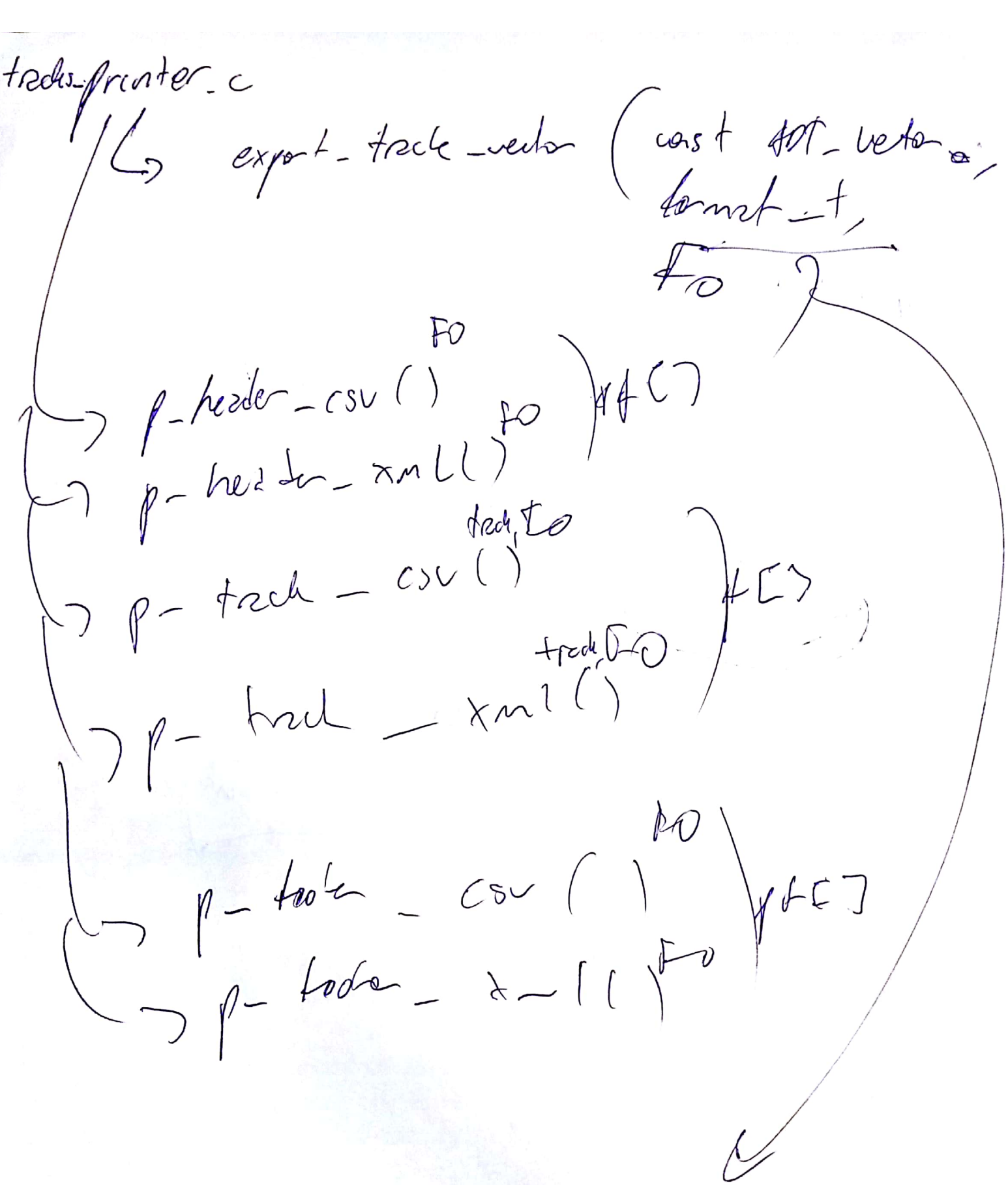
```
/***   main.c  ****/

# 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"

int main (int argc, char * argv [])
{
        status_t st;
        FILE * file_track_list;
        FILE * file_mp3;
        track_list_format_t track_list_format;
        track_sort_type_t track_sort_type;
        destructor_t destructor;
        clone_t clone;
        ADT_Vector_t * ADT_Vector;
        ADT_Track_t ADT_Track;
        size_t mp3_file_index;
        size_t mp3_files_quantity;
        context_t context;
        printer_t printers [NUMBER_OF_PRINTERS_FUNCTIONS] =
        {
                ADT_Track_export_as_csv,
                ADT_Track_export_as_xml,
        };
        comparer_t comparers [NUMBER_OF_COMPARATORS_FUNCTIONS ] =
        {
                ADT_Track_compare_by_name,
                ADT_Track_compare_by_artist,
                ADT_Track_compare_by_genre,
        };


        clone = ADT_Track_clone;
        destructor = ADT_Track_destroy;
        if ((st = validate_arguments (argc, argv, &track_list_format, &track_sort_type, &mp3_files_quantity
)) != OK)
        {
                print_error_msg (st);
                return st;
        }
        if ((st = set_context (&context, mp3_files_quantity)) != 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 < mp3_files_quantity; mp3_file_index ++)
        {
                if ((file_mp3 = fopen (argv [mp3_file_index + CMD_ARG_POSITION_FIRST_MP3_FILE], "rb"))
== NULL)
                {
```


tracks_printer.c
export_track_vector (const std_vector&
format_t,
fo)
p_header_csv ( ) fo
p_header_xml ( ) fo
p_track_csv ( ) track, fo
p_track_xml ( ) track, fo
p_footer_csv ( ) fo
p_footer_xml ( ) fo

```
                        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, clone, &ADT_Track, mp3_file_index)) !=
 OK)
                {
                        ADT_Vector_destroy (&ADT_Vector, destructor);
                        print_error_msg (st);
                        fclose (file_track_list);
                        fclose (file_mp3);
                        return st;
                }
                fclose (file_mp3);
        }
        if ((st = ADT_Vector_sort (&ADT_Vector, comparers [track_sort_type])) != OK)
        {
