Objetivo

El objetivo del presente trabajo práctico es desarrollar un aplicativo en lenguaje de programación ANSI C capaz de indexar una serie de archivos en formato MP3 ubicados en un directorio de trabajo a especificar por el usuario. Dicho índice debe ser exportado en un formato a elección en tiempo de ejecución.

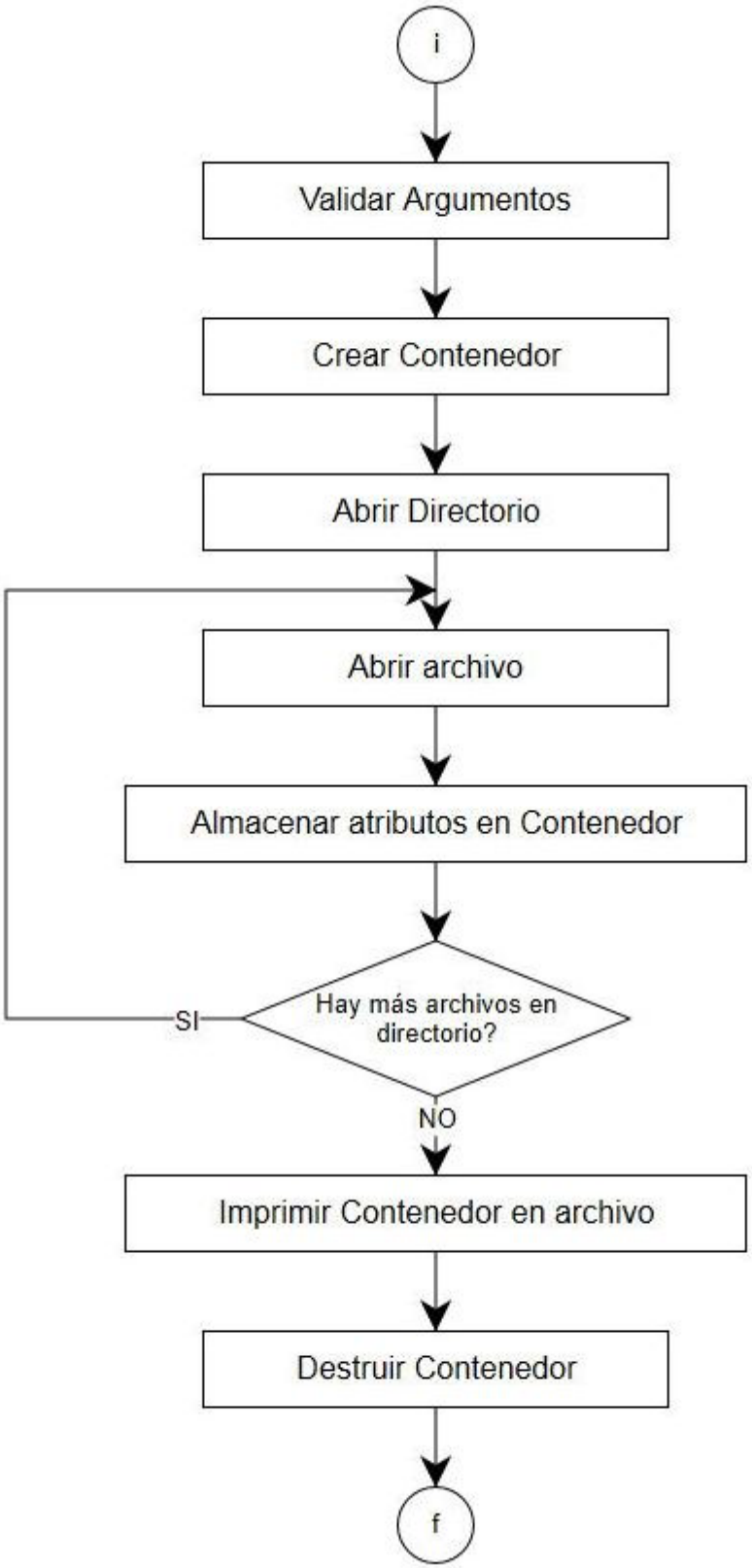
Alcance del T.P.

