Informe TP1 ~ ESCAPE POKEMON

Padrón: 108951

Nombre: Luciano Andrés Lorenzo

Compilación

El programa se compila con `gcc`, el comando es:

```
gcc escape_pokemon.c src/*.c -o escape_pokemon
```

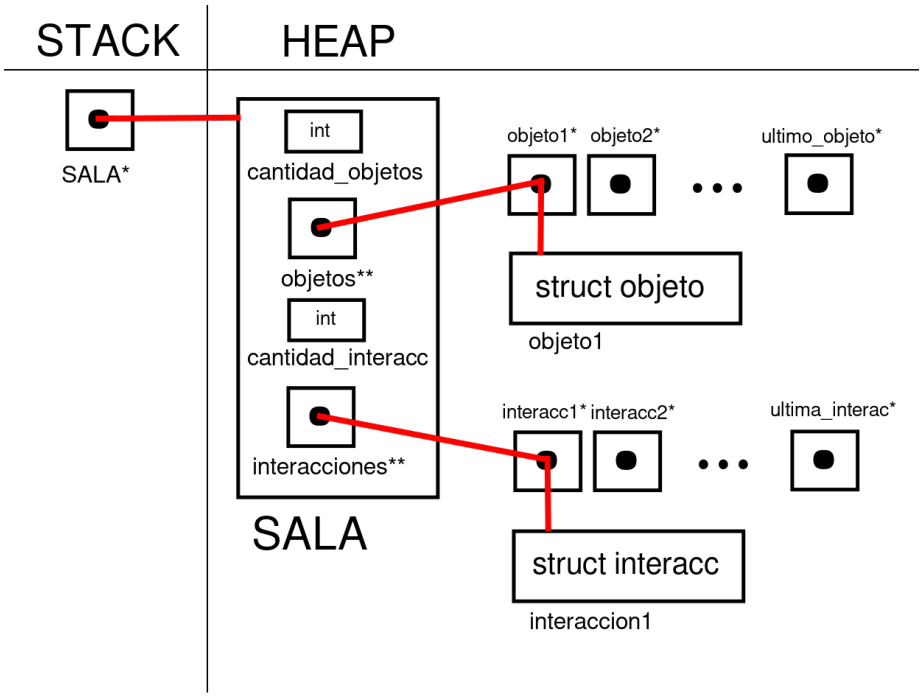
Paso por parametro todos los archivos `.c` que necesita mi programa para correr, con la flag `-o` que tiene el nombre que tendrá el ejecutable (si compila exitosamente el programa). Para correr este ejecutable es tan simple como llamarlo de la siguiente manera:

```
./escape_pokemon
```

Manejo de memoria

Este programa guarda toda la información que necesita sobre la sala de escape en el `heap`, es decir que es alocada dinámicamente. Esto permite que sea más fácil manipular la memoria (poder cambiar el tamaño del bloque comodamente), a cambio de tener que liberar los bloque requeridos cuando ya no van a seguir siendo utilizados.

Lo único que se guarda en el stack es el puntero que apunta hacia el bloque de memoria de la sala que se encuentra en el heap, y sobre este bloque se encuentran todos los punteros hacia los demás bloques de memoria que reserva el programa. Esquema de la memoria del programa:



Para alocar dinámicamente memoria se utilizan las funciones `malloc` y `realloc`, ambas reciben la cantidad de tamaño en `bytes` para alocar en el heap y devuelven un puntero a ese bloque de memoria, que luego debe ser liberado con la función `free`. La única diferencia entre ambas es que `realloc` tambien recibe (opcionalmente) un puntero hacia un bloque dinámico, el cual modifica y le asigna el nuevo tamaño.

En el programa todo es liberado en la función `sala_destruir`, que recorre cada bloque alocado en la sala y los libera. Libera empezando desde los objetos/interacciones hasta la sala, ya que si liberase la sala primero, perdería todos los punteros para liberar los objetos y las interacciones.

Lectura de archivos

Para leer los archivos se utilizan funciones estándar de `C`.

Para abrir el archivo utilizo la función `fopen`, que recibe el nombre del archivo, un char que indica el modo en el que abro el archivo y devuelve un puntero de tipo `FILE`.

Para leer cada linea del archivo, se usa la función `fgets`, que recibe una string (vector de chars) en donde va a ser guardada los caracteres de la linea del archivo, el máximo de chars que puede leer y el puntero al archivo donde se debe leer. Esta función devuelve un `char*`, en donde estará el texto si es que salió todo correctamente (en caso contrario devuelve `NULL`).

Al no conocer la cantidad de líneas que tiene el archivo, aprovecho que la función devuelve `NULL` cuando no hay más líneas y utilizo un while que se ejecutara mientras que la función devuelva diferente de `NULL`:

```
char linea[20];
while (fgets(linea, 20, archivo_objetos) != NULL){
   /* procesamiento de línea */
}
```