34

Carnet por puntos 1

La DGT nos ha pedido ayuda para gestionar el *carnet por puntos*. Los conductores están identificados de manera unívoca por su DNI y la cantidad de puntos de un conductor está entre 0 y 15 puntos inclusives.

La implementación del sistema se deberá realizar como un TAD `carnet_puntos` con las siguientes operaciones:

- `nuevo(dni)`: añade un nuevo conductor identificado por su `dni` (un `string`), con 15 puntos. En caso de que el DNI esté duplicado, la operación lanza una excepción `domain_error` con mensaje `Conductor duplicado`.

- `quitar(dni, puntos)`: le resta puntos a un conductor tras una infracción. Si a un conductor se le quitan más puntos de los que tiene, se quedará con 0 puntos. En caso de que el conductor no exista, lanza una excepción `domain_error` con mensaje `Conductor inexistente`.

- `consultar(dni)`: devuelve los puntos actuales de un conductor. En caso de que el conductor no exista, lanza una excepción `domain_error` con mensaje `Conductor inexistente`.

- `cuantos_con_puntos(puntos)`: devuelve cuántos conductores tienen un determinado número de puntos. En caso de que el número de puntos no esté entre 0 y 15 lanza una excepción `domain_error` con mensaje `Puntos no validos`.

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.

Salida

Para cada caso de prueba se escribirán los datos que se piden. Las operaciones que generan salida son:

- `consultar`, que debe escribir una línea con `Puntos de DNI: N`, donde `DNI` es del DNI por el que se ha preguntado y `N` sus puntos;

- `cuantos_con_puntos`, que debe escribir una línea con `Con N puntos hay M`, donde `N` son los puntos por los que se ha preguntado y `M` cuántos tienen esos puntos.

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

Si una operación produce un error, entonces se escribirá una línea con `ERROR:`, seguido del error que devuelve la operación, y no se escribirá nada más para esa operación.

Entrada de ejemplo

```
nuevo 123A
nuevo 456B
nuevo 666
cuantos_con_puntos 15
cuantos_con_puntos 0
quitar 666 15
cuantos_con_puntos 0
quitar 456B 9
consultar 456B
quitar 123A 10
cuantos_con_puntos 5
quitar 456B 1
cuantos_con_puntos 5
FIN
nuevo 123A
nuevo 123A
cuantos_con_puntos 20
quitar 456B 2
FIN
```

Salida de ejemplo

```
Con 15 puntos hay 3
Con 0 puntos hay 0
Con 0 puntos hay 1
Puntos de 456B: 6
Con 5 puntos hay 1
Con 5 puntos hay 2
---
ERROR: Conductor duplicado
ERROR: Puntos no validos
ERROR: Conductor inexistente
---
```

Autor: Alberto Verdejo.

35
Carnet por puntos 2

La DGT nos ha pedido ayuda para gestionar el *carnet por puntos*. Los conductores están identificados de manera unívoca por su DNI y la cantidad de puntos de un conductor está entre 0 y 15 puntos inclusives.

La implementación del sistema se deberá realizar como un TAD `carnet_puntos` con las siguientes operaciones:

