



FACULTAD DE INGENIERIA

Universidad de Buenos Aires

Año 2018 – 1.º Cuatrimestre

Algoritmos y programación I (95.11)

Curso: Ing. Cardozo

Trabajo práctico n.º 2

Tema: Indexación automática de archivos MP3

Integrantes:

Apellido y nombres	Email	Padrón
Báez, Facundo	baezfacundo1994@gmail.com	97733
Torales, Cristian Valentín	crist.torales@gmail.com	95549

FECHA: 04/07/2018

Enunciado

1) Objetivo

El objetivo de este T.P. consiste en desarrollar un aplicativo de consola con comandos en línea de órdenes y escrito en lenguaje ANSI C, que permita construir un índice de los archivos MP3 indicados como parámetro, ordenado de acuerdo con un criterio particular y presentado en formato texto (CSV, HTML, XML).

2) Alcance

Mediante el presente T.P. se busca que el alumno adquiera y aplique conocimientos sobre los siguientes temas:

- programas en modo consola;
- argumentos en línea de órdenes (CLA);
- modularización;
- makefile;
- archivos de texto y binarios;
- memoria dinámica;
- punteros a función;
- tipo de dato abstracto, T.D.A. Vector. T.D.A. ad hoc;
- estructura del encabezado de un archivo MP3 (ID3v1);
- métodos de ordenamiento;
- estructura básica de un archivo CSV;
- estructura básica de un documento XML;
- estructura básica de un documento HTML (optativo).

3) Desarrollo

En este T.P. se pide escribir un programa ejecutable que genere documentación sobre pistas de audio en formato MP3.

El programa ejecutable, denominado "mp3explorer.exe" (WinXX) o "mp3explorer" (Unix), debe ser invocado de la siguiente forma:

WinXX:

mp3explorer -fmt <formato> -sort <criterio> -out <salida> <arch 1>...<arch N>

UNIX:

./mp3explorer -fmt <formato> -sort <criterio> -out <salida> <arch 1>...<arch N>

Los comandos en línea de órdenes utilizados por la aplicación son los indicados a continuación.

Formato del índice a generar

Comando	Descripción	Valor	Tipo de dato	Observaciones
-fmt	Formato del índice a generar	"csv"	Cadena de caracteres	Obligatorio
		"xml"	Cadena de caracteres	Obligatorio
		"html"	Cadena de caracteres	Optativo

Criterio de ordenamiento para los temas

Comando	Descripción	Valor	Tipo de dato	Observaciones
-sort	Criterio de ordenamiento	"name"	Cadena de caracteres	Nombre del tema (obra).
		"artist"	Cadena de caracteres	Autor de la obra.
		"genre"	Cadena de caracteres	Género de la obra.

Si se ordena, por ejemplo, por género, no se pide aplicar un nuevo criterio para todos los elementos con el mismo género.

Archivo de salida <salida>

Es la ruta del archivo índice que se desea construir.

Archivos de entrada <arch j>

Se trata del archivo MP3 j-ésimo de entrada.

Nota: Se puede asumir que los comandos en línea de órdenes estarán en cualquier orden (en pares), si esta estrategia simplifica el desarrollo de la presente aplicación, pero se debe consignar la decisión en el informe.

Operación

El programa debe analizar los archivos MP3 suministrados al programa y generar una tabla en memoria con los atributos de cada tema, para su posterior ordenamiento e impresión del índice en el formato correspondiente.

Formato de entrada

Se debe extraer la información de cada tema a partir del frame header del archivo MP3 mediante la lectura de los últimos 128 bytes del archivo. Se sugiere utilizar la función de biblioteca fseek(). Se debe trabajar con archivos ID3v1.

Formato de salida

a) Formato CSV:

El formato del documento CSV de salida a generar es:

<nombre de tema> <artista> <género>
...
<nombre de tema> <artista> <género>

b) Formato XML:

El formato del documento XML de salida a generar es:

```
<?xml version="1.0" ?>
<tracks>
  <track>
    <name>...</name>
    <artist>...</artist>
    <genre>...</genre>
  </track>
  ...
</tracks>
```

c) Formato HTML (optativo):

Formato HTML tabular libre, a elección del desarrollador.

4) Restricciones

La realización de este programa está sujeta a las siguientes restricciones:

- Se debe recurrir al uso del T.D.A. Vector para almacenar los elementos de información correspondientes a una pista de audio.
- Se debe recurrir al uso de punteros a función a fin de parametrizar la impresión del índice.
- Se deben utilizar funciones y una adecuada modularización.
- Se debe construir un proyecto mediante la utilización de makefile.
- Hay otras cuestiones que no han sido especificadas intencionalmente en este requerimiento, para darle al desarrollador la libertad de elegir implementaciones que, según su criterio, resulten más favorables en determinadas situaciones. Por lo tanto, se debe explicitar cada una de las decisiones adoptadas y el o los fundamentos considerados para ellas.

5) Entrega del Trabajo Práctico

Se debe presentar la correspondiente documentación de desarrollo. Ella se debe entregar en forma impresa y encarpeta, de acuerdo con la numeración siguiente:

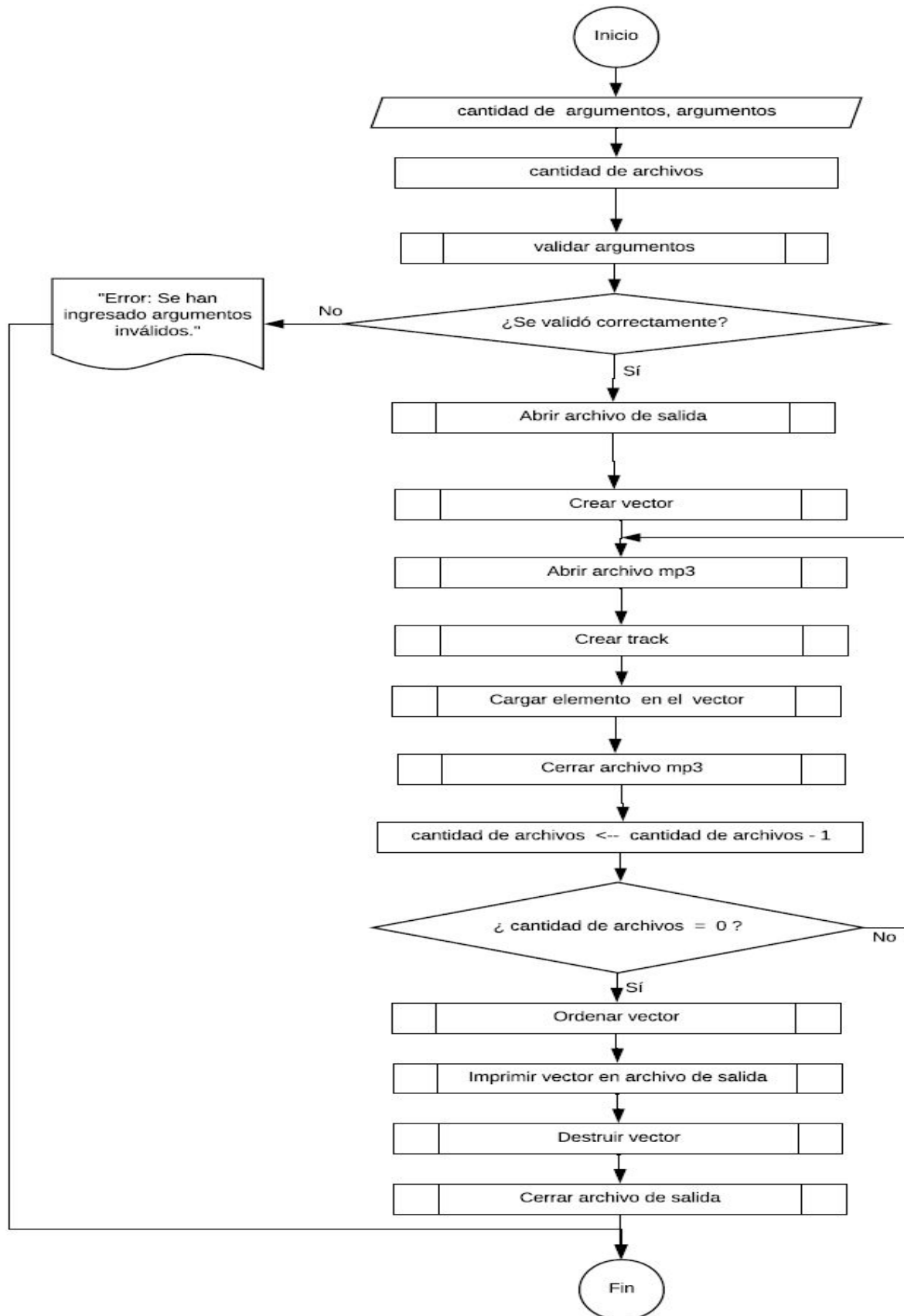
- 1) Carátula. Incluir una dirección de correo electrónico.
- 2) Enunciado (el presente documento).
- 3) Diagrama de flujo básico y simplificado del programa (1 carilla A4).
- 4) Estructura funcional del programa desarrollado (diagrama de funciones, 1 carilla A4).
- 5) Explicación de las alternativas consideradas y las estrategias adoptadas.
- 6) Resultados de la ejecución (corridos) del programa, en condiciones normales e inesperadas de entrada.
- 7) Reseña de los problemas encontrados en el desarrollo del programa y las soluciones implementadas para subsanarlos.
- 8) Conclusiones.
- 9) Script de compilación.
- 10) Código fuente.

