Task Overview Sheet

TASK		MEXICO	POINTS	BLACKBOX
Working directory		/home/ioi06/day2/mexico/	/home/ioi06/day2/points/	/home/ioi06/day2/blackbox/
Input file		mexico.in	points.in	N/A
Output file		mexico.out	points.out	blackboxK.out
Time limit per task		0.75 seconds	1 second	N/A
Memory limit		16 MB	32 MB	N/A
Compiler options	**C**	`-pipe –O2 -static –lm`		N/A
	C++	`-pipe –include /usr/include/stdlib.h –O2 –static –lm`		N/A
	Pascal	`-O1 –XS`		N/A
Detailed feedback enabled		NO	YES	NO
Maximum total points		100	100	100
Program header comment when using C		/* PROB: mexico LANG: C */	/* PROB: points LANG: C */	Not applicable, the output files you submit shall always begin with `#FILE blackbox K` where *K* is the number of the input file being solved
Program header comment when using C++		/* PROB: mexico LANG: C++ */	/* PROB: points LANG: C++ */	

Program header comment when using Pascal		{
		PROB: mexico
		LANG: PASCAL
		}
			{
			PROB: points
			LANG: PASCAL
			}	

Solution acceptance rule		Example input is solved correctly		Output file is formatted correctly

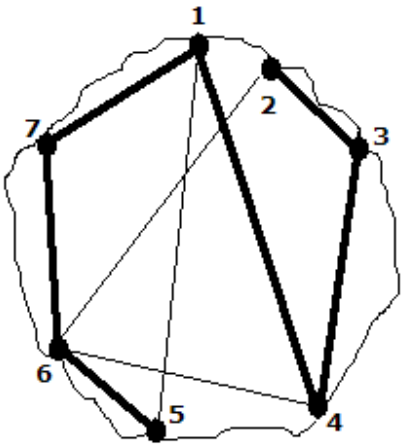
THE VALLEY OF MEXICO

Mexico City is built in a beautiful valley known as the Valley of Mexico which, years ago, was mostly a lake. Around the year 1300, Aztec religious leaders decreed that the lake's center be filled in order to build the capital of their empire. Today, the lake is completely covered.

Before the Aztecs arrived, *c* cities were located around the lake on its shores. Some of these cities established commercial agreements. Goods were traded, using boats, between cities that had a commercial agreement. It was possible to connect any two cities by a line segment through the lake.

Eventually, the kings of the cities decided to organize this commerce. They designed a commerce route that connected every city around the lake. The route met the following requirements:

- It could start in any of the cities, visited each of the cities around the lake, and finally ended in another city different from the starting city.
- The route visited each city exactly once.
- Every pair of consecutively visited cities in the route had a commercial agreement.
- Every pair of consecutively visited cities in the route was connected by a line segment.
- To avoid crashes between boats, the route never crossed itself.



The figure shows the lake and the cities around it. The lines (both thick and thin) represent commercial agreements between cities. The thick lines represent a commerce route starting in city 2 and ending in city 5.

This route never crosses itself. It would not be legal, for example, to construct a route that went from 2 to 6 to 5 to 1, since the route would cross itself.

Cities in the lake are numbered from 1 through *c* moving in clockwise direction.

TASK

Write a program that, given both the count *c* of cities and a list of the commercial agreements between them, constructs a commerce route that meets the above requirements.

CONSTRAINTS

$3 \le c \le 1000$ Number of cities around the lake.

INPUT

Your program must read the following data from the file `mexico.in`

`mexico.in`	DESCRIPTION
7 9 1 4 5 1 1 7 5 6 2 3 3 4 2 6 4 6 6 7	**LINE 1:** Contains integer *c* **LINE 2:** Contains an integer *n* that represents the number of commercial agreements **NEXT *n* LINES:** Each line represents a unique commercial agreement. Every line contains two space-separated integers that represent the two cities involved in the agreement.

OUTPUT

Your program must output the following data to the file `mexico.out`

`mexico.out`	DESCRIPTION
2 3 4 1 7 6 5	If it's possible to construct the commerce route, write *c* lines, each with an integer that represents the order in which the cities are visited in the commerce route. If it's not possible to construct a commerce route that meets all the requirements, output a single line containing the number `-1`.

NOTE: If there is more than one commerce route that meets the requirements, any of them you output will be considered correct.