- `nuevo(dni)`: añade un nuevo conductor identificado por su `dni` (un `string`), con 15 puntos. En caso de que el DNI esté duplicado, la operación lanza una excepción `domain_error` con mensaje `Conductor duplicado`.
- `quitar(dni, puntos)`: le resta puntos a un conductor tras una infracción. Si a un conductor se le quitan más puntos de los que tiene, se quedará con 0 puntos. En caso de que el conductor no exista, lanza una excepción `domain_error` con mensaje `Conductor inexistente`.
- `recuperar(dni, puntos)`: le añade puntos a un conductor enmendado. Si debido a una recuperación un conductor supera los 15 puntos, se quedará con 15 puntos. En caso de que el conductor no exista, lanza una excepción `domain_error` con mensaje `Conductor inexistente`.
- `consultar(dni)`: devuelve los puntos actuales de un conductor. En caso de que el conductor no exista, lanza una excepción `domain_error` con mensaje `Conductor inexistente`.
- `cuantos_con_puntos(puntos)`: devuelve cuántos conductores tienen un determinado número de puntos. En caso de que el número de puntos no esté entre 0 y 15 lanza una excepción `domain_error` con mensaje `Puntos no validos`.
- `lista_por_puntos(puntos)`: produce una lista con los DNI de los conductores que poseen un número determinado de puntos. La lista estará ordenada por el momento en el que el conductor pasó a tener esos puntos, primero el que menos tiempo lleva con esos puntos. En caso de que el número de puntos no esté entre 0 y 15 lanza una excepción `domain_error` con mensaje `Puntos no validos`.

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.

Salida

Para cada caso de prueba se escribirán los datos que se piden. Las operaciones que generan salida son:

- `consultar`, que debe escribir una línea con `Puntos de DNI: N`, donde `DNI` es del DNI por el que se ha preguntado y `N` sus puntos;
- `cuantos_con_puntos`, que debe escribir una línea con `Con N puntos hay M`, donde `N` son los puntos por los que se ha preguntado y `M` cuántos tienen esos puntos;
- `lista_por_puntos`, que escribe una línea con `Tienen N puntos:`, seguida de los DNIs de los que tienen esos puntos, separados por espacios en blanco.

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

Si una operación produce un error, entonces se escribirá una línea con `ERROR:`, seguido del error que devuelve la operación, y no se escribirá nada más para esa operación.

Entrada de ejemplo

```
nuevo 123A
nuevo 456B
nuevo 666
cuantos_con_puntos 15
cuantos_con_puntos 0
lista_por_puntos 15
quitar 666 15
lista_por_puntos 0
quitar 456B 9
consultar 456B
quitar 123A 10
recuperar 123A 1
lista_por_puntos 6
recuperar 123A 1
lista_por_puntos 6
lista_por_puntos 7
FIN
nuevo 123A
nuevo 123A
cuantos_con_puntos 20
quitar 456B 2
FIN
```

Salida de ejemplo

```
Con 15 puntos hay 3
Con 0 puntos hay 0
Tienen 15 puntos: 666 456B 123A
Tienen 0 puntos: 666
Puntos de 456B: 6
Tienen 6 puntos: 123A 456B
Tienen 6 puntos: 456B
Tienen 7 puntos: 123A
---
ERROR: Conductor duplicado
ERROR: Puntos no validos
ERROR: Conductor inexistente
---
```

Autor: Alberto Verdejo.

36
Consultorio médico

Queremos implementar un TAD `consultorio` que simule el comportamiento de un consultorio médico simplificado. Se hará uso de los tipos `medico` y `paciente` que se representan con un `string`, y `fecha` que es una clase con tres datos: día, hora y minuto.

Las operaciones del consultorio son las siguientes:

- `nuevoMedico(m)`: da de alta un nuevo médico en el consultorio. Si el médico ya estaba en el consultorio, este no se modifica.
- `pideConsulta(p,m,f)`: el paciente `p` pide consulta con el médico `m` para una fecha `f`. Si el médico no está dado de alta se lanzará una excepción con el mensaje `Medico no existente`. Si el médico ya tiene reservada esta fecha, se lanzará una excepción con el mensaje `Fecha ocupada`. Un paciente puede tener varias citas con el mismo médico, siempre que sean en distinto momento.
- `siguientePaciente(m)`: consulta el paciente al que le toca el turno de ser atendido por el médico `m`. Suponemos que el siguiente paciente es el que tiene una fecha menor. Si el médico no está dado de alta se lanzará una excepción con el mensaje `Medico no existente`. Si el médico no tiene pacientes asignados se lanzará una excepción con mensaje `No hay pacientes`.
- `atiendeConsulta(m)`: elimina el siguiente paciente del médico `m`. Suponemos que el siguiente paciente es el que tiene una fecha menor. Si el médico no está dado de alta se lanzará una excepción con el mensaje `Medico no existente`. Si el médico no tiene pacientes asignados se lanzará una excepción con mensaje `No hay pacientes`.
- `listaPacientes(m,d)`: devuelve la lista de pacientes del médico `m` que tienen cita el día `d`, ordenada de menor a mayor por hora de consulta. Se supone que el día es un número entero correcto. Si el médico no está dado de alta se lanzará una excepción con el mensaje `Medico no existente`. Si el médico no tiene pacientes ese día, la lista devuelta será vacía.

