Trabajo Práctico 1 — Centro Pokemon

[7541/9515] Algoritmos y Programación II

Segundo cuatrimestre de 2021

Alumno:	González Calderón, Julián
Número de padrón:	107938
Email:	jgonzalezc@fi.uba.ar

1. Introducción

El trabajo práctico constaba del uso de memoria dinámica para administrar los registros de un centro pokemon. Tanto los pokemónes que están allí atendidos, como los entrenadores matriculados en el mismo.

La idea era definir una serie de funciones que ayuden a manejar la información del hospital con más facilidad, como indicaba el documento dado por la cátedra.

2. Funcionamiento

A continuación voy a mencionar algunas funciones importantes y explicar la lógica detrás de ellas.

- **`leer_linea()`**

  Esta función lee una línea completa del archivo y la almacena.

  Para obtener esto, lo que hace es leer una cantidad inicial de caracteres del **stream**. Si el anteúltimo carácter leído no es un `\n`, quiere decir que no llego al final de la línea, por lo que reserva mas memoria para el **string** y continua leyendo del archivo, almacenando los nuevos caracteres a partir del ultimo elemento.

  Una vez que detecta el fin de una línea, remplaza el `\n` por un `\0`, ya que no lo necesitamos. En caso de llegar al fin del archivo, simplemente devuelve el string hasta donde leyo.

- **`hospital_leer_archivo()`**

  Esta función recibe el nombre de una archivo de texto en el formato indicado por la cátedra, y almacena su información en los registros del hospital.

  Para lograr esto, lee cada línea del archivo y la separa en distintos **substrings** usando la biblioteca **split**. Una vez dividido, reserva mas memoria para el vector de pokemones (dependiendo de la cantidad de **substrings**) y de entrenadores (un solo entrenador por línea).

Luego, agrega la información al hospital. Los primeros dos **substrings** pertenecen a la lista de entrenadores, mientras que el resto pertenecen a la lista de pokemones.

Los **substrings** que contienen un nombre se almacenan directamente en el vector del hospital, mientras que los substrings que contienen números se transforman a un tipo de dato numerico y luego se liberan. Esto se debe a que la función `atoi()` devuelve un entero, por lo que ya no necesitamos el puntero original.

- **`ordenar_indice_pokemones()`**

  Esta función asigna a cada elemento del vector valores ascendentes empezando por el `0`. Luego ordena este vector de menor a mayor, pero comparando no el número en sí, sino los nombres de los pokemones almacenados en ese índice.

- **`hospital_a_cada_pokemon()`**

  Esta función aplica una función a cada pokemon del hospital en orden alfabético.

  Utiliza la función `ordenar_indice_pokemones()`. Así obtiene un vector de índices que representa el orden en el que se tienen que ejecutar las función

  Luego, simplemente recorre este vector y aplica la función al pokemon que se encuentra en el índice de cada posición del vector.

- **`destruir_hospital()`**

  Esta función se encarga de liberar toda la memoria que fue reservada por el hospital. Itera a través de los vectores del mismo y libera cada uno de los elementos. Luego libera los vectores en sí. Finalmente, libera el hospital.

