the original image (at the same place, of course). Have fun ...

1991 ACM Scholastic Programming Contest
European Regional Final

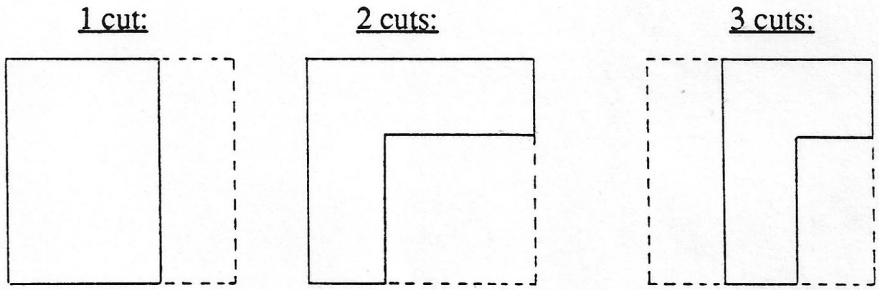
Problem D : FLOORTILES

INPUT FILE : **FLOOR.IN**
OUTPUT FILE : **FLOOR.OUT**
SOURCE CODE : **FLOOR.PAS**

A castle needs new floortiles. The king does not like cut floortiles, so he asks to use as few cuts as possible. Each room of the castle is rectangular and there are zero or more rectangular pillars in the rooms. No pillar touches another and no pillar touches the wall. All dimensions are integers ranging from 1 to 32000. The number of pillars is unlimited, as long as the pillars fit into the room. The pillars have a minimum dimension of 1 by 2 length units.

The tiles used are squares and have a size of three by three length units. They can be cut at one or two times the unit length. In no case there may be a cut that is not adjacent to a wall or a pillar. A 'cut' is defined as cutting of a piece of the floortile along a straight line of any length. What remains of the floortile after the cut is thrown away and is not reused later (although this part may be greater than the part that is used). The floortiles should be laid in a regular rectangular pattern (this means that if you think away the pillars, what remains is a checkboard-pattern). It is your task to design a program that gives the minimum number of cuts for each room.

Examples of cuts:



The input file consists of a number of blocks. The first line of each block gives the dimensions of the room (X and Y coordinate: two integers). All other lines give the coordinates of the pillars, if any (X1, Y1, X2, Y2 coordinates: four integers, defining the opposing corners). After each block there is one empty line. The output file consists of a number of lines, one for each block. On each line there is the minimum number of cuts needed for the corresponding block. Each line, including the last one, ends with a carriage return.

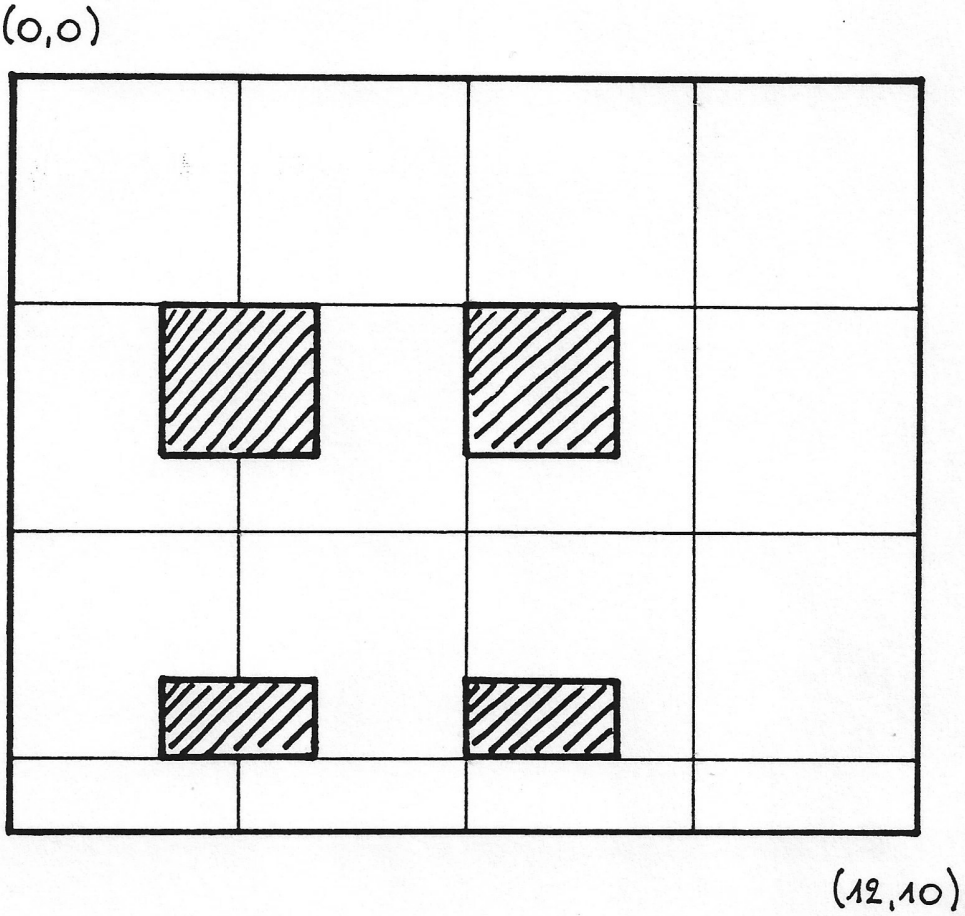
Continue....