Requisitos de implementación.

Se implementará un tipo de datos `fecha` con tres campos: día, hora y minuto; con métodos para acceder a los datos; y con el operador de comparación menor.

Los métodos del TAD `consultorio` 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 comienza con una línea en la que se indica el número de operaciones que se llevarán a cabo en el consultorio. En las líneas siguientes se muestran las operaciones, una por línea: primero aparece el nombre de la operación, seguido de sus argumentos. Los nombres de los médicos y pacientes no tienen blancos y todas las fechas (expresadas con tres números, día, hora y minuto) pueden considerarse correctas.

Salida

Para cada caso de prueba se escribirán los datos que se piden. Las operaciones que generan datos de salida son:

- `siguientePaciente`, que escribe dos líneas: en la primera debe poner `Siguiente paciente doctor` seguido del nombre del doctor que atiende y en la segunda el nombre del paciente;
- `listaPacientes`, que escribe como cabecera `Doctor` seguido del nombre del doctor, seguido de `dia`, seguido del día del que se pide el listado. En las líneas siguientes se escribe el nombre seguido de la hora y del minuto de la cita de todos los pacientes de ese día con ese doctor, ordenados por hora de consulta. La hora se expresa en el formato `HH:MM`.

Cada operación termina con una línea con tres guiones, y cada caso termina con una línea con seis.

Si alguna operación produce una excepción se mostrará el mensaje de la excepción como resultado de la operación, seguido de una línea con tres guiones para cerrar la operación.

Entrada de ejemplo

```
12
nuevoMedico Hernandez
nuevoMedico Alvarez
pideConsulta Ana Alvarez 16 12 30
pideConsulta Antonio Alvarez 16 12 10
pideConsulta Alvaro Alvarez 17 10 00
pideConsulta Alba Alvarez 17 10 15
pideConsulta Anacleto Alvarez 17 10 55
listaPacientes Alvarez 16
siguientePaciente Alvarez
atiendeConsulta Alvarez
listaPacientes Alvarez 16
listaPacientes Alvarez 17
6
nuevoMedico Hernandez
pideConsulta Alvarez Ana 16 12 30
pideConsulta Helena Hernandez 17 10 00
atiendeConsulta Hernandez
listaPacientes Hernandez 17
atiendeConsulta Hernandez
```

Salida de ejemplo

```
Doctor Alvarez dia 16
Antonio 12:10
Ana 12:30
---
Siguiente paciente doctor Alvarez
Antonio
---
Doctor Alvarez dia 16
Ana 12:30
---
Doctor Alvarez dia 17
Alvaro 10:00
Alba 10:15
Anacleto 10:55
---
------
Medico no existente
---
Doctor Hernandez dia 17
---
No hay pacientes
---
------
```

Autores: Isabel Pita y Alberto Verdejo.

