FACULTAD DE INGENIERIA
Universidad de Buenos Aires

95.11 - Algoritmos y Programación I

Trabajo práctico N.° 2: Altas/Bajas/Modificaciones

Nombre	Correo Electrónico	Padrón
Santiago Viegas Bordeira	santiago.bordeira@gmail.com	99634
Matilde Cuenca	maticuenca84@gmail.com	99319
Pedro Aguirre	pedro.aguirre2395@gmail.com	97603

Fecha de entrega: 27/10/2017

Corrector: Patricio Moreno

2° cuatrimestre de 2017.

Índice

1. Objetivo

El objetivo del trabajo consiste en la realización de un conjunto de aplicaciones a través de la consola por medios de línea de comandos, que manipulen un conjunto de archivos, en forma binaria y de texto plano, y permitan la modificación de su contenido, ya sea agregando, quitando o modificando información.

2. Alcance

Mediante el presente trabajo se busca que los alumnos adquieran y apliquen conocimientos sobre los siguientes temas:

- Arreglos/Vectores
- Memoria dinámica
- Argumentos en línea de orden
- Estructuras
- Archivos
- Modularización

3. Desarrollo

3.1 Funcionamiento del programa

El conjunto de programas consiste en la manipulación y modificación de archivos por medio de la utilización de argumentos de línea de orden (CLA).

La idea general es la de invocar a un programa que reciba por CLA un archivo que contenga la base de datos, un archivo que contenga los registros a modificar y otro donde se almacenan estructuras erróneas, es decir, ids ya existentes en la operación de altas y/o ids inexistentes en la operación de bajas o modificaciones. En dicha invocación, se especificará si el usuario desea realizar una alta, una baja o una modificación.

Como la base de datos a recibir es un archivo binario, se debe realizar un programa aparte que reciba un archivo CSV y devuelva como salida un archivo binario.

El usuario podría desear, luego de realizar una alta, baja o modificación, corroborar que dicha modificación al archivo binario se haya realizado correctamente. Para ello, se realiza un programa que realice una desbinarización de dicho archivo y muestre su contenido por pantalla.

Se ve entonces que el trabajo requiere de 3 programas, que serán explicados por separado a lo largo del presente informe.

3.2 Descripción del código y criterios utilizados

Para el presente informe, se trabaja con un archivo en formato CSV que contiene información sobre películas, como se puede ver en la figura 1.

```
1,Jaws,Steven Spielberg,Peter Benchley,20/06/1975,56,951
2,Titanic,James Cameron,James Cameron,19/12/1997,45,56
3,Gravity,Alfonso Cuaron,Alfonso Cuaron,28/08/2013,99,465
4,The Terminator,James Cameron,James Cameron,26/07/1984,88,95
5,The Expendable,Sylvester Stallone,Avi Lerner,6/12/1997,45,962
6,The Fast and the Furious, Vin Diesel, Gary Gray,22/05/2001,45,66,66
7,Harry Potter, El prisionero de Azkaban,01/05/07,6.45,76,54
```

Figura 1: archivo en formato CSV a utilizar en el programa

La estructura del archivo CSV se puede apreciar en la tabla 1.