Durante el desarrollo del trabajo se aplicaron los siguientes conceptos:

- Programas en modo consola.
- Bibliotecas, compilación y enlace.
- Argumentos en Línea de Ordenes (CLA).
- Makefiles.
- Archivos de texto y binario.
- Memoria dinámica.
- Punteros a funciones.
- Tipo de Dato Abstracto (TDA). TDA Vector/ TDA ad-hoc.
- Estructura de headers de archivos MP3 (*ID3v1).*
- Estructura básica de un archivo CSV(**).

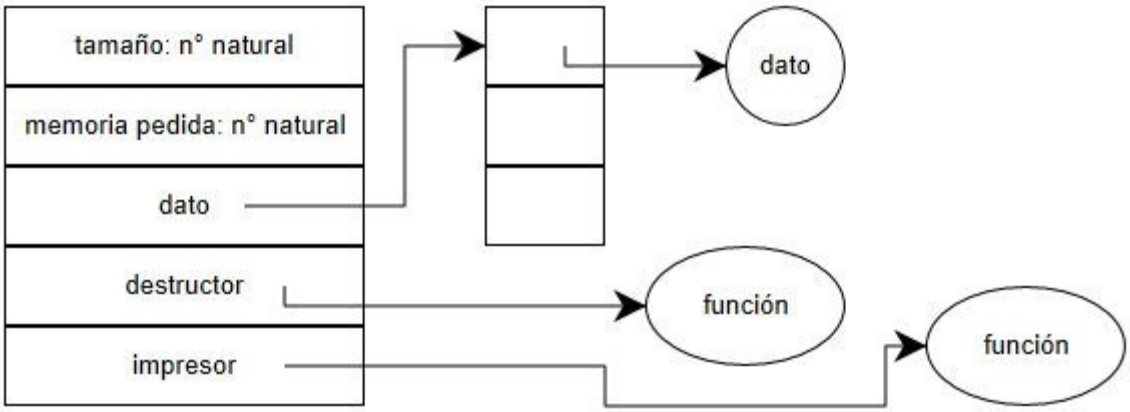
Diagrama en bloques

En primer lugar, en orden de realizar el trabajo encomendado, se trazó un diagrama en bloques del algoritmo que utilizaría el programa:

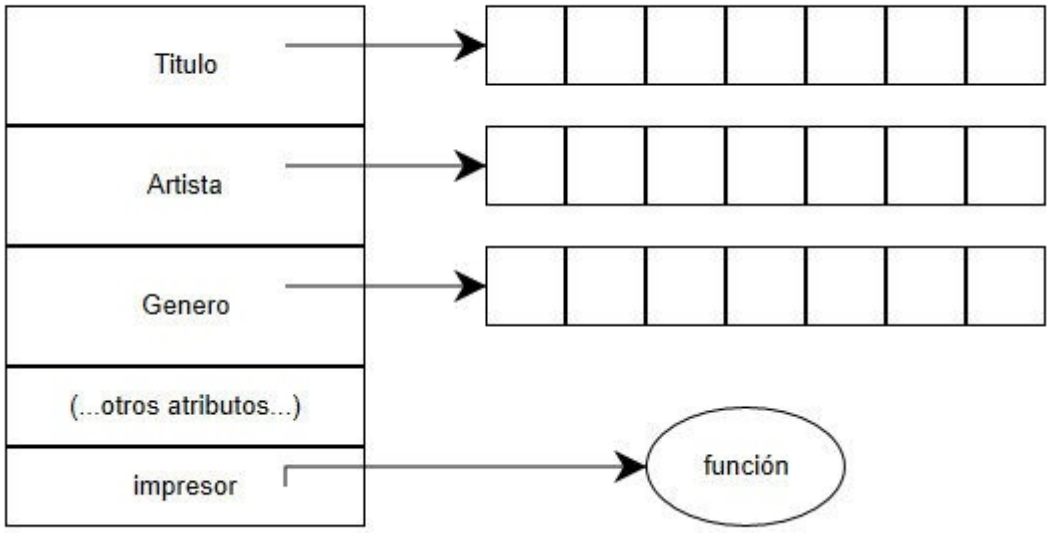

i
Validar Argumentos
Crear Contenedor
Abrir Directorio
Abrir archivo
Almacenar atributos en Contenedor
Hay más archivos en directorio?
SI
NO
Imprimir Contenedor en archivo
Destruir Contenedor
f