37
Autoescuela

Se desea diseñar un TAD para gestionar los alumnos de los distintos profesores de una autoescuela (tanto alumnos como profesores se identifican por su nombre, que es un `string`). Para ello se desea disponer de las siguientes operaciones:

- constructora: al comienzo ningún profesor tiene asignados alumnos.
- `alta(A, P)`: sirve tanto para dar de alta a un alumno como para cambiarle de profesor. Si el alumno no estaba matriculado en la autoescuela se le da una puntuación de cero. Si ha cambiado de profesor, se le da de alta con el nuevo, con la puntuación que tuviera, y se le da de baja con el anterior. La puntuación determinará quién se puede examinar.
- `es_alumno(A, P)`: comprueba si el alumno `A` está matriculado actualmente con el profesor `P`.
- `puntuacion(A)`: devuelve los puntos que tiene el alumno `A`. Si el alumno no está dado de alta con ningún profesor, entonces se lanza una excepción `domain_error` con mensaje `El alumno A no esta matriculado`.
- `actualizar(A, N)`: aumenta en una cantidad `N` la puntuación del alumno `A`. Si el alumno no está dado de alta, entonces se lanza una excepción `domain_error` con mensaje `El alumno A no esta matriculado`.
- `examen(P, N)`: obtiene una lista con los alumnos del profesor `P`, ordenados alfabéticamente, que se presentarán a examen por tener una puntuación mayor o igual a `N` puntos.
- `aprobar(A)`: el alumno `A` aprueba el examen y es borrado de la autoescuela, junto con toda la información que de él existiera. Si el alumno no está dado de alta, entonces se lanza una excepción `domain_error` con mensaje `El alumno A no esta matriculado`.

Requisitos de implementación.
Seleccionar un tipo de datos adecuado para representar la información. En la cabecera de cada función debe indicarse el coste de la misma. 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 consiste en una serie de líneas donde se muestran las operaciones a realizar, sobre una autoescuela inicialmente vacía: el nombre de la operación seguido de sus argumentos, si los tiene. Cada caso termina con una línea con la palabra `FIN`. Los nombres de alumnos y profesores no tienen espacios en blanco.

Salida

Para cada caso de prueba se escribirán los datos que se piden. Las operaciones que generan salida son:

- `es_alumno`, que debe escribir una línea con `A es alumno de P` o `A no es alumno de P`, según corresponda (donde `A` es el nombre del alumno y `P` el del profesor);
- `puntuacion`, que debe escribir una línea con `Puntuacion de A:` seguido de los puntos del alumno `A`;
- `examen`, que escribe una línea con `Alumnos de P a examen:`, seguida de una línea por cada alumno que se pueda presentar al examen, con su nombre, ordenados alfabéticamente.

Cada caso termina con una línea con seis guiones.
Si una operación produce un error, entonces se escribirá una línea con `ERROR`, y no se escribirá nada más para esa operación.

Entrada de ejemplo

```
alta Luis PACO
alta Marta PACO
es_alumno Luis PACO
es_alumno Marta RAMON
examen PACO 0
actualizar Marta 10
examen PACO 10
examen JUAN 10
alta Marta JUAN
puntuacion Marta
actualizar Marta 10
examen JUAN 20
actualizar Miguel 5
aprobar Marta
examen JUAN 20
FIN
puntuacion Alejandro
alta Alejandro ISABEL
actualizar Alejandro 20
puntuacion Alejandro
aprobar Alejandro
puntuacion Alejandro
FIN
```

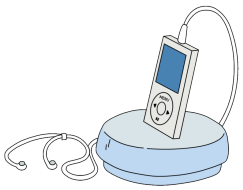
Salida de ejemplo

```
Luis es alumno de PACO
Marta no es alumno de RAMON
Alumnos de PACO a examen:
Luis
Marta
Alumnos de PACO a examen:
Marta
Alumnos de JUAN a examen:
Puntuacion de Marta: 10
Alumnos de JUAN a examen:
Marta
ERROR
Alumnos de JUAN a examen:
------
ERROR
Puntuacion de Alejandro: 20
ERROR
------
```

Autor: Alberto Verdejo.