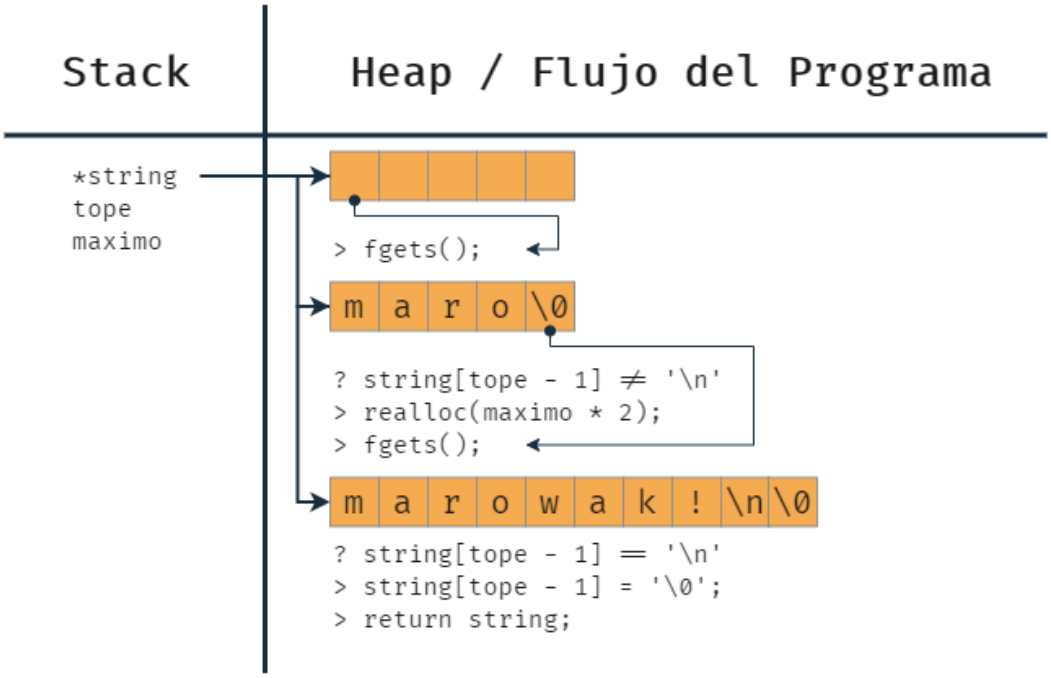
- **`destruir_hospital()`**

  Esta función se encarga de crear el hospital. Primero reserva memoria para el hospital y luego inicializa sus componentes con valores nulos, para deshacerse de la basura.

3. Diagramas

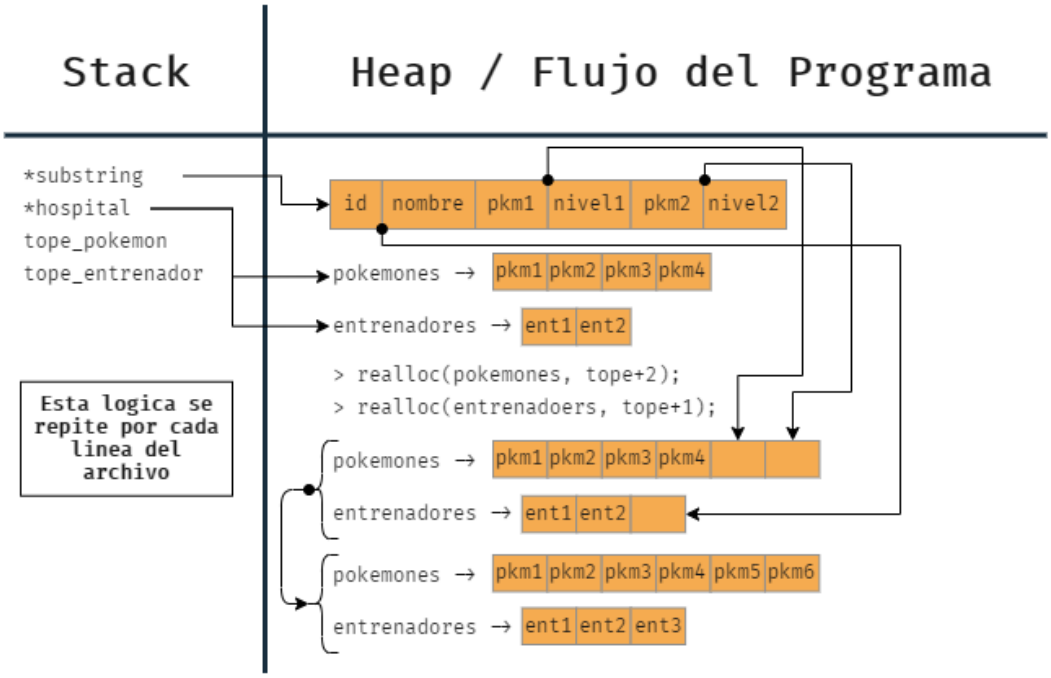
Algunos diagramas para profundizar la explicación de los algoritmos.

- **`leer_linea()`**



Este es un ejemplo simple de leer una línea de un archivo. Crea el **string** vacio y lee hasta donde le alcanza la memoria reservada inicialmente, luego, como no leyó el `\n`, reserva mas memoria y vuelve a leer. Ahora si leyó el `\n`, por lo que lo cambia a un `\0` y devuelve el **string**.

- **`hospital_leer_archivo()`**



En este diagrama se puede ver como el algoritmo lee la primera línea de un archivo y la almacena en el hospital. Partimos del vector de **substrings**, luego reservamos la memoria que necesitemos (en este caso vamos a reservar memoria para un entrenador y para dos pokemones). Una vez reservada la memoria la guardamos en el vector. En este diagrama se simplificó bastante el procedimiento para que sea más fácil de seguir.