6) Bibliografía

Se debe incluir la referencia a toda bibliografía consultada para la realización del trabajo: libros, artículos, etc.

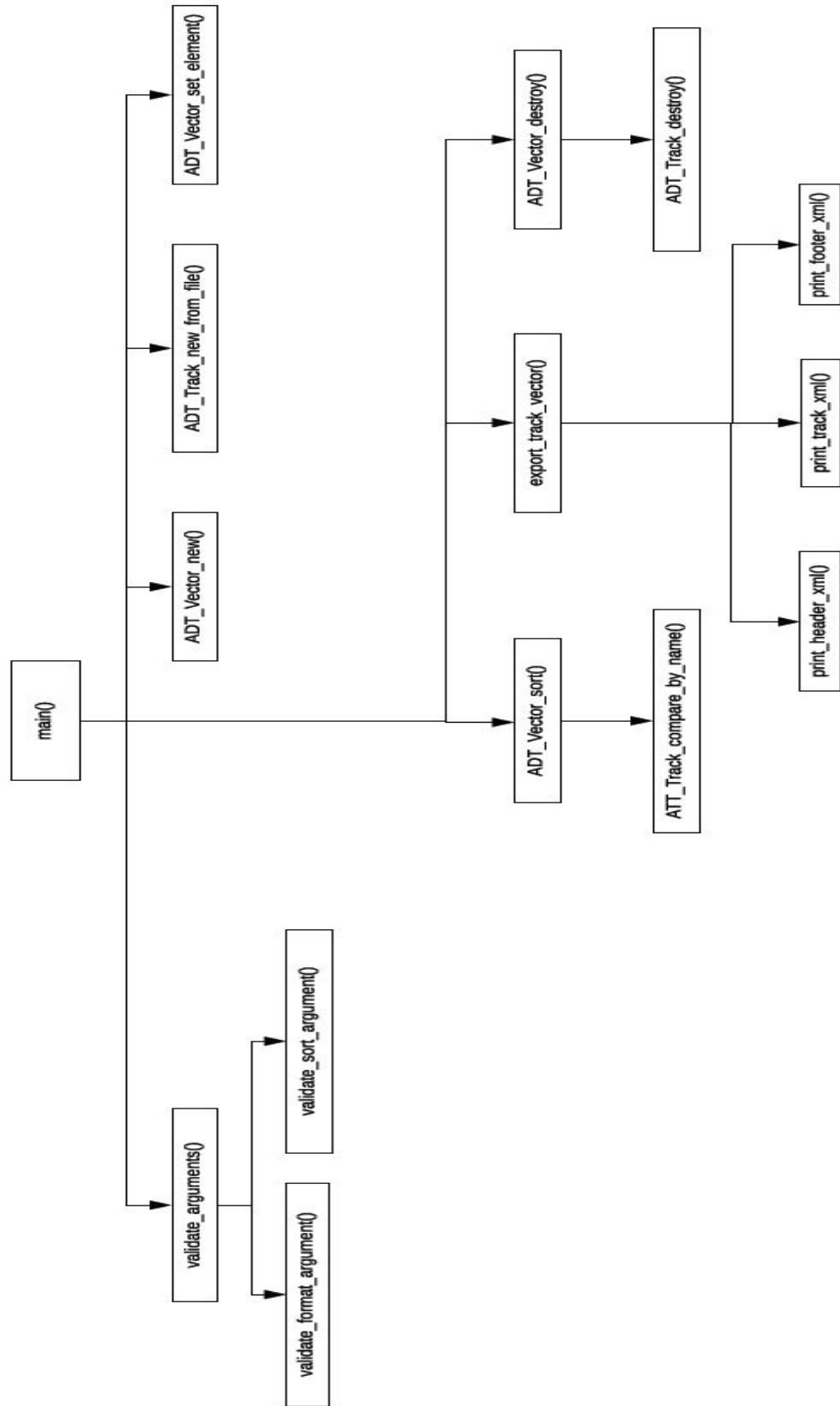
7) Última fecha de entrega: sábado 30/06/2018

Diagrama de flujo



Estructura funcional del programa desarrollado

Estructura funcional del programa desarrollado (Diagrama de funciones)



Explicación de las alternativas consideradas y las estrategias adoptadas

Para el desarrollo del trabajo se ha adoptado la estrategia “Top-Down”, esto es porque primero se realizó el planteo a nivel general, es decir, lo que pertenece a la función principal. Luego, se fueron desarrollando los casos particulares. Esto ayudó a entender mejor la aplicación, y al momento de desarrollar el código, ver las estrategias más convenientes para los casos particulares. Además permitió discriminar a la entrada, al proceso y a la salida del programa, ya que al realizar un planteo general, resulta mucho más sencillo de ver, comparado con codificar módulos específicos, y luego tratar de entender qué rol cumplen cada uno de ellos en el esquema del software.

Al utilizar el programa, el usuario podrá ingresar una cantidad indefinida de archivos mp3, se tomó la decisión de abrir un archivo a la vez, procesarlo y cargar los datos en el vector, por último cerrarlo y luego hacer lo mismo con el siguiente. El proceso se realiza en un ciclo que necesita de la cantidad de archivos que se van a procesar. Ésta cantidad se obtiene a partir de la resta entre la cantidad total de argumentos ingresados y la cantidad de argumentos fijos (cantidad de argumentos que no pertenecen a nombres de los archivos mp3).

Cuando se decidió como estaría definido ADT_Track se evaluó la posibilidad de que el campo género sea una variable del tipo unsigned char, pero finalmente se descartó y se optó por utilizar un tipo enumerativo, debido a que esta variable tenía un significado muy concreto.

En los casos de ordenamiento por el tipo de dato del Track y las indicaciones de como imprimir, se utilizaron definiciones de tipos enumerativos. Esto permite que en el futuro se puedan agregar más opciones de ordenamiento y el modos de impresión. Por lo tanto, esta estrategia favorece al mantenimiento evolutivo.

Con respecto al procedimiento de carga del ADT_Vector, para hacer la configuración de cada elemento se tomó la decisión de retornar un estado de error en el caso de que la posición a cargar esté ocupada, ya que si se permitía cargar un elemento en una posición ocupada por otro se perdería la información contenida en este último.

Para el ordenamiento de los datos de los archivos mp3, las alternativas fueron: “bubble sort”, “selection sort” e “insertion sort”.

No se utilizó el método “bubble sort” porque realiza las comparaciones con elementos adyacentes, a diferencia del “selection sort” que compara con un mínimo.

Se utilizó el método de ordenamiento de “selection sort”, el cual consiste en tener un pivote con el que se realizan las comparaciones y se ordenan, las ventajas son la facilidad de implementación y que el rendimiento es de orden cuadrado. Si la cantidad de registros a ordenar es muy grande, como realiza muchas comparaciones no sería un algoritmo eficiente. Al realizar la comparación con los otros dos métodos considerados, éste es el que mejor se comporta en el caso de tener los datos muy desordenados.

Resultados de la ejecución (corridas) del programa, en condiciones normales e inesperadas de entrada.

En primer lugar se procedió a ejecutar al programa con argumentos en línea de orden incorrectos, para poder visualizar su comportamiento.

```
facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$ ./mp3explorer -fmt xml
-sort name -in in.xml Sonne.mp3 Libertango.mp3 Jamming.mp3 Thriller.mp3 Jeremy.m
p3
Los argumentos en línea de orden son inválidos
facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$
```

Como es debido el programa informa al usuario que los argumentos en línea de orden son inválidos.

Luego ingresando CLA aparentemente correctos, se intentó correr el programa:

```
facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$ ./mp3explorer -fmt csv
-sort artist -out out.csv Sonne.mp3 Libertango.mp3 Jamming.mp3 Thriller.mp3 Jere
my.mp3 Träumst du?.mp3
Archivo .mp3 inválido
facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$
```

Pero surgió un inconveniente debido a el nombre del archivo “Träumst du?.mp3” con tiene un espacio, lo que conlleva que el programa lo lea como dos argumentos distintos (“Träumst” y “du?.mp3”) y que éste lo informe como un error.

Editando el nombre archivo se volvió a intentar ejecutar el programa:

```
facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$ ./mp3explorer -fmt csv
-sort artist -out out.csv Sonne.mp3 Libertango.mp3 Jamming.mp3 Thriller.mp3 Jere
my.mp3 Träumst_du?.mp3
facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$
```

Se obtuvo el el resultado esperado, un archivo csv, con los *headers* ordenados por artista.

