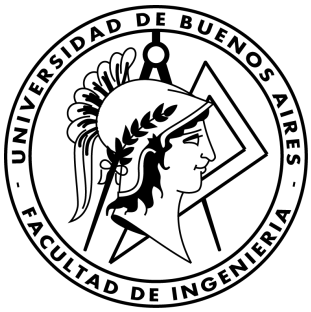
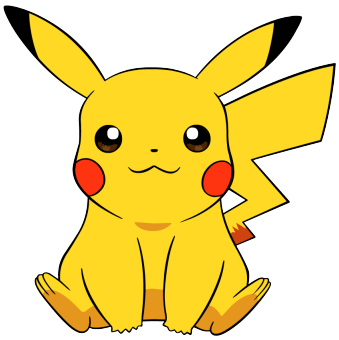


Trabajo Práctico 1
Sala de escape Pokemon



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

Alumno:	Davies, Alen
Número de padrón:	107084
Email:	adavies@fi.uba.ar

Índice

1. Introducción

La idea general del TP es crear una sala de escape que dentro tiene objetos con los cuales se puede interactuar. Los objetos de la sala y las interacciones que se pueden realizar con ellos los debemos crear a partir de dos archivos de texto, proporcionados por la cátedra en un archivo .zip, que poseen la información de cada objeto y cada interacción. Luego, a partir de la sala, los objetos y las interacciones creados, se pide imprimir por pantalla los objetos y chequear si son válidas o no algunas de las interacciones detalladas en la consigna del TP según la información en el archivo de texto.
Dentro del archivo de objetos se detalla para cada uno:

- Nombre del objeto.
- Una descripción del objeto.
- un indicador: es true si el objeto es asible y sino false.

Para cada interacción, en el archivo de texto, se provee lo siguiente:

- Un objeto con el que se puede interactuar.
- El verbo de la interacción a realizar con el objeto.
- Un segundo objeto en caso de que sea necesario para realizar la interacción.
- Una acción que posee la siguiente información:
  - El tipo de acción, que puede ser:
    - Descubrir objetos.
    - Reemplazar objetos.
    - Eliminar objetos.
    - Mostrar un mensaje.
  - El nombre del objeto resultante de la acción realizada.
  - Un mensaje relacionado con la acción.

Además de los dos archivos de texto, en el archivo .zip provisto se encuentra otra carpeta la cual contiene archivos .c y .h con estructuras de datos y funciones que se deben implementar, y luego utilizar para cumplir con el objetivo, a partir de su firma y una descripción de cada una.

2. Teoría

1. ¿Cómo se compila y corre un programa?

   El código escrito en un lenguaje de programación se denomina código fuente, una vez que éste código está listo se utiliza un compilador para traducir el código fuente en lenguaje que una computadora pueda ejecutar. En este caso se usó el compilador del lenguaje C, llamado gcc y algunos flags de compilación para detectar más errores en el código.
   Luego de haber compilado el código, se crea un archivo ejecutable del programa que se puede correr para probar su funcionamiento.
   Si se quiere verificar errores de memoria en el programa se puede utilizar valgrind al ejecutar el programa.
   Las líneas que deben utilizarse para compilar y ejecutar mi programa serán detalladas en la sección 3.

2. ¿Cómo funciona la lectura de archivos?

   C trata a los archivos como punteros, por ello se define un nuevo tipo de dato llamado **FILE*** que es un puntero a un archivo. El uso de archivos posee algunas ventajas:

- Permite que persista en el tiempo la información de nuestros programas (a diferencia de los arreglos). Es decir que queda almacenada una vez que el algoritmo termino da hacer uso de la misma. La única forma de perderlos es debido a un error del algoritmo o de un problema del dispositivo de almacenamiento.
- Su tamaño puede variar en tiempo de ejecución, esto hace que virtualmente pueda contener cualquier cantidad de elementos.