GRADING

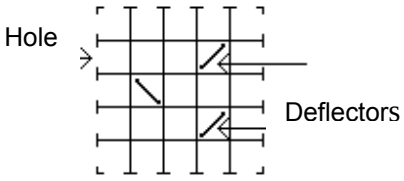
For a set of test cases worth a total of 40 points, each test case will meet the following requirements:

$3 \leq c \leq 20$

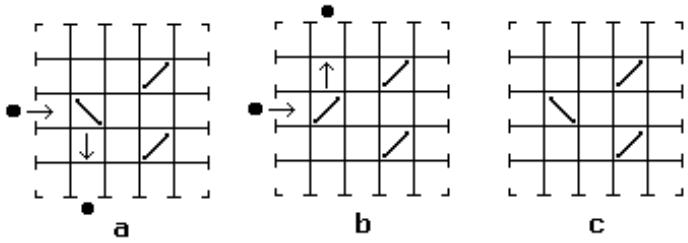
A BLACK BOX GAME

The Black Box Game is played with a square-shaped black box lying flat on a table. Each of its four sides has *n* holes (for a total of 4*n* holes) into which a ball can be thrown. A thrown ball will eventually exit from one of the 4*n* holes, potentially the same hole into which it was thrown.

The black box's internals can be envisioned as an *n* x *n* grid. The holes in the sides are the starts and ends of rows and columns. Each of the box's squares is either empty or occupied by a *deflector*. A deflector is a piece of hardware that changes the direction of the ball by 90 degrees. Consider this example of a 5x5 box.



A ball thrown into the box follows a straight line until it either hits a deflector or exits the box. When a ball hits a deflector, the ball changes direction and the deflector toggles its position (by "toggle" we mean rotate 90 degrees). The examples below show the action of a deflector.



a) A ball is thrown through a hole; it hits a deflector and changes direction.
b) After the first ball was thrown, the deflector has toggled its position. A new ball is thrown into the same hole, hits the deflector and is deflected in a direction opposite to that of the first ball.
c) The deflector toggles every time it is hit.

Whenever a deflector is hit, it makes a beep. The number of times the ball was deflected can be deduced by counting the beeps. It can be proved that the ball always exits the box. The box has a button that resets it to its original state and another button that toggles all of its deflectors.

TASK

You will be provided with an interface to 15 black boxes via a library of Pascal or C/C++ functions. You must determine the internals of each one of them as best as possible and submit a file describing each. You will also be provided with a method to define your own black boxes for testing.

CONSTRAINTS

$1 \le n \le 30$

OUTPUT

You must submit a file containing the following data for each one of the 15 black-boxes

`blackboxK.out`	DESCRIPTION

`#FILE blackbox K`
`.....`
`.../.`
`.\...`
`.../.`
`.?.?.`
	LINE 1: The file header. The file header must contain
	`#FILE blackbox K`
	where K (range 1..15) corresponds to the box being solved.
	***n* LINES:** Each line describes a row of the box, starting from the topmost row to the bottom row. Each line must contain exactly *n* characters; each character corresponds to a column (running from left to right).
	• '.' means that the square is empty.
	• '/' means the square contains a deflector with initial position '/'
	• '\' means the square contains a deflector with initial position '\'
	• '?' means that you were unable to determine the initial contents of that square

LIBRARY

You are given a library that provides the following functions

FUNCTION	Description

PASCAL
`function Initialize(box: integer): integer;`
C/C++
`int Initialize(int box);`
	Initializes the library, must be called once at the start of your program. It returns *n*, the number of holes on each side of the box.
	The parameter *box* must contain an integer between 1 and 15 indicating the box you want to use or 0 if you want to use a box created by you.

PASCAL
`function throwBall(holeIn, sideIn: integer;`
`var holeOut, sideOut: integer): longint;`
C
`int throwBall(int holeIn, int sideIn,`
`int *holeOut, int *sideOut);`
C++
`int throwBall(int holeIn, int sideIn,`
`int &holeOut, int &sideOut);`
	Throws a ball into the box through hole number *holeIn* in side *sideIn*, sides are numbered as 1 – Top, 2 – Right, 3 – Bottom and 4 – Left. Holes are numbered from left to right and from top to bottom starting from number 1 in each side. In *holeOut* and *sideOut* you'll receive the hole and side number where the ball exits the box. The function *throwBall* returns the number of beeps caused by the ball hitting a deflector.

