Trabajo Práctico 1 — Hospital Pokémon

[7541/9515] Algoritmos y Programación II
Segundo cuatrimestre de 2021

Alumno:	Aldazabal, Lucas Rafael
Número de padrón:	107705
Email:	laldazabal@fi.uba.ar

Índice

1. Introducción **2**

2. Detalles de implementación **2**
2.1. hospital_crear 2
2.2. hospital_leer_archivo 2
2.3. Como se interpreta el archivo 3
2.4. Otras funciones 4
2.5. ¿Por qué se almacenan los entrenadores? 4

3. Modelo de un Hospital **5**

1. Introducción

En este proyecto se plantea la digitalización de información del ingreso de pokemones a un hospital mediante archivos de texto que el usuario puede proporcionar. El proyecto es capaz de crear y eliminar un registro de hospital, levantar la información de los archivos, llevar una lista de entrenadores y pokemones que se encuentran, y hacer operaciones con ellos.

Una vez uno crea un hospital puede proporcionar archivos con el siguiente formato:

ID_ENTRENADOR;NOMBRE_ENTRENADOR;POKEMON;NIVEL;POKEMON;NIVEL;...
ID_ENTRENADOR;NOMBRE_ENTRENADOR;POKEMON;NIVEL;POKEMON;NIVEL;...
ID_ENTRENADOR;NOMBRE_ENTRENADOR;POKEMON;NIVEL;POKEMON;NIVEL;...

Cada renglón representa un ingreso de un entrenador con sus pokemones proporcionando id y nombre del entrenador, y nombre y nivel de sus pokemones. El sistema esta preparado para almacenar dinámicamente una cantidad virtualmente infinita (tanto como el hardware permita) de pokemones y entrenadores para una implementación real.

El sistema permite además de la carga de información, consultar cantidad de entrenadores, cantidad de pokemones, el nivel de un pokémon, el nombre de un pokémon, y aplicar una acción a cada pokémon ingresado.

2. Detalles de implementación

Como se mencionó previamente, la implementación se hizo siempre priorizando el almacenamiento de la información ingresada, marcando el enfoque a la digitalización de datos.

Para esto se crea la estructura hospital para almacenar un vector de entrenadores y otro de pokemones, con sus cantidades a modo de tope. Para almacenar toda la información se utiliza el heap, en el caso del hospital es para que se pueda acceder con un puntero que este se pueda ir pasando para leer o modificar el mismo, y en el caso de los vectores tanto de pokemones como de entrenadores es para poder aumentar dinámicamente la cantidad de elementos que contienen y no tener que fijar un limite arbitrario que seria poco coherente en la implementación del usuario.

A continuación se van a detallar técnicamente algunas funciones

2.1. hospital_crear

Esta, como indica en nombre, es la función encargada de reservar en el heap un hospital y inicializar sus datos.

El hospital es un struct con el siguiente formato

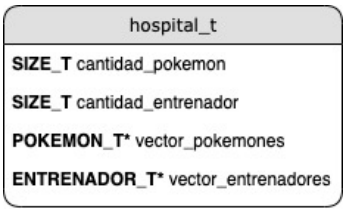

Figura 1: Formato de un hospital.

2.2. hospital_leer_archivo

La función esta encargada de leer un archivo y guardar en el hospital los datos levantados con el formato mencionado en la Introducción, para esto lee linea por linea los datos separando por punto y coma para poder almacenar los primeros dos valores del renglón como el entrenador y los pokemones que le continúan. En caso de que la linea no tenga al menos dos valores (nombre e id del entrenador), o una cantidad par de datos (nombre e id del entrenador, mas 2 por pokémon),

que haya algún error leyendo el archivo o un error reservando memorias, esta devolverá false, en caso de que todo salga perfectamente devolverá true.

2.3. Como se interpreta el archivo

Cuando se recibe el archivo y se abre correctamente se ejecuta la función leer_linea_completa, que hace una lectura de linea en el archivo de un tamaño determinado almacenándolo en el heap, verifica si esta termina con un salto de linea confirmando que sea el renglón completo, y en caso de no serlo agranda el espacio en memoria para leer otra porción de la linea, y así hasta completarla. El resultado final es un string dinámico con el siguiente formato

Figura 2: Ejemplo de string en el heap leido por leer_linea_completa.

Una vez se tiene la linea completa almacenada, se ejecuta la función split que se encarga de separar los elementos por un separador que en este caso es el ';', y devuelve un vector dinámico con punteros a los strings, también dinámicos, de cada una de las porciones del string original y termina con un NULL.

