Trabajo Práctico 1 — Hospital Pokémon

[7541/9515] Algoritmos y Programación II

Segundo cuatrimestre de 2021

Alumno:	APELLIDO, Nombre
Número de padrón:	107951
Email:	msanchezf@fi.uba.ar

1. Introducción

El trabajo practico consistió en la lectura de un archivo, el cual contiene diversos campos separados por `;`. Estos campos debían ser almacenados en una estructura de tipo `hospital_t`. La dificultad consistió en que el tamaño de este archivo no era fijo, por lo que fue necesario implementar memoria dinamica.

Por el otro lado, la estructura provista no tenia los campos necesarios para almacenar la informacion del archivo. En consecuencia, tuve que agregar diversos campos a la estructura que seran explicados en la seccion de **detalles de implementacion**.

Para poder resolver el trabajo practico, dividí el proceso en varias partes para facilitar la resolucion. Estas fueron las siguientes:

1. Leer el archivo y extraer la informacion correctamente
2. Almacenar la informacion del archivo en las correspondientes estructuras
3. Procesar la informacion acorde a lo pedido en la consigna

Una vez realizadas estas 3 partes, lo unico que restaba era comprobar que no haya ningun tipo de error en cuanto a la memoria dinámica.

2. Detalles de implementación

Con el fin de almacenar la informacion del archivo, cree y modifique algunas de las estructuras presentes en el archivo. Las estructuras que plantee inicialmente para el desarrollo del trabajo practico fueron las siguientes:

```
struct _entrenador_pkm_t{
    char* nombre;
    int id;
};

struct _pkm_t{
    char* nombre;
    size_t nivel;
    int id_entrenador;
};

struct _hospital_pkm_t{
    size_t cantidad_actual_pokemon;
    size_t cantidad_maxima_pokemon;
    pokemon_t* vector_pokemon;
    size_t cantidad_actual_entrenadores;
    size_t cantidad_maxima_entrenadores;
    entrenador_t* vector_entrenadores;
};
```

Sin embargo, al leer detalladamente cada funcion, me di cuenta que ninguna pide los datos de los entrenadores. En otras palabras, el registro de tipo entrenador no tiene utilidad alguna. No obstante, decidi mantenerlo a la hora de desarrollar el programa ya que me parecio que era una buena idea preservarlo para futuras implementaciones en caso que se vuelva a trabajar con este mismo trabajo practico.

Por el otro lado, para realizar el trabajo practico, dividi mis funciones en 3 archivos distintos:

- **hospital.c/.h**: contiene todas las funciones principales del trabajo practico. Es decir, aquellas cuyo funcionamiento se comprueba en `pruebas.c`. Es la interfaz publica de la implementacion.
- **split.c/.h**: contiene las funciones usadas para el TP0 y una nueva funcion llamada `liberar_elementos_split()`. Esta ultima funcion sirve para realizar un `free()` a todos los elementos del split ya que esto no era parte del TP0, sino que el usuario se hacia cargo. La finalidad de esta biblioteca es dividir las lineas del archivo en base a sus separadores y guardar la informacion en un vector dinamico. En otras palabras, me permite extraer la informacion del archivo con mayor facilidad.
- **hospital_interno.c/.h**: como su nombre lo indica, esta biblioteca contiene funciones internas que facilitan el desarrollo de las funciones de `hospital.c`. La razon detras de esta biblioteca es para reutilizar el codigo ya escrito y, a su vez, mantener orden dentro de las otras bibliotecas separando aquellas funciones que son privadas de las publicas. Asimismo, me permite realizar pruebas con las funciones internas que, de otro modo, no podria.

Me gustaria destacar que la funcion `split()` junto a `leer_linea()` y `parsear_linea()` se complementan y son esenciales para extraer la informacion del archivo. Ambas son

usadas en la funcion `hospital_leer_archivo()` de **hospital.c**.

Otro detalle a mencionar es que moví la definicion y declaracion de todos los registros desde **hospital.h** hacia **hospital_interno.h**. Esto me permite utilizar los registros en las funciones de **hospital.c** y **hospital_interno.c** sin tener que reescribir los registros en ambos archivos (ya que **hospital.h** incluye a **hospital_interno.h**) y sin perder la "privacidad" de los registros.

