



UNIVERSIDAD DE BUENOS AIRES  
FACULTAD DE INGENIERÍA  
Año 2018 - 1.<sup>er</sup> Cuatrimestre

## ALGORITMOS Y PROGRAMACIÓN I (95.11)

TRABAJO PRÁCTICO N.º 2

TEMA: Indexación automática de archivos MP3

FECHA: 28/06/2018

INTEGRANTES:

Puy, Gonzalo - #99784

<gonza.puy@gmail.com>

Reigada, Maximiliano Daniel - #100565

<reigada.maximiliano@gmail.com>

# Índice

1. Enunciado	2
2. Estructura lógica	5
3. Estructura Funcional	6
4. Consideraciones y Estrategias	7
5. Resultados de ejecución	8
6. Problemas encontrados y soluciones implementadas	12
7. Conclusiones	13
8. Bibliografía consultada	14
9. Script de compilación	15
10. Anexo	15

# 1. Enunciado

## TRABAJO PRÁCTICO N.º 2 – Indexación automática de archivos MP3

### 1. Objetivos

El objetivo de este trabajo práctico 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 a un criterio particular y presentado en formato texto (CSV, XML).

### 2. Alcance

Mediante el presente trabajo 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;
- estructura del encabezado de un archivo MP3 (ID3v1);
- métodos de ordenamiento;
- estructura básica de un archivo CSV;
- estructura básica de un archivo XML;

### 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:

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

**Figura 1.1:** Formato del índice a generar

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.

**Figura 1.2:** criterio de ordenamiento para los temas

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

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

**Figura 1.3:** Formato del documento CSV a generar

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

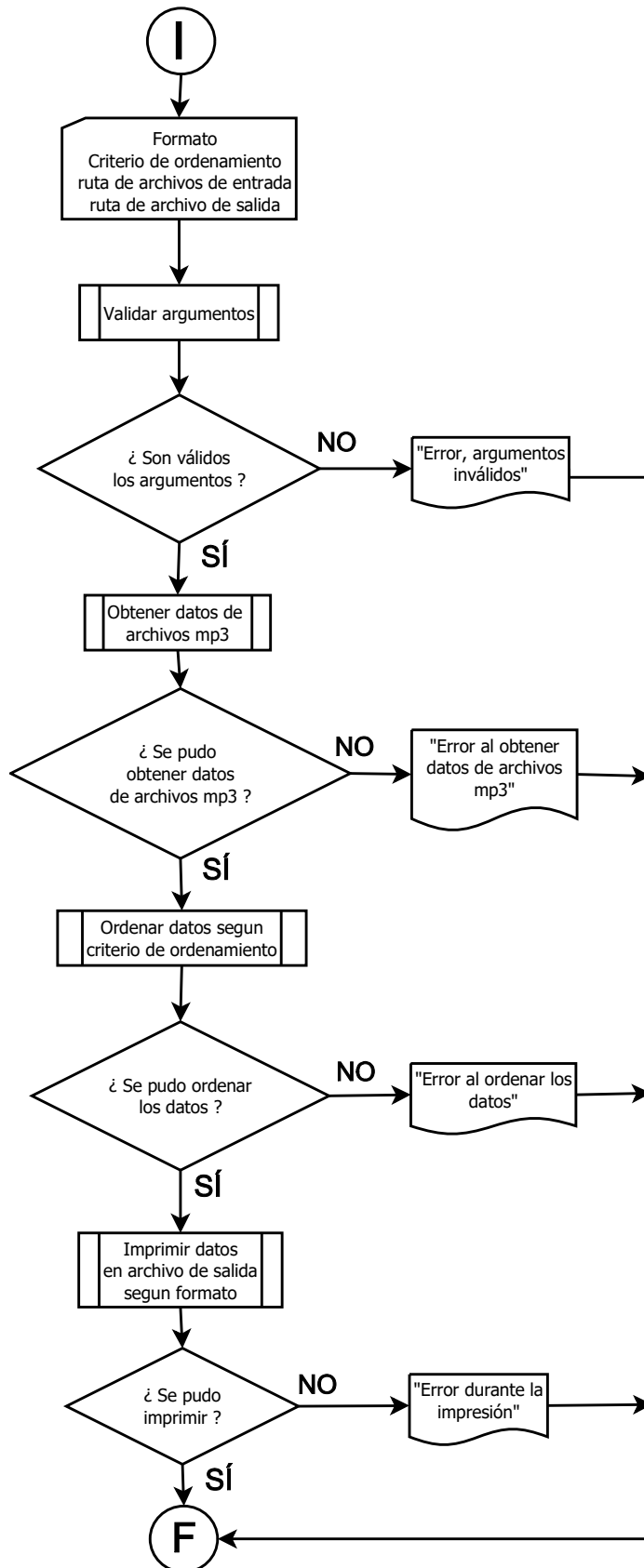
**Figura 1.4:** Formato del documento XML a generar

