Ejercicio 3

3 puntos

Debemos simular un sistema de batallas por turnos entre héroes y villanos del juego *Fantasia Final*. Cuando un héroe o un villano aparecen en la batalla, tienen que esperar a que todos los personajes que estuvieran presentes anteriormente en la batalla ataquen. El personaje al que le toca el turno, ya sea héroe o villano, realiza un ataque y vuelve a esperar su turno para volver a atacar. Si el ataque derrota al adversario, éste desaparece definitivamente de la batalla. Los villanos tienen un nombre, una determinada cantidad de puntos de vida y un valor de daño al atacar. Los héroes, por otro lado, además de su nombre y sus puntos de vida, tienen una lista de ataques que van aprendiendo a lo largo de la batalla, cada uno con su valor de daño correspondiente. El nombre de cada personaje es único, es decir, no existen un villano y un héroe con el mismo nombre.

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

- `aparece_villano(v,s,a)`: Registra el nuevo villano `v` en la batalla con `s` puntos de vida y valor de ataque `a`. Si el villano no había aparecido en la batalla, se registra. Si ya había un personaje (héroe o villano) en la batalla con ese mismo nombre, se lanza una excepción del tipo `invalid_argument` con el mensaje `Personaje ya existente`.
- `aparece_heroe(h,s)`: Registra el nuevo héroe `h` en la batalla con `s` puntos de vida. Si el héroe no había aparecido en la batalla, se registra. Si ya existe otro personaje en la batalla con el mismo nombre, se lanza una excepción del tipo `invalid_argument` con el mensaje `Personaje ya existente`.
- `aprende_ataque(h,a,v)`: Añade el ataque `a` con daño `v` a la lista de ataques del héroe `h`. Si el héroe no está en la batalla, se lanza una excepción del tipo `invalid_argument` con el mensaje `Heroe inexistente`. Si el héroe conoce ya ese ataque, se lanza una excepción del tipo `invalid_argument` con el mensaje `Ataque repetido`.
- `mostrar_ataques(h)`: Devuelve un vector de pares *(nombre, daño)* con los ataques aprendidos por el héroe `h`, ordenados lexicográficamente por nombre. Si el héroe no está registrado, se lanza una excepción del tipo `invalid_argument` con el mensaje `Heroe inexistente`.
- `mostrar_turnos()`: Devuelve un vector de pares *(nombre, puntos_vida)* con el nombre de todos los héroes y villanos acompañados de su cantidad correspondiente de puntos de vida, ordenados según su turno. El primer elemento de la lista corresponde al personaje que tiene el turno, el segundo elemento contiene el personaje al que le toca después, y así sucesivamente.
- `villano_ataca(v,h)`: El villano `v` ataca al héroe `h`, restándole a los puntos de vida del héroe el valor de daño de `v`. Tras el ataque, `v` se vuelve a colocar al final del turno. Si el villano o el héroe no están registrados, se lanza una excepción con el mensaje `Villano inexistente` o `Heroe inexistente`, respectivamente. Si no es el turno del villano pasado como parámetro, se lanza una excepción con el mensaje `No es su turno`. Si el héroe atacado pierde todos sus puntos de vida, desaparece completamente de la batalla y la función devuelve el valor `true`. En caso contrario, devuelve `false`.
- `heroe_ataca(h,a,v)`: El héroe `h` ataca al villano `v`, restándole a los puntos de vida del villano el valor de daño del ataque `a`. Tras el ataque, `h` se vuelve a colocar al final del turno. Si el villano o el héroe no están registrados, se lanza una excepción con el mensaje `Villano inexistente` o `Heroe inexistente`, respectivamente. Si no es el turno del héroe pasado como parámetro, se lanza una excepción con el mensaje `No es su turno`. Si el héroe `h` no conoce el ataque `a`, se lanza una excepción con el mensaje `Ataque no aprendido`. Si el villano atacado pierde todos sus puntos

de vida, desaparece completamente de la batalla y la función devuelve el valor `true`. En caso contrario, devuelve `false`.

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 el **coste** de cada una de ellas.

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 héroes y villanos son cadenas de caracteres sin blancos. Las cantidades de puntos de vida y los valores de ataque son números enteros positivos menores que 10^9.

Salida

Para cada caso de prueba se escribirán los datos que se piden. Las operaciones `aparece_villano`, `aparece_heroe`, y `aprende_ataque` no imprimen nada, salvo en caso de error. La comprobación de las excepciones indicadas hay que llevarlas a cabo en el orden indicado.

Las operaciones que generan datos de salida son:

- `mostrar_ataques`: si no ha producido ninguna excepción, imprime una primera línea con el mensaje `Ataques de` y el nombre del héroe, seguida de varias líneas (ordenadas alfabéticamente) con el nombre y daño de cada ataque.
- `mostrar_turnos`: imprime una primera línea con `Turno:`, seguida de cada nombre y la cantidad de puntos de vida de héroes y villanos, en orden del turno que tienen.
- `villano_ataca`: si no ha producido ninguna excepción, imprime una línea con el nombre del villano, seguido de `ataca a`, seguido del nombre del héroe. En caso de que el héroe sea derrotado se imprime una segunda línea con el nombre del héroe seguido de la palabra `derrotado`.
- `heroe_ataca`: si no ha producido ninguna excepción, imprime una línea con el nombre del héroe, seguido de `ataca a`, seguido del nombre del villano. En caso de que el villano sea derrotado se imprime una segunda línea con el nombre del villano seguido de la palabra `derrotado`.

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

```
aparece_villano seymour 10 6
aparece_villano edea 12 10
aparece_heroe tidus 30
aprende_ataque tidus ataque_superpoderoso 10
aprende_ataque tidus ataque_impresionante 20
mostrar_ataques tidus
mostrar_turnos
villano_ataca seymour tidus
villano_ataca edea tidus
mostrar_turnos
heroe_ataca tidus ataque_impresionante seymour
villano_ataca edea tidus
mostrar_turnos
heroe_ataca tidus ataque_superpoderoso edea
mostrar_turnos
FIN
aparece_villano v1 5 4
aparece_heroe h1 3
aparece_villano v1 7 20
aparece_heroe h1 10
aprende_ataque h2 a1 10
aprende_ataque h1 a1 10
mostrar_ataques h2
villano_ataca v1 v2
heroe_ataca h1 a1 h2
villano_ataca v1 h1
heroe_ataca h1 a1 v1
aparece_heroe h1 10
heroe_ataca h1 a1 v1
villano_ataca v1 h1
heroe_ataca h1 a1 v1
aprende_ataque h1 a1 30
heroe_ataca h1 a1 v1
aparece_villano v1 5 4
FIN
```

Salida de ejemplo

```
Ataques de tidus
ataque_impresionante 20
ataque_superpoderoso 10
Turno:
seymour 10
edea 12
tidus 30
seymour ataca a tidus
edea ataca a tidus
Turno:
tidus 14
seymour 10
edea 12
tidus ataca a seymour
seymour derrotado
edea ataca a tidus
Turno:
tidus 4
edea 12
tidus ataca a edea
Turno:
edea 2
tidus 4
---
ERROR: Personaje ya existente
ERROR: Personaje ya existente
ERROR: Heroe inexistente
ERROR: Heroe inexistente
ERROR: Heroe inexistente
ERROR: Villano inexistente
v1 ataca a h1
h1 derrotado
ERROR: Heroe inexistente
ERROR: No es su turno
v1 ataca a h1
ERROR: Ataque no aprendido
h1 ataca a v1
v1 derrotado
---
```