38

iPud



Los ingenieros de la empresa Apel están actualmente diseñando el software para su nuevo reproductor de música *iPud*, el cual debe permitir añadir y eliminar canciones, añadir una canción existente a la lista de reproducción, avanzar a la siguiente canción, obtener el tiempo total de la lista de reproducción, y obtener una lista con las canciones escuchadas recientemente.

En particular, el TAD `iPud` debe contar con las siguientes operaciones:

- `addSong(S,A,D)`: Añade la canción `S` (un `string`) del artista `A` (un `string`) con duración `D` (un `int`) al iPud. Si ya existe una canción con el mismo nombre la operación dará error.
- `addToPlaylist(S)`: Añade la canción `S` al final de la lista de reproducción. Si la canción ya se encontraba en la lista entonces no se añade (es decir, la lista no tiene canciones repetidas). Si la canción no está en el iPud se devuelve error.
- `current()`: Devuelve la primera canción de la lista de reproducción. Si la lista de reproducción es vacía se devuelve error.
- `play()`: La primera canción de la lista de reproducción abandona la lista de reproducción y se registra como reproducida. Si la lista es vacía la acción no tiene efecto.
- `totalTime()`: Devuelve la suma de las duraciones de las canciones que integran la lista de reproducción actual. Si es vacía se devuelve 0.
- `recent(N)`: Obtiene la lista con las `N` (mayor que 0) últimas canciones que se han reproducido (mediante la operación `play`), de la más reciente a la más antigua. Si el número de canciones reproducidas es menor que `N` se devolverán todas. La lista no tiene repeticiones, de manera que si una canción se ha reproducido más de una vez solo figurará la reproducción más reciente.
- `deleteSong(S)`: Elimina todo rastro de la canción `S` del iPud. Si la canción no existe la operación no tiene efecto.

Se puede asumir que las canciones quedan unívocamente determinadas por su nombre (un `string`).

Requisitos de implementación.

Se elegirá un tipo representante que permita implementar todas las operaciones con coste constante, salvo `recent` que tendrá un coste lineal en su argumento `N`.

Entrada

La entrada consiste en varios casos de prueba. Cada caso de prueba está formado por una lista de llamadas a operaciones del iPud, terminada por la palabra `FIN`. Cada una de estas llamadas comienza por el nombre de la operación, seguido de sus argumentos, en caso de tenerlos. Los nombres de canciones y artistas no contienen espacios.

Salida

Las operaciones que producen salida son las siguientes:

- `play`: se escribirá `Sonando` y el nombre de la canción. Si la lista de reproducción es vacía se escribirá `No hay canciones en la lista`.
- `totalTime`: se escribirá `Tiempo total` y el tiempo de duración de la lista de reproducción.
- `recent`: se escribirá en una línea `Las N más recientes` (donde `N` es el número de canciones realmente mostradas) y en las líneas siguientes los nombres de las canciones, indentadas a la derecha cuatro espacios. Si la lista es vacía se escribirá `No hay canciones recientes` en su lugar.

Si una operación produce un error, entonces se escribirá `ERROR` seguido del nombre de la operación. Y no producirá más salida.

Cada caso se terminará con una línea con cuatro guiones (`----`).

Entrada de ejemplo

```
addSong HumanTouch BruceSpringsteen 392
addSong BornToRun BruceSpringsteen 270
addToPlaylist BornToRun
addToPlaylist HumanTouch
play
totalTime
addSong LuckyTown BruceSpringsteen 207
addToPlaylist LuckyTown
play
play
deleteSong HumanTouch
recent 2
FIN
play
current
FIN
```

Salida de ejemplo

```
Sonando BornToRun
Tiempo total 392
Sonando HumanTouch
Sonando LuckyTown
Las 2 mas recientes
    LuckyTown
    BornToRun
----
No hay canciones en la lista
ERROR current
----
```

Autor: Examen junio 2016.

