75.41 95.15 95.12 Algoritmos y Programación II Curso 4
TP1 - Hospital Pokemon

16 de septiembre de 2021

1. Introducción

Los hospitales Pokemon son centros médicos especializados en la atencion de Pokemon. Un familiar tuyo hace poquito se puso un hospital de estas características pero anda necesitando un sistema para llevar la cuenta de cada uno de los pokemon que estan siendo atendidos en el mismo. Por ahora se viene manejando con un anotador y una birome, pero le gustaría... informatizar un poco el asunto. Oh casualidad, vos estas estudiando y sabes algo de programación. Seguro que podes ayudar en algo.

Te proponemos lo siguiente. Tuvimos acceso a los listados que se estan usando y, por suerte, son bastante simples. En base a estos listados, te pedimos que implementes una solución para facilitar el listado y búsqueda de Pokemon.

2. Formato de los listados

Por ahora los listados se venian haciendo a mano en un anotador, pero por ahora decidieron pasarlos a computadora en un formato estilo csv con la información mínima posible. En un futuro el formato puede cambiar, pero por ahora nos basta con esta información. El archivo contiene una o mas líneas. Cada línea contiene diferentes campos separados por punto y coma.

El primer elemento de la línea es el ID del entrenador, luego le sigue su nombre, y a continuación le siguen los pokemon y nivel de cada uno de los entrenadores. A continuación se muestra un ejemplo de un archivo con 3 entrenadores:

```
ID_ENTRENADOR1;NOMBRE_ENTRENADOR1;POKEMON1;NIVEL;POKEMON2;NIVEL;POKEMON3;NIVEL
ID_ENTRENADOR2;NOMBRE_ENTRENADOR2;POKEMON1;NIVEL;POKEMON2;NIVEL
ID_ENTRENADOR3;NOMBRE_ENTRENADOR3;POKEMON1;NIVEL;POKEMON2;NIVEL;POKEMON3;NIVEL;POKEMON4;NIVEL
```

En esta oportunidad nos vamos a asegurar de que los archivos vengan sin errores. Pueden venir archivos vacíos, pero si el archivo no está vacío, por lo menos va a tener una línea con un entrenador y por lo menos un pokemon. Nunca va a haber entrenador sin pokemon o pokemon sin la información del entrenador.

2.1. Funcionalidad pedida

Se pide que el sistema pueda almacenar la información de los archivos. Debe ser posible abrir y uno o mas de los archivos con el formato pedido y luego buscar en el hospital la información básica. Para asegurarnos de que no haya problemas, te vamos a dar un archivo header de C con la interfaz pública del sistema pedido. Si cumplis con lo que te pide, va a funcionar todo bien.

Obviamente, para asegurarnos de que va a funcionar, vas a tener que subir tu programa al sistema de pruebas, así nos aseguramos de que todo está bien. Además te dejamos un **Makefile** para que puedas compilar y probar tu sistema. Ya sabes, a futuro puede ser que los requerimientos cambien, asegurate de trabajar de forma prolija y entendible. Ojo con los errores de memoria.

En esta oportunidad, se presenta una estructura de directorios con un **makefile**, **pruebas** y un par de archivos **c** y **h**. Dentro del directorio **src** podes crear tantos archivos **c** y **h** extras como quieras. Asegurate de no modificar el archivo **hospital.h**. Podes modificar el resto, siempre que cumpla con los requisitos pedidos. Recordá que nosotros vamos a compilar todo con el **makefile** original.

2.2. Entrega

Una vez completada la implementación, tenes que subirla al sistema de entregas, como siempre, para verificar su correcto funcionamiento. Además de la implementación, tenes que escribir un informe y entregarlo (en formato PDF). En el informe **tenés que explicarle a tu corrector** qué algoritmos diseñaste, cómo funcionan, cómo manejas la memoria y podes (y recontra re recomendamos que lo hagas) incluír diagramas de memoria o de algún otro tipo que ayuden a entender tu implementación. **El informe es parte de la nota**.

Las recomendaciones de siempre (repetidas, claro):

- No se pueden asumir tamaños máximos: No vale decir 'Asumo que nunca me van a mandar un string mas grande que 5000 caracteres'. No. Usen memoria dinámica, que para eso está.
- Dicho esto, no todo en la vida es memoria dinámica. Úsenla dónde sea apropiado. El stack sirve para colocar variables locales, úsenlo.
- Modularización ante todo. No es tan importante en este punto si recorren dos veces el vector para resolver el problema. Es preferible hacer código claro que recorra dos veces el string a un código inentendible que recorre una sola vez.
- Tienen un **Makefile** que corre pruebas automatizadas para saber si su implementación cumple con lo básico que se pide. Aprovechenlo. Pueden agregar mas pruebas si lo creen necesario.
- La modularización es importante (¿Ya lo había dicho?). No me hagan un choclo.
- Los nombres de variable son importantes. Mejor **'int cantidad_de_substrings'** a **'int n'**.
- Los nombres de función también son importantes. Mejor **'buscar_posición_separador'** que **'buscar'**.