Detalles de funciones

1. `leer_linea()` Esta funcion se basa en agarrar caracter por caracter de la linea agregandolos a un string dinamico siempre y cuando ese caracter no sea ni `'\n'` ni `'\0'`. Una vez que se topa con uno de esos caracteres, la funcion devuelve el string que almaceno hasta el momento. Si el vector esta lleno, entonces duplica su tamaño (esto reduce en gran cantidad las asignaciones de memoria dinamica.

   Para leer cada caracter, la funcion usa `fgetc()` ya que al momento de escribirla, no estaba enterado de `fgets()`. Sin embargo, tiempo despues mencionaron esta funcion en Slack y me di cuenta que probablemente habria sido mas facil utilizarla pero decidi no cambiarla ya que:

   - La tenia hecha
   - Funcionaba bien
   - Le da un toque de originalidad :)

2. `inicializar_pokemon()` y `inicializar_entrenador()` Son funciones que, como bien indica su nombre, inicializan un registro de tipo `pokemon_t` y `entrenador_t` respectivamente. Es interesante destacar que estas 2 funciones son de tipo bool, y devuelven `true` si se ejecuto correctamente y `false` si fallo la asignacion de memoria para el nombre. En cuanto a esta asignacion, se realiza una copia del string dinamico devuelto en split en vez de directamente usar ese string dinamico ya que de esta forma es mas facil liberar los elementos del split una vez que ya no son necesarios. Estas 2 funciones son muy similares, sin embargo, despues de darle vueltas un rato, no pude pensar ninguna manera de agruparlas en una unica funcion sin complicar mucho la comprension del codigo. Es por esta razon que preferí dejarlo como 2 funciones separadas.

3. `agregar_pokemon()` y `agregar_entrenador()` Son funciones que agregan registros de `pokemon_t` y `entrenador_t` a un vector de memoria dinamica con el correspondiente tipo. Si el vector no estuviera inicializado, lo inicializa y agrega el primer valor. Con el objetivo de minimizar la cantidad de asignaciones en cuanto a espacios de memoria, lo que hace la funcion es tomar el tamaño maximo del vector y duplicarlo una vez que este lleno. En estas 2 funciones sucede lo mismo que con el caso de `inicializar_pokemon()` y `inicializar_entrenador()` en cuanto a la similitud de las mismas pero la dificultad a la hora de agruparlas en una unica funcion.

4. `parsear_linea()` Esta funcion extrae la informacion del vector devuelto por `split()` y la asigna correctamente en cada campo del hospital. Para esto, tomo