#### 4. Restricciones

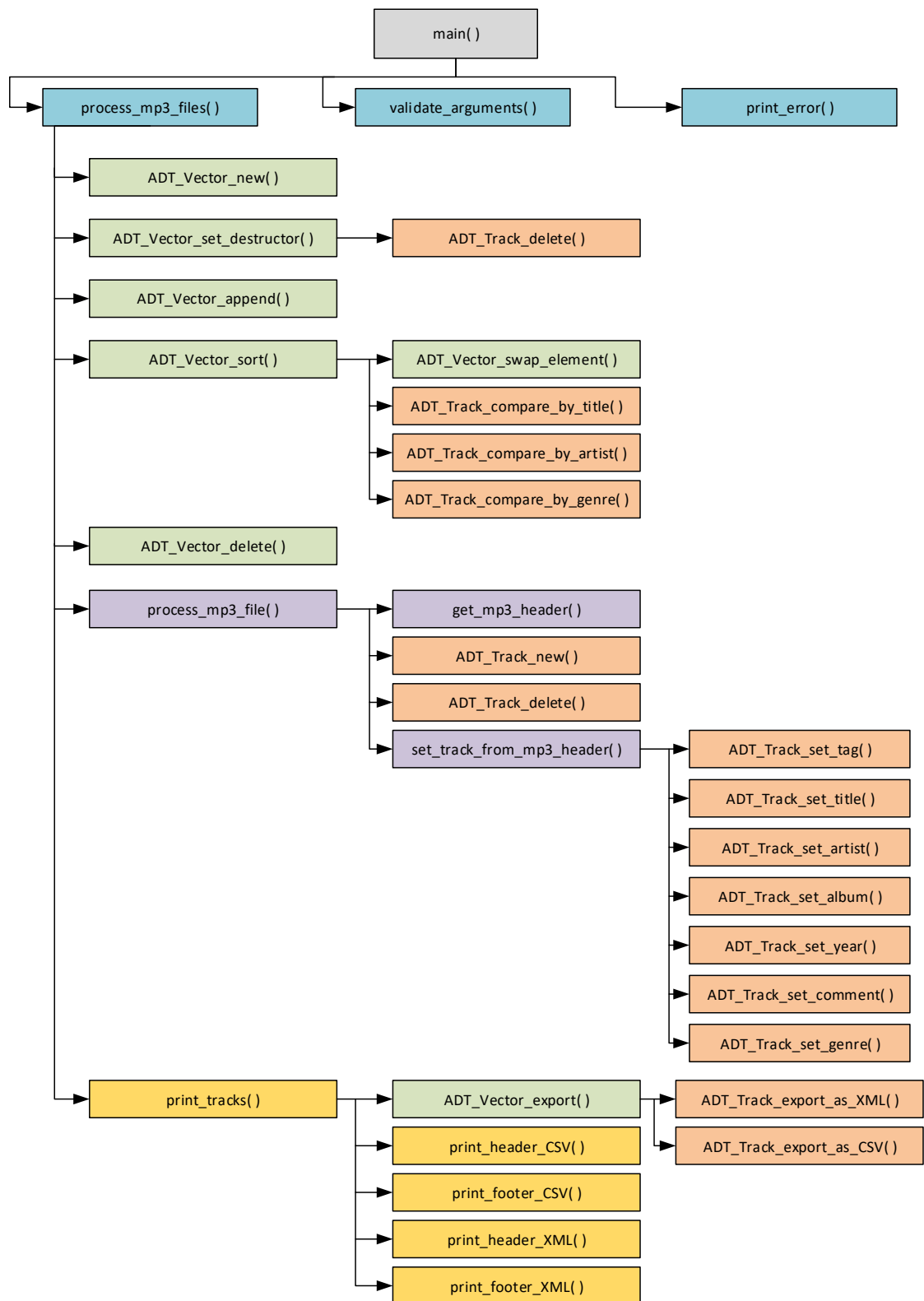
La realización de este programa está sujeta a las siguientes restricciones:

- Se debe recurrir al uso de 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.

## 2. Estructura lógica



### 3. Estructura Funcional



## 4. Consideraciones y Estrategias

Tras haber analizado el presente trabajo práctico y determinar los procedimientos necesarios para poder cumplir con el fin esperado, se optó por dividir el desarrollo en diversos procesos esenciales, de manera que la modularización estuviese justificada y cuente como punto favorable al momento de implementar el programa. En cada uno de estos se pudo identificar una o varias (según cual) funciones necesarias para completar dicha tarea.

El primer proceso fue el de validación de argumentos en el que se determinó si la cantidad de parámetros, los **flags** indicativos, el criterio de ordenamiento, el formato de archivo de salida, los archivos de entrada y la ruta de archivo de salida, ingresados por el usuario como argumentos en línea de órdenes coincidían con los necesarios para que el programa desarrollado funcionara correctamente. En este también se determinó la configuración que más tarde fue usada por el resto del sistema. Cabe destacar que se asumió que los comandos en línea de órdenes fueron ingresados en cualquier orden (en pares), y que las direcciones de los archivos de salida se ingresaron como último parámetro.

