



Trabajo Práctico N2 — ESCAPE POKEMON

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

Alumno:	Dall'Acqua Denise
Número de padrón:	108645
Email:	Denisedallacqua10@gmail.com

Índice

1.	Enunciado	2
2.	Introducción	3
3.	Compilación del programa	3
4.	Detalles de implementación	4
5.	Interfaz	5
6.	Cambios en las pruebas	6

1. Enunciado

Vamos a reutilizar la mayor parte del TP1 implementado al principio del cuatrimestre. Para esta segunda parte se pide incorporar algunos cambios y explicar algunas cuestiones detalladas en los siguientes items:

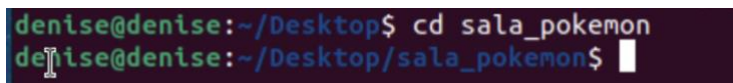
- La definicion de la estructura de la sala fue movida al archivo sala.c.
Haga las modificaciones que crea necesarias a la estructura sala, de modo tal que se utilicen los TDA implementados a lo largo del cuatrimestre.
- Explique y justifique por qué utiliza los TDA seleccionados para cada caso. ¿Falla alguna prueba al realizar estos cambios? ¿Por qué?
En caso de que alguna prueba falle, muestre como arreglarla.
- La definicion del enumerado de acciones fue movido al .h de sala. Recuerde que este archivo es el único archivo de cabecera visible para el usuario de la biblioteca (el único archivo que puede ser incluido desde el o los archivos que implementen la lógica del juego.)
- Se agrega al enumerado el tipo de acción ESCAPAR que corresponde al caracter 'g' en los archivos de acción.
- Implemente las nuevas funciones agregadas a sala.h
- Implemente en el archivo escape pokemon.c la lógica del juego (utilizando la sala).
- Si lo necesita, puede crear archivos extra dentro de src para separar funcionalidades. Recuerde que NO está permitido incluir los archivos de cabecera privados. Solamente está permitido hacer include de sala.h.
- Una vez completada la implementación, Hay que subirla al sistema de entregas, para verificar su correcto funcionamiento. Además de la implementación, se debe entregar un informe (ya sea en formato PDF o video de 5 minutos o menos). En el informe tenés que explicarle a tu corrector cómo se compila y corre tu programa y se deben discutir las cuestiones pedidas anteriormente. Adicionalmente se deben agregar todas las consideraciones que sean importantes para el corrector. El informe es parte de la nota.

2. Introducción

En este informe desarrollaré los conceptos e implementaciones que se han utilizado para el desarrollo de este trabajo práctico. Se trata de un juego inicializado por medio de dos archivos de texto, los cuales poseen la información necesaria sobre los elementos que lo componen y las interacciones entre estos.

3. Compilación del programa

Para poder acceder al juego debemos ingresar a la carpeta “sala_pokemon” (puede que al compimir los archivos la carpeta se llame “escape_pokemon_denise_dallacqua”) desde el directorio donde se haya descargado el archivo comprimido “denise_dallacqua_escape_pokemon.zip”. Cuando accedemos sacamos el makefile y pruebas.c para que nos centremos en la ejecución del juego.



```
denise@denise:~/Desktop$ cd sala_pokemon
denise@denise:~/Desktop/sala_pokemon$
```

Figura 1. Primer paso para la compilación del juego

Luego al acceder a la carpeta, debemos compilar con el comando gcc los archivos escape_pokemon.c e interfaz.c con el comando *.c y los archivos que están dentro de la carpeta src con el comando src/*.c y luego los correspondientes flags que se verán a continuación en la figura 2.



```
denise@denise:~$ gcc *.c src/*.c -o juego -std=c99 -Werror -Wall -Wconversion -Wtype-limits
```

Figura 2. Segundo paso para la compilación del juego

Cuando se ha realizado la compilación exitosamente, para que pueda correr nuestro juego necesitamos la presencia de dos archivos fundamentales: uno que contenga los objetos del juego y en otro las interacciones que hay entre ellos. Se puede insertar los objetos e interacciones que queramos, dándole la temática al juego mientras tengan el formato correspondiente para cada archivo.

En el archivo de objetos, su formato para la extracción correcta de la información es “nombre;descripción;es_asible”. Nombre del objeto, alguna descripción que se haga de él y si se puede agarrar o no.

Luego es el archivos de interacción, su formato para la extracción de la información es “objeto;verbo;objeto_parametro;tipo_accion:objeto_accion:mensaje”. Objeto y objeto parámetro son los objetos que están involucrados en la interacción, verbo es lo que se está realizando con ellos, el tipo de acción esta pautado por 5 tipos: MOSTRAR_MENSAJE, ELIMINAR_OBJETO, DESCUBRIR_OBJETO, ESCAPAR, REEMPLAZAR_OBJETO; el objeto en consecuencia a la acción y un mensaje.