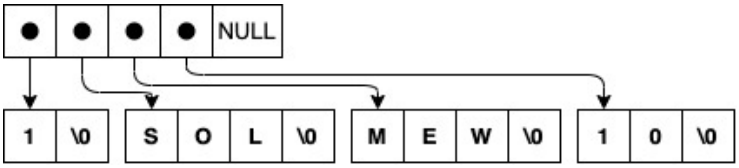

Figura 3: Ejemplo de vector de strings separados por split.

Luego, verifica que la cantidad de elementos sea correcta, que sean pares para poder almacenar entrenador y pokemones sin que a alguno le falten datos (2 del entrenador mas dos por cada pokemon) o que haya al menos dos elementos que serian las del entrenador. Una vez verificado, se ejecuta la función hospital_agregar_entrenador_y_pokemones para almacenar en el hospital los elementos obtenidos en el paso anterior. Acá se copian los números a los structs correspondientes y se reutilizan los punteros de los strings ya almacenados en el heap ahorrando una reserva innecesaria que no solo consume recursos, sino que es agregar posibilidades de fallo.

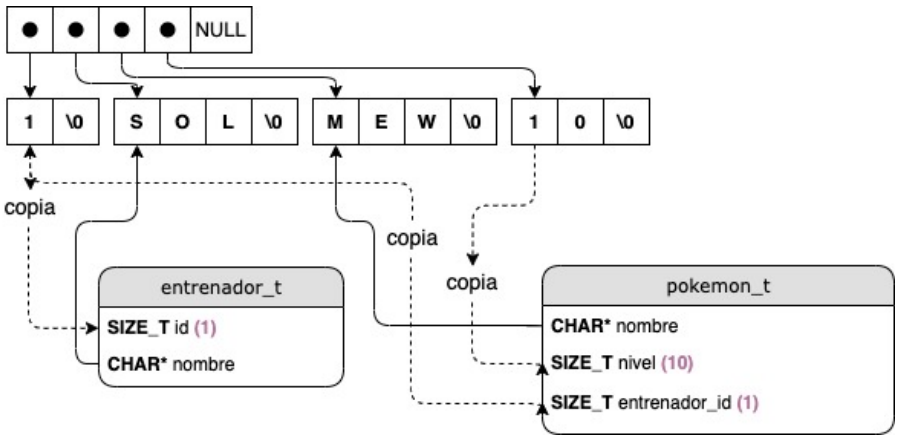

Figura 4: Ejemplo de guardado de elementos.

Luego, cuando toda la información de la linea ya se encuentra impactada en el hospital se libera la memoria utilizada por el split (la memoria del leer_linea_completa ya había sido liberada luego del split) donde se libera únicamente los datos numéricos porque como se mencionó previamente, los punteros a strings son reutilizados, y luego se libera el vector de vectores.

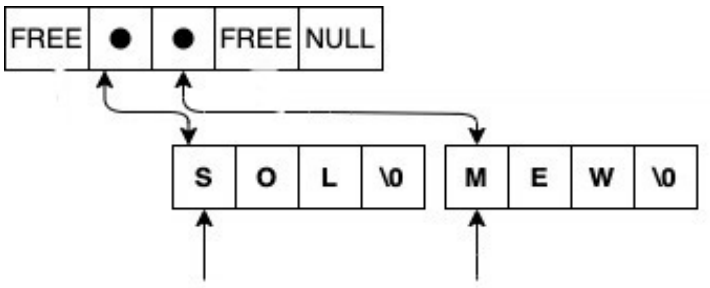

Figura 5: Ejemplo de guardado de elementos.

Luego de esto se lee la siguiente linea y se repite el proceso. La función encargada de liberar la memoria del split que libera únicamente los números, también tiene la posibilidad de vaciar el split entero si hay algún error o motivo para no utilizar esa información y por ende no reutilizar los punteros a strings.

2.4. Otras funciones

Otras funcionalidades que da el programa son las de ver datos de los entrenadores y pokemones registrados, para eso se crean múltiples getters ya que las estructuras son privadas para el usuario final. Algunas de ellas son:

- hospital_cantidad_pokemon
- hospital_cantidad_entrenadores
- pokemon_nivel
- pokemon_nombre

estas reciben el objeto y devuelven datos almacenados adentro.

También hay una función hospital_destruir que recibe un hospital y se ocupa de liberar los nombres de pokemones y entrenadores, sus vectores y finalmente el hospital mismo.

Por ultimo otra función a mencionar es hospital_a_cada_pokemon, es una función que recibe un hospital y un puntero a función, su funcionalidad consiste en que recorre alfabéticamente los pokemones del hospital ejecutando la función recibida pasándole como parámetro el pokémon de turno, si la función devuelve falsa retorna la cantidad de elementos recorridos hasta el momento, si recibe true continua ejecutando hasta el final de los pokemones donde devuelve el total de pokemones. Para el orden se utiliza un bubble sort mejorado que verifica que por cada vuelta se haga algún cambio, y en caso de no hacerlo entiende que ya esta ordenado y se ahorra continuar dando vueltas no necesarias.