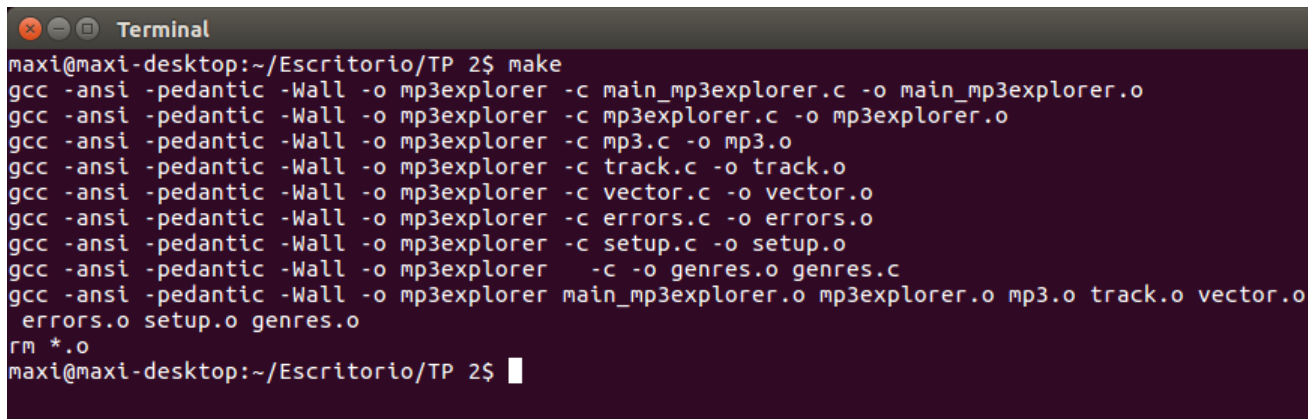
Tras haber validado los argumentos, se comenzó a procesar los archivos de entrada, para esto se optó por utilizar una variable del tipo de dato abstracto `ADT_Vector_t` cuyas componentes se referenciaron a los datos propios de cada *track*, que fueron guardados en variables del tipo `ADT_Track_t`. Una vez cargados todos los datos en el vector, se procedió a realizar el ordenamiento de este según el criterio elegido por el usuario. Para esto se implementó el método de ordenamiento de burbuja, cuyo rendimiento algorítmico dependió esencialmente de que tan ordenado estuvo el vector al ser cargado, y la cantidad de *tracks* que fueron analizados, ya que, si esta hubiese sido muy elevada, la cantidad de comparaciones efectuadas también lo hubiera sido, y en el caso que es el vector hubiera estado ordenado en orden inverso, la situación hubiese sido desfavorable.

El último proceso fue de impresión de los *tracks* ya ordenados, en el que se dio la posibilidad de elegir entre dos formatos, CSV y XML. Para esto se decidió que fuese una función propia del tipo de dato vector la encargada de imprimir de manera recursiva sus componentes, llamando mediante el uso de puntero a función, a la encargada de imprimir cada *track* en el formato adecuado.

Como era de esperar, en cada proceso se evaluó la posibilidad de que se produjera un error, por lo que fue necesario desarrollar una manera de gestionarlos. La función `print_error()` en conjunto con el diccionario de errores en español fueron desarrollados con este fin. Sin embargo, fue el módulo principal (main) el único que entró en contacto con esta implementación. De hecho, la aislación entre módulos fue lo que posibilitó su simplificación, como también la estrategia principal de desarrollo.