.... Continued

Example:

```
FLOOR.IN :48   120          FLOOR.OUT :    0
                                          10
          4   30                          16

          12   10
          2   3   4   5
          2   8   4   9
          6   3   8   5
          6   8   8   9
```

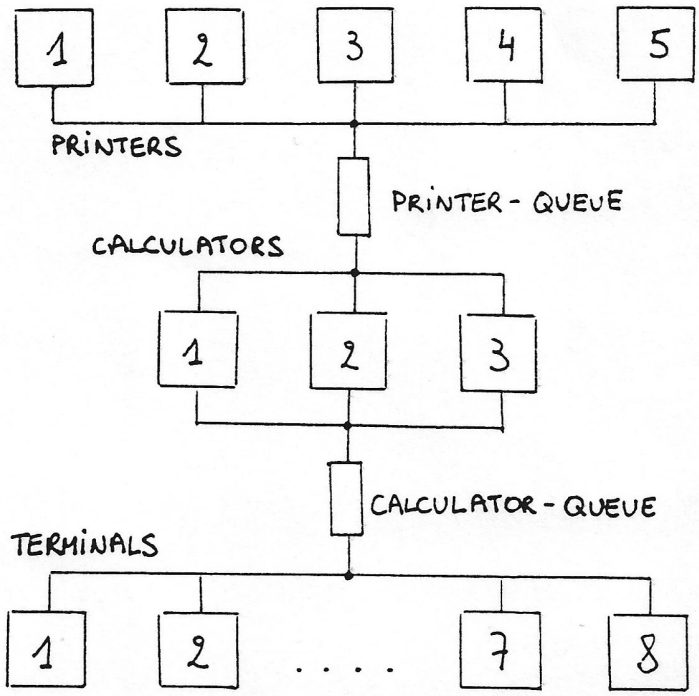


1991 ACM Scholastic Programming Contest
European Regional Final

Problem E : NETWORK

INPUT FILE : **NETWORK.IN**
OUTPUT FILE : **NETWORK.OUT**
SOURCE CODE : **NETWORK.PAS**

Since this is a programming contest, and since most of the computers seem to be linked together in a network nowadays (only your computers are not, because you would be cheating by stealing each other's submissions all the time), we will devote our attention a while to the difficulties arising on such networks, such as "Who can do what at what time on what machine ?" . Catch my meaning ? ...



As an example we will use the '835'-network that you can see on the chart: it consists of 8 terminals, linked to 3 'calculators' and further onto 5 printers, in exactly the same manner as you can see on the chart. The terminals send requests to the network, asking for a certain amout of calculation time and afterwards a certain amount of printing time. The requests first go into a calculator-queue before being calculated, and after calculation go into a printer-queue before being printed. These queues both work differently, but we'll return to this later.