	A
1	Jamming Bob Marley 1977 Reggae
2	Thriller Michael Jackson 1982 Pop
3	Träumst du? Oomph! 2009 Metal
4	Jeremy Pearl Jam 1991 Grunge
5	Sonne Rammstein 2004 Metal
6	Libertango Astor Piazzolla 1974 Tango
7	

Después se intentó correr el programa de forma tal que dé como resultado, un archivo xml que contenga los *headers* ordenados por el nombre de las canciones:

```
facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$ ./mp3explorer -fmt xml
-sort name -out out.xml Sonne.mp3 Libertango.mp3 Jamming.mp3 Thriller.mp3 Jeremy
.mp3 Träumst_du?.mp3
facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$
```

Pero se manifestó otro problema, el nombre de uno de los artistas contenía un acento, y el título de una canción una diéresis por lo que el archivo xml marcaba los siguientes errores:

Error de análisis XML: malformado

Ubicación: file:///home/facundo/Escritorio/Algo/TPN%C2%B02/Programa/out.xml
Línea 17, columna 12:

-----<Artist> Träumst du? ^

Error de análisis XML: malformado

Ubicación: file:///home/facundo/Escritorio/Algo/TPN%C2%B02/Programa/out.xml
Línea 34, columna 12:

-----<Name> Tr ^

Modificando los headers de los archivos .mp3 para que no aparezcan estos errores, se volvió a ejecutar el programa con los mismos CLA.

Caso de impresión en formato XML:

En la siguiente figura se verifica el ejemplo del caso correspondiente a la impresión de los datos de los archivos mp3 en XML:

```

-<Tracks>
  -<Track>
    <Name> Jamming </Name>
    <Artist> Bob Marley </Artist>
    <Year> 1977 </Year>
    <Genre> Reggae </Genre>
  </Track>
  -<Track>
    <Name> Jeremy </Name>
    <Artist> Pearl Jam </Artist>
    <Year> 1991 </Year>
    <Genre> Grunge </Genre>
  </Track>
  -<Track>
    <Name> Libertango </Name>
    <Artist> Astor Piazzolla </Artist>
    <Year> 1974 </Year>
    <Genre> Tango </Genre>
  </Track>
  -<Track>
    <Name> Sonne </Name>
    <Artist> Rammstein </Artist>
    <Year> 2004 </Year>
    <Genre> Metal </Genre>
  </Track>
  -<Track>
    <Name> Thriller </Name>
    <Artist> Michael Jackson </Artist>
    <Year> 1982 </Year>
    <Genre> Pop </Genre>
  </Track>
  -<Track>
    <Name> Traeumst du? </Name>
    <Artist> Oomph! </Artist>
    <Year> 2009 </Year>
    <Genre> Metal </Genre>
  </Track>
</Tracks>

```

Ejecución del programa en condiciones normales:

En la siguiente figura se presenta un ejemplo de ejecución del programa con su correcto funcionamiento, en el cual se ingresan las opciones de formato a imprimir en csv, comparaciones para ordenar por género, el archivo de salida y los archivos mp3 utilizados para las pruebas:

```

facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$ ./mp3explorer -fmt csv
-sort genre -out out.csv Sonne.mp3 Libertango.mp3 Jamming.mp3 Thriller.mp3 Jerem
y.mp3 Traeumst_du?.mp3
facundo@facundo-System:~/Escritorio/Algo/TPN°2/Programa$

```

Caso de impresión a formato CSV:

Como se realizó la impresión en formato csv, se presenta en la siguiente figura el ejemplo del archivo csv del cual se procesaron los archivos mp3:

	A	B
1	Jeremy Pearl Jam 1991 Grunge	
2	Sonne Rammstein 2004 Metal	
3	Traeumst du? Oomph! 2009 Metal	
4	Thriller Michael Jackson 1982 Pop	
5	Jamming Bob Marley 1977 Reggae	
6	Libertango Astor Piazzolla 1974 Tango	
7		

Reseña de los problemas encontrados en el desarrollo del programa y las soluciones implementadas para subsanarlos.

Surgió un inconveniente la primera vez que se quiso obtener un *header* e imprimirlo en un archivo, debido a que aparecían caracteres no imprimibles como resultado, el origen del problema era el editor de los *headers* incluido en el reproductor mp3, ya que al cambiarlo por otro el inconveniente se solucionó. Además se notó que al intentar imprimir en un archivo xml caracteres como acentos, diéresis, etc. este daba un aviso de error, esto algo que no ocurre con el formato csv, por lo tanto esto es algo que se debe informar a los usuarios del programa.

Cuando se intentaron codificar las funciones de creación y destrucción de ADT_Vector, se introdujo como parámetro a recibir un `**void`, esto es solo un puntero a un puntero a una posición de la memoria ocupada por una variable no especificada, el problema surgió porque este tipo de punteros solo se puede desreferenciar una vez (por lo tanto solo se puede desreferenciar un nivel de puntero a void) y para estas funciones se necesitaba hacerlo dos veces. Como solución a este inconveniente, se codificaron las funciones de creación y destrucción de forma tal que reciban como argumento un puntero doble a ADT_Vector.

Al confeccionar las funciones de importación apareció el inconveniente de que las mismas deben abstenerse completamente del tipo de impresión, lo que acomplexó el método de impresión, que se presumía sencillo. Para subsanar este inconveniente se utilizaron punteros a funciones para los “headers”, “tracks” y “footers”. De esta manera se permite agregar más funciones de impresión a futuro, beneficiando al mantenimiento evolutivo.

Conclusiones

Al utilizar el concepto de TDA en el desarrollo del programa, se concluye que tiene ventajas como el mantenimiento evolutivo y permite la reutilización del código.

Los punteros a función y a void se utilizan para obtener un mejor desacoplamiento de los módulos.

La ventaja de la utilización de ADT_Vector es que es reutilizable para otros programas, ya que las primitivas utilizan punteros a void y esto permite brindar un contexto al vector, sin necesidad de que éste lo contenga en su módulo.

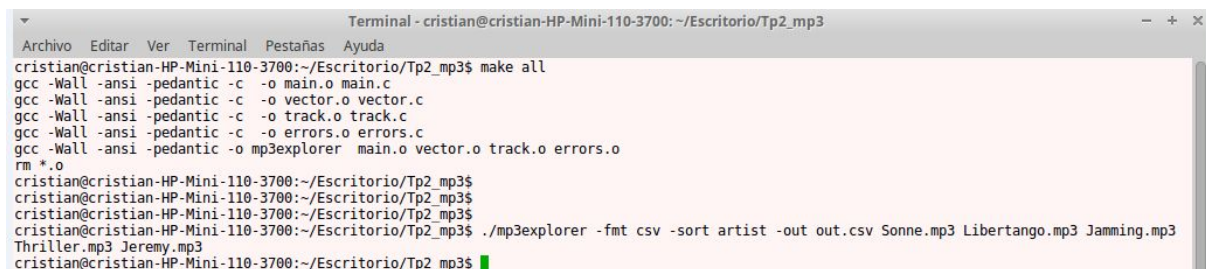
Al utilizar el T.D.A. ad hoc ADT_Track se logra empaquetar los datos de un *header* de un archivo .mp3, si bien esto se podría lograr con una estructura común, la diferencia radica en que el T.D.A. ad hoc tiene asociado a él funciones, las cuales son las únicas que pueden acceder/recibir este tipo de dato, esto permite que el programa sea mucho más verificable.

Bibliografía

B. W. Kernighan y D. M. Ritchie, *El lenguaje de programación C*, 2da edición, Prentice-Hall, México, 1991.

Script de compilación

Se utilizó la herramienta “*makefile*” para la compilación del programa, esto permite que la compilación sea parametrizable, rápida y fácil de realizarla. Para esto, se utiliza el comando “*make all*”. En la siguiente figura se puede verificar un ejemplo de su utilización:



```
Terminal - cristian@cristian-HP-Mini-110-3700: ~/Escritorio/Tp2_mp3
Archivo Editar Ver Terminal Pestañas Ayuda
cristian@cristian-HP-Mini-110-3700:~/Escritorio/Tp2_mp3$ make all
gcc -Wall -ansi -pedantic -c -o main.o main.c
gcc -Wall -ansi -pedantic -c -o vector.o vector.c
gcc -Wall -ansi -pedantic -c -o track.o track.c
gcc -Wall -ansi -pedantic -c -o errors.o errors.c
gcc -Wall -ansi -pedantic -o mp3explorer main.o vector.o track.o errors.o
rm *.o
cristian@cristian-HP-Mini-110-3700:~/Escritorio/Tp2_mp3$
cristian@cristian-HP-Mini-110-3700:~/Escritorio/Tp2_mp3$
cristian@cristian-HP-Mini-110-3700:~/Escritorio/Tp2_mp3$
cristian@cristian-HP-Mini-110-3700:~/Escritorio/Tp2_mp3$ ./mp3explorer -fmt csv -sort artist -out out.csv Sonne.mp3 Libertango.mp3 Jamming.mp3
Thriller.mp3 Jeremy.mp3
cristian@cristian-HP-Mini-110-3700:~/Escritorio/Tp2_mp3$
```

```

/***** Makefile *****/

CC=gcc
CFLAGS=-Wall -ansi -pedantic -c
LFLAGS=-Wall -ansi -pedantic

all: mp3explorer clean

mp3explorer: main.o vector.o track.o errors.o tracks_printer.o
    $(CC) $(LFLAGS) -o mp3explorer main.o vector.o track.o errors.o
    tracks_printer.o

main.o: main.c main.h setup.h mp3.h errors.h types.h vector.h track.h
    $(CC) $(CFLAGS) -o main.o main.c

vector.o: vector.c errors.h vector.h types.h
    $(CC) $(CFLAGS) -o vector.o vector.c

track.o: track.c errors.h track.h mp3.h types.h setup.h
    $(CC) $(CFLAGS) -o track.o track.c

tracks_printer.o: tracks_printer.c errors.h track.h mp3.h types.h setup.h
    vector.h tracks_printer.h
    $(CC) $(CFLAGS) -o tracks_printer.o tracks_printer.c

errors.o: errors.c errors.h
    $(CC) $(CFLAGS) -o errors.o errors.c

clean:
    rm *.o