Estructuras de contenedores elegidas

Vector



MP3



Luego se desprende del diagrama en bloques y de la estructura elegida que el TDA Vector elegido debería poseer las siguientes primitivas:

- Crear
- Destruir
- Setear destructor
- Setear impresor
- Imprimir
- Agregar dato

Por otro lado, el TDA MP3:

- Crear
- Destruir
- Setear atributos
- Obtener atributos
- Setear impresor
- Imprimir

A continuación se detalla el pseudocódigo informal con el que se dio paso más adelante a la codificación del programa:

```
-Validar argumentos
-Crear Vector (primitiva)
-Setear impresor del vector (primitiva)

-Cargar Vector (a partir de un directorio)
  -Abrir directorio
  -Leer directorio hasta que no haya más archivos
    -Abrir archivo
    -Analizar archivo y agregar en vector
      -Crear track TDA MP3(primitiva)
      -Leer bytes de c/atributo
      -Guardar en miembro correspondiente de TDA_MP3(primitiva)
      -Setear printer del track(primitiva)
      -Añadir track al vector(primitiva)
    -Cerrar archivo
  -Cerrar directorio

[ Ordenar() <- no codificado ]
Imprimir() #puntero a funcion
  -Abrir archivo de salida
  -Imprimir Vector (primitiva)
    -Imprimir MP3 (primitiva)
  -cierro archivo

-Destruir Vector (primitiva)
  -Destruir MP3
```

Modularización

En lo que respecta a la modularización del proyecto, la filosofía adoptada fue la de facilitar la tarea de agregar mejores funcionalidades en el futuro a la aplicación. Este es el caso de las bibliotecas. Básicamente, la idea es que al modificar parte del código (alguna función o procedimiento) se necesite recompilar única y estrictamente la parte del programa que se encarga de realizar la tarea modificada. Por ejemplo, esta configuración permite agregar nuevas formas de exportación (diferentes formatos, PDF, etc) del índice de canciones en el futuro teniendo que compilar únicamente la nueva biblioteca y el módulo de variables globales.

De esta manera, el programa quedó subdividido en los siguientes módulos:

TDA Vector

- vector.c
- vector.h

TDA MP3

- mp3.c
- mp3.h

CSV

- csv.c
- csv.h

HTML –sin codificar-

XML –sin codificar

Utilidades

- utilities.c
- utilities.h

Variables globales

- globals.c

Paquete de idioma inglés

- english.c

Las dependencias de todos los archivos fuente y los headers puede leerse claramente en el makefile.

Mapa de PROTOTIPOS

```
/****************** PROTOTIPOS TDA VECTOR *****************/
status_t ADT_Array_Create(ADT_Array_t **);
    void ADT_Array_Destroy(ADT_Array_t **);
status_t ADT_Array_Set_destructor(ADT_Array_t *, destructor_t );
status_t ADT_Array_Append(ADT_Array_t *, void *);
  size_t ADT_Array_Get_size(ADT_Array_t *);
status_t ADT_Array_Get_element_at(ADT_Array_t *, size_t , void **);
/*status_t ADT_Array_Export2CSV (ADT_Array_t *, string );*/
status_t ADT_Array_Print (ADT_Array_t *, FILE *);
status_t ADT_Array_Set_printer (ADT_Array_t *, Array_printer_t );
/*********************************************************/

/****************** PROTOTIPOS TDA MP3 ********************/
status_t ADT_MP3_Create (ADT_MP3_t **);
status_t ADT_MP3_Set_Title (ADT_MP3_t *, string);
status_t ADT_MP3_Set_Artist (ADT_MP3_t *, string);
status_t ADT_MP3_Set_Genre (ADT_MP3_t *, string);
status_t ADT_MP3_Get_Title (ADT_MP3_t *, string *);
status_t ADT_MP3_Get_Artist (ADT_MP3_t *, string *);
status_t ADT_MP3_Get_Genre (ADT_MP3_t *, string *);
/*status_t ADT_MP3_Print_CSV (ADT_MP3_t *, FILE *);*/
status_t ADT_MP3_Print (ADT_MP3_t *, FILE *);
status_t ADT_MP3_Set_printer (ADT_MP3_t *, MP3_printer_t );
void ADT_MP3_Destroy (ADT_MP3_t **);
void ADT_MP3_Destroy_Title (ADT_MP3_t *);
void ADT_MP3_Destroy_Artist (ADT_MP3_t *);
void ADT_MP3_Destroy_Genre (ADT_MP3_t *);
/*********************************************************/

/************ PROTOTIPOS DE FUNCIONES DE DATOS ************/
status_t Load_Vector (string , ADT_Array_t *);
status_t Load_Track (ADT_Array_t *, FILE *);
void destroy_atributes (void *);
status_t Validate_and_SetOptions(int , char *[], string *);
status_t array_printer (const void *, FILE *);
status_t track_printer (const void *, FILE *);
/*********************************************************/

/********* PROTOTIPOS UTILIDADES *********/
status_t strdup(string *, const string );
status_t my_strcat(string *, string );
void logmsg(string );
void logstr(string , string);
void logint(string , int );
/*****************************************/

/*************** PROTOTIPOS CSV ***************/
status_t Export2CSV (ADT_Array_t *, string );
status_t Print_CSV (ADT_MP3_t *, FILE *);
/**********************************************/
/*************** PROTOTIPOS HTML ***************/
status_t Export2HTML (ADT_Array_t *, string );
```

```
status_t Print_HTML (ADT_MP3_t *, FILE *);
/*********************************************/
/*************** PROTOTIPOS XML **************/
status_t Export2XML (ADT_Array_t *, string );
status_t Print_XML (ADT_MP3_t *, FILE *);
/*********************************************/
```

