FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

# Trabajo Práctico 1

*Lucas Franciulli*

*[7541/9515] Algoritmos y Programación II*

*Segundo cuatrimestre 2021*



| | |
|---|---|
| Nombre: | Franciulli, Lucas |
| Padrón: | 107059 |
| Email: | lfranciulli@fi.uba.ar |

# *Índice*

# 1. Introducción

El objetivo de este trabajo es actualizar el sistema de un hospital que se usa para llevar la cuenta de cada uno de los Pokemon que están siendo atendidos en este. Para lograr esto se crearon varias funciones que cumplen con todas las acciones necesarias para lograrlo. En este trabajo se usaron los datos proporcionados por el archivo **CSV** (*comma-separated values*).

Las funciones creadas están en los archivos hospital.h y split.h.

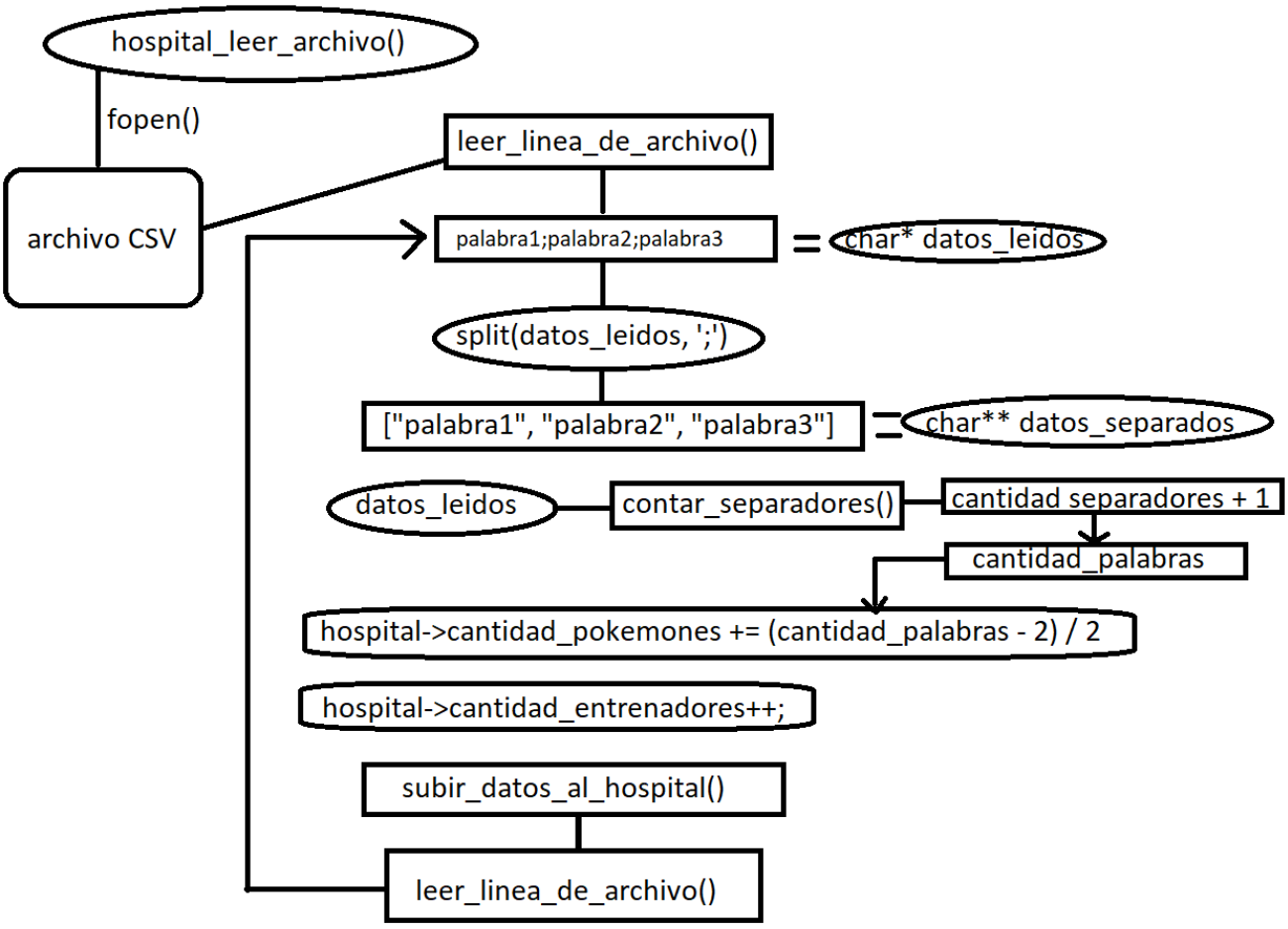
El funcionamiento específico de estas se encuentra en sus respectivos archivos .c (hospital.c y split.c). En este informe se detallará la lógica de algunas funciones que aparecen en estos.