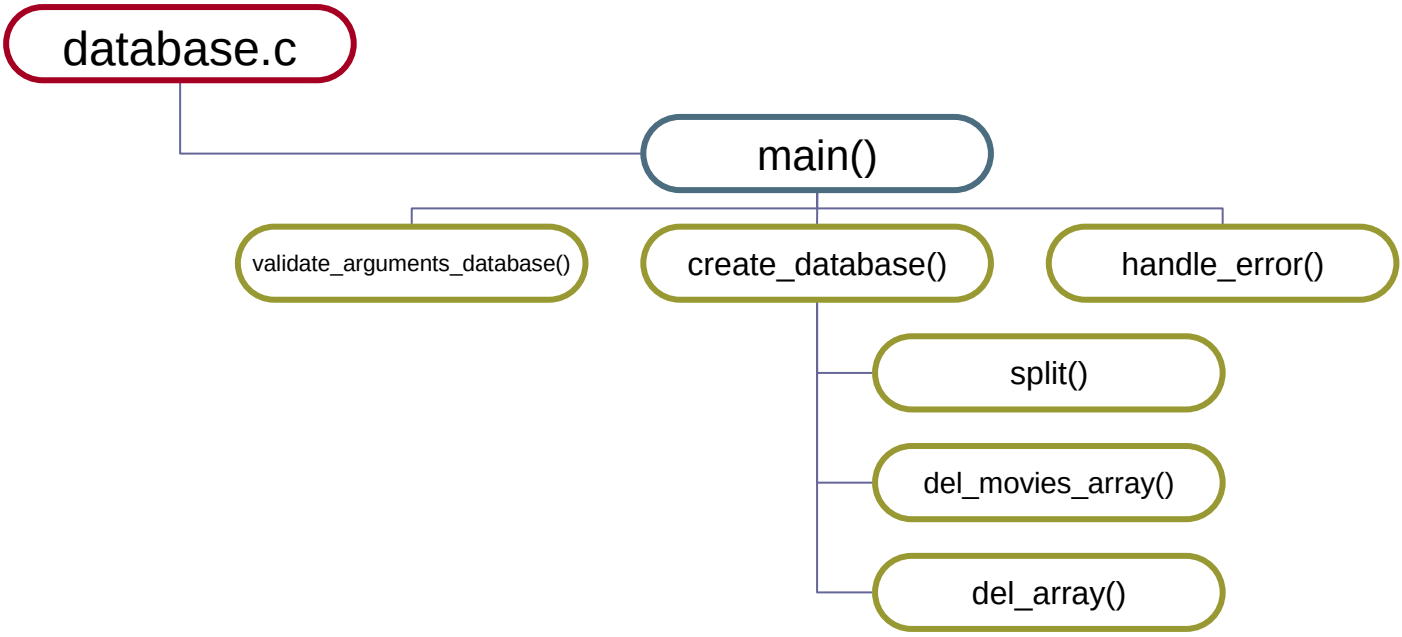
Campo	1	2	3	4	5	6	7
Contenido	id	titulo	guion	director	fecha	puntaje	reviews
tipo de dato	size_t	char[200]	char[200]	char[200]	time_t	double	size_t

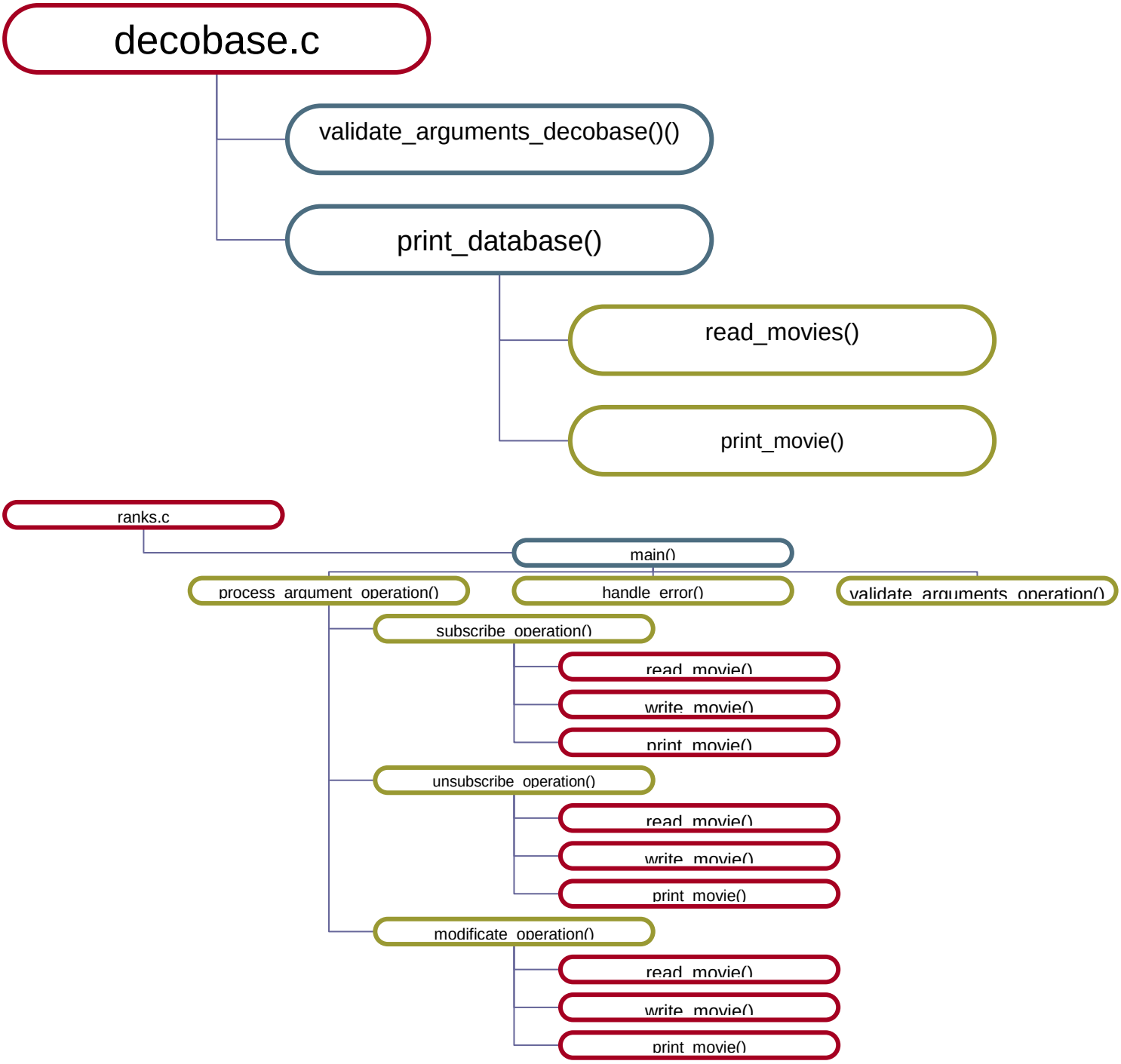
Tabla 1: estructura del archivo CSV a utilizar