Para manipular archivos en C, utilizamos funciones incluidas en la biblioteca estándar de C, **stdio.h**. Lo primero que debemos hacer para utilizar un archivo, es abrirlo con la función **fopen()** y creando una variable de tipo **FILE***.

```
fopen(const char *filename , const char *mode);
```

Donde **filename** es el nombre del archivo que se quiere abrir y **mode** es el modo de apertura de ese archivo, puede ser **"r"** para lectura, **"w"** para escritura, entre otros. Ésta función devuelve un puntero a un archivo y puede devolver **NULL** en caso que haya habido un error con la apertura del archivo, por esto siempre debemos chequear que se haya podido abrir correctamente el archivo antes de continuar con el código.
Una vez abierto el archivo deseado, podemos empezar a leerlo utilizando alguna función como **fgets()** o **scanf()** que nos permiten leer el archivo línea por línea. Siempre debemos chequear si se llegó al final del archivo cuando estemos leyéndolo, validando si las funciones antes mencionadas devuelven **NULL** o también se puede verificar con otra función, **feof()**.

```
fgets(char *string, int n, FILE *archivo);
```

El primer parámetro **string** es un puntero a un espacio de memoria del tipo char (podríamos usar un arreglo de char). El segundo parámetro es **n** que es el limite en cantidad de caracteres a leer para la función fgets. Y por último el puntero del **archivo** que es la forma en que fgets sabrá que archivo debe leer. Esta función devuelve el mismo parámetro **string** o **NULL** si se llegó al final del archivo, no pudo leer nada o si ocurrió un error durante la lectura, por ello mismo también debemos chequear que no devuelva **NULL** antes de continuar con el programa.
Cuando terminamos de utilizar un archivo previamente abierto, siempre debemos cerrarlo para que no se produzcan errores en nuestro programa. Esto se hace con la función **fclose()** pasándole por parámetro el puntero al archivo que queremos cerrar.

```
fclose(FILE *archivo);
```

3. ¿Cómo se maneja la memoria dinámica?

La memoria dinámica es aquella que será reservada y utilizada únicamente en tiempo de ejecución, y será manejada por el programador. La memoria dinámica es reservada en el heap del programa.
Para utilizar este tipo de memoria en C se utilizan funciones de la biblioteca estándar:

- **malloc()**: Se usa para decirle al sistema que se desea utilizar memoria dinámica. Esta función reserva los bytes especificados por parámetro y devuelve un puntero al bloque de memoria reservado. Puede devolver NULL en caso de error al reservar la memoria pedida.

```
malloc(size_t tamanio);
```

- **calloc()**: Reserva memoria para cualquier arreglo de n elementos del tamaño pasado por parámetro y devuelve un puntero al inicio de la memoria reservada o NULL si hubo algún error. Toda la memoria se inicializa en cero.

```
calloc(size_t n_elementos, size_t tamanio);
```

- **realloc()**: Modifica el tamaño del bloque de memoria apuntado por el puntero en la cantidad de bytes especificados por parámetro. El contenido del bloque de memoria permanecerá sin cambios.

```
realloc(void *puntero , size_t tamanio);
```

- **free()**: Libera el espacio de memoria apuntado por el puntero que fue previamente reservado con **malloc()**, **calloc()** o **realloc()**.

```
free(void *puntero);
```

Cuando reservamos memoria dinámica podemos utilizar el operador **sizeof()** que calcula el tamaño de cualquier estructura o tipo de dato.
Como se menciona más arriba, todas las funciones que sirven para reservar memoria dinámica devuelven **NULL** en caso de que haya habido algún error, por esto es muy importante siempre chequear lo devuelto por esas funciones antes de continuar con nuestro programa.
Ya que la memoria dinámica es controlada exclusivamente por el programador, él decide cuando pedirla y cuando liberarla. Si no se libera la memoria dinámica reservada se pueden producir problemas en el programa, por esto siempre de debe chequear que se libere toda la memoria reservada, cuando ya no se necesite, al final de nuestro programa. Para controlar que no se esté perdiendo memoria en nuestro programa se utiliza Valgrind.

