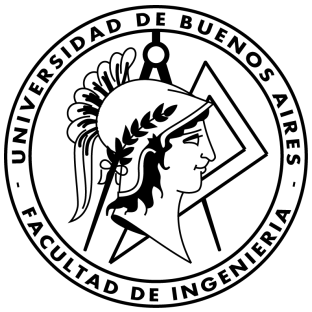

Trabajo Práctico 1 — Escape Pokemon

[7541/9515] Algoritmos y Programación II
Primer cuatrimestre de 2022

Alumno:	De San Vicente, Hernan
Número de padrón:	108800
Email:	hdesanvicente@fi.uba.ar

Índice

1. Introducción

En este trabajo práctico se nos pidió implementar diferentes funciones y procedimientos utilizando memoria dinámica y lectura de archivos. Se nos daba un archivo de estructuras donde ya estaban los structs armados y las funciones a implementar divididas en sus respectivos archivos .c y .h. La regla de estilos a seguir es la sugerida por el equipo de desarrollo del Kernel de Linux.

2. Detalles de implementación

Este trabajo se implementó evitando recorrer los archivos más de una vez y verificando cada dato recibido. Se compila con el comando "gcc escape_pokemon.c src/*.c -o tp1z se ejecuta con la linea "./tp1 <nombre archivo objetos><nombre archivo interacciones>". En fase de desarrollo se utilizó el comando "gcc -std=c99 -Wall -Wconversion -Wtype-limits -pedantic -Werror -O0 escape_pokemon.c src/*.c -o tp1 -g"para compilar y el comando "valgrind –tool=memcheck –leak-check=yes –show-reachable=yes –num-callers=20 –track-fds=yes ./tp1 ejemplo/objetos.txt ejemplo/interacciones.txt" para ejecutar y poder ver si había perdida de memoria.

2.1. Detalles de la función sala_crear_desde_archivos

Esta es la función central del trabajo, que utiliza la mayoría del resto de funciones para cumplir con su objetivo, inicializar completamente una sala y retornar su dirección en memoria.

Empieza declarando e inicializando una variable del struct sala en el heap, junto con sus dos vectores y sus respectivos topes en 0 (Figura 1).

Luego comienza la lectura de cada uno de los archivos, cuyo nombre se recibe por parámetro. Comenzando por el archivo de objetos, cada linea leída es pasada a la función objeto_crear_desde_string y el resultado es guardado en el vector de objetos de la sala (Figura 2), aumentando, a su vez, la variable que indica la cantidad de objetos almacenados. Antes de agregar cualquier objeto al vector y luego de leer la linea se verifica que haya suficiente memoria reservada para almacenarlo, en caso contrario se duplica el tamaño anterior. Al finalizar, se cierra el archivo y se sigue con las interacciones, que siguen el mismo proceso pero llamando a la función interaccion_crear_desde_string y guardando el resultado en el vector de interacciones de la sala.

2.2. Detalles de la función objeto_crear_desde_string

Esta función tiene el objetivo de crear un struct objeto a partir de un string recibido por parámetro que siga el formato preestablecido.

Primero se verifica que el string recibido no sea nulo. Luego se reserva espacio en memoria para el objeto que luego se va a llenar y retornar. Se utiliza sscanf tanto para dividir el string en sus respectivas variables como para verificar que el formato sea el correcto. Por último, se verifica el valor de la última variable para pasarla a tipo booleano. En caso de no poder, se libera la memoria y se retorna null por formato inválido.

2.3. Detalles de la función interaccion_crear_desde_string

Esta función tiene el objetivo de crear un struct interaccion a partir de un string recibido por parámetro que siga el formato preestablecido. El funcionamiento es muy parecido al de la función anterior (objeto_crear_desde_string) salvo en algunos detalles.

Acá se tiene que crear varias variables auxiliares para guardar datos que tienen un formato diferente en la forma en que se van a guardar del string recibido. Estas variables son el objeto_parametro, el objeto de la acción y el tipo de acción. Las primeras dos tiene que ser verificadas, ya que si en el string se encuentran como un "_.^{en}tonces se guardan vacíos. El tipo de acción es muy especifico y debe pasarse a su valor en el enum de estructuras. En caso de no poder pasar uno de estos valores, se libera la memoria y se retorna null por formato inválido.

2.4. Detalles de la función sala_destruir

Esta función se encarga de liberar la memoria de cada nivel del struct sala recibido, yendo de abajo hacia arriba.

Primero se recorren los vectores de objetos e interacciones, liberando cada ítem. Luego se liberan los vectores en sí. Finalmente se libera el struct completo, quedando como en la Figura 3. En caso de la sala ser nula, no se hace nada.

2.5. Detalles de la función sala_obtener_nombre_objetos

Esta función recibe un puntero a una sala reservada en memoria y un puntero a lo que va a ser el tope del vector resultante. Tiene como objetivo crear y retornar un vector reservado en memoria donde se guarden los nombres de los objetos que se encuentran en el vector de objetos de la sala que se recibe por parámetro. Si la sala y la cantidad son nulos, se retorna nulo y no se hace nada. Si solo la sala es nula, se pone cantidad en -1 y se retorna nulo. Si solo la cantidad es nula, aún así se llena y devuelve el vector de nombres.

Simplemente se reserva espacio en memoria para el vector, se recorre el vector de objetos y se van pasando los nombres de un vector a otro. Al final se devuelve el puntero al vector.