```

Código fuente

```
/****** main.c *****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "errors.h"
#include "types.h"
#include "main.h"
#include "setup.h"
#include "mp3.h"
#include "vector.h"
#include "track.h"
#include "tracks_printer.h"

comparator_t comparers [NUMBER_OF_COMPARATORS_FUNCTIONS] =
{
    ADT_Track_compare_by_name,
    ADT_Track_compare_by_artist,
    ADT_Track_compare_by_genre,
};

int main (int argc, char * argv [])
{
    status_t st;
    FILE * file_track_list;
    FILE * file_mp3;
    destructor_t destructor;
    ADT_Vector_t * ADT_Vector;
    void * ADT_Track;
    size_t mp3_file_index;
    config_mp3_t config;

    destructor = ADT_Track_destroy;

    if ((st = validate_arguments (argc, argv, &config)) != OK)
    {
        print_error_msg (st);
        return st;
    }

    if ((file_track_list = fopen (argv [CMD_ARG_POSITION_OUTPUT_FILE], "wt")) ==
    NULL)
    {
        print_error_msg (ERROR_OUTPUT_FILE);
        return ERROR_OUTPUT_FILE;
    }

    if ((st = ADT_Vector_new (&ADT_Vector)) != OK)
    {
        print_error_msg (st);
        fclose (file_track_list);
    }
}
```

```

        return st;
    }

    for (mp3_file_index = 0; mp3_file_index < config.mp3_files_quantity;
        mp3_file_index++)
    {
        if ((file_mp3 = fopen (argv [mp3_file_index +
            CMD_ARG_POSITION_FIRST_MP3_FILE], "rb")) == NULL)
        {
            ADT_Vector_destroy (&ADT_Vector, destructor);
            print_error_msg (ERROR_INPUT_MP3_FILE);
            fclose (file_track_list);

            return ERROR_INPUT_MP3_FILE;
        }

        if ((st = ADT_Track_new_from_file (&ADT_Track, file_mp3)) != OK)
        {
            ADT_Vector_destroy (&ADT_Vector, destructor);

            print_error_msg (st);

            fclose (file_track_list);
            fclose (file_mp3);

            return st;
        }

        if ((st = ADT_Vector_set_element (&ADT_Vector, ADT_Track,
            mp3_file_index)) != OK)
        {
            ADT_Vector_destroy (&ADT_Vector, destructor);
            ADT_Track_destroy(ADT_Track);

            print_error_msg (st);

            fclose (file_track_list);
            fclose (file_mp3);

            return st;
        }

        fclose (file_mp3);
    } /* Fin del ciclo for */

    if ((st = ADT_Vector_sort (&ADT_Vector, comparers [config.track_sort_type]))
        != OK)
    {
        ADT_Vector_destroy (&ADT_Vector, destructor);

        print_error_msg (st);

        fclose (file_track_list);
    }

```

```

        return st;
    }

    if ((st = export_track_vector(ADT_Vector, config.track_list_format,
file_track_list)) != OK)
    {
        ADT_Vector_destroy (&ADT_Vector, destructor);

        print_error_msg (st);

        fclose (file_track_list);

        return st;
    }

    if ((st = ADT_Vector_destroy (&ADT_Vector, destructor)) != OK)
    {
        print_error_msg (st);

        fclose (file_track_list);

        return st;
    }

    fclose (file_track_list);

    return OK;
}

status_t validate_arguments (int argc, char * argv [], config_mp3_t * config)
{
    status_t st;
    track_list_format_t format;
    track_sort_type_t sort;

    if (argv == NULL || config == NULL)
        return ERROR_NULL_POINTER;

    if ((st = validate_format_argument (argv, &format)) != OK)
        return st;

    if ((st = validate_sort_argument (argv, &sort)) != OK)
        return st;

    config -> track_list_format = format;
    config -> track_sort_type = sort;

    if (strcmp (argv [CMD_ARG_POSITION_FLAG_OUTPUT_FILE],
CMD_ARG_FLAG_OUTPUT_FILE))
        return ERROR_PROG_INVOCATION;

    if (argc < CMD_ARG_POSITION_FIRST_MP3_FILE)

```



```

        return ERROR_PROG_INVOCATION;

    config -> mp3_files_quantity = argc - CMD_ARG_POSITION_FIRST_MP3_FILE;

    return OK;
}

status_t validate_format_argument (char * argv [], track_list_format_t *
track_list_format)
{
    if (argv == NULL || track_list_format == NULL)
        return ERROR_NULL_POINTER;

    if (strcmp (argv [CMD_ARG_POSITION_FLAG_FORMAT], CMD_ARG_FLAG_FORMAT))
        return ERROR_PROG_INVOCATION;

    if (!strcmp (argv [CMD_ARG_POSITION_FORMAT], CMD_ARG_CSV_FORMAT))
        *track_list_format = TRACK_LIST_FORMAT_CSV;

    if (!strcmp (argv [CMD_ARG_POSITION_FORMAT], CMD_ARG_XML_FORMAT))
        *track_list_format = TRACK_LIST_FORMAT_XML;

    if (*track_list_format != TRACK_LIST_FORMAT_XML && *track_list_format !=
TRACK_LIST_FORMAT_CSV)
        return ERROR_PROG_INVOCATION;

    return OK;
}

status_t validate_sort_argument (char * argv [], track_sort_type_t *
track_sort_type)
{
    if (argv == NULL || track_sort_type == NULL)
        return ERROR_NULL_POINTER;

    if (strcmp (argv [CMD_ARG_POSITION_FLAG_SORT], CMD_ARG_FLAG_SORT))
        return ERROR_PROG_INVOCATION;

    if (!strcmp (argv [CMD_ARG_POSITION_SORT], CMD_ARG_NAME_SORT))
        *track_sort_type = TRACK_SORT_BY_NAME;

    if (!strcmp (argv [CMD_ARG_POSITION_SORT], CMD_ARG_ARTIST_SORT))
        *track_sort_type = TRACK_SORT_BY_ARTIST;

    if (!strcmp (argv [CMD_ARG_POSITION_SORT], CMD_ARG_GENRE_SORT))
        *track_sort_type = TRACK_SORT_BY_GENRE;

    if (*track_sort_type != TRACK_SORT_BY_NAME && *track_sort_type !=
TRACK_SORT_BY_ARTIST
    && *track_sort_type != TRACK_SORT_BY_GENRE)
        return ERROR_PROG_INVOCATION;

    return OK;
}

```

```

/***** main.h *****/

# ifndef MAIN__H
# define MAIN__H

# include <stdio.h>
# include "errors.h"
# include "setup.h"
# include "types.h"

# define CMD_ARG_POSITION_FIRST_MP3_FILE 7
# define CMD_ARG_POSITION_FLAG_FORMAT 1
# define CMD_ARG_POSITION_FORMAT 2
# define CMD_ARG_POSITION_FLAG_SORT 3
# define CMD_ARG_POSITION_SORT 4
# define CMD_ARG_POSITION_FLAG_OUTPUT_FILE 5
# define CMD_ARG_POSITION_OUTPUT_FILE 6
# define CMD_ARG_FLAG_FORMAT "-fmt"
# define CMD_ARG_CSV_FORMAT "csv"
# define CMD_ARG_XML_FORMAT "xml"
# define CMD_ARG_FLAG_SORT "-sort"
# define CMD_ARG_NAME_SORT "name"
# define CMD_ARG_ARTIST_SORT "artist"
# define CMD_ARG_GENRE_SORT "genre"
# define CMD_ARG_FLAG_OUTPUT_FILE "-out"

status_t validate_arguments (int argc, char * argv [], config_mp3_t * config);
status_t validate_format_argument (char * argv [], track_list_format_t *
track_list_format);
status_t validate_sort_argument (char * argv [], track_sort_type_t *
track_sort_type);

# endif

/***** types.h *****/
# ifndef TYPES__H
# define TYPES__H

# include <stdio.h>
# include "errors.h"
# include "vector.h"

typedef char * string;
typedef unsigned short ushort;

typedef enum
{
    TRUE,
    FALSE
} bool_t;

typedef status_t (* destructor_t) (void *);