Continue....

....Continued

The time is measured in arbitrary time-pulses. At time 0 all queues are empty and all calculators and printers are available.

The input text file consists of an unknown number of lines. The first line describes the network using three numbers N1, N2, N3 (all within [1..10]) : the number of terminals, the number of calculators and the number of printers. All other lines (no more than 100) contain the data for the requests. Each request is described on a separate line by five numbers, all separated by blanks, and they are respectively : the time the request is sent [1..100], the number of the requesting terminal [1..N1], the priority code [1..5] where 5 has the highest priority, the amount of calculation time requested [1..20] and finally the amount of printing time requested [1..20]. Input will be such that all occuring times will fit into one byte (so all requests will be completely processed before time 256).

These requests first go into the calculator-queue. At any time this queue is ordered first by priority (higher priorities first), then by terminal number (lower terminal numbers first) and last by request time (lower request times first). No terminal will put two requests at the same time.

The calculators, when they finish processing a request, take the first request from the calculator-queue. If two or more calculators become available at the same time, the one with the lowest number takes the first request. They calculate upon a request for as long as is defined by the calculation-time of that request. Calculation can only start at the next time-pulse after the request is put, e.g. a request entered at time 1 will start calculation at time 2 (if there is a calculator available, of course). This means that transferring events to and from a queue does not consume any time.

When a calculator finishes processing a request, it puts the request into the printer-queue. This queue is an ordinary FIFO-queue (first in, first out) without priorities. When two calculators finish calculating at the same time, the calculator with the lowest number puts it's printing request first.

The printers, when they finish processing a request, take the first request from the printer-queue. If two or more printers become available at the same time, the one with the lowest number takes the first request. They print upon a request for as long as is defined by the printing-time of that request.

Continue....

....Continued

The input file contains the terminal requests, sorted upon request time (only upon time, no further sorting).

The output text file should contain one line for each request, and the requests should occur in the same order as in the input file. Each line should contain four numbers: the calculator number that processes the request, the time at wich calculation starts, the printer number that processes the request and the time at which printing starts ; all numbers should be separated by exactly one space.

Example: NETWORK.IN :

```
8   3  5
1   1  1   10  10
1   2  1   10  10
1   3  2   12  12
1   4  1   10  10
5  10  5   1   20
6  10  5   1   20
7  10  5   1   20
10  1  1   10  10
```

NETWORK.OUT:

```
2 2 1 12
3 2 2 12
1 2 5 14
1 14 5 26
2 12 3 13
3 12 4 13
2 13 1 22
3 13 2 23
```

1991 ACM Scholastic Programming Contest
European Regional Final

Problem F : BUYING A HOUSE

INPUT FILE : **HOUSE.IN**
OUTPUT FILE : **HOUSE.OUT**
SOURCE CODE : **HOUSE.PAS**

For this next problem, we're going to plunge into the magic of 'real estate'. Since house-prices are rising far above the means of normal mortal souls, a solution has to be found. The solution we will concentrate upon is that of people putting their money together to buy a house. But more people may be interested than are really needed...

The input file consists of a number of blocks, with each block containing the data for one house (blocks are separated by one blank line). The first line contains the price of the house (this price, as well as all amounts of money, are guaranteed to be non-zero positive integers), the second line the number of people that are interested (in the range [1..20]) and the rest of the block describes these people. Each of these last lines describes one person: his number ("We are very discrete, Sir, everybody is anonymous !"), one or more blanks, and the amount of money he possesses. These lines are not guaranteed to be sorted in any way.

We define a **combination** as a group of one or more people who are able to buy the house when they put their money together, but when any of the persons is taken away, they're no longer able to buy the house (everyone in the combination is necessary to buy the house).

For example, the house costs 50, person 1 and 2 each possess 26 and person 3 possesses 10 : although they are able to buy the house, they don't form a valid combination since person 1 and 2 can buy the house by themselves without needing person 3.

Your task is to find the number of valid combinations for each house, and to write these numbers on separate lines in the output file (without leading or trailing spaces).