# 2. Funciones

### 2.1 hospital_leer_archivo ()

La función hospital_leer_archivo() abre el archivo CSV que le pasaron por parámetro en modo lectura. Después de esto entra en el loop principal el cual hace lo siguiente: lee una línea del archivo con la función leer_linea_de_archivo(). A la línea leída se encarga de separarla en diferentes strings con split(). Una vez teniendo los datos de la línea separados calculo la cantidad de pokemones y teniendo toda esta información la mando a la función subir_datos_al_hospital() la cual se encarga de subir estos al hospital_t. Luego vuelve a leer una línea y repite el proceso hasta que llegue al final del archivo o haya algún error al subir los datos.

### 3.2.0 Diagrama del loop principal

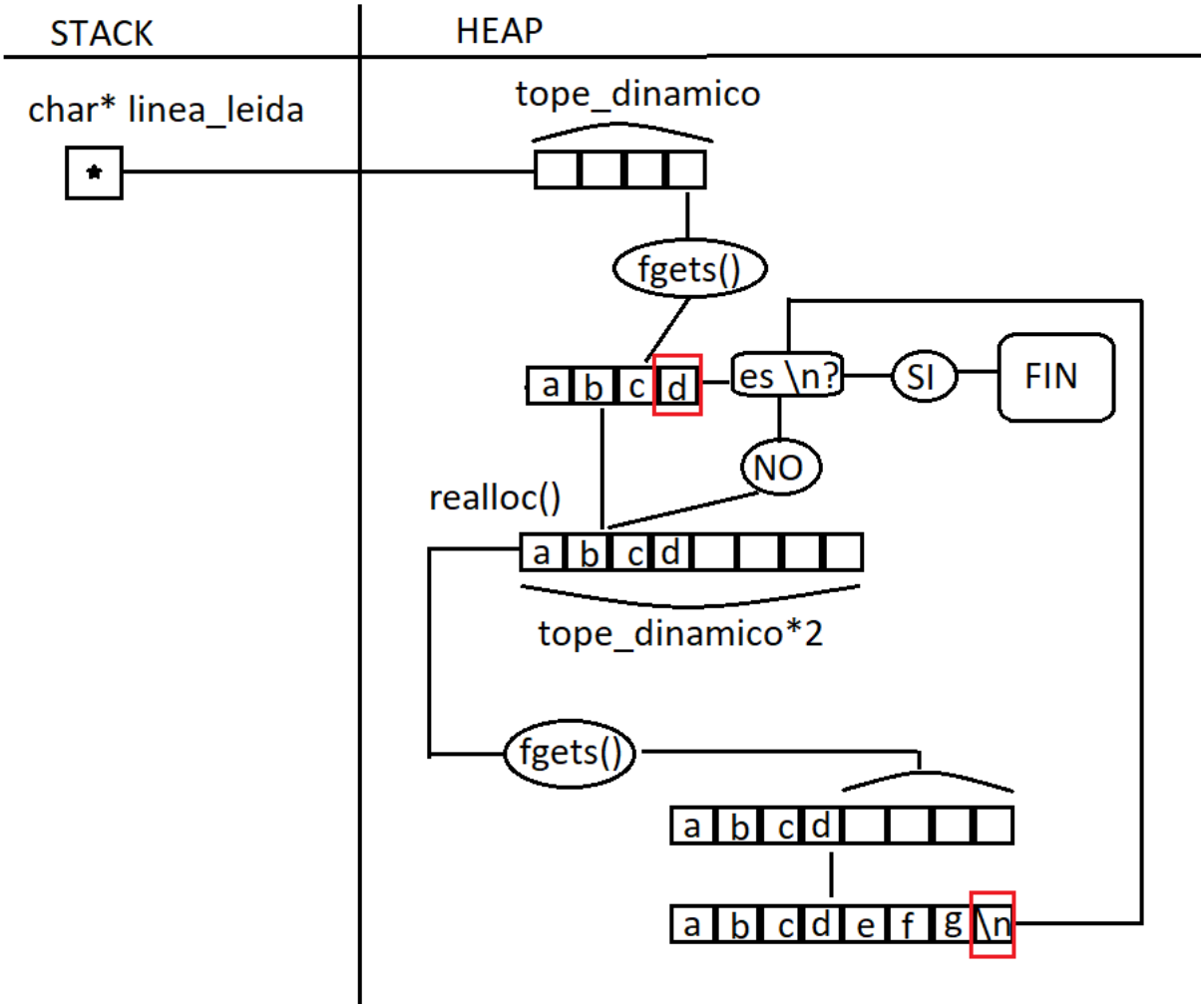



### 3.2.1 leer_linea_de_archivo()

La función leer_linea_de_archivo() tiene como objetivo leer una línea de un archivo con memoria dinámica. Esta función comienza pidiendo memoria en el heap para la línea que se está por leer. Después entra en el loop principal el cual pide leer una línea del archivo hasta un tope_dinamico y guarda lo leído en la memoria pedida. Una vez guardado esto se fija si el último carácter leído es un salto de línea (ósea un "\n"). Si este es un salto de línea significa que toda la línea entro en el tope_dinamico y la función cumplió su cometido por eso devuelve la línea leída, reemplazando el salto de línea por un "\0" señalando

el fin del string. De no ser este el caso entonces el tope_dinamico debe agrandarse. Para lograr esto se hace un realloc() suponiendo como nuevo tope al doble del anterior, así no hacemos tantos reallocs. Una vez hecho este para no tener que volver a leer todo lo que se leyó simplemente le decimos que esta vez lea desde el ultimo carácter leído. Luego de esto simplemente se repite el proceso hasta que se encuentre con un salto de línea.

