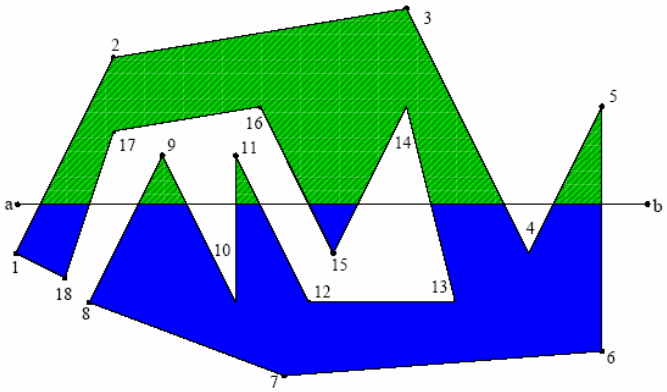
Electric Fence (Verja eléctrica)

Input file: `--` **100 points**
Output file: `--` **Time limit: 3 sec**
Source code: `fence.pas/.c/.cpp` **Memory limit: 64 MB**

El granjero G tiene un gran campo de pasto rodeado por una verja eléctrica. La verja consiste en postes y segmentos rectilíneos de cables donde cada segmento conecta a dos postes vecinos. Obviamente, la verja no se cruza a ella misma, por lo que ningún segmento cruza a otro segmento. Han informado al granjero G que se va a construir una nueva carretera rectilínea, y que podría atravesar a su campo. El fue a su campo y verifico que los dos extremos de la carretera habían sido marcados mediante dos postes, ***a*** y ***b***. Se dio cuenta que la línea de la carretera divide la parte interior de su campo en diversas regiones disjuntas.

El granjero G quiere determinar cuantas regiones se formarán a ambos lados de la carretera. Ve que ningún poste de la verja está sobre la línea de la carretera. Además si un segmento de la verja intersecciona con la línea de la carretera, el punto de intersección está entre los extremos ***a*** y **b**.

Desgraciadamente, el granjero G no dispone de instrumentos para medir la distancia entre dos postes. El solo puede observar la orientación de los postes, y puede caminar hacía cualquier poste ***p*** (recuerda que los extremos de la carretera son también postes) y, mirando hacía un poste ***q***, puede ver si un tercer poste ***r*** está situado a su izquierda o a su derecha o si los tres son colineales. Afortunadamente, el granjero G tiene su portátil y efectua complejas computaciones.



Task

Escribe un programa que calcule el número de regiones disjuntas localizadas a la izquierda y a la derecha de la carretera, como resultado de dividir el campo de pasto por la carretera.

Library

Para realizar llamadas, te dan la librería **`lookup`** con tres operaciones:

- **`GetN`**, se llama una vez al principio, sin argumentos; devuelve ***N***, el número de postes de la verja. **`GetN`** debe llamarse antes de la primera llamada a **`Drift`**.
- **`Drift`**, debe llamarse con tres etiquetas de poste como argumentos. **`Drift(x,y,z)`** devuelve ***1*** si el poste ***z*** está a la izquierda cuando miro desde el poste ***x*** hacía el poste ***y***, devuelve ***-1*** si ***z*** está a la derecha, y devuelve ***0*** si los tres postes son colineales. Los postes de la verja están etiquetados con los números de ***1*** a ***N***, los extremos de la carretera ***a*** y ***b*** están etiquetados mediante ***N+1*** y ***N+2,*** respectivamente. Los segmentos de la verja conectan postes etiquetados por ***i*** y

(***i*** módulo ***N)*** + ***1***. **`Drift`** devuelve ***0***, también, si por lo menos dos de sus argumentos son iguales.

- **`Answer`**, se llama una sola vez al final; devuelve la solución y termina la ejecución de tu programa. **`Answer`** tiene dos argumentos enteros. El primer y el Segundo argumento debe ser el número de regions disjuntas situadas a la izquierda y a a la derecha de la carretera, respectivamente. (**`Drift(a,b,p)`** devuelve ***1*** o ***-1*** si el poste ***p*** está a la izquierda o a la derecha de la carretera, respectivamente.)

Instrucción para Pascal: incluye la sentencia

```
uses lookup;
```

en tu código.

Instrucciones para C/C++: usa la directiva

```
#include "lookup.h"
```

en tu código fuente, crea un proyecto en el directorio de trabajo, añade los ficheros `fence.c` (`fence.cpp`), `lookup.h` y `lookup.o` en este proyecto, y entonces *compile* y/o *make* tu programa. (Using Dev-C++ IDE, escoje el Project/Project Options/Files menu, selecciona el fichero `lookup.o`, desactiva "include in compilation" y activa "include in linking").

Command line compilation:

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

Experimentation

Proveemos las herramientas que contienen las librerías para WinXP y Linux. Puedes bajarlo desde el servidor de la competición como un fichero zip. Copia los ficheros de la librería adecuada en tu directorio de trabajo.

Las heramientas incluyen el generador de juegos de prueba **`testgener`** que produce el fichero `fence.in` que contiene un ejemplo al azar válido. **`testgener`** necesita un parámetro entero de entrada, ***N***, el número aproximado de postes de verja. Si ***N < 300*** entonces **`testgener`** también crea un fichero postscript `fence.ps` que muestra el aspecto de la verja (puedes verlo usando gsview u otro visualizador postscript). El juego de prueba generado es diferente para ***N's*** pares e impares; pruebalo y miralo! Atención: **`testgener`** no puede generar todos los posibles juegos de prueba.

La solución enviada por **`Answer`** se escribirá en el fichero `fence.out`.

También puedes crear tu propio juego de prueba, creando el fichero `fence.in`. La primera línea debe contener cuatro enteros, las coordenadas de los extremos de la carretera. La segunda línea debe contener ***N***, el número de postes de la verja. Cada una de las siguientes ***N*** líneas deben contener un par de enteros, ***x y*** (***-20 000 ≤ x, y ≤ 20 000***); el par en la línea ***i+2*** define las coordenadas del poste etiquetado con ***i***.

Constraints

- Para el número de postes de verja ***N,*** tenemos que $3 \le N \le 100\,000$.
- 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);
```