Electric Fence

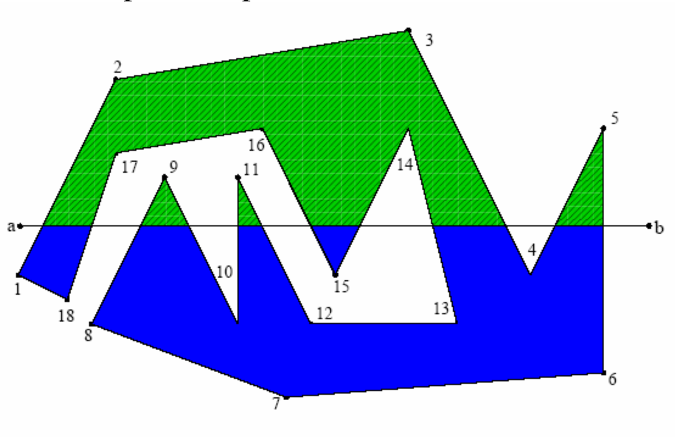
Input file: --
Output file: --
Source code: fence.pas/.c/.cpp

100 points
Time limit: 3 sec
Memory limit: 64 MB

Farmer G has a large pasture-field surrounded by an electric fence. The fence consists of fence-posts and straight-line segments of wires where each segment connects two neighboring posts. The fence is obviously not self-crossing, i.e. no wire-segment crosses any other wire-segment. Farmer G had been informed that a new straight-line road will be built that might cross his field. He went to the field and noticed that the two endpoints of the road have already been marked by two posts, ***a*** and ***b***. He realized that the line of the road splits the interior part of his pasture-field into several disjoint regions.

Farmer G wants to determine how many regions will form on both sides of the road. He finds that none of the fence-posts lay on the line of the road. Moreover, if a wire-segment intersects with the line of the road, then the intersection lays between the endpoints ***a*** and **b**.

Unfortunately, Farmer G has no instrument to measure the distance of two posts. He can only observe the orientation of the posts, i.e. he can walk to any post ***p*** (recall that road endpoints are also posts) and, looking towards post ***q***, he can see whether a third post ***r*** stands on his left, or on his right, or whether the three posts are collinear. Fortunately, farmer G has his laptop with him (as usual), so he can do even complex computations.



Task

You are to write a program that computes the number of disjoint regions located on the left and on the right side of the planned road as the result of splitting the pasture-field by the road.

Library

To perform queries, you are given a library **`lookup`** with three operations:

- **`GetN`**, to be called once at the beginning, without arguments; it returns ***N***, the number of fence-posts. **`GetN`** must be called before the first call to **`Drift`**.
- **`Drift`**, to be called with three post labels as arguments. **`Drift(x,y,z)`** returns ***1*** if post ***z*** stands on the left when looking from ***x*** towards post ***y***, it returns ***-1*** if ***z*** stands on the right, and it returns ***0*** if the three posts are collinear. The fence-posts are labeled by the numbers from ***1*** to ***N***, the road endpoints ***a*** and ***b*** are labeled by ***N+1*** and by ***N+2,*** respectively. The wire-segments of the fence connect fence-posts labeled by ***i*** and (***i*** modulo ***N)+1***. **`Drift`** returns ***0***, too, if at least two of its arguments are equal.

- **`Answer`**, to be called once in the end; it reports the solution and it properly terminates the execution of your program. **`Answer`** has two integer arguments. The first and second arguments must be the number of disjoint regions located on the left and right side of the road, respectively. (**`Drift(a,b,p)`** returns ***1*** or ***-1*** if fence-post ***p*** stands on the left or right side of the road, respectively.)

Your program is not allowed to read or write files. Input and output is handled by the library.

Instruction for Pascal programmers: include the import statement

```
uses lookup;
```

in your source code.

Instructions for C/C++ programmers: use the directive

```
#include ”lookup.h”
```

in your source code, create a project file in the task directory, add the files `fence.c` (`fence.cpp`), `lookup.h` and `lookup.o` into this project, and then *compile* and/or *make* your program. (Using Dev-C++ IDE, choose the Project/Project Options/Files menu, select the file `lookup.o,` unset “include in compilation” and set “include in linking”).

Command line compilation:

```
gcc/g++ -O2 -static -o fence fence.c lookup.o -lm
```

Experimentation

You are provided with a toolset that contains the libraries both for WinXP and Linux. You can download it from the competition server as a zip archive. Copy the appropriate library files into your task directory.

The toolset includes a test generator **`testgener`** to produce the file `fence.in` containing valid random sample input. **`testgener`** needs an integer input parameter, ***N***, the approximate number of fence-posts. If ***N < 300*** then **`testgener`** also creates a postscript file `fence.ps` that visualizes the layout of the fence (you can view it using gsview or another postscript viewer). The generated test data is considerably different for even and odd ***N***s; try and see it! Warning: **`testgener`** can not generate all possible inputs.

The solution submitted by **`Answer`** will be written into the file `fence.out`.

You can also create your own input by creating a text file `fence.in`. The first line must contain four integers, the coordinates of the endpoints of the road. The second line must contain ***N***, the number of fence-posts. Each of the following ***N*** lines must contain a pair of integers, ***x y*** ($-20\,000 \le x, y \le 20\,000$); the pair in line ***i+2*** defines the coordinates of the fence-post labeled by ***i***.

Constraints

- For the number of fence-posts ***N,*** we have $3 \le N \le 100\,000$.
- FreePascal library file names: `lookup.ppu` and `lookup.o` for WinXP and `lookup.o` for Linux.
- Pascal function declarations:
```
function GetN: longint;
function Drift(x, y, z: longint): integer;
procedure Answer(x, y: longint);
```
- C/C++ library file names: `lookup.h`, and `lookup.o`
- C/C++ function declarations:
```
long GetN(void);
int Drift(long x, long y, long z);
void Answer(long left, long right);
```