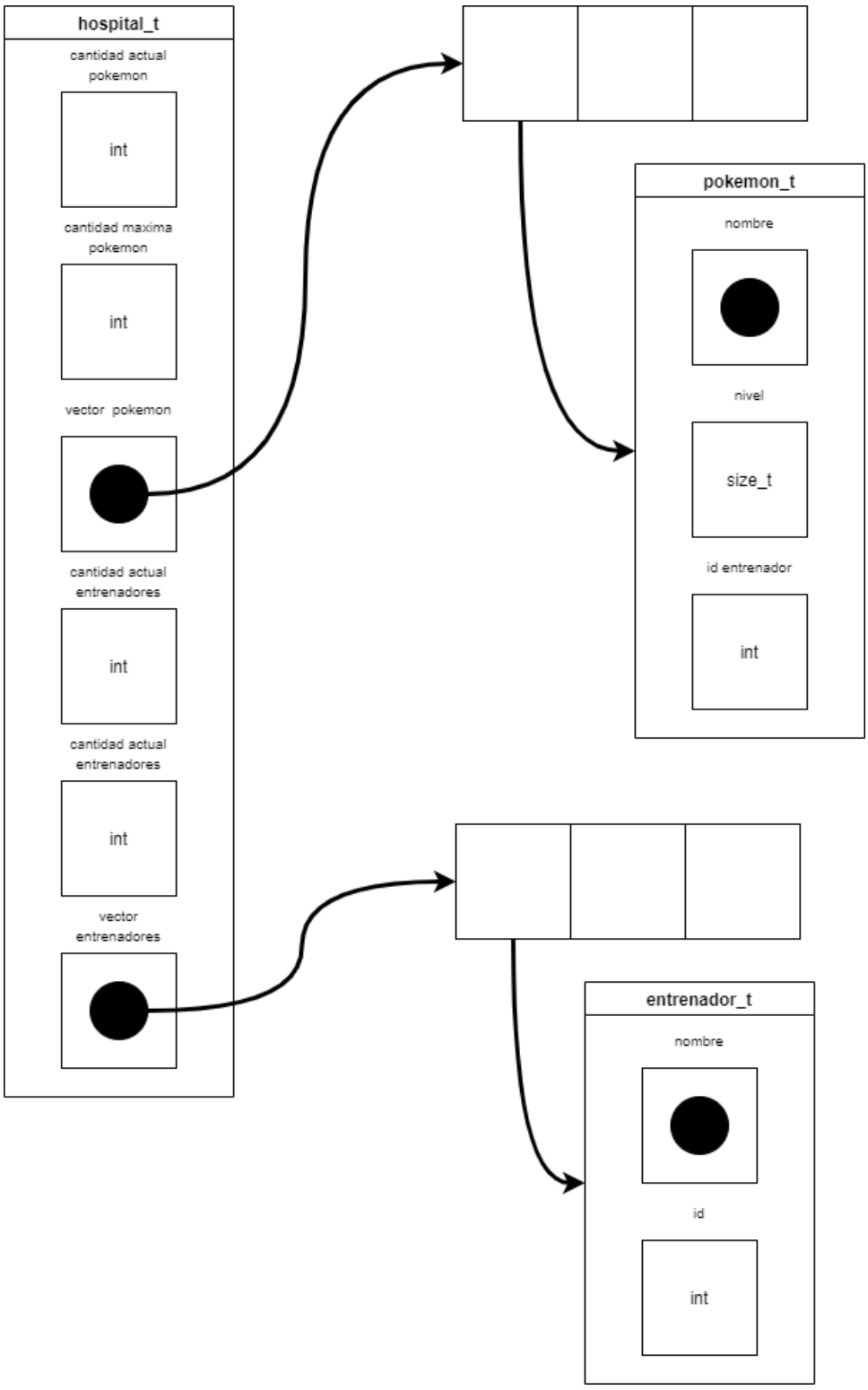
que en la primera y segunda posicion del archivo esta el id y nombre del entrenador respectivamente. A partir de la posicion 2, en los numeros pares tengo el nombre de los pokemones y en los numeros impares tengo el nivel de cada 1. Tomando todo esto en cuenta, recorro los vectores agarrando la informacion y asignandola como corresponde en el vector de pokemon y entrenador, y a su vez aumentando el tope.

5. `hospital_a_cada_pokemon` En esta funcion, opte por ordenar alfabeticamente el vector de pokemon ya que es mas simple que pasarle cada pokemon en orden alfabetico a la funcion pasada por parametro.

6. `hospital_leer_archivo` El detalle de implementacion que me gustaria destacar en esta funcion es que hago uso de la funcion `hospital_copiar()` para poder tener una copia del hospital en su estado inicial en caso de que la lectura del archivo o carga de datos falle. Si esto llegara a suceder, el hospital original sera devuelto a su estado inicial de antes de aplicar la funcion.