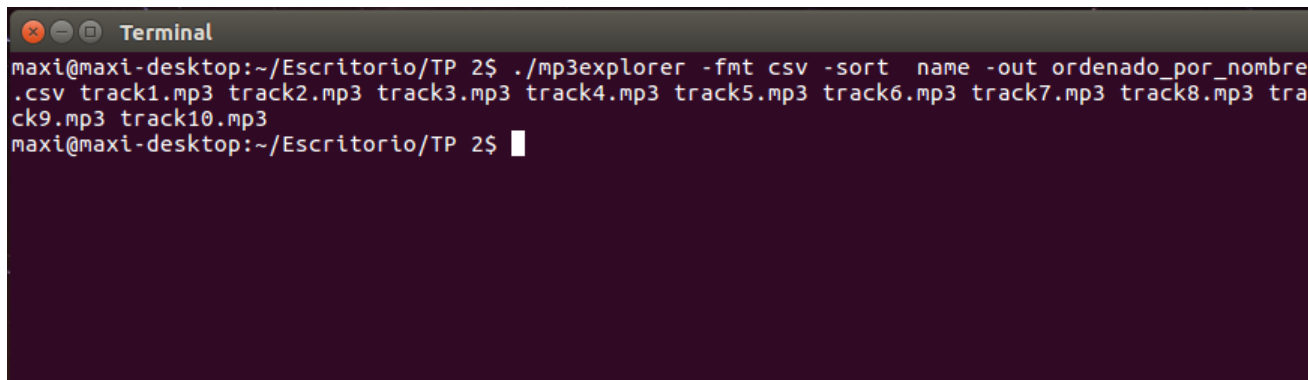


## 5. Resultados de ejecución



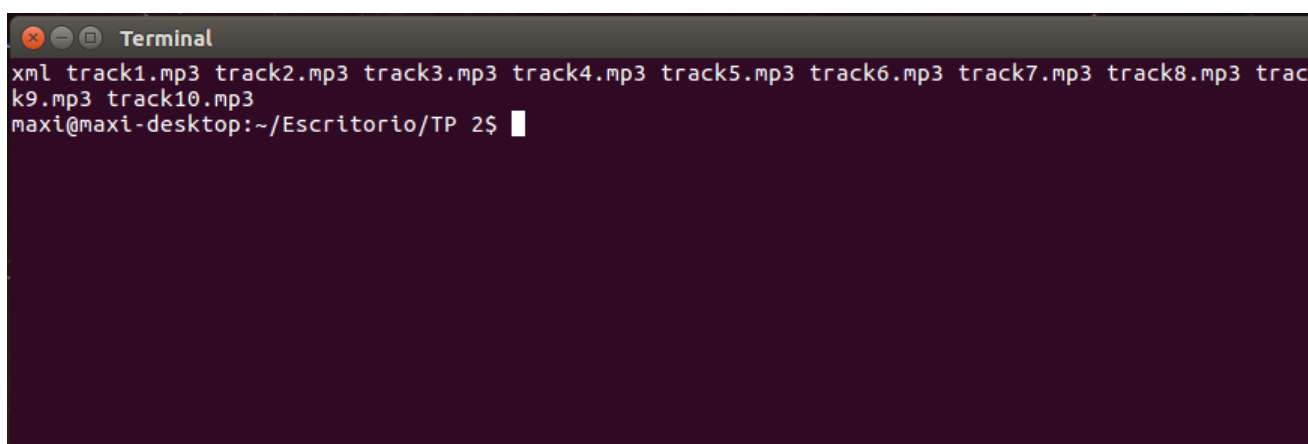
```
maxi@maxi-desktop:~/Escritorio/TP 2$ make
gcc -ansi -pedantic -Wall -o mp3explorer -c main_mp3explorer.c -o main_mp3explorer.o
gcc -ansi -pedantic -Wall -o mp3explorer -c mp3explorer.c -o mp3explorer.o
gcc -ansi -pedantic -Wall -o mp3explorer -c mp3.c -o mp3.o
gcc -ansi -pedantic -Wall -o mp3explorer -c track.c -o track.o
gcc -ansi -pedantic -Wall -o mp3explorer -c vector.c -o vector.o
gcc -ansi -pedantic -Wall -o mp3explorer -c errors.c -o errors.o
gcc -ansi -pedantic -Wall -o mp3explorer -c setup.c -o setup.o
gcc -ansi -pedantic -Wall -o mp3explorer -c -o genres.o genres.c
gcc -ansi -pedantic -Wall -o mp3explorer main_mp3explorer.o mp3explorer.o mp3.o track.o vector.o
errors.o setup.o genres.o
rm *.o
maxi@maxi-desktop:~/Escritorio/TP 2$
```