```

```

typedef status_t (* clone_t ) (const void *, void ** );
typedef status_t (* exporter_t) (const void * pvoid, const void * pcontext, FILE *
fo);
typedef int (* comparator_t) (const void * pvoid1, const void * pvoid2);

# endif

/***** setup.h *****/
# ifndef SETUP__H
# define SETUP__H

# include <stdio.h>

# define CSV_DELIMITER '|'
# define XML_PROCESSING_INSTRUCTION "<?xml version=\"1.0\" ?>"
# define XML_MAX_TAG_LENGTH 40
# define XML_NUMBER_OF_TAG 12
# define XML_OPEN_TRACKS_TAG "<Tracks>"
# define XML_CLOSE_TRACKS_TAG "</Tracks>"
# define XML_OPEN_TRACK_TAG "<Track>"
# define XML_CLOSE_TRACK_TAG "</Track>"
# define XML_OPEN_NAME_TAG "<Name>"
# define XML_CLOSE_NAME_TAG "</Name>"
# define XML_OPEN_ARTIST_TAG "<Artist>"
# define XML_CLOSE_ARTIST_TAG "</Artist>"
# define XML_OPEN_YEAR_TAG "<Year>"
# define XML_CLOSE_YEAR_TAG "</Year>"
# define XML_OPEN_GENRE_TAG "<Genre>"
# define XML_CLOSE_GENRE_TAG "</Genre>"

# define CSV_HEADER_NAME "Name"
# define CSV_HEADER_ARTIST "Artist"
# define CSV_HEADER_YEAR "Year"
# define CSV_HEADER_GENRES "Genres"
# define CSV_CLOSE_TRACKS_TAG "End CSV file."

typedef enum
{
    TRACK_LIST_FORMAT_CSV = 0,
    TRACK_LIST_FORMAT_XML = 1
} track_list_format_t;

typedef enum
{
    TRACK_SORT_BY_NAME,
    TRACK_SORT_BY_ARTIST,
    TRACK_SORT_BY_GENRE
} track_sort_type_t;

typedef struct {
    track_list_format_t track_list_format;
    track_sort_type_t track_sort_type;
    size_t mp3_files_quantity;
}config_mp3_t;

```

```
# endif
```

```
/****** errors.c *****/
```

```
# include <stdio.h>
# include <string.h>
# include "errors.h"
```

```
static char * errors [MAX_ERRORS] =
{
    MSG_OK,
    MSG_ERROR_NULL_POINTER,
    MSG_ERROR_NO_MEMORY,
    MSG_ERROR_INPUT_MP3_FILE,
    MSG_ERROR_OUTPUT_FILE,
    MSG_ERROR_DISK_SPACE,
    MSG_ERROR_CORRUPTED_FILE,
    MSG_ERROR_PROG_INVOCATION,
    MSG_ERROR_OCUPPIED_MEMORY
};
```

```
status_t print_error_msg (status_t st)
{
    fprintf (stderr, "%s\n", errors [st]);

    return OK;
}
```

```
/****** errors.h *****/
```

```
# ifndef ERRORS__H
# define ERRORS__H
```

```
# include <stdio.h>
```

```
typedef enum
{
    OK,
    ERROR_NULL_POINTER,
    ERROR_NO_MEMORY,
    ERROR_INPUT_MP3_FILE,
    ERROR_OUTPUT_FILE,
    ERROR_DISK_SPACE,
    ERROR_CORRUPTED_FILE,
    ERROR_PROG_INVOCATION,
    ERROR_OCUPPIED_MEMORY
} status_t;
```

```
# define MAX_ERRORS 9
# define MSG_OK "OK"
# define MSG_ERROR_NULL_POINTER "Puntero nulo"
# define MSG_ERROR_NO_MEMORY "Memoria insuficiente en el sistema"
```

```
# define MSG_ERROR_INPUT_MP3_FILE "Archivo .mp3 inválido"
# define MSG_ERROR_OUTPUT_FILE "Archivo de salida inválido"
# define MSG_ERROR_DISK_SPACE "Falta de espacio en disco"
# define MSG_ERROR_CORRUPTED_FILE "Archivo corrupto"
# define MSG_ERROR_PROG_INVOCATION "Los argumentos en linea de orden son inválidos"
# define MSG_ERROR_OCUPPIED_MEMORY "Se intento sobrecribir una posición de
memoria ya ocupada"
```

```
status_t print_error_msg (status_t st);
```

```
# endif
```

```
/***** mp3.h *****/
```

```
# ifndef MP3__H
# define MP3__H
# include <stdio.h>
# include "errors.h"
# include "vector.h"

# define MP3_HEADER_SIZE      128
# define LEXEM_START_TAG      0
# define LEXEM_SPAN_TAG       3
# define LEXEM_START_TITLE    3
# define LEXEM_SPAN_TITLE     30
# define LEXEM_START_ARTIST    33
# define LEXEM_SPAN_ARTIST     30
# define LEXEM_START_ALBUM     63
# define LEXEM_SPAN_ALBUM      30
# define LEXEM_START_YEAR      93
# define LEXEM_SPAN_YEAR       4
# define LEXEM_START_COMMENT   97
# define LEXEM_SPAN_COMMENT    30
# define LEXEM_START_GENRE     127
# define LEXEM_SPAN_GENRE      1
# define NUMBER_OF_GENRES     126
# define GENRE_BLUES "Blues"
# define GENRE_CLASSIC_ROCK "Classic Rock"
# define GENRE_COUNTRY "Country"
# define GENRE_DANCE "Dance"
# define GENRE_DISCO "Disco"
# define GENRE_FUNK "Funk"
# define GENRE_GRUNGE "Grunge"
# define GENRE_HIP_HOP "Hip Hop"
# define GENRE_JAZZ "Jazz"
# define GENRE_METAL "Metal"
# define GENRE_NEW_AGE "New Age"
# define GENRE_OLDIES "oldies"
# define GENRE_OTHER "OTHER"
# define GENRE_POP "Pop"
# define GENRE_R_AND_B "R&B"
# define GENRE_RAP "Rap"
# define GENRE_REGGAE "Reggae"
# define GENRE_ROCK "Rock"
# define GENRE_TECHNO "Techno"
```

```
# define GENRE_INDUSTRIAL "Industrial"
# define GENRE_ALTERNATIVE "Alternative"
# define GENRE_SKA "Ska"
# define GENRE_DEATH_METAL "Death Metal"
# define GENRE_PRANKS "Pranks"
# define GENRE_SOUNDTRACK "Soundtrack"
# define GENRE_EURO_TECHNO "Euro Techno"
# define GENRE_AMBIENT "Ambient"
# define GENRE_TRIP_HOP "Trip Hop"
# define GENRE_VOCAL "Vocal"
# define GENRE_JAZZ_PLUS_FUNK "Jazz + Funk"
# define GENRE_FUSION "Fusion"
# define GENRE_TRANCE "Trance"
# define GENRE_CLASSICAL "Classical"
# define GENRE_INSTRUMENTAL "Instrumental"
# define GENRE_ACID "Acid"
# define GENRE_HOUSE "House"
# define GENRE_GAME "Game"
# define GENRE_SOUND_CLIP "Sound Clip"
# define GENRE_GOSPEL "Gospel"
# define GENRE_NOISE "Noise"
# define GENRE_ALTERNROCK "AlternRock"
# define GENRE_BASS "Bass"
# define GENRE_SOUL "Soul"
# define GENRE_PUNK "Punk"
# define GENRE_SPACE "Space"
# define GENRE_MEDITATIVE "Meditative"
# define GENRE_INSTRUMENTAL_POP "Instrumental Pop"
# define GENRE_INSTRUMENTAL_ROCK "Instrumental Rock"
# define GENRE_ETHNIC "Ethnic"
# define GENRE_GOTHIC "Gothic"
# define GENRE_DARKWAVE "Darkwave"
# define GENRE_TECHNO_INDUSTRIAL "Techno Industrial"
# define GENRE_ELECTRONIC "Electronic"
# define GENRE_POP_FOLK "Pop Folk"
# define GENRE_EURODANCE "Eurodance"
# define GENRE_DREAM "Dream"
# define GENRE_SOUTHERN_ROCK "Southern Rock"
# define GENRE_COMEDY "Comedy"
# define GENRE_CULT "Cult"
# define GENRE_GANGSTA "Gangsta"
# define GENRE_TOP_40 "Top 40"
# define GENRE_CHRISTIAN_RAP "Christian Rap"
# define GENRE_POP_FUNK "Pop Funk"
# define GENRE_JUNGLE "Jungle"
# define GENRE_NATIVE_AMERICAN "Native American"
# define GENRE_CABARET "Cabaret"
# define GENRE_NEW_WAVE "New Wake"
# define GENRE_PSYCHADELIC "Psychadelic"
# define GENRE_RAVE "Rave"
# define GENRE_SHOWTUNES "Showtunes"
# define GENRE_TRAILER "Trailer"
# define GENRE_Lo-Fi "Lo-Fi"
# define GENRE_TRIBAL "Tribal"
```

```
# define GENRE_ACID_PUNK "Acid Punk"
# define GENRE_ACID_JAZZ "Acid Jazz"
# define GENRE_POLKA "Polka"
# define GENRE_RETRO "Retro"
# define GENRE_MUSICAL "Musical"
# define GENRE_ROCK_AND_ROLL "Rock&Roll"
# define GENRE_HARD_ROCK "Hard Rock"
# define GENRE_FOLK "Folk"
# define GENRE_FOLK_ROCK "Folk Rock"
# define GENRE_NATIONAL_FOLK "National Folk"
# define GENRE_SWING "Swing"
# define GENRE_FAST_FUSION "Fast Fusion"
# define GENRE_BEBOB "Bebob"
# define GENRE_LATIN "Latin"
# define GENRE_REVIVAL "Revival"
# define GENRE_CELTIC "Celtic"
# define GENRE_BLUEGRASS "Bluegrass"
# define GENRE_AVANTGARDE "Avantgarde"
# define GENRE_GOTHIC_ROCK "Gothic Rock"
# define GENRE_PROGRESSIVE_ROCK "Progressive Rock"
# define GENRE_PSYCHEDELIC_ROCK "Psychedelic Rock"
# define GENRE_SYMPHONIC_ROCK "Symphonic Rock"
# define GENRE_SLOW_ROCK "Slow Rock"
# define GENRE_BIG_BAND "Big Band"
# define GENRE_CHORUS "Chorus"
# define GENRE_EASY_LISTENING "Easy Listening"
# define GENRE_ACOUSTIC "Acoustic"
# define GENRE_HUMOUR "Humour"
# define GENRE_SPEECH "Speech"
# define GENRE_CHANSON "Chanson"
# define GENRE_OPERA "Opera"
# define GENRE_CHAMBER_MUSIC "Chamber Music"
# define GENRE_SONATA "Sonata"
# define GENRE_SYMPHONY "Symphony"
# define GENRE_BOOTY_BRASS "Booty Brass"
# define GENRE_PRIMUS "Primus"
# define GENRE_PORN_GROOVE "Porn Groove"
# define GENRE_SATIRE "Satire"
# define GENRE_SLOW_JAM "Slow Jam"
# define GENRE_CLUB "Club"
# define GENRE_TANGO "Tango"
# define GENRE_SAMBA "Samba"
# define GENRE_FOLKLORE "Folklore"
# define GENRE_BALLAD "Ballad"
# define GENRE_POWEER_BALLAD "Poweer Ballad"
# define GENRE_RHYTMIC_SOUL "Rhytmic Soul"
# define GENRE_FREESTYLE "Freestyle"
# define GENRE_DUET "Duet"
# define GENRE_PUNK_ROCK "Punk Rock"
# define GENRE_DRUM_SOLO "Drum Solo"
# define GENRE_A_CAPELA "A Capela"
# define GENRE_EURO_HOUSE "Euro House"
# define GENRE_DANCE_HALL "Dance Hall"
```

```
typedef enum
{
    BLUES = 0,
    CLASSIC_ROCK = 1,
    COUNTRY = 2,
    DANCE = 3,
    DISCO = 4,
    FUNK = 5,
    GRUNGE = 6,
    HIP_HOP = 7,
    JAZZ = 8,
    METAL = 9,
    NEW_AGE = 10,
    OLDIES = 11,
    OTHER = 12,
    POP = 13,
    R_AND_B = 14,
    RAP = 15,
    REGGAE = 16,
    ROCK = 17,
    TECHNO = 18,
    INDUSTRIAL = 19,
    ALTERNATIVE = 20,
    SKA = 21,
    DEATH_METAL = 22,
    PRANKS = 23,
    SOUNDTRACK = 24,
    EURO_TECHNO = 25,
    AMBIENT = 26,
    TRIP_HOP = 27,
    VOCAL = 28,
    JAZZ_PLUS_FUNK = 29,
    FUSION = 30,
    TRANCE = 31,
    CLASSICAL = 32,
    INSTRUMENTAL = 33,
    ACID = 34,
    HOUSE = 35,
    GAME = 36,
    SOUND_CLIP = 37,
    GOSPEL = 38,
    NOISE = 39,
    ALTERNROCK = 40,
    BASS = 41,
    SOUL = 42,
    PUNK = 43,
    SPACE = 44,
    MEDITATIVE = 45,
    INSTRUMENTAL_POP = 46,
    INSTRUMENTAL_ROCK = 47,
    ETHNIC = 48,
    GOTHIC = 49,
    DARKWAVE = 50,
    TECHNO_INDUSTRIAL = 51,
```