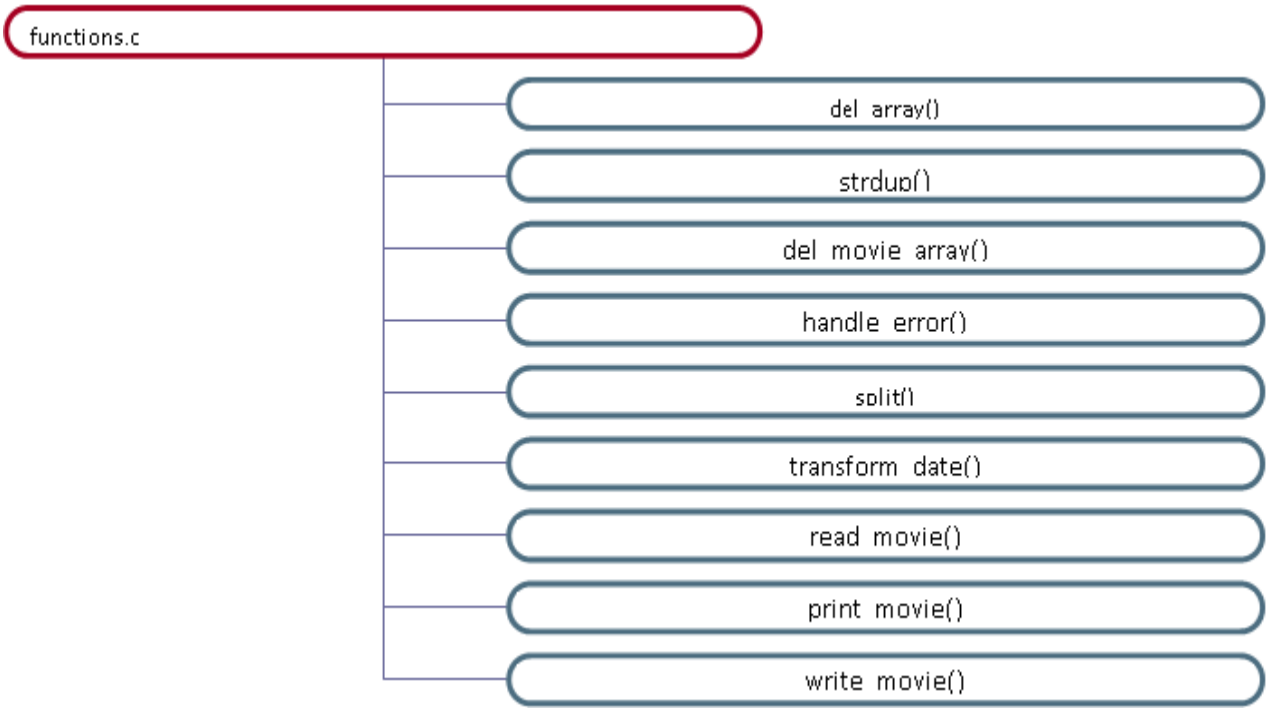
El tipo de dato que contiene a los elementos de la tabla 1, nombrado `movies_t`, se definió en un archivo de biblioteca llamado `movie.h`. Dentro de éste, se utiliza la palabra reservada `typedef` para crear un tipo de dato y asignarle un sinónimo a través de datos anteriormente definidos. Es así como el compilador puede designar qué tipo de dato será guardado en este nuevo tipo de dato. Dentro del mismo se define el tipo `status_t` donde se asignan los distintos tipos de estados que puede tener el programa.