Figura 5.1: Compilación



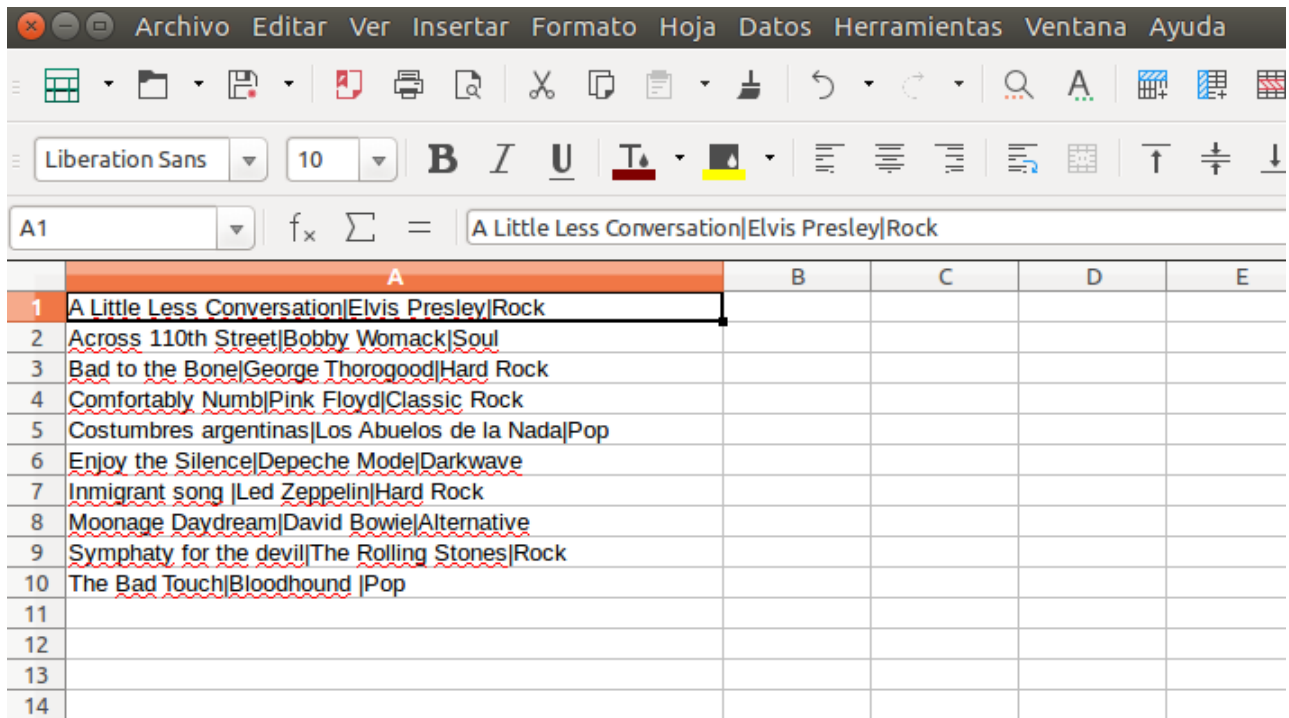
```
maxi@maxi-desktop:~/Escritorio/TP 2$ ./mp3explorer -fmt csv -sort name -out ordenado_por_nombre
.csv track1.mp3 track2.mp3 track3.mp3 track4.mp3 track5.mp3 track6.mp3 track7.mp3 track8.mp3 tra
ck9.mp3 track10.mp3
maxi@maxi-desktop:~/Escritorio/TP 2$
```

Figura 5.2: Prueba de ejecución en condiciones normales (CSV)



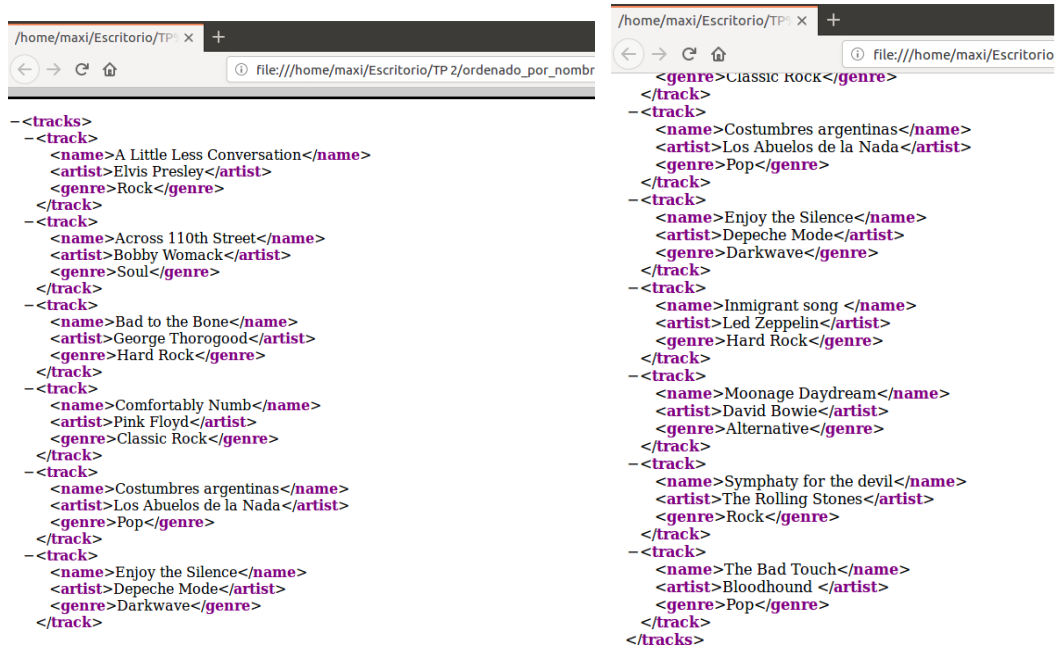
```
xml track1.mp3 track2.mp3 track3.mp3 track4.mp3 track5.mp3 track6.mp3 track7.mp3 track8.mp3 trac
k9.mp3 track10.mp3
maxi@maxi-desktop:~/Escritorio/TP 2$
```