3. Diagramas

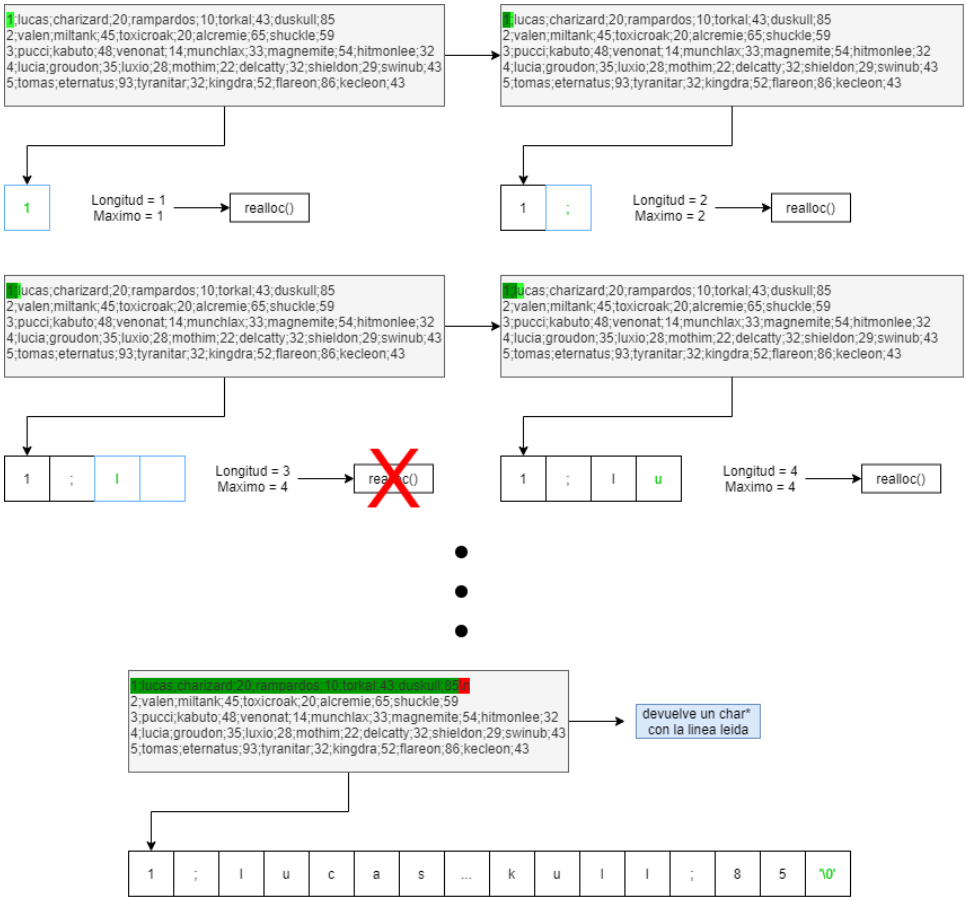
1. Diagrama de los registros



2. Funcionamiento de `leer_linea()`

   La explicacion del funcionamiento se encuentra en los detalles de las

funciones.


1;lucas;charizard;20;rampardos;10;torkal;43;duskull;85
2;valen;miltank;45;toxicroak;20;alcremie;65;shuckle;59
3;pucci;kabuto;48;venonat;14;munchlax;33;magnemite;54;hitmonlee;32
4;lucia;groudon;35;luxio;28;mothim;22;delcatty;32;shieldon;29;swinub;43
5;tomas;eternatus;93;tyranitar;32;kingdra;52;flareon;86;kecleon;43
1;lucas;charizard;20;rampardos;10;torkal;43;duskull;85
2;valen;miltank;45;toxicroak;20;alcremie;65;shuckle;59
3;pucci;kabuto;48;venonat;14;munchlax;33;magnemite;54;hitmonlee;32
4;lucia;groudon;35;luxio;28;mothim;22;delcatty;32;shieldon;29;swinub;43
5;tomas;eternatus;93;tyranitar;32;kingdra;52;flareon;86;kecleon;43
1
Longitud = 1
Maximo = 1
realloc()
1
;
Longitud = 2
Maximo = 2
realloc()
1;lucas;charizard;20;rampardos;10;torkal;43;duskull;85
2;valen;miltank;45;toxicroak;20;alcremie;65;shuckle;59
3;pucci;kabuto;48;venonat;14;munchlax;33;magnemite;54;hitmonlee;32
4;lucia;groudon;35;luxio;28;mothim;22;delcatty;32;shieldon;29;swinub;43
5;tomas;eternatus;93;tyranitar;32;kingdra;52;flareon;86;kecleon;43
1;lucas;charizard;20;rampardos;10;torkal;43;duskull;85
2;valen;miltank;45;toxicroak;20;alcremie;65;shuckle;59
3;pucci;kabuto;48;venonat;14;munchlax;33;magnemite;54;hitmonlee;32
4;lucia;groudon;35;luxio;28;mothim;22;delcatty;32;shieldon;29;swinub;43
5;tomas;eternatus;93;tyranitar;32;kingdra;52;flareon;86;kecleon;43
1
;
l
Longitud = 3
Maximo = 4
realloc()
1
;
l
u
Longitud = 4
Maximo = 4
realloc()
1;lucas;charizard;20;rampardos;10;torkal;43;duskull;85
2;valen;miltank;45;toxicroak;20;alcremie;65;shuckle;59
3;pucci;kabuto;48;venonat;14;munchlax;33;magnemite;54;hitmonlee;32
4;lucia;groudon;35;luxio;28;mothim;22;delcatty;32;shieldon;29;swinub;43
5;tomas;eternatus;93;tyranitar;32;kingdra;52;flareon;86;kecleon;43
devuelve un char*
con la linea leida
1
;
l
u
c
a
s
...
k
u
l
l
;
8
5
'\0'