Para lograr el desarrollo independiente e íntegro de cada programa por separado, se definió un tipo de archivo `.c` conteniendo en cada caso una función `main`. Esta función única dentro de cada programa, se encarga de invocar a las demás funciones y comparar los estados que estas devuelven.

A continuación la estructura funcional del programa:




decobase.c
validate_arguments_decobase()()
print_database()
read_movies()
print_movie()
ranks.c
main()
process_argument_operation()
handle_error()
validate_arguments_operation()
subscribe_operation()
read_movie()
write_movie()
print_movie()
unsubscribe_operation()
read_movie()
write_movie()
print_movie()
modificate_operation()
read_movie()
write_movie()
print_movie()



En principio, cada programa contiene una función encargada de la validación de datos de modo general y otra función similar pero más específica encargada del procesamiento de datos. Todas estas devuelven algún caso de estado, que es chequeado en `main`. En el caso de devolver un estado erróneo, `main` se encarga de invocar una función denominada `handle_error`.

La función `handle_error` contiene una construcción del lenguaje C para decisiones múltiples, denominada `switch`, donde para cada `case` (casos con escenarios esperados e inesperados) de la misma se contemplan todos los posibles errores del programa para los cuales se debe realizar alguna acción. Luego de detectar qué tipo de problema ha ocurrido, se imprime por pantalla un mensaje que informa al usuario el motivo por el cual el programa no ha sido ejecutado con éxito. Estos mensajes están definidos en un archivo denominado `messages_error.h` a través de la definición de constantes simbólicas definidas con la orden al preprocesador `#define`.

Se tuvo como consideración el uso de constantes simbólicas para evitar generar un código *hardcodeado*. El término recién mencionado, “hace referencia a una mala práctica en el desarrollo de software que consiste en incrustar datos directamente en el código fuente del programa, en lugar de obtener esos datos de una fuente externa como un fichero de configuración o parámetros de la línea de comandos, o un archivo de recursos”.[(2)]Siempre que fue necesario, se definieron dichas constantes simbólicas en archivos de extensión `.h` agrupadas según un criterio de utilidad o uniformidad de los mismos. Dichas constantes simbólicas, son utilizadas por los archivos de extensión `.c`, por lo que los archivos `.h` deben ser incluidos en los archivos .c. Además, para las mismas, se utilizó una protección para evitar las inclusiones múltiples que resultan en un aumento de tiempo de compilación y un error de programación notorio.

Para el desarrollo, se agruparon las funciones según el programa que las invoque. Es decir, los programas que tengan funciones que son de uso exclusivo se incluyen en el archivo .c que contenga las invocaciones de dichas funciones. A su vez, cada prototipo de función fue explicitado en el archivo `.h` referido a cada programa específico junto con lo necesario para el correcto funcionamiento de las mismas.

Los prototipos de funciones que son de uso común a más de un programa, se las incluyen en un archivo de extensión .h nombrado `common.h`. Para el caso de funciones no específicas del programa, pero con utilidades generales se definieron sus prototipos en un archivo `.h` aparte, llamado `utilities.h`. En dicho archivo se encuentran funciones que manejan estado de error o conversión de tipo de datos. Si bien responden a funciones distintas, ambas fueron implementadas en un mismo archivo de código .c denominado `functions.c`.

