Misterios de Pekín *(1ª parte)*

Misterios de Pekín es un juego de mesa para niños, en el que los jugadores toman el rol de detectives que deben atrapar al culpable de un crimen. Para ello, los jugadores disponen de una lista de sospechosos y saben que el culpable está entre ellos. Cada sospechoso puede tener una serie de rasgos en su apariencia física (color de pelo, bigote, cicatriz, etc.) y cada jugador ha de interrogar a distintos testigos repartidos a lo largo del tablero con el fin de obtener información sobre el culpable. Las pistas proporcionadas por los testigos permiten a un jugador descartar a algunos de los sospechosos en base a sus rasgos, aunque algunos testigos podrían mentir en sus declaraciones. Cuando un jugador ha descartado a todos los sospechosos menos uno, puede proceder a la detención del culpable.

En este ejercicio has de implementar un TAD `MisteriosDePekin` que permita almacenar la información del juego. Debes implementar cada una de las operaciones que se describen a continuación e indicar su **coste** en tiempo.

- `MisteriosDePekin(const string &culpable)` *(constructor)*

  Construye una nueva instancia del juego, en el que `culpable` es el nombre de la persona que ha cometido el crimen. Esta persona se registra como una de las sospechosas.

- `void anyadir_rasgo(const string &sospechoso, const string &rasgo)`

  Indica que el `sospechoso` dado tiene el `rasgo` pasado como parámetro. Si el sospechoso no se encontraba registrado en el sistema, este método lo registrará.

  Solo es posible llamar a este método si no hay jugadores registrados en el sistema. De lo contrario, esta operación lanzará una excepción `domain_error` con el mensaje `Juego ya empezado`.

- `vector<string> sospechosos() const`

  Devuelve la lista con los nombres de los sospechosos registrados en el TAD. Esta lista debe estar ordenada por orden lexicográfico (el orden definido en los `string`).

- `void nuevo_jugador(const string &nombre)`

  Registra un nuevo jugador con el `nombre` dado en el sistema. Si el jugador ya existe, debe lanzar una excepción `domain_error` con el mensaje `Jugador existente`.

- `void jugador_descarta(const string &jugador, const string &rasgo)`

  Indica que el `jugador` dado, tras preguntar a un testigo, ha decidido descartar a los sospechosos que tengan el `rasgo` pasado como parámetro. Si el jugador no está registrado en el sistema, se lanzará una excepción `domain_error` con el mensaje `Jugador no existente`.

- `bool jugador_enganyado(const string &jugador) const`

  Indica si el `jugador`, en una de las llamadas previas a `jugador_descarta`, ha descartado al sospechoso culpable del juego. Si el jugador no está registrado en el sistema, se lanzará una excepción `domain_error` con el mensaje `Jugador no existente`.

- `bool puede_detener_culpable(const string &jugador) const`

  Indica si el `jugador` pasado como parámetro ha descartado a todos los sospechosos menos al culpable y, por tanto, puede proceder a la detención de este último. Si el jugador no está registrado en el sistema, se lanzará una excepción `domain_error` con el mensaje `Jugador no existente`.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba comienza con una línea que indica el nombre de la persona culpable del crimen, que será la que debe pasarse al constructor del TAD. A continuación aparecen una serie de líneas en las que se describen las operaciones que deben invocarse en el TAD. Cada línea consiste en el nombre de la operación seguido de sus argumentos. Suponemos que los nombres de sospechosos, jugadores y rasgos están formados por caracteres alfanuméricos sin espacios. Cada caso de prueba finaliza con una línea con la palabra `FIN`.

Salida

Tras llamar al constructor del TAD debe imprimirse una línea con el texto `OK`. Después debe imprimirse una línea con el resultado de cada operación:

- `sospechosos`: Deben imprimirse los elementos de la lista de sospechosos, separados por espacios.
- `jugador_enganyado`: Debe imprimirse `J ha sido enganyado` o `J no ha sido enganyado`, según el resultado de la operación, donde `J` es el nombre del jugador pasado como parámetro.
- `puede_detener_culpable`: Debe imprimirse `J puede detener al culpable` o `J no puede detener al culpable`, según el resultado de la operación, donde `J` es el nombre del jugador pasado como parámetro.
- Resto de operaciones: debe imprimirse `OK`, salvo que la operación haya producido una excepción.

Si alguna de las operaciones produce una excepción, debe imprimirse una línea con el texto `ERROR:` seguido del mensaje de la excepción, y no debe imprimirse nada más para esa operación.

Al finalizar el caso de prueba, debe imprimirse una línea con tres guiones: `---`.

Entrada de ejemplo

```
ChenXin
anyadir_rasgo ZhouJian Hombre
anyadir_rasgo ChenXin Mujer
anyadir_rasgo LiKun Gafas
anyadir_rasgo ChenXin LlevaSombrero
anyadir_rasgo LiKun Bigote
anyadir_rasgo ZhouJian LlevaSombrero
anyadir_rasgo ZhouJian Bigote
anyadir_rasgo LiKun Calvo
nuevo_jugador Wolfgang
nuevo_jugador Markus
anyadir_rasgo ChenXin PeloMoreno
sospechosos
jugador_descarta Markus Gafas
jugador_enganyado Markus
jugador_descarta Markus LlevaSombrero
jugador_enganyado Markus
puede_detener_culpable Wolfgang
jugador_descarta Wolfgang Bigote
puede_detener_culpable Wolfgang
FIN
LiKun
anyadir_rasgo ZhouJian Hombre
anyadir_rasgo LiKun Gafas
nuevo_jugador Peter
jugador_descarta Peter Gafas
nuevo_jugador Peter
puede_detener_culpable Peter
puede_detener_culpable Martha
jugador_enganyado Peter
FIN
```

Salida de ejemplo

```
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
ERROR: Juego ya empezado
ChenXin LiKun ZhouJian
OK
Markus no ha sido enganyado
OK
Markus ha sido enganyado
Wolfgang no puede detener al culpable
OK
Wolfgang puede detener al culpable
---
OK
OK
OK
OK
OK
ERROR: Jugador existente
Peter no puede detener al culpable
ERROR: Jugador no existente
Peter ha sido enganyado
---
```

Créditos

Manuel Montenegro