### 3.1.2 Diagrama leer línea

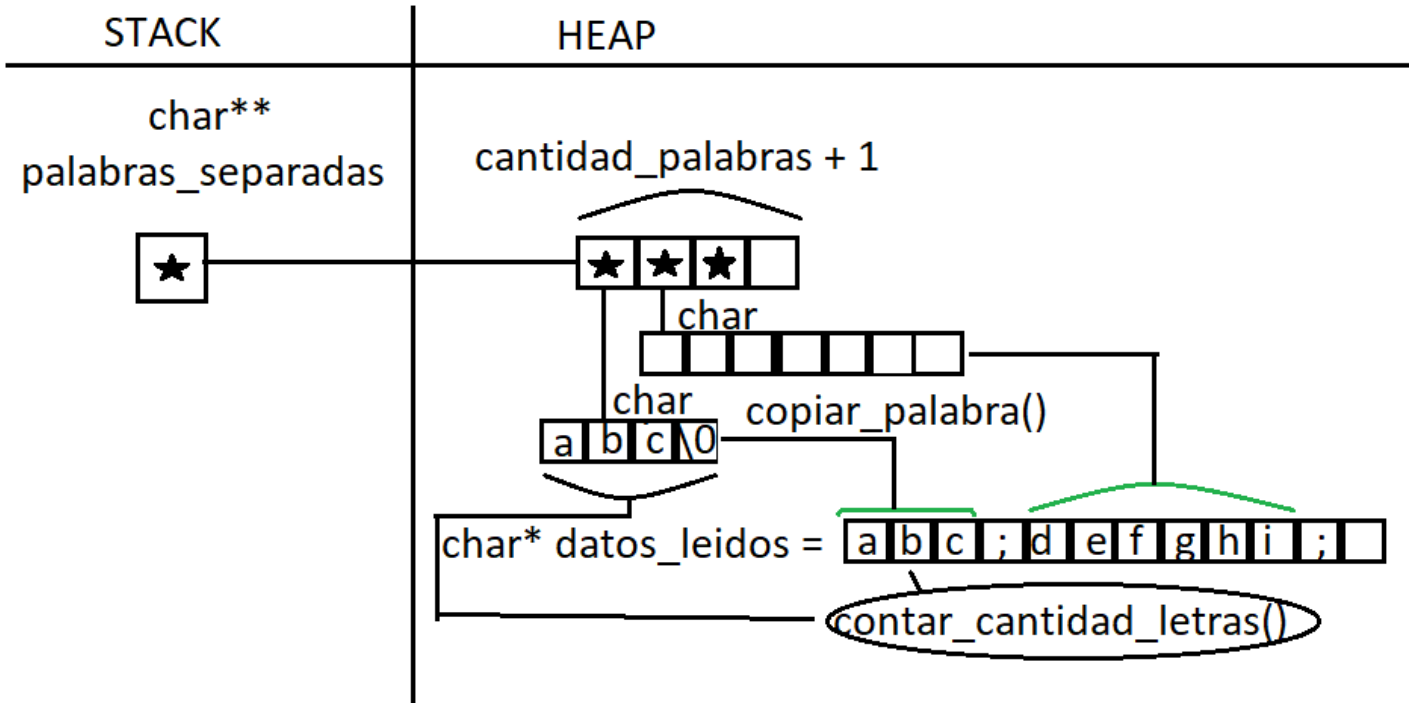



### 3.1.3 split()

El objetivo de split() es separar las palabras que están entre un separador especificado y devolverlas cada una en un vector de strings. Esta

función arranca calculando la cantidad de palabras que tiene el string dado. Luego pide memoria en el heap para esa cantidad de palabras como punteros. Después de esto entra en un loop el cual debe contar la cantidad de letras que ocupa la palabra que esta por copiar, hace espacio en el heap para esta y copia letra por letra. Una vez termina esto se desplaza una posición en el vector de strings y también en el string de donde copia la información. Al finalizar esto vuelve a repetir este proceso una "cantidad_palabras" veces.

### 2.1.2 Diagrama split




### 2.1.4 subir_datos_al_hospital()

La función subir_datos_al_hospital() se encarga de actualizar los datos de un hospital_t con los datos proporcionados por datos_separados.

En primer lugar agranda los vectores pokemon y entrenadores sumandole la cantidad de estos que hay en la linea leida. Esto lo logra gracias a realloc(). Una vez agrando ambos vectores existen 2 funciones que se encargan de actualizar los datos de cada una.

1- actualizar_pkmnes_en_hospital() arranca pidiendo memoria en el heap para el largo del nombre del pokemon que se va a copiar en el hopital. Luego de copia el nombre del pokemon proporcionado por datos_separados. En el caso de los niveles se usa la función atoi() la cual transforma un string en un entero y