2.5. ¿Por qué se almacenan los entrenadores?

Como podrán ver de momento no hay ninguna funcionalidad para el usuario que aproveche la información almacenada de los entrenadores, tanto el del vector de entrenadores como que se guarde el id del entrenador dueño de cada pokémon, pero volviendo al objetivo planteado en la

introducción, lo principal es poder digitalizar la información que los usuarios tenían en papel, y eso incluye a los entrenadores, por lo que aunque de momento no se utilice para nada, seria un error no guardarlo ya que se podrían ampliar las funcionalidades en algún futuro y esa información perdida podría ser vital. Por el mismo motivo, a la hora de agregar un entrenador nuevo hay un parámetro que de momento se encuentra desactivado que se ocupa de revisar si ya estaba registrado el entrenador mediante el id, para evitar repeticiones, de momento se almacenan repeticiones ya que no se utiliza esta información y es meramente un registro, pero si luego se hace una actualización se pueden eliminar repetidos y agregar funcionalidad para estos sin problemas.

3. Modelo de un Hospital

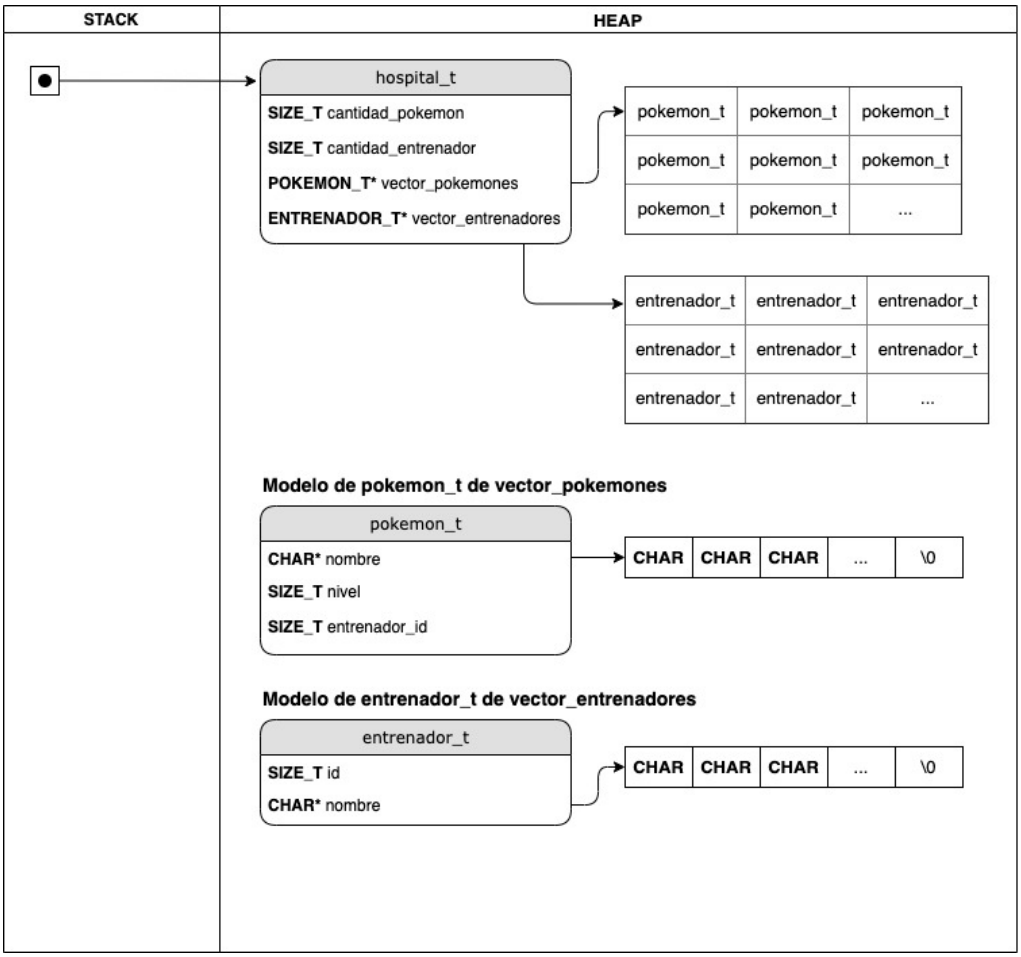

Figura 6: Modelo de un hospital.