3. Detalles de implementación

Para implementar lo pedido en la consigna, primero en la función de sala.c, sala_crear_desde_archivo, desde el main contenido en escape_pokemon.c se pasaron por parámetro los archivos de texto a utilizar en el programa. Ya dentro de la función, lo primero fue abrir ambos archivos en formato lectura y reservar memoria para la estructura sala que se debía crear llenándola con el contenido de los archivos. Ya teniendo los archivos abiertos y la memoria reservada, se comenzó a leer los archivos línea por línea.
Cada línea fue pasada por parámetro a las funciones crear_vector_objeto y crear_vector_interacción, ambas en sala.c, en las cuales se crean los vectores dinámicos contenidos en la estructura sala. Para crear estos vectores hubo que reservar memoria por cada elemento que se quería agregar a estos. Los elementos a agregar a los vectores dinámicos también debieron ser creados en sus respectivas funciones a las cuales se les pasó por parámetro las líneas de los archivos. Las funciones encargadas de esto fueron objeto_crear_desde_string contenida en objeto.c y a interacción_crear_desde_string contenida en interacción.c donde cada una fue parseada para crear cada objeto y cada interacción presente en los archivos. Luego de parsear cada una de las líneas de los archivos, hubo que reservar memoria también para cada objeto e interacción a crear que luego fueron devueltos a la función sala_crear_desde_archivo para incorporarlos a la sala.
Ya teniendo la sala totalmente creada, con sus respectivos campos llenos, se devolvió al main para ser utilizada más adelante.
Además de crear la sala, en la función de sala.c, sala_obtener_nombre_objetos, se pedía crear un vector con los nombres de los objetos de la sala de forma dinámica, para el cual también se debió reservar memoria antes de llenarlo. Ésta función recibió desde el main la sala ya cargada y una variable cantidad que debía ser devuelta al main con el valor de la cantidad de los objetos de la sala. Luego de tener el vector cargado de nombres, ésta función se utilizó en el main para imprimir por pantalla los nombres de los objetos como se pedía en la consigna del TP.
En la consigna también se pedía validar algunas interacciones según la información contenida en el archivo de texto, para ello se utilizó la función de sala.c, sala_es_interacción_valida la cual recibió por parámetros

la sala cargada, un verbo, un objeto y un segundo objeto en caso de ser necesario para realizar la interacción. Estos parámetros fueron comparados con los campos de la estructura interacción y en caso de coincidir, la función debía devolver true y sino false. Luego en base a esto se pedía imprimir por pantalla desde el main, si dichas interacciones a validar eran efectivamente válidas o inválidas.
Al terminar de cumplir con todos los items de la consigna, ya podíamos liberar desde el main toda la memoria dinámica que reservamos durante nuestro programa. Para esto se utilizó otra función de sala.c, sala_destruir.c la cual se encargaba de liberar toda la memoria reservada pero que pertenecía a la estructura sala. Para el vector de nombres, se liberó la memoria a parte.
Las funciones utilizadas en la implementación del programa serán detalladas en la siguiente sección.

Mi programa se compila con la siguiente línea:

```
gcc escape_pokemon.c src/*.c -o escape_pokemon -g -O0 -std=c99 -Wall -Wconversion -Wtype-limits -
    Werror
```

Para ejecutar el programa desde la consola:

```
./escape_pokemon ejemplo/objetos.txt ejemplo/interacciones.txt
```

Y para chequear errores de memoria en el mismo:

```
valgrind ./escape_pokemon ejemplo/objetos.txt ejemplo/interacciones.txt
```

3.1. sala_crear_desde_archivo()

En ésta sección se dará una explicación más detallada de lo que ocurre dentro de ésta función que es la función principal del programa.
Cómo se mencionó más arriba, ésta función recibe por parámetro el path de los archivos que serán utilizados para crear la sala de escape y sus respectivos campos.
Dentro de ésta función se deben abrir ambos archivos de texto en formato de lectura y chequear que se hayan podido abrir correctamente, luego se crea un puntero a sala y se reserva memoria para ésta con malloc (siempre chequeando que no retorne NULL). Luego de esto, se llama a la función privada crear_vector_objetos para comenzar a crear el vector dinámico de objetos la cual recibe por parámetros la variable FILE* del archivo de objetos y un tope que se utilizará para llenar el vector.
Dentro de ésta función privada se comienza a leer el archivo de texto, linea por línea, utilizando en un while la función fgets() con la cual chequeamos que no se haya llegado al final del archivo. Dentro de éste while, la primera vez que se ingresa se reserva memoria con malloc y se crea un puntero a la primer posición del bloque de memoria reservado que será la primera posición del vector dinámico. Las demás veces que se entre al while para leer otra línea, se debe agrandar el bloque de memoria, previamente reservado, con realloc (o crear un nuevo bloque más grande) para poder agregar los siguientes punteros a objetos al vector. (cada vez que se reserva memoria se debe chequear que no esté retornando NULL como se mencionó anteriormente).
Cuando se termine de leer el archivo de texto, se sale del while y se retorna el puntero doble y el tamaño del vector a la función sala_crear_desde_archivo, en la cual se asignan a los respectivos campos de la sala (si no hubo ningún error en el proceso).
Todo lo detallado para el vector dinámico de objetos, es análogo para el vector dinámico de interacciones.