2.6. Detalles de la función sala_es_interaccion_valida

Esta función recibe por parámetro algunos de los valores de una interacción y un puntero a una sala reservada en memoria. Ésta determina si las parámetros de interacción recibidos conforman una interacción válida (Que se encuentre en el vector de interacciones de la sala) y en ese caso devolver true, o en el caso contrario false. En caso de encontrarse algún error se devuelve false.

3. Diagramas

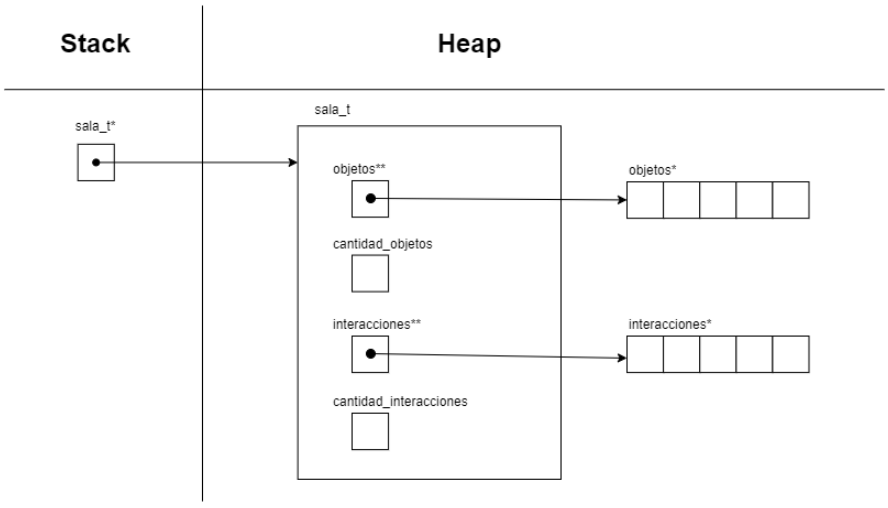

Figura 1: Diagrama de sala con valores principales inicializados.

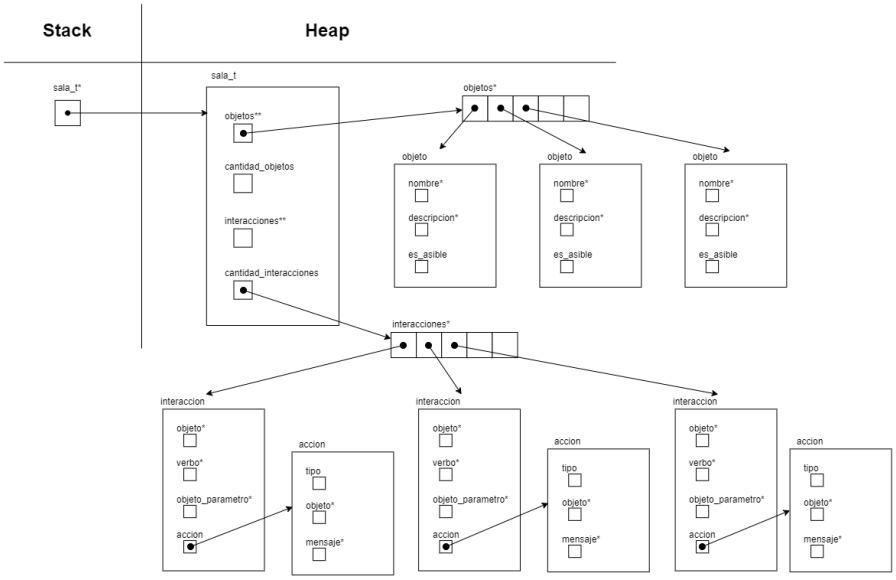

Figura 2: Diagrama de sala con todo inicializado.

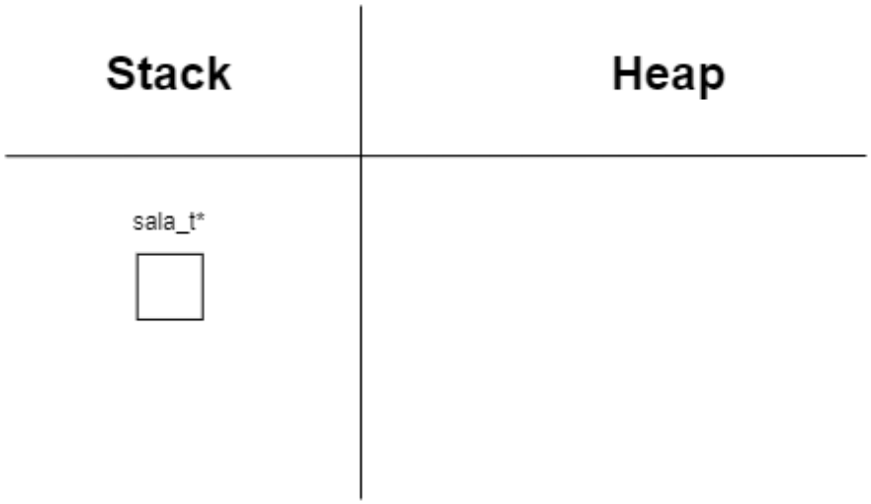

Figura 3: Diagrama de sala luego de ser destruida.