despues de hacerlo lo copia en el vector_pokemones. La forma en la cual se sabe donde esta el nombre de un pokemon o el nivel de este es sencilla. En el vector datos_separados sabemos que en el index 2 esta el primer nombre de un pokemon, si queremos ir al siguente nombre de un pokemon simplemente tenemos que sumar 2 a este index. Y para los niveles siempre estan a la derecha de los nombre, por eso haciendo index actual + 1 encontramos el nivel que le corresponde a cada uno.
2- actualizar_entrenadores_hospital() hace algo muy parecido a la función anterior pero con la diferencia que se preocupa por la posicion 0 y 1 del vector datos_separados ya que estos son el id y el nombre respectivamente.

### 2.2 ordernar_pokemones_alfabeticamente()

Esta función es utilizada en hospital_a_cada_pokemon(). Sirve para ordenar el vector_pokemones en orden alfabético. El algoritmo que usa para poder ordenarlo es burbujeo. Este consiste en agarrar los primeros dos

elementos del vector y compararlos. Si el primero esta antes en el alfabeto que el segundo simplemente nos quedamos con el segundo y lo comparamos con el tercero. De no ser así se intercambian de lugar y se compara el que estaba primero con el tercero. Este proceso se va repitiendo hasta llegar al final del vector donde sabemos que ese elemento que quedo último es el que esta mas abajo en el alfabeto. Al terminar esto volvemos a repetir este proceso, pero ahora el último elemento con el cual hacemos una comparación va a ser uno menos que la vez anterior.

### 2.2.1 Burbujeo