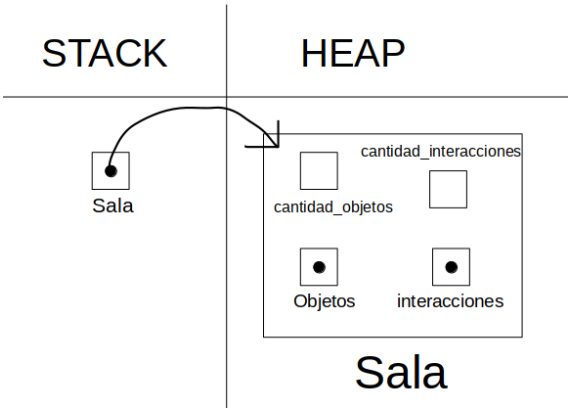
3.2. objeto_crear_desde_string()

Ésta función es llamada por la función privada crear_vector_objetos y ésta a su vez se llama desde sala_crear_desde_archivo donde finalmente se crea la sala con sus campos.
La función objeto_crear_desde_string recibe por parámetro un string que es una línea del archivo de objetos. Ésta línea se debe parsear para poder crear un objeto ya que ésta tiene el formato **nombre_objeto;descripción;flag**.

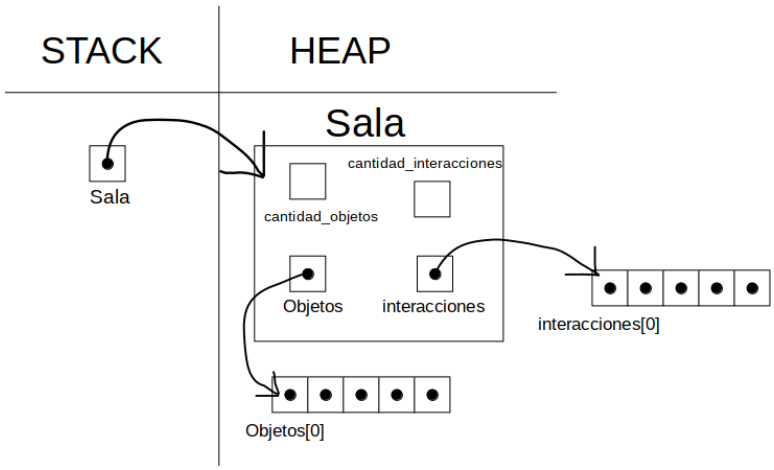
Para parsearla se utilizó la función sscanf que divide la línea y llena las variables que se le indiquen con los datos extraídos de la línea parseada. Ésta función devuelve la cantidad de parámetros que pudo llenar correctamente por lo que se debe chequear que hayan sido los correctos antes de continuar con el programa. Ya teniendo los datos de la línea parseados y guardados en variables, se crea un puntero a un objeto y se reserva memoria para éste con malloc (chequeando que no devuelva NULL). Si no ocurre ningún error durante el proceso, entonces se puede asignar los valores extraídos del string parseado al puntero creado y devolverlo a la función crear_vector_objetos.
Para crear las interacciones se siguió el mismo procedimiento.

4. Diagramas

Cuando se reserva memoria con malloc para la estructura sala, se crea un puntero a la sala. La sala a su vez adentro tiene dos enteros y dos vectores dinámicos. Esto se puede representar con el siguiente diagrama:



Luego se reserva un bloque de memoria con malloc para los vectores dinámicos, objetos e interacciones, creando un puntero a la primera posición del bloque reservado. A medida que se van llenando los vectores dinámicos con punteros a objetos e interacciones respectivamente, se va ampliando el bloque de memoria previamente reservado (o reservando un bloque nuevo más grande) con realloc. Como se ve a continuación en el diagrama:



Todo visto en los anteriores diagramas ocurre dentro de la función sala_crear_desde_archivo, ahora lo que se muestra es cuando se va cargando cada vector dinámico. En este proceso se llama a las funciones

objeto_crear_desde_string e interacción_crear_desde_string dependiendo de cual vector se esté llenando. Dentro de esas funciones se reserva memoria para una interacción o un objeto utilizando malloc. Como resultado de esto se crea un puntero a un objeto o un puntero a una interacción que es devuelto a las funciones privadas crear_vector_interacción o a crear_vector_objetos que a su vez son llamadas por sala_crear_desde_archivo.