Estructura funcional

main()
- Validate_and_SetOptions()
- ADT_Array_Create()
- ADT_Array_Set_printer()
- Load_Vector()
  - Load_Track()
    - ADT_MP3_Create()
    - ADT_MP3_Set_Title()
    - ADT_MP3_Set_Artist()
    - ADT_MP3_Set_Genre()
    - ADT_MP3_Set_printer()
    - ADT_Array_Append()
- pExport[export_id]() # colección de punteros a función, variable global #
  - ADT_Array_Print()
    - printer() # = array_printer() #
      - ADT_MP3_Print()
        - printer() # = track_printer #
          - MP3_printer[export_id]() # ídem pExport #
            - ADT_MP3_Get_Title()
            - ADT_MP3_Get_Artist()
            - ADT_MP3_Get_Genre()
- ADT_Array_Set_Destructor()
- ADT_Array_Destroy()
  - destructor() # = destroy_atributes #
    - ADT_MP3_Destroy_Title()
    - ADT_MP3_Destroy_Artist()
    - ADT_MP3_Destroy_Genre()

Si bien, a simple vista, daría la impresión que el proceso de exportación genera un apilamiento de subrutinas que sobrecarga el stack, esto es solo aparente debido a que la mayoría de estas funciones simplemente llaman a otras funciones pasándose un único dato que solo será leído en la función de más arriba de la pila. En otras palabras, son funciones livianas que no aportan retardo significativo.

En un principio, se intentó que el impresor del vector apunte directamente a la primitiva de impresión del MP3. Sin embargo, esto fue imposible debido a que el impresor del vector debe recibir un puntero a void, mientras que la primitiva de impresión del MP3 recibe un puntero a MP3 (TDA). Además, se quiso que el impresor del MP3 apunte directamente a la función que sabe el formato a imprimir, aunque esto no fue posible por la misma razón.

La solución fue optar por funciones que únicamente llaman a otras funciones pasándose el dato de una a otra.

Conclusión

Es de destacar que la implementación compila y corre adecuadamente y acorde a lo esperado. Sin embargo, se descubrió, luego de preparar este informe, que cuando la primitiva de impresión del TDA Vector llama a su impresor, la función que se le asigne a este podría llamar directamente a la función que sabe imprimir el dato en el formato elegido, sin necesidad de pasar por la primitiva de impresión del TDA MP3 (que encima implica pasar por otra función en el medio por incompatibilidades con el tipo de dato).

No obstante, se considera que el trabajo que tomó resolver todos los problemas presentados a lo largo del desarrollo posibilitó una práctica muy constructiva de la modularización de bibliotecas y enlaces y por sobre todo, de la utilización del tipo de dato abstracto.