39
Venta de libros por internet

Se desea diseñar una aplicación para gestionar un sistema de venta de libros por Internet. La implementación hará uso del tipo `libro` que se representa con un `string`.

Las operaciones son las siguientes:

- `nuevoLibro(n,x)`: Añade `n` ejemplares de un libro `x` al sistema. Si `n` toma el valor cero significa que el libro está en el sistema, aunque no se tienen ejemplares disponibles. Si el libro ya está en el sistema se añaden los ejemplares.
- `comprar(x)`: Un usuario compra un libro `x`. Si no existen ejemplares disponibles del libro `x` se produce un error de tipo `out_of_range` con el mensaje `No hay ejemplares`. Si el libro no está dado de alta en el sistema se produce un error de tipo `invalid_argument` con el mensaje `Libro no existente`.
- `estaLibro(x)`: Indica si un libro `x` se ha añadido al sistema. El resultado será cierto si el libro está en el sistema, aunque no haya ejemplares disponibles, y será falso si no está en el sistema.
- `elimLibro(x)`: Elimina el libro `x` del sistema. Si el libro no existe la operación no tiene efecto.
- `numEjemplares(x)`: Devuelve el número de ejemplares de un libro `x` que hay disponibles en el sistema. Si el libro no está dado de alta se produce un error de tipo `invalid_argument` con el mensaje `Libro no existente`.
- `top10()`: Obtiene una lista con los 10 libros que más se han vendido. La lista estará ordenada por número de ventas, de mayor a menor, y los libros que se hayan vendido el mismo número de veces se ordenan del que tenga la venta más reciente a la más antigua, así hasta completar la lista de 10 libros. Si no se han vendido 10 libros distintos se listarán todos ellos.

Requisitos de implementación.

Seleccionar un tipo de datos adecuado para representar la información. En la cabecera de cada función debe indicarse el coste de la misma.

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 comienza con una línea en la que se indica el número de operaciones que se llevará a cabo en el sistema de venta. En las líneas siguientes se muestran las operaciones, una por cada línea: el nombre de la operación seguido de sus argumentos, si los tiene. Los títulos de los libros pueden tener espacios en blanco.

Salida

Para cada caso de prueba se escribirán los datos que se piden. Las operaciones que generan datos de salida son:

- `estaLibro`, que debe escribir `El libro x esta en el sistema` en caso de que el libro esté dado de alta y `No existe el libro x en el sistema` en caso de que no esté dado de alta;
- `numEjemplares`, que debe escribir `Existen m ejemplares del libro x` en caso de que el libro esté dado de alta y `No existe el libro x en el sistema` en caso de que no esté dado de alta;
- `top10`, que escribe los libros más comprados, uno en cada línea ordenados del más comprado al menos comprado. Los libros que se han comprado el mísmo número de veces se listan del comprado más actual al más antiguo.

Cada operación termina con una línea con tres guiones, y cada caso termina con una línea con seis.

Si alguna operación produce una excepción se mostrará el mensaje de la excepción como resultado de la operación, seguido de una línea con tres guiones para cerrar la operación.

Entrada de ejemplo

```
5
nuevoLibro 20 Heidi
nuevoLibro 30 Caperucita roja
comprar Heidi
numEjemplares Caperucita roja
top10
8
nuevoLibro 1 La vuelta al mundo en 80 dias
comprar La vuelta al mundo en 80 dias
estaLibro La vuelta al mundo en 80 dias
nuevoLibro 5 Viaje al centro de la tierra
comprar Viaje al centro de la tierra
comprar Viaje al centro de la tierra
top10
comprar La isla misteriosa
```

Salida de ejemplo

```
Existen 30 ejemplares del libro Caperucita roja
---
Heidi
---
------
El libro La vuelta al mundo en 80 dias esta en el sistema
---
Viaje al centro de la tierra
La vuelta al mundo en 80 dias
---
Libro no existente
---
------
```

Autores: Isabel Pita y Alberto Verdejo.