ELECTRONIC = 52,
POP_FOLK = 53,
EURODANCE = 54,
DREAM = 55,
SOUTHERN_ROCK = 56,
COMEDY = 57,
CULT = 58,
GANGSTA = 59,
TOP_40 = 60,
CHRISTIAN_RAP = 61,
POP_FUNK = 62,
JUNGLE = 63,
NATIVE_AMERICAN = 64,
CABARET = 65,
NEW_WAVE = 66,
PSYCHADELIC = 67,
RAVE = 68,
SHOWTUNES = 69,
TRAILER = 70,
LO_FI = 71,
TRIBAL = 72,
ACID_PUNK = 73,
ACID_JAZZ = 74,
POLKA = 75,
RETRO = 76,
MUSICAL = 77,
ROCK_AND_ROLL = 78,
HARD_ROCK = 79,
FOLK = 80,
FOLK_ROCK = 81,
NATIONAL_FOLK = 82,
SWING = 83,
FAST_FUSION = 84,
BEBOB = 85,
LATIN = 86,
REVIVAL = 87,
CELTIC = 88,
BLUEGRASS = 89,
AVANTGARDE = 90,
GOTHIC_ROCK = 91,
PROGRESSIVE_ROCK = 92,
PSYCHEDELIC_ROCK = 93,
SYMPHONIC_ROCK = 94,
SLOW_ROCK = 95,
BIG_BAND = 96,
CHORUS = 97,
EASY_LISTENING = 98,
ACOUSTIC = 99,
HUMOUR = 100,
SPEECH = 101,
CHANSON = 102,
OPERA = 103,
CHAMBER_MUSIC = 104,
SONATA = 105,

```

        SYMPHONY = 106,
        BOOTY_BRASS = 107,
        PRIMUS = 108,
        PORN_GROOVE = 109,
        SATIRE = 110,
        SLOW_JAM = 111,
        CLUB = 112,
        TANGO = 113,
        SAMBA = 114,
        FOLKLORE = 115,
        BALLAD = 116,
        POWEER_BALLAD = 117,
        RHYTMIC_SOUL = 118,
        FREESTYLE = 119,
        DUET = 120,
        PUNK_ROCK = 121,
        DRUM_SOLO = 122,
        A_CAPELA = 123,
        EURO_HOUSE = 124,
        DANCE_HALL = 125

    } track_genre_t;

# endif

/***** vector.c *****/

# include <stdio.h>
# include <stdlib.h>
# include "errors.h"
# include "types.h"
# include "vector.h"

/*Crea un nuevo Vector (tipo de dato abstracto), precarga INIT_CHOP elementos y los
inicializa a NULL,
indica la cantidad de elementos almacenados con alloc_size (inicialmente cero)
y indica la capacidad de almacenamiento de elementos con size */

status_t ADT_Vector_new ( ADT_Vector_t ** ADT_Vector)
{
    size_t i;

    if (ADT_Vector == NULL)
        return ERROR_NULL_POINTER;

    if ((*ADT_Vector) = (ADT_Vector_t*) malloc (sizeof (ADT_Vector_t))) == NULL)
        return ERROR_NO_MEMORY;

    if ((*ADT_Vector) -> elements = (void **) malloc (INIT_CHOP * sizeof ( void
*)) == NULL)
    {
        free (*ADT_Vector);
    }
}

```

```

        *ADT_Vector = NULL;

        return ERROR_NO_MEMORY;
    }

    for (i = 0; i < INIT_CHOP; i++)
        (*ADT_Vector) -> elements [i] = NULL;

    (*ADT_Vector) -> alloc_size = 0;
    (*ADT_Vector) -> size = INIT_CHOP;

    return OK;
}

/*Destruye un nuevo Vector (tipo de dato abstracto), libera la memoria pedida para
el Vector y para
los elementos almacenados en él, requiere un puntero a una función destructora de
los elementos
correspondientes*/

status_t ADT_Vector_destroy (ADT_Vector_t ** ADT_Vector, status_t (*pf) (void *))
{
    status_t st;
    size_t i;

    if (pf == NULL || ADT_Vector == NULL)
        return ERROR_NULL_POINTER;

    for (i = 0; i < (*ADT_Vector) -> size; ++i)
    {
        if ( (*ADT_Vector) -> elements [i] != NULL)
        {
            st = (*pf) ((*ADT_Vector) -> elements [i]);

            if ( st != OK )
                return st;

            (*ADT_Vector) -> elements [i] = NULL;
        }
    }

    free ( (*ADT_Vector) -> elements);
    (*ADT_Vector) -> elements = NULL;

    free (*ADT_Vector);
    *ADT_Vector = NULL;

    return OK;
}

/* Setea en un Vector (tipo de dato abstracto) en la posición indicada por index,
requiere una función de creación de los elementos correspondientes.
Si existia previamente un elemento en la posición index, en la cual se queria
cargar un elementos, se coincidiera un error*/