Continue....

....Continued

Example:

HOUSE.IN :

```
15
6
1 3
2 5
3 7
4 9
5 11
6 13

25
4
1 10
2 10
3 10
4 10
```

HOUSE.OUT :

```
11
4
```

1991 ACM Scholastic Programming Contest
European Regional Final

Problem G : CHECK !

INPUT FILE : **CHECK.IN**
OUTPUT FILE : **CHECK.OUT**
SOURCE CODE : **CHECK.PAS**

Since all computer programmers know how to play chess, here is a simple problem for everyone. Your program has to determine whether white's king is in check, in a series of positions. The standard rules of chess apply on an 8x8 board.

The input file is formatted as follows :

<white piece data>]
<blank line>] repeated for each position
<black piece data>]
<blank line>]

For each position, the white piece data and the black piece data are sequences of lines. These lines contain three characters each :

- the first character is from the set {'**p**', '**r**', '**n**', '**b**', '**q**', '**k**'} and denotes the piece whose location is being described: **p**awn, **r**ook, **k**night, **b**ishop, **q**ueen and **k**ing respectively.
- the second and third character indicate the location of this piece, coded as a letter {'**a**' ... '**h**'} followed by a number {'**1**' ... '**8**'}.

Note that square e1 is that occupied by the white king at the start of a game.

The output file should contain one line for each position described in the input file. Each line may contain one of two words :

check: if white's king is under attack
safe: if there is no piece threatening white's king

Continue....

....Continued

All the rules of chess apply. Note that check should be reported even if the threatening piece is pinned. You should assume that each position described in the input file is legal. That is, there are no errors such as two pieces occupying the same position, or the white king being absent.

Example files:

CHECK.IN	CHECK.OUT
kd3	check
pe4	safe
kb8	
nc5	
bg6	
kh3	
pf3	
rb3	
kf8	

1991 ACM Scholastic Programming Contest
European Regional Final

Problem H : WE GET AROUND ...

INPUT FILE : **TOUR.IN, BUS.IN**
OUTPUT FILE : **TOUR.OUT**
SOURCE CODE : **TOUR.PAS**

The Poucher family are on holiday in an unfamiliar part of the country and want to organise some round-trip excursions. However, the only information they have on possible routes is a leaflet listing all bus services to some nearby towns and their prices. The Pouchers have drawn up a list of places to visit on a tour, but they don't know which tours, if any, are possible by bus. Your program should combine the bus service data and their tour list to produce a list of possible tours and the total cost of each tour. All the bus services are two-way and run between exactly two towns. Since this is a sight-seeing holiday, the Pouchers only want to use a particular bus service once.

The bus service information will be in a text file BUS.IN. Each line of this file will contain two entirely alphabetic strings of maximum length 10 followed by a single integer, all separated by a single space. These represent the two towns connected by a bus service and the price of the journey. There are at most fifty such lines.

The Poucher's shortlist of towns is stored in TOUR.IN. This file contains a number of blocks, each block describing a tour. A block contains a number of place names, one per line. The first line indicates the town in which the tour must start and end. The remaining lines list the names of places which must be visited on the tour. These places must be visited at least once in the tour, but may be visited in any order. Blocks are separated from each other by one blank line.

For each of the blocks in the TOUR.IN file, your program should report either ...

Tour costs n franks.

where n is the lowest possible price for the tour, or

No route possible.

These responses should be written to TOUR.OUT, one per line

Continued....

Example

bus.in

```
lovendegem evergem 1
nevele lovendegem 1
gent evergem 3
gent lovendegem 4
nevele gent 4
gent wetteren 4
melle gent 2
melle aalst 6
wetteren aalst 8
aalst haaltert 1
haaltert opwijk 5
aalst opwijk 2
buggenhout opwijk 3
```

tour.in

```
gent
evergem
nevele

gent
aalst

haaltert
lovendegem

gent
buggenhout
```

tour.out

```
Tour costs 7 franks.
Tour costs 20 franks.
Tour costs 35 franks.
No route possible.
```