En este caso, se utilizarán los archivos objetos.txt e interacciones.txt de la carpeta info_obj_int utilizando el path **info_obj_int/objetos.txt** **info_obj_int/interacciones.txt** para acceder a los archivos correspondientes. Y los escribimos luego del nombre del ejecutable. Y finalmente acceder al juego.

```
denise@denise:~/Desktop/sala_pokemon$ gcc escape_pokemon.c src/*.c -o juego -std=c99 -Werror -Wall -Wconversion -Wtype-limits
denise@denise:~/Desktop/sala_pokemon$ ./juego info_obj_int/objetos.txt info_obj_int/interacciones.txt
```

Si se quiere jugar con los archivos que he implementado de ejemplo, se debe realizar el mismo algoritmo que lo anterior, nada más que al ejecutarlo hay que escribir:

```
denise@denise:~/Desktop/sala_pokemon$ ./juego mi_ejemplo_juego/objetos_pruebas.txt mi_ejemplo_juego/ejemplo_interacciones.txt
```

4. Detalles de implementación

En la estructura que lleva la información sobre todo el lugar y lo que el jugador puede ver, conocer y poseer; llamada “sala”; he seleccionado los TDA lista y hash para su implementación.

Dentro de sala se pueden observar cuatro punteros:

Uno de ellos es un puntero de tipo lista_t, que contiene las interacciones. Me ha parecido eficiente utilizar este TDA ya que, no hay problemas con el almacenamiento de las mismas interacciones, si quería utilizar un hash, por ejemplo, iba a ser mucho más complicado porque todos los campos de la estructura “interacciones” pueden repetirse y en consecuencia reemplazarse una con otra; y la implementación del iterador externo para la función ejecutar_interacciones fue de gran ayuda para las comparaciones y la implementación de sus acciones en relación al tipo de acción.

Luego, los tres punteros restantes son de tipo hash_t, en donde se almacenan los objetos dependiendo de la situación en la que se encuentra dentro del juego: el primero almacena todos los objetos del juego, el segundo los que el jugador conoce y el tercero los que el jugador posee. Me ha parecido el TDA más eficiente por su búsqueda por medio de la clave, que la he definido en base a su nombre ya que los objetos no se repiten, y el rápido manejo de la inserción, eliminación, obtención de los elementos para el desarrollo del juego.

Y finalmente, tenemos una variable booleana que nos indica si pudo escapar de la sala o no, es decir, si pudo ganar o no.

```
struct sala
{
    hash_t *objetos;
    lista_t *interacciones;
    hash_t *objetos_conocidos;
    hash_t *objetos_poseidos;
    bool pudo_escapar;
};
```