                ADT_Vector_destroy (&ADT_Vector, destructor);
                print_error_msg (st);
                fclose (file_track_list);
                return st;
        }

        if ((st = ADT_Vector_export (ADT_Vector, &context, file_track_list, printers [track_list_format])) !=
 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 [], track_list_format_t * track_list_format,
        track_sort_type_t * track_sort_type, size_t * mp3_files_quantity )
{
        status_t st;

        if (argv == NULL || track_list_format == NULL || track_sort_type == NULL || mp3_files_quantity
 == NULL)
                return ERROR_NULL_POINTER;
        if ((st = validate_format_argument (argv, track_list_format)) != OK)
                return st;
        if ((st = validate_sort_argument (argv, track_sort_type)) != OK)
                return st;
        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;
        *mp3_files_quantity = argc - CMD_ARG_POSITION_FIRST_MP3_FILE;
        return OK;
}
```

```
# endif


/*** types.h ***/

# ifndef TYPES__H
# define TYPES__H

# include <stdio.h>
# include "errors.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 status_t (* printer_t) (const void * pvoid, const void * pcontext, FILE * fo);
typedef int (* comparer_t) (const void * pvoid1, const void * pvoid2);

# endif
```

comparer_t

```
/*Exporta un Vector (tipo de dato abstracto) en el stream fo, requiere un función que imprima elementos
en un formato correspondiente, y un contexto de impresion.*/

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 st;
        size_t i;

        if (pf == NULL || ADT_Vector == NULL || context == NULL)
                return ERROR_NULL_POINTER;
        for (i = 0; i < ADT_Vector -> alloc_size; ++i)
        {
                if ((st = (*pf) (ADT_Vector -> elements [i], context, fo ) != OK))
                        return st;
        }
        return OK;
}
```

```
/*Exporta un ADT_Track (tipo de dato abstracto) con formato csv en un flujo fo, requiere un
contexto de impresion.*/

status_t ADT_Track_export_as_csv (const void * pvoid, const void * pcontext, FILE * fo)
{
        ADT_Track_t * ptrack;
        context_t * context;

        static 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,
```