PASCAL
`procedure ResetBox;`
C/C++
`void ResetBox();`
	Resets every deflector in the box to its initial position.

PASCAL
`procedure ToggleDeflectors;`
C/C++
`void ToggleDeflectors();`
	Toggles every deflector in the box.

PASCAL
`procedure Finalize;`
C/C++
`void Finalize();`
	Gracefully ends the interaction with the box. It should be called at the end of your program.

To be able to use the library in your program do as follows:

- **FreePascal:** In the task directory you will find the files `pbblib.o` and `pbblib.ppu` to be able to use them include the following statement.
  ```
  uses pbblib;
  ```
  The file `pblackbox.pas` gives an example of how to use the library.

- **C:** In the task directory you will find the files `cbblib.o` and `cbblib.h`, to be able to use them include the following statement in your code.
  ```
  #include "cbblib.h"
  ```
  The file `cblackbox.c` gives an example of how to use the library. In order to compile your code you will need to use the following command
  ```
  gcc –o yourprogram cbblib.o yourprogram.c
  ```

- **C++:** In the task directory you will find the files `cppbblib.o` and `cppbblib.h`, to be able to use them include the following statement in your code.
  ```
  #include "cppbblib.h"
  ```
  The file `cppblackbox.cpp` gives an example of how to use the library. In order to compile your code you will need to use the following command
  ```
  g++ –o yourprogram cppbblib.o yourprogram.cpp
  ```

NOTE: At any given time only one program using the library can be running.

SAMPLE INTERACTION

A sample interaction for the box in the previous figure could be:

FUNCTION CALL	VALUE RETURNED BY FUNCTION
`Initialize(0);`	Assuming that the box used is the one in the previous figure. Returns 5 indicating that *n* = 5

PASCAL
`throwBall(3,4,holeOut,sideOut);`
C
`throwBall(3,4,&holeOut,&sideOut);`
C++
`throwBall(3,4,holeOut,sideOut);`
	A ball is thrown into hole number 3 (third from the top) on the left side. Returns 1, indicating that the ball hit 1 deflector. When the function returns, *holeOut* will equal 2 and *sideOut* will equal 3 indicating that the ball exited through hole 2 (second from the left) of the bottom side of the box.

EXPERIMENTATION

If you pass an integer value of 0 to the function `Initialize`, the library will read the box internals from the file `blackbox.in`. In this way you can experiment with the library. The format for the file `blackbox.in` is described below.

`blackbox.in`	DESCRIPTION

`5`
`3`
`2 3 \`
`4 2 /`
`4 4 /`
	LINE 1: Contains *n,* the number of holes on each side.
	LINE 2: Contains an integer *d* that indicates the number of deflectors in the box.
	NEXT *d* LINES: There must be one line for every deflector in the box. Each line contains two space-separated integers that represent the column and the row of the deflector respectively, and a character separated with a space from the second integer that can be either '/' or '\' that represents the original position of the deflector.

NOTE; The example blackbox.in describes the black box shown in the figure at the top of page 1.

ERROR MESSAGES

In the case of any anomalies the library will output an error message to the standard error. The possible error message that you can get and their meanings are shown in the table below.

ERROR MESSAGE	MEANING
ERR 1 More than one app	Only one application at a time can interact with the black boxes, please restart all applications and start only one at a time
ERR 2 Invalid box	The box number you input is not in the range from 0 to 15
ERR 3 Invalid deflector	The file blackbox.in has a deflector in an invalid position
ERR 4 Invalid symbol	The file blackbox.in has an invalid symbol
ERR 5 Invalid size	The size of the black box in blackbox.in is invalid.
ERR 6 Invalid input hole	Either the side or the input hole that you entered are invalid
ERR 7 ALARM	Please call the technical staff

GRADING

For each box you must submit a text file that describes the internals of the black box as best as possible. For each box:

- If your submission has a '.', '/' or '\' character in an incorrect position, you'll get zero points for that box.
- Let B_m be the maximum number of discovered positions among all of the correct submissions, and let B_y be the number of discovered positions in your submission, then percentage of your score for that box will be

$$100\ B_y / B_m$$

NOTE: The official solution for this task can programmatically discover 100% of the initial contents of any of the boxes in a time less than 8 minutes.