```

```

status_t ADT_Vector_set_element (ADT_Vector_t ** ADT_Vector, void * pvoid, size_t
index)
{
    void ** aux;
    size_t i;

    if ( ADT_Vector == NULL)
        return ERROR_NULL_POINTER;

    if ((*ADT_Vector) -> alloc_size == (*ADT_Vector) -> size)
    {
        if ((aux = (void **) realloc ((*ADT_Vector) -> elements ,
(((*ADT_Vector) -> alloc_size + CHOP_SIZE) * sizeof ( void *))) == NULL)
            return ERROR_NO_MEMORY;

        (*ADT_Vector) -> elements = aux;
        (*ADT_Vector) -> size += CHOP_SIZE;

        for(i = ( 1 + ((*ADT_Vector) -> alloc_size)); i < (*ADT_Vector) ->
size ; i++)
            (*ADT_Vector) -> elements[i] = NULL;

    }

    if ((*ADT_Vector) -> elements [index] != NULL)
        return ERROR_OCUPPIED_MEMORY;
/*
    st = (*pf) (pvoid, (&(*ADT_Vector) -> elements [index]));
    if (st != OK)
        return st;
*/

    ((* ADT_Vector) -> elements [index])= pvoid;

    (*ADT_Vector) -> alloc_size ++;

    return OK;
}

```

/*Ordena un Vector (tipo de dato abstracto) con el metodo SELECTION SORT, requiere un función que compare (segun un criterio) los elementos del vector.*/

```

status_t ADT_Vector_sort (ADT_Vector_t ** ADT_Vector, int (* pf_comparer) (const
void * pvoid1, const void * pvoid2))
{
    size_t i, j;
    int min;
    void * aux_swap;

    if ( pf_comparer == NULL || ADT_Vector == NULL)
        return ERROR_NULL_POINTER;

```

```

        for (i = 0; i < (*ADT_Vector) -> alloc_size - 1; i++)
        {
            min = i;

            for (j = i + 1; j < (*ADT_Vector) -> alloc_size ; j ++)
            {
                if (((* pf_comparer) ( (*ADT_Vector) -> elements [j],
(*ADT_Vector) -> elements [min])) < 0)
                {
                    min = j;
                }
            }

            aux_swap = (*ADT_Vector) -> elements [i];

            (*ADT_Vector) -> elements [i] = (*ADT_Vector) -> elements [min];
            (*ADT_Vector) -> elements [min] = aux_swap;
        }

        return OK;
    }

/***** vector.h *****/
# ifndef VECTOR__H
# define VECTOR__H

# include <stdio.h>
# include "errors.h"
# include "types.h"

# define INIT_CHOP 1
# define CHOP_SIZE 2

typedef struct
{
    void ** elements;
    size_t size;
    size_t alloc_size;
} ADT_Vector_t;

status_t ADT_Vector_new (ADT_Vector_t ** ADT_Vector);
status_t ADT_Vector_destroy (ADT_Vector_t ** ADT_Vector, status_t (*pf) (void *));
status_t ADT_Vector_set_element (ADT_Vector_t ** ADT_Vector, void * pvoid, size_t
index);

status_t ADT_Vector_export (const ADT_Vector_t * ADT_Vector, void * context, FILE *
fo,
    status_t (*pf) (const void * pvoid, const void * pcontext, FILE * fo));

status_t ADT_Vector_sort (ADT_Vector_t ** ADT_Vector, int (* pf_comparer) (const
void * pvoid1, const void * pvoid2));

# endif

```

```
/****** track.c *****/
```

```
# include <stdio.h>
# include <string.h>
# include <stdlib.h>
# include "errors.h"
# include "mp3.h"
# include "types.h"
# include "track.h"
# include "setup.h"
```

```
string genres [NUMBER_OF_GENRES] =
{
```

```
    GENRE_BLUES,
    GENRE_CLASSIC_ROCK,
    GENRE_COUNTRY,
    GENRE_DANCE,
    GENRE_DISCO,
    GENRE_FUNK,
    GENRE_GRUNGE,
    GENRE_HIP_HOP,
    GENRE_JAZZ,
    GENRE_METAL,
    GENRE_NEW_AGE,
    GENRE_OLDIES,
    GENRE_OTHER,
    GENRE_POP,
    GENRE_R_AND_B,
    GENRE_RAP,
    GENRE_REGGAE,
    GENRE_ROCK,
    GENRE_TECHNO,
    GENRE_INDUSTRIAL,
    GENRE_ALTERNATIVE,
    GENRE_SKA,
    GENRE_DEATH_METAL,
    GENRE_PRANKS,
    GENRE_SOUNDTRACK,
    GENRE_EURO_TECHNO,
    GENRE_AMBIENT,
    GENRE_TRIP_HOP,
    GENRE_VOCAL,
    GENRE_JAZZ_PLUS_FUNK,
    GENRE_FUSION,
    GENRE_TRANCE,
    GENRE_CLASSICAL,
    GENRE_INSTRUMENTAL,
    GENRE_ACID,
    GENRE_HOUSE,
    GENRE_GAME,
    GENRE_SOUND_CLIP,
    GENRE_GOSPEL,
    GENRE_NOISE,
    GENRE_ALTERNROCK,
```

GENRE_BASS,
GENRE_SOUL,
GENRE_PUNK,
GENRE_SPACE,
GENRE_MEDITATIVE,
GENRE_INSTRUMENTAL_POP,
GENRE_INSTRUMENTAL_ROCK,
GENRE_ETHNIC,
GENRE_GOTHIC,
GENRE_DARKWAVE,
GENRE_TECHNO_INDUSTRIAL,
GENRE_ELECTRONIC,
GENRE_POP_FOLK,
GENRE_EURODANCE,
GENRE_DREAM,
GENRE_SOUTHERN_ROCK,
GENRE_COMEDY,
GENRE_CULT,
GENRE_GANGSTA,
GENRE_TOP_40,
GENRE_CHRISTIAN_RAP,
GENRE_POP_FUNK,
GENRE_JUNGLE,
GENRE_NATIVE_AMERICAN,
GENRE_CABARET,
GENRE_NEW_WAVE,
GENRE_PSYCHADELIC,
GENRE_RAVE,
GENRE_SHOWTUNES,
GENRE_TRAILER,
GENRE_Lo-Fi,
GENRE_TRIBAL,
GENRE_ACID_PUNK,
GENRE_ACID_JAZZ,
GENRE_POLKA,
GENRE_RETRO,
GENRE_MUSICAL,
GENRE_ROCK_AND_ROLL,
GENRE_HARD_ROCK,
GENRE_FOLK,
GENRE_FOLK_ROCK,
GENRE_NATIONAL_FOLK,
GENRE_SWING,
GENRE_FAST_FUSION,
GENRE_BEBOB,
GENRE_LATIN,
GENRE_REVIVAL,
GENRE_CELTIC,
GENRE_BLUEGRASS,
GENRE_AVANTGARDE,
GENRE_GOTHIC_ROCK,
GENRE_PROGRESSIVE_ROCK,
GENRE_PSYCHEDELIC_ROCK,
GENRE_SYMPHONIC_ROCK,

```

    GENRE_SLOW_ROCK,
    GENRE_BIG_BAND,
    GENRE_CHORUS,
    GENRE_EASY_LISTENING,
    GENRE_ACOUSTIC,
    GENRE_HUMOUR,
    GENRE_SPEECH,
    GENRE_CHANSON,
    GENRE_OPERA,
    GENRE_CHAMBER_MUSIC,
    GENRE_SONATA,
    GENRE_SYMPHONY,
    GENRE_BOOTY_BRASS,
    GENRE_PRIMUS,
    GENRE_PORN_GROOVE,
    GENRE_SATIRE,
    GENRE_SLOW_JAM,
    GENRE_CLUB,
    GENRE_TANGO,
    GENRE_SAMBA,
    GENRE_FOLKLORE,
    GENRE_BALLAD,
    GENRE_POWEER_BALLAD,
    GENRE_RHYTHMIC_SOUL,
    GENRE_FREESTYLE,
    GENRE_DUET,
    GENRE_PUNK_ROCK,
    GENRE_DRUM_SOLO,
    GENRE_A_CAPELA,
    GENRE_EURO_HOUSE,
    GENRE_DANCE_HALL,
};

/*Crea un ADT_Track (tipo de dato abstracto), a traves de un archivo .mp3
y lo almacena en una variable estatica.*/

status_t ADT_Track_new_from_file (void ** pvoid, FILE * file_mp3)
{
    size_t length;
    char * temp;
    char header[MP3_HEADER_SIZE];
    char buf[MP3_HEADER_SIZE];
    ADT_Track_t ** ptrack;

    if (file_mp3 == NULL || pvoid == NULL)
        return ERROR_NULL_POINTER;

    ptrack = (ADT_Track_t **) pvoid;

    if (( *ptrack = (ADT_Track_t *) malloc (sizeof (ADT_Track_t))) == NULL)
        return ERROR_NO_MEMORY;

```



```

        fseek(file_mp3, 0, SEEK_END);                /*manda el puntero al
final del archivo*/
        length = ftell(file_mp3);                    /*da la distancia al
comienzo*/
        fseek(file_mp3, length - MP3_HEADER_SIZE, SEEK_SET); /*se para en el header
MP3*/

```

```

        fread(header, sizeof(char), MP3_HEADER_SIZE, file_mp3);

```

```

        memcpy(buf, header + LEXEM_START_TITLE, LEXEM_SPAN_TITLE);
        buf[LEXEM_SPAN_TITLE] = '\0';
        strcpy ((*ptrack) -> name, buf);