Una de las consideraciones del trabajo práctico fue el uso de la biblioteca `time.h`, el trato con las funciones y el tipo de dato `time_t` definido en ésta. La causa de uso de la misma en el trabajo proviene del programa que se encarga de crear la base de datos binario a través del archivo CSV. Cuando se obtiene el dato, éste está conformado por una cadena de caracteres que contiene la fecha, delimitada por el carácter '/' (por ejemplo, 06/05/1953). Esta cadena de caracteres no es un tipo de dato `time_t`, por lo que imposibilita utilizar las funciones de esta biblioteca (al menos en forma directa). Por lo tanto, se tuvo que definir una función llamada `transform_date` que convierte dicha cadena de caracteres en un dato de tipo `struct_tm`. Esta función recibe la cadena de caracteres que contiene como dato la fecha, la duplica y luego la "corta" según el delimitador que esté definido, para insertarlo según sea requerido en la estructura `struct_tm` definida en `time.h`. Una vez adquirida esta estructura, se utiliza la función predefinida `mktime`, que convierte `struct_tm` a un tipo `time_t` como resultado esperado.

Para lograr la impresión del `time_t` en el programa, también fueron requeridas las funciones de esta biblioteca. Para lograr el formato deseado, se utilizó la función `localtime`, que convierte un `time_t` en un `struct_tm`, con el objetivo de utilizar la función `strtime`, que imprime en una cadena de caracteres la estructura. Esta función permite elegir el formato en que se imprime en la cadena de caracteres la estructura `struct_tm`.

3.2.1 Creación de la base de datos

Como fue mencionado anteriormente, antes de pensar en realizar una modificación a un archivo binario primero hay que crearlo. Para dicho propósito, se crea un programa que se debe invocar por línea de comandos del siguiente modo

```
./crear_base entrada salida
```

donde:

`entrada` es el archivo CSV que se debe binarizar.

`salida` es el archivo binario que contiene todos los datos del CSV de entrada, pero en estructuras.

La estructura de este programa está definida de manera principal/mayoritaria en el archivo `database.c` (usando, también, funciones presentes en functions.c). En dicho archivo se encuentra el orden de invocación de las funciones que logran crear una base de datos.

En primera instancia la función `validate_arguments_database` recibe, por medio de un puntero, los archivos que fueron ingresados por el usuario. Su función básica es la de contabilizar de forma correcta la cantidad de argumentos y lograr la apertura de los archivos. Éstos serán abiertos por medio de la función `fopen` en modo lectura para el archivo de entrada y en modo escritura binaria para el archivo de salida. En caso de que ocurra algún error, el mismo será devuelto por la función. En caso de no presentarse errores, la función devolverá un estado exitoso.

Una vez abiertos los archivos de manera exitosa, el archivo procede a invocar la función `create_database`, que recibe como argumentos estos archivos. Su función es leer las líneas que integran el archivo CSV e implementar de manera iterativa la escritura de estos datos en el archivo binario. Esto, en el programa, se realiza por medio de la función

`while(fgets(line,MAX_STR,fin)!=NULL)` (1)

donde `fin` es el archivo CSV de donde se leen las líneas que se almacenan en la variable `line`. El ciclo termina cuando el archivo se lee completamente y se llega a `EOF`(end of file).

Siguiendo el proceso del programa, una vez que se obtiene una de las líneas del archivo, la variable `line` almacena algo como muestra el siguiente ejemplo

`1,Jaws,Steven Spielberg,Peter Benchley,20/06/1975,56,951`

Se puede apreciar que los campos están aún todos unidos, por lo que es necesario crear una función que logre separar los campos. Dicha función, empleada en el trabajo, es la función `split`. Una vez obtenida dicha línea, ésta ingresa a la función `split`, que se encarga de separar los campos del archivo CSV por su carácter delimitador ‘,’. Una vez dentro de esta función se duplica cada línea, luego se separan los campos y se los ingresa de manera iterativa, de forma distinta según el tipo de dato, en la estructura del tipo `movies_t` que será lo que se escribe el archivo final. Cuando no se pueda leer más líneas, será porque ya se leyeron todas las líneas del CSV (suponiendo el correcto funcionamiento del programa) y el programa finaliza, dando como resultado la creación de un archivo binario con el contenido del CSV.

