A
Poli y cacos

Vamos a simular el juego de polis y cacos, en el que un policía persigue a ladrones que se esconden en diferentes lugares. Cada caco posee un nombre único, una velocidad única y comienza en un lugar escondido. Éste puede descubrir nuevos lugares a los que escapar y abandonar su escondite actual cuando lo considere oportuno para ir al siguiente que descubrió cronológicamente. Además, los cacos pueden abandonar la partida en cualquier momento. Por otro lado, el poli visita un lugar y atrapa al caco que esté escondido con menor velocidad. Por último, el poli de la siguiente partida será aquel caco que haya sido pillado primero, y que no haya abandonado la partida.

Se pide implementar un TAD `PoliCacos` que implemente las siguientes operaciones:

- `void nuevo_caco(string j, int velocidad, string primer_lugar)`: Crea un nuevo caco con el nombre dado, una velocidad específica y ubicado en el primer lugar de escondite. Lanza una excepción de tipo `std::invalid_argument`, con el texto `caco ya existe`, si el jugador `j` ya estaba registrado. cierto.

- `void descubre_nuevo_lugar(string j, string nuevo_lugar)`: El jugador `j` descubre un nuevo lugar al que puede escapar. Se garantiza que un caco nunca va a descubrir dos veces el mismo lugar. Lanza una excepción de tipo `std::invalid_argument`, con el texto `caco no existe`, si el jugador `j` no existe. Si el jugador ya ha sido pillado, lanza una excepción con el texto `caco pillado`.

- `string cambia_lugar(string j)`: El jugador `j` cambia al siguiente lugar de escondite siguiendo el orden de descubrimiento. Si ya ha visitado todos los lugares, vuelve a empezar desde el primer lugar que visitó. Devuelve el nombre del nuevo lugar. Lanza una excepción de tipo `std::invalid_argument`, con el texto `caco no existente`, si el jugador `j` no existe. Si el jugador ya ha sido pillado, lanza una excepción con el texto `caco pillado`.

- `void abandona_juego(string j)`: Elimina al jugador `j` del juego y borra su información de todas las estructuras de datos utilizadas.

- `string poli_busca_en(string l)`: El policía visita un lugar y captura al ladrón con la velocidad más baja en ese lugar. Devuelve el nombre del ladrón capturado. Lanza una excepción de tipo `sstd::invalid_argument`, con el texto `lugar vacío`, si el lugar `l` estaba vacío.
  Importante: Al ser capturado, el ladrón deja de estar escondido en l.
- `string siguiente_poli()`: Devuelve el nombre del primer ladrón que fue pillado y que permanece en el juego actualmente o una excepción con el texto `nadie ha sido pillado` si ningún jugador actual ha sido pillado.

Requisitos de implementación.

La implementación de las operaciones debe ser lo más eficiente posible. Por tanto, debes elegir una representación adecuada para el TAD, implementar las operaciones y **justificar la complejidad resultante**.

Los métodos del TAD no deben mostrar nada por pantalla. El manejo de la entrada y salida de datos se realizará en funciones externas al TAD.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso está formado por una serie de líneas, en las que se muestran las operaciones a llevar a cabo, una por cada línea: el nombre de la operación seguido de sus argumentos. La palabra `FIN` en una línea indica el final de cada caso.

Los nombres de los cacos y los lugares son cadenas de caracteres sin blancos. El valor de velocidad es un número entero positivo menor que 10^9.

Salida

Para cada caso de prueba se escribirán los datos que se piden. La comprobación de las excepciones indicadas hay que llevarlas a cabo en el orden indicado.

La salida de las operaciones son:

- `void nuevo_caco(string j, int velocidad, string primer_lugar)`: En caso de no saltar la excepción se mostrará `j es el nuevo jugador`.
- `void descubre_nuevo_lugar(string j, string nuevo_lugar)`: En caso de no saltar la excepción, se mostrará `j descubre nuevo_lugar`.
- `string cambia_lugar(string j)`: En caso de no saltar la excepción, se mostrará `j se refugia siguiente_lugar`, donde `siguiente_lugar` es el string que devuelve la operación.
- `void abandona_juego(string j)`: En caso de no saltar la excepción, se mostrará `j abandona el juego`.
- `string poli_busca_en(string l)`: En caso de no saltar la excepción, se muestra `j ha sido cazado`, donde `j` es el string que devuelve la operación.
- `string siguiente_poli()`: En caso de no saltar la excepción, se muestra `j es el siguiente poli`, donde `j` es el string que devuelve la operación.

Si alguna operación produce una excepción se mostrará el mensaje `ERROR:` seguido del mensaje de la excepción como resultado de la operación, y nada más.

Cada caso termina con una línea con tres guiones (`---`).

Entrada de ejemplo

```
nuevo_caco JUAN 20 bajo_la_mesa
nuevo_caco JUAN 30 en_la_chimenea
poli_busca_en bajo_la_mesa
FIN

descubre_nuevo_lugar PEDRO bajo_la_mesa
poli_busca_en en_el_coche
nuevo_caco JUAN 30 en_la_chimenea
abandona_juego JUAN
poli_busca_en en_la_chimenea
FIN

nuevo_caco JUAN 20 bajo_la_mesa
descubre_nuevo_lugar JUAN en_la_chimenea
cambia_lugar JUAN
nuevo_caco PEDRO 30 bajo_la_mesa
poli_busca_en bajo_la_mesa
siguiente_poli
abandona_juego PEDRO
poli_busca_en en_la_chimenea
siguiente_poli
FIN
```

Salida de ejemplo

```
JUAN es el nuevo jugador
ERROR: caco ya existe
JUAN ha sido cazado
---
ERROR: caco no existe
ERROR: lugar vacio
JUAN es el nuevo jugador
JUAN abandona el juego
ERROR: lugar vacio
---
JUAN es el nuevo jugador
JUAN descubre en_la_chimenea
JUAN se refugia en_la_chimenea
PEDRO es el nuevo jugador
PEDRO ha sido cazado
PEDRO es el siguiente poli
PEDRO abandona el juego
JUAN ha sido cazado
JUAN es el siguiente poli
---
```