```

```

        memcpy(buf, header + LEXEM_START_ARTIST, LEXEM_SPAN_ARTIST);
        buf[LEXEM_SPAN_ARTIST] = '\0';
        strcpy ((*ptrack) -> artist, buf);

```

```

        memcpy(buf, header + LEXEM_START_YEAR, LEXEM_SPAN_YEAR);
        buf[LEXEM_SPAN_YEAR] = '\0';
        (*ptrack) -> year = strtol (buf, &temp, 10);

```

```

        if (*temp)
            return ERROR_CORRUPTED_FILE;

```

```

        memcpy(buf, header + LEXEM_START_GENRE, LEXEM_SPAN_GENRE);
        (*ptrack) -> genre = buf [0];

```

```

        return OK;

```

```

}

```

/*Destruye un ADT_Track (tipo de dato abstracto), libera la memoria pedida y pone sus campos a un valor seguro.*/

```

status_t ADT_Track_destroy (void * pvoid)

```

```

{

```

```

    ADT_Track_t * ptrack;

```

```

    if (pvoid == NULL)
        return ERROR_NULL_POINTER;

```

```

    ptrack = (ADT_Track_t *) pvoid;

```

```

    strcpy (ptrack -> name, DEFAULT_TRACK_NAME);
    strcpy (ptrack -> artist, DEFAULT_TRACK_ARTIST);

```

```

    ptrack -> year = DEFAULT_TRACK_YEAR;
    ptrack -> genre = OTHER;

```

```

    free (ptrack);
    ptrack = NULL;

```

```

    return OK;

```

```

}

```

```

/*Compara 2 ADT_Track (tipo de dato abstracto) por campo nombre*/

int ADT_Track_compare_by_name (const void * pvoid1, const void * pvoid2)
{
    ADT_Track_t * ptrack1;
    ADT_Track_t * ptrack2;

    if (pvoid1 == NULL || pvoid2 == NULL)
        return ERROR_NULL_POINTER;

    ptrack1 = (ADT_Track_t *) pvoid1;
    ptrack2 = (ADT_Track_t *) pvoid2;

    return strcmp (ptrack1 -> name, ptrack2 -> name);
}

/*Compara 2 ADT_Track (tipo de dato abstracto) por campo Artista*/

int ADT_Track_compare_by_artist (const void * pvoid1, const void * pvoid2)
{
    ADT_Track_t * ptrack1;
    ADT_Track_t * ptrack2;

    if (pvoid1 == NULL || pvoid2 == NULL)
        return ERROR_NULL_POINTER;

    ptrack1 = (ADT_Track_t *) pvoid1;
    ptrack2 = (ADT_Track_t *) pvoid2;

    return strcmp (ptrack1 -> artist, ptrack2 -> artist);
}

/*Compara 2 ADT_Track (tipo de dato abstracto) por campo genero*/

int ADT_Track_compare_by_genre (const void * pvoid1, const void * pvoid2)
{
    ADT_Track_t * ptrack1;
    ADT_Track_t * ptrack2;

    if (pvoid1 == NULL || pvoid2 == NULL)
        return ERROR_NULL_POINTER;

    ptrack1 = (ADT_Track_t *) pvoid1;
    ptrack2 = (ADT_Track_t *) pvoid2;

    return ptrack1 -> genre - ptrack2 -> genre;
}

/***** track.h *****/

# ifndef TRACK__H
# define TRACK__H

# include <stdio.h>

```

```

# include "errors.h"
# include "mp3.h"
# include "types.h"

# define MAX_TRACK_NAME_LENGTH 31
# define MAX_TRACK_ARTIST_LENGTH 31
# define DEFAULT_TRACK_NAME "Unknown"
# define DEFAULT_TRACK_ARTIST "Unknown"
# define DEFAULT_TRACK_YEAR 1900
# define NUMBER_OF_PRINTERS_FUNCTIONS 2
# define NUMBER_OF_COMPARATORS_FUNCTIONS 3

typedef struct
{
    char name [MAX_TRACK_NAME_LENGTH];
    char artist [MAX_TRACK_ARTIST_LENGTH];
    ushort year;
    track_genre_t genre;
} ADT_Track_t;

status_t ADT_Track_new_from_file (void ** pvoid, FILE * file_mp3);
status_t ADT_Track_destroy (void * pvoid);
status_t ADT_Track_clone (const void * pvoid1, void ** pvoid2);

int ADT_Track_compare_by_name (const void * pvoid1, const void * pvoid2);
int ADT_Track_compare_by_artist (const void * pvoid1, const void * pvoid2);
int ADT_Track_compare_by_genre (const void * pvoid1, const void * pvoid2);

# endif

/***** tracks_printer.c *****/

# include <stdio.h>
# include <string.h>
# include "errors.h"
# include "types.h"
# include "setup.h"
# include "vector.h"
# include "track.h"
# include "mp3.h"
# include "tracks_printer.h"

extern string genres [NUMBER_OF_GENRES];

print_header_t print_header[NUMBER_OF_PRINTERS_FUNCTIONS] =
{
    print_header_csv,
    print_header_xml
};

print_track_t print_track[NUMBER_OF_PRINTERS_FUNCTIONS] =
{

```

```

        print_track_csv,
        print_track_xml
};

print_footer_t print_footer[NUMBER_OF_PRINTERS_FUNCTIONS] =
{
    print_footer_csv,
    print_footer_xml
};

status_t export_track_vector(const ADT_Vector_t * vector, track_list_format_t
format, FILE * fo)
{
    size_t i;

    if (fo == NULL || vector == NULL )
        return ERROR_NULL_POINTER;

    (*(print_header[format]))(fo);

    for(i=0; i < vector -> alloc_size; i++)
        (*(print_track[format]))(vector -> elements[i], fo);

    (*(print_footer[format]))(fo);

    return OK;
}

status_t print_header_xml(FILE * fo)
{
    if(fo == NULL)
        return ERROR_NULL_POINTER;

    fprintf(fo, "%s\n", XML_OPEN_TRACKS_TAG );

    return OK;
}

status_t print_footer_xml(FILE * fo)
{
    if(fo == NULL)
        return ERROR_NULL_POINTER;

    fprintf(fo, "%s\n", XML_CLOSE_TRACKS_TAG );

    return OK;
}

status_t print_track_xml(const ADT_Track_t * track, FILE * fo)
{
    if(track == NULL || fo == NULL)
        return ERROR_NULL_POINTER;

```

```

        fprintf(fo, "\t%s\n", XML_OPEN_TRACK_TAG);

        fprintf(fo, "\t\t%s%s\n", XML_OPEN_NAME_TAG, track -> name,
XML_CLOSE_NAME_TAG);
        fprintf(fo, "\t\t%s%s\n", XML_OPEN_ARTIST_TAG, track -> artist,
XML_CLOSE_ARTIST_TAG);
        fprintf(fo, "\t\t %s %hu %s\n", XML_OPEN_YEAR_TAG, track -> year,
XML_CLOSE_YEAR_TAG);
        fprintf(fo, "\t\t%s%s\n", XML_OPEN_GENRE_TAG, genres [track -> genre],
XML_CLOSE_GENRE_TAG);

        fprintf(fo, "\t%s\n", XML_CLOSE_TRACK_TAG);

        return OK;
}

status_t print_track_csv(const ADT_Track_t * track, FILE * fo)
{
    if(track == NULL || fo == NULL)
        return ERROR_NULL_POINTER;

    fprintf (fo,"%s%c%s%c%hu%c%s\n", track -> name, CSV_DELIMITER, track ->
artist,
        CSV_DELIMITER, track -> year, CSV_DELIMITER , genres [track-> genre]);

    return OK;
}

status_t print_header_csv(FILE * fo)
{
    if(fo == NULL)
        return ERROR_NULL_POINTER;

    fprintf (fo,"%s%c%s%c%s%c%s\n", CSV_HEADER_NAME, CSV_DELIMITER,
CSV_HEADER_ARTIST,
        CSV_DELIMITER, CSV_HEADER_YEAR, CSV_DELIMITER , CSV_HEADER_GENRES);

    return OK;
}

status_t print_footer_csv(FILE * fo)
{
    if(fo == NULL)
        return ERROR_NULL_POINTER;

    fprintf(fo, "%s\n", CSV_CLOSE_TRACKS_TAG );

    return OK;
}

```

```

/***** tracks_printer.h *****/

# ifndef TRACKS_PRINTER__H
# define TRACKS_PRINTER__H

# include <stdio.h>
# include "errors.h"
# include "types.h"
# include "vector.h"
# include "track.h"

typedef status_t (* print_header_t)(FILE * fo);
typedef status_t (* print_track_t)(const ADT_Track_t * track, FILE * fo);
typedef status_t (* print_footer_t)(FILE * fo);

status_t export_track_vector(const ADT_Vector_t * vector, track_list_format_t
format, FILE * fo);

status_t print_header_xml(FILE * fo);
status_t print_track_xml(const ADT_Track_t * track, FILE * fo);
status_t print_footer_xml(FILE * fo);

status_t print_header_csv(FILE * fo);
status_t print_track_csv(const ADT_Track_t * track, FILE * fo);
status_t print_footer_csv(FILE * fo);

# endif

```