3.2.2 Altas/Bajas/Modificaciones

En el caso del programa de altas, bajas y modificaciones, se busca realizar alguna clase de modificación al archivo fuente invocado por el usuario. Un caso particular de este programa es que, al encontrar un error, no lo informa por pantalla, sino que está especificado en la línea de comandos un archivo en el cual se escribirán los errores producidos durante la operación designada, pero ninguno de éstos deberá significar una finalización del programa.

El usuario deberá invocar el programa de la siguiente manera:

`./ranking operacion -if db -f data -log log`

donde:

`operacion` indica si se quiere realizar una alta, baja o modificación. La misma puede ser [A]LTA, [B]AJA o [M]ODIFICACION.

`-if` indica el archivo que contiene la base de datos, cuyo nombre es `db`, es

decir, debe recibir el nombre por CLA.

`-f` indica el archivo que contiene los registros a modificar.

`-log` indica un archivo donde almacenar los mensajes de error.

Este programa está definido de manera explícita en el archivo `ranks.c` (usando, también, funciones presentes en functions.c).

En un caso análogo con el programa anterior, en primera instancia se invoca una función que valide los argumentos ingresados para chequear si cumplen con las especificaciones. Esta función, llamada `validate_arguments_operations`, se asegura que la cantidad de argumentos ingresados sea correcta, como también, que se respete el prefijo que identifica a cada archivo y, por último, se realiza a través de `fopen` la apertura de los archivos seleccionados.

Se procede a invocar la función `process_arguments_operations`, la cual utiliza el primer carácter de la operación ingresada por el usuario para introducirse a la operación deseada. Cada operación está identificada como un caso en `switch`. A su vez, estas operaciones devuelven un estado que será devuelto por la función general en caso de error.

Para el caso de las operaciones, si bien éstas cumplen distintas funciones, el algoritmo utilizado para desarrollar estas tareas es muy similar. Se crea un archivo de manera auxiliar, donde se escribe momentáneamente el resultado de la operación, para que luego éste reemplace de manera final a la base de datos al cierre de la operación con la función `rename`. Luego, se debe leer una línea perteneciente a cada archivo y realizar una comparación entre las mismas a razón de la operación ejecutada. El campo que se utilizó como identificador de cada línea fue el `id`. Según la operación, se procede de distinta manera:

-En la función `unsubscribe_operation`, al encontrar la igualdad en el caso de bajas, se evita escribir la línea en el archivo auxiliar, de modo que la línea queda eliminada de la futura base de datos. En caso de no existir igualdad, informará un error.

-En la función `subscribe_operation`, al no encontrar la igualdad en el caso de altas, se escribe la línea en el archivo auxiliar, de modo que la línea queda agregada en la futura base de datos. En caso de igualdad, informa un error.

-En la función `modification_operation`, al encontrar la igualdad, se reemplaza la línea de la base de datos con la leída en la base de modificaciones y se escribe la línea en el archivo auxiliar, de modo que la línea queda reemplazada en la futura base de datos. En caso de no existir igualdad, informa un error.

La implementación de escritura utilizada en la función `database` (que genera el archivo binario) genera que la lectura o bien la impresión de los datos que ésta contiene deba ser mediante funciones específicas para cada caso. Esto se debe a que, para el caso de los datos de tipo char *, dentro de la estructura se analiza primero el número de caracteres que cada cadena posee. Esto simplifica la escritura, lectura e impresión de los mismos. Según corresponda, se utilizan las funciones `fread`, `fwrite` y `fprintf`, ya que éstas manejan de manera integral los archivos fuentes, los cuales son utilizados en esta implementación.

En todos los casos se considera la situación de que la base de modificaciones o bien la base de datos se acabe. Cuando esto suceda, ya no habrá valores a comparar. Sin embargo, se debe contemplar el hecho de que el programa pueda seguir escribiendo líneas en el archivo auxiliar. De no ser así, la base de datos sería recortada o bien no se llevarían a cabo todas las altas o no se informarían las bajas o modificaciones erróneas. Estas fallas quedaran escritas en un archivo creado para tal fin denominado log.

3.2.3 Desbinarización de la base de datos

Este programa es el encargado de imprimir un archivo binario por pantalla. El mismo se invoca de la siguiente manera:

```
./decobase entrada
```

donde:

`entrada` es un archivo binario compuesto por estructuras.