Figura 4. Estructura de la sala

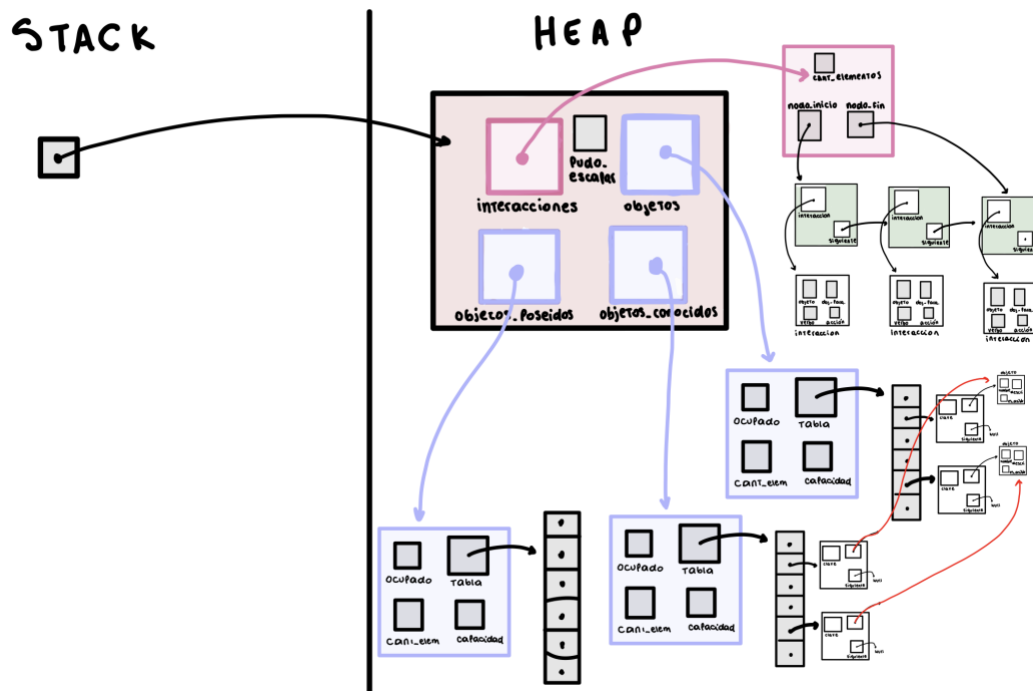


Figura 5. Representación en memoria de la sala inicializada. Los valores de las cantidades son solamente de ejemplo y no ilustran las cantidades reales que están en los archivos de ejemplo.

5. Interfaz

Como el juego era en base a Pokémon, he tratado de seguir la temática desarrollando una interfaz en la cual aparece uno de los personajes del anime, “pikachu”, explicando los comandos correspondientes para el desarrollo del juego: agarrar objeto, ayuda, describir objeto y salir del juego. Las interacciones internas del juego no se han puesto en el menú ya que como estas dependen del archivo que le insertamos, las interacciones como los objetos pueden cambiar, y además no se puede acceder directamente a la estructura de las interacciones para poder recorrer la lista y guardarme las interacciones para mostrárselas al jugador, así que he tomado como norma que el jugador debe descubrir esas acciones claves para que pueda conocer los objetos he interactuar con ellos.

Para la implementación de la interfaz gráfica he creado una biblioteca “*interfaz.h*” que tiene las funciones necesarias para utilizar en *escape_pokemon.c*. Además, para probar mi trabajo con otros archivos me he creado mis objetos e interacciones (ubicados en la carpeta *mi_ejemplo_juego*) ambientado en un aula de un colegio o facultad. Es la misma temática de búsqueda que los ejemplos implementados en *chanutron't* y el ejemplo dado.

Para poder interactuar con el juego, el jugador debe introducir el comando dependiendo de lo que quiera realizar: si su comando lleva dos palabras, entonces le debe de agregar la palabra “pika” (en referencia a Pikachu); si lleva tres palabras, entonces solo pone esas dos. Si el jugador pone un comando que no se reconoce, se le avisa con un cartel recordándole como introducir el comando. Además, se ha aclarado en las instrucciones del juego que por contexto va a tener que adivinar que verbos se utilizan para la ejecución de cada acción, ya que, al no poder acceder a la estructura de interacciones, no se puede saber que interacciones están inicializadas.

6. Cambios en las pruebas

Para las pruebas tuve que modificar la mayoría en las que se relacionaban con los vectores dinámicos que se habían implementado anteriormente en el trabajo practico 1.

- En las pruebas “pruebasCrearObjeto” y “PruebasCrearInteracciones” no se han hecho cambios, porque la obtención de la información por medio de los archivos es la misma.
- En la prueba “pruebas_crear_sala”, para poder saber si se ha insertado la cantidad correcta de objetos e interacciones al inicializarse la sala, tuve que utilizar las funciones primitivas de cada TDA.
- En la prueba “pruebas_nombre_objeto” en la primera parte donde se obtiene los vectores, por medio de una sala null, cantidad null y correctamente no se han modificado, pero si se tuvo que modificar la parte en la que se recorre al vector comparando cada lugar para ver si corresponde al objeto indicado. Como el hash inserta dependiendo del número que nos da como resultado su “función de hash” y no inserta en un orden que se sepa; al utilizar la función “hash_con_cada_clave”, que va recorriendo todo el hash desde la posición cero hasta su tope, la inserción al vector va a depender de cómo se haya insertado, entonces comparar de posición a posición con el vector “esperados” que tenemos en las pruebas no es lógico. Entonces, se ha modificado el bucle de manera tal que recorra el vector hasta que encuentre el objeto con el que se compara. Sin embargo, esto tiene una pequeña falla porque si no se encuentra el objeto, el bucle puede quedar en un bucle infinito de comparaciones, entonces agregue una variable llamada “vueltas” que cuenta cuantas vueltas realiza en el vector de objetos2, cada vez que mi variable “j” (la que recorre el vector) vuelve a cero, significando que puede que dio media vuelta o una vuelta completa este incrementa, y si la j vuelve otra vez a cero(cuando llego al tope), el bucle se corta porque ya recorrió todo el vector y no encontró el elemento.
- En las pruebas “pruebas_interacciones” no se han hecho modificaciones.
- Por último, se han implementado funciones para las nuevas funciones.