JOINING POINTS

"Joining points" is a single-player game. To play it, choose two integers greater than two and call them *g* and *r.* Then draw four points at the vertices of a square making the top two points green and the bottom two points red. Draw green points and red points inside the square taking care that no three points, including the four initial ones, are in the same line. Continue until the total number of green points equals *g* and the total number of red points equals *r*.

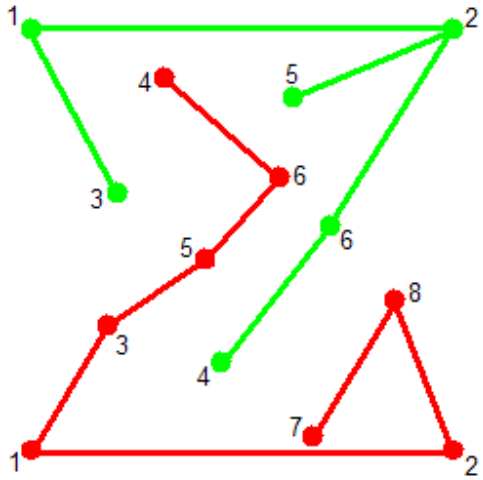
After the board is drawn, start joining points. Any two points can be joined by a line segment as long as:

- The two points to be joined are of the same color, and
- The line segment joining the points does not intersect any other previously drawn line segment (other than at the endpoints).

Two points *u* and *v* are said to be in the same *component* if it is possible to traverse from point *u* to point *v* using the line segments already drawn.

You win the game if you get all the green points in one component using exactly *g*-1 line segments, and all the red points in another component using exactly *r*-1 line segments. **It can be proven that if the points are drawn as described above, then there is always a way to win the game.**

You will be given a square board of size *s* with *g* green and *r* red points whose coordinates are represented by integer pairs (x_i, y_i). Green points are numbered from 1 to *g* with the top-left point at (0,*s*) being 1, the top-right at (*s*,*s*) being 2, and interior points ranging from 3 to *g* in arbitrary order. Red points are numbered from 1 to *r* with the bottom-left point at (0,0) being 1, the bottom-right at (*s*,0) being 2, and interior points ranging from 3 to *r* in arbitrary order.



The figure shows a sample game all the green points are joined into one component and all the red points are joined into another component.

You can see that no three points are in the same line, and that no two line segments intersect each other except at their endpoints.

TASK

Write a program that, given the coordinates of the *g* green points and the coordinates of the *r* red points, decides how to draw *g* - 1 green line segments and *r* - 1 red line segments so that all of the green points are in the same component, all of the red points are in another component, and no two line segments intersect each other.

CONSTRAINTS

$3 \le g \le 50\,000$	Number of green points.
$3 \le r \le 50\,000$	Number of red points.
$0 < s \le 200\,000\,000$	

INPUT

Your program must read the following data from the file `points.in`

`points.in`	DESCRIPTION

`6`
`0 1000`
`1000 1000`
`203 601`
`449 212`
`620 837`
`708 537`
`8`
`0 0`
`1000 0`
`185 300`
`314 888`
`416 458`
`614 622`
`683 95`
`838 400`
	LINE 1: Contains the integer *g.*
	NEXT *g* LINES: Each line contains two space-separated integers that represent the coordinates x_i and y_i of each of the *g* green points, starting from 1 to *g.*
	LINE *g+2:* Contains integer *r*.
	NEXT *r* LINES: Each line contains two space-separated integers that represent the coordinates x_i and y_i of each of the *r* red points, starting from 1 to *r.*

OUTPUT

Your program must write the following data to the file `points.out`

`points.out`	DESCRIPTION

`1 3 g`
`3 1 r`
`3 5 r`
`4 6 r`
`6 5 r`
`4 6 g`
`1 2 g`
`1 2 r`
`5 2 g`
`2 6 g`
`7 8 r`
`8 2 r`
	Your output file must contain $(g - 1) + (r - 1)$ lines, one for every line segment drawn to join points.
	Each line must contain three space-separated entities: two integers and a character.The two integers represent the numbers of two points joined by that line segment. The character must be a ***g*** if the points are green and an ***r*** if the joined points are red.
	The order in which you list the line segments does not matter; neither does the order of the endpoints of each line segment.

GRADING

For a set of test cases worth a total of 35 points, every test run will meet the following requirements:
$3 \le g \le 20$
$3 \le r \le 20$