Este programa está definido de manera explícita en el archivo `decobase.c`.(usando, también,funciones presentes en functions.c).

Como primera instancia, la función `validate_arguments_decobase`, se realiza una validación de cantidad de argumentos por línea de comando y se realiza, con `fopen`, la inclusión del archivo binario en modo lectura binaria con "*rb*". Luego de esto, se prosigue con la operación `print_database()` el cual recibe el archivo binario con argumento. Aquí, se lee el último elemento del archivo, que contiene la información de la cantidad de películas que contiene el archivo en cuestión. Dicha información se almacena en una variable, llamada `amount`. Posterior, se ejecuta una función que se encarga de leer la información de una película (en binario) y cargar dicha información a una estructura del tipo `movies_t`. Esta función es implementada en el programa de la siguiente manera:

```
read_movie (FILE * stream, movies_t * movie)
```

donde `stream` es el archivo binario recibido, y `movie` es la estructura del tipo `movies_t` donde se carga la información leída del archivo binario. Esta función se ejecuta tantas veces como películas haya en el archivo binario. Una vez realizado esto, se imprime por pantalla la información cargada en `movie`, por medio de la función

```
print_movie (const movies_t movie)
```

que recibe la estructura cargada. Esta función se ejecuta, también, tantas veces como películas haya en el archivo binario.

4. Resultados de ejecución y reseñas sobre problemas encontrados en el programa

Durante el desarrollo del código, se fueron presentando grandes números de fallas y errores al momento de compilar.

El manejo del tipo de dato `time_t`, incluida en la biblioteca `time.h` generó inconvenientes a la hora del desarrollo. En el caso de la impresión de este tipo de dato, la función `strftime()` permite elegir el formato para la impresión de la fecha. Para respetar el formato que especifica el enunciado, se debería imprimir de manera (aaaa-mm-dd), lo cual es implementado en el código como "%Y-%m-%d". Sin embargo, se presentan problemas al momento de compilar, ya que (por alguna razón que no se pudo determinar), este formato es marcado como erróneo. Por esta razón, se utiliza el formato "%x" que imprime la fecha de manera (mm/dd/aa).

También, existe un problema en la validación de la función `strtol()` en el código correspondiente a `database.c`. Se analizó por qué sucede la falla y la razón resultante es que el puntero `endptr` apunta a `NULL` luego de la conversión. Sin embargo, el condicional impone que éste apunte a un '\0' al finalizar la conversión de la cadena. Esto es debido a que la cadena que recibe para convertir sólo consta de números.

Para evitar esta falla, la validación fue comentada y el número obtenido para la variable `reviews` ha sido el deseado, aunque es necesario mencionar que no es una solución.

```
/*Asigno Reviews*/
movies[used_size].reviews=strtoul(csv_fields[REVIEWS_FIELD_POS],&endptr,10);
/*if(*endptr!='\0'||*endptr==' '){
    printf("Entro el el strtol if reviews\n");
    printf("endptr:%s\n",endptr);
    del_movies_array(&movies,&used_size);
    del_array(&csv_fields,&n);
    return ST_ERROR_NULL_POINTER;
}*/
if(fwrite(&movies[used_size].reviews,sizeof(size_t),1,fout)!=1){
    return ST_ERROR_INVALID_DATA;
}
```

Figura 4: Validación de la función `strtoul()` comentada

5. Conclusiones

El manejo de estructuras, como también el de archivos, es de suma importancia en el ámbito de la programación, por la cantidad de beneficios que su buena implementación conlleva. Lo mismo se puede decir de la implementación de memoria dinámica, que permite la optimización de los recursos del ordenador. Sin embargo, no tener un dominio adecuado de estas herramientas implica la aparición de errores como los mencionados anteriormente.

6. Script de compilación

```
database
gcc -ansi -Wall -pedantic functions.c database.c -o database

decobase
gcc -ansi -Wall -pedantic functions.c decobase.c -o decobase

ranks
gcc -ansi -Wall -pedantic functions.c ranks.c -o ranks
```

7. Bibliografía

(1) Función `fgets`, Tutorial Points, https://www.tutorialspoint.com/c_standard_library/c_function_fgets.htm

(2) Descripción del término "harcodeado", Wikipedia,https://es.wikipedia.org/wiki/Hard_code