Figura 5.3: Prueba de ejecución en condiciones normales (XML)



	A	B	C	D	E
1	A Little Less Conversation Elvis Presley Rock				
2	Across 110th Street Bobby Womack Soul				
3	Bad to the Bone George Thorogood Hard Rock				
4	Comfortably Numb Pink Floyd Classic Rock				
5	Costumbres argentinas Los Abuelos de la Nada Pop				
6	Enjoy the Silence Depeche Mode Darkwave				
7	Inmigrant song  Led Zeppelin Hard Rock				
8	Moonage Daydream David Bowie Alternative				
9	Symphaty for the devil The Rolling Stones Rock				
10	The Bad Touch Bloodhound  Pop				
11					
12					
13					
14					

Figura 5.4: Ejemplo de resultado CSV



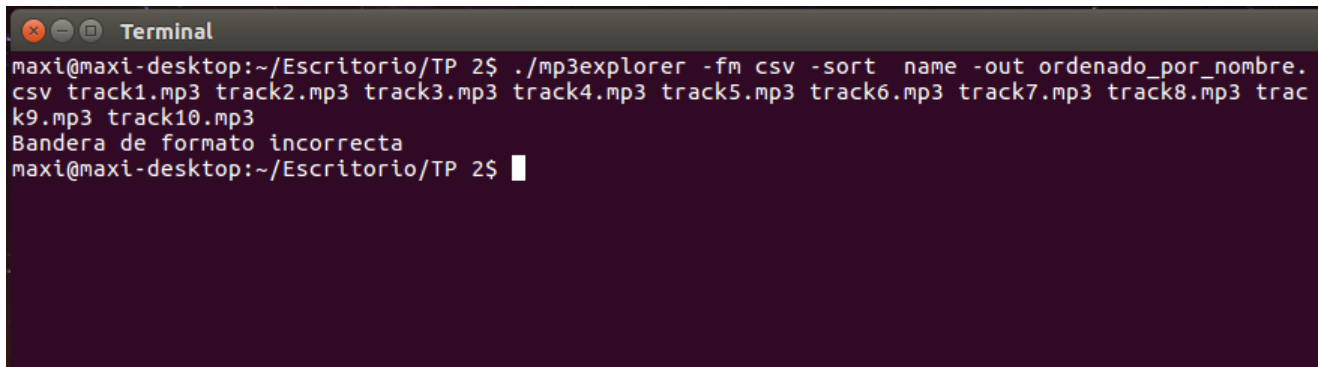
```

- <tracks>
- <track>
  <name>A Little Less Conversation</name>
  <artist>Elvis Presley</artist>
  <genre>Rock</genre>
</track>
- <track>
  <name>Across 110th Street</name>
  <artist>Bobby Womack</artist>
  <genre>Soul</genre>
</track>
- <track>
  <name>Bad to the Bone</name>
  <artist>George Thorogood</artist>
  <genre>Hard Rock</genre>
</track>
- <track>
  <name>Comfortably Numb</name>
  <artist>Pink Floyd</artist>
  <genre>Classic Rock</genre>
</track>
- <track>
  <name>Costumbres argentinas</name>
  <artist>Los Abuelos de la Nada</artist>
  <genre>Pop</genre>
</track>
- <track>
  <name>Enjoy the Silence</name>
  <artist>Depeche Mode</artist>
  <genre>Darkwave</genre>
</track>
- <track>
  <name>Inmigrant song </name>
  <artist>Led Zeppelin</artist>
  <genre>Hard Rock</genre>
</track>
- <track>
  <name>Moonage Daydream</name>
  <artist>David Bowie</artist>
  <genre>Alternative</genre>
</track>
- <track>
  <name>Symphaty for the devil</name>
  <artist>The Rolling Stones</artist>
  <genre>Rock</genre>
</track>
- <track>
  <name>The Bad Touch</name>
  <artist>Bloodhound </artist>
  <genre>Pop</genre>
</track>
</tracks>

```

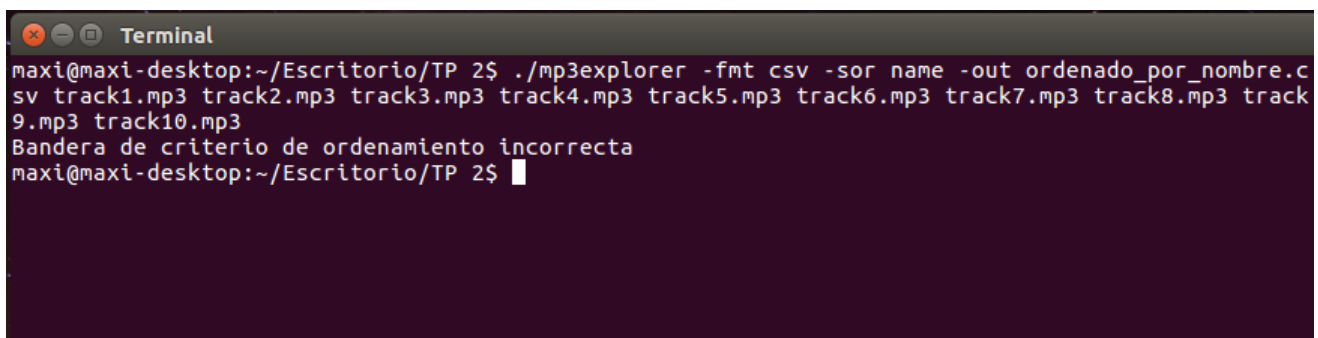
Figura 5.5: Ejemplo de resultado XML

Las siguientes imágenes muestran el funcionamiento de algunos de los errores, se eligió los errores más comunes que pueden llegar a ser cometidos por un usuario.



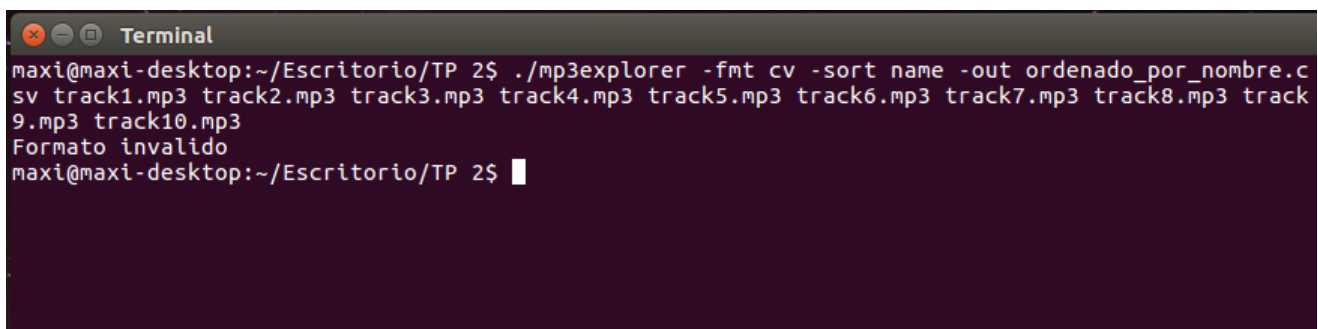
```
maxi@maxi-desktop:~/Escritorio/TP 2$ ./mp3explorer -fm csv -sort name -out ordenado_por_nombre.csv track1.mp3 track2.mp3 track3.mp3 track4.mp3 track5.mp3 track6.mp3 track7.mp3 track8.mp3 track9.mp3 track10.mp3
Bandera de formato incorrecta
maxi@maxi-desktop:~/Escritorio/TP 2$
```

Figura 5.6: Error bandera de formato



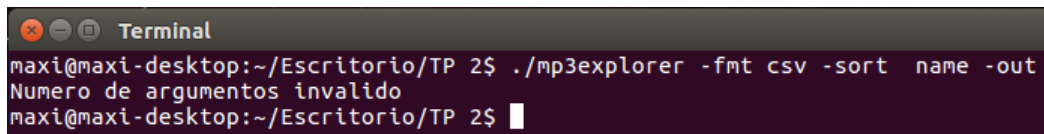
```
maxi@maxi-desktop:~/Escritorio/TP 2$ ./mp3explorer -fmt csv -sor name -out ordenado_por_nombre.csv track1.mp3 track2.mp3 track3.mp3 track4.mp3 track5.mp3 track6.mp3 track7.mp3 track8.mp3 track9.mp3 track10.mp3
Bandera de criterio de ordenamiento incorrecta
maxi@maxi-desktop:~/Escritorio/TP 2$
```

Figura 5.7: Error bandera de ordenamiento



```
maxi@maxi-desktop:~/Escritorio/TP 2$ ./mp3explorer -fmt cv -sort name -out ordenado_por_nombre.csv track1.mp3 track2.mp3 track3.mp3 track4.mp3 track5.mp3 track6.mp3 track7.mp3 track8.mp3 track9.mp3 track10.mp3
Formato invalido
maxi@maxi-desktop:~/Escritorio/TP 2$
```

Figura 5.8: Error formato invalido

A terminal window titled "Terminal" with standard window controls. The prompt is "maxi@maxi-desktop:~/Escritorio/TP 2\$". The command entered is "./mp3explorer -fmt csv -sort name -out". The output is "Numero de argumentos invalido". The prompt is repeated on the next line with a cursor.

```
maxi@maxi-desktop:~/Escritorio/TP 2$ ./mp3explorer -fmt csv -sort name -out
Numero de argumentos invalido
maxi@maxi-desktop:~/Escritorio/TP 2$
```

**Figura 5.9:** Error cantidad de argumentos

## 6. Problemas encontrados y soluciones implementadas

El principal problema encontrado a la hora de codificar el programa estuvo en el uso de T.D.A y punteros a función. Debido a que su implementación es más útil, pero compleja que otros métodos aplicables, se encontraron varios errores a la hora de verificar el funcionamiento del código. Esto se resolvió prestando sumo cuidado a la hora de utilizarlo, verificando en material de apoyo cual era la manera correcta de utilizarlos según el caso.

Otro problema surgió en el manejo del campo 'género', originalmente situado en el encabezado de cada archivo MP3. Esto se debió a que gran parte de los archivos MP3 utilizados para la verificación parcial del código poseían en algunos casos un estándar distinto al ID3v1, y en otros las etiquetas estaban mal configuradas. Para este problema se decidió por modificar los metadatos de los archivos MP3 mediante *software* externo, como por ejemplo el programa '*VLC media player*'

Para finalizar, otro conflicto que se presentó fue en cuanto a decisiones de diseño del programa. Se debatió de manera exhaustiva la forma correcta de realizar la modularización, problema que si bien no impidió en ningún momento el buen funcionamiento del sistema, al haber sido tantas las variables y funciones utilizadas, fue más que recurrente.

## 7. Conclusiones

A modo de conclusión se destaca que la modularización del código facilitó el desarrollo y puesta en prueba, y además dejó abierta la posibilidad de reutilizarlo en futuras implementaciones, como también de modificarlo para expandir los formatos o maneras de expresar la lista de *tracks* una vez ordenada.

A su vez, se observó que el desarrollo y uso de tipos de dato abstracto permitió el ocultamiento de información, y facilitó su procesamiento, de manera que, si se lo tuviese que utilizar en otra aplicación no solo sería fácil implementarlo, sino que lo más probable es que no haya que realizar muchas adiciones, ya que se idearon primitivas para las acciones más comunes y solicitadas que se pueden realizar con una variable.

Por otro lado, cabe destacar que cada variable utilizada para cargar los datos de un *track* fue completada con todos los datos que el encabezado de un archivo en formato MP3 contuviese, si bien en este trabajo esto no era necesario, para futuras implementaciones valió la pena haberlo hecho.

Para finalizar, es necesario señalar que este programa fue diseñado con el fin de crear una lista de *tracks* en formato MP3 y cuya versión ID3 fuese la primera, de haber querido extender la cantidad de posibles etiquetas de género a reconocer en el programa se podría haber hecho con tan solo modificar el diccionario en el que estos se encuentran, sin embargo, esto hubiese atentado contra el estándar al que nos apegamos durante todo el desarrollo.

## 8. Bibliografía consultada

- Kernigham, Brian - Ritchie, Dennis. '*The C Programming Language*', Estados Unidos, Editorial Prentice Hall, 1978
- Deitel, Paul - Deitel, Harvey. '*C How to program*', Estados Unidos, Editorial Prentice Hall, 2010

## 9. Script de compilación

El *script* de compilación utilizado fue el siguiente:

```

1 CFLAGS=-ansi -pedantic -Wall -o mp3explorer
2 CC=gcc
3
4 all: makefile clean
5
6 makefile: main_mp3explorer.o mp3explorer.o mp3.o track.o track_print.o vector.o errors.o setup
    .o genres.o
7     $(CC) $(CFLAGS) main_mp3explorer.o mp3explorer.o mp3.o track_print.o track.o
    vector.o errors.o setup.o genres.o
8
9 main_mp3explorer.o: main_mp3explorer.c main_mp3explorer.h mp3explorer.h types.h errors.h
10     $(CC) $(CFLAGS) -c main_mp3explorer.c -o main_mp3explorer.o
11
12 mp3explorer.o: mp3explorer.c types.h vector.h track.h track_print.h mp3explorer.h mp3.h
    setup.h
13     $(CC) $(CFLAGS) -c mp3explorer.c -o mp3explorer.o
14
15 mp3.o: mp3.c types.h mp3.h track.h vector.h
16     $(CC) $(CFLAGS) -c mp3.c -o mp3.o
17
18 track.o: track.c track.h types.h genres.h
19     $(CC) $(CFLAGS) -c track.c -o track.o
20
21 track_print.o: track_print.c track_print.h types.h track.h vector.h
22     $(CC) $(CFLAGS) -c track_print.c -o track_print.o
23
24 vector.o: vector.c vector.h types.h
25     $(CC) $(CFLAGS) -c vector.c -o vector.o
26
27 errors.o: errors.c errors.h types.h
28     $(CC) $(CFLAGS) -c errors.c -o errors.o
29
30 setup.o: setup.c setup.h types.h
31     $(CC) $(CFLAGS) -c setup.c -o setup.o
32
33 genre.o: genre.c genres.h
34     $(CC) $(CFLAGS) -c genres.c -o genres.o
35
36 clean:
37     rm *.o

```

**Código 1:** Script de compilación

## 10. Anexo

Se anexa con el presente trabajo el código fuente del programa.