06/27/18 07:33:27 /home/ivan/Desktop/Repositorio-Final/main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "main.h"

/*Biblioteca de formatos*/
char * format_dictionary[MAX_FORMATS] =
{
    CSV_FORMAT,
    XML_FORMAT
};


/*Biblioteca de ordenamientos*/
char * sort_dictionary[MAX_SORTS] =
{
    SORT_BY_NAME,
    SORT_BY_ARTIST,
    SORT_BY_GENRE
};


/**********Externs**********/
extern char context_csv;
extern char * context_xml[MAX_XML_CONTEXTS];
extern status_t (*format_output[MAX_FORMATS])(void *, const void *, FILE *);
extern int (*sort_output[MAX_SORTS]) (void *, void *);
extern char * errors_dictionary[MAX_ERRORS];
extern setup_t setup;
/**********Externs**********/

int main(int argc, char *argv[])
{
    size_t i;
    size_t out_index;
    FILE *file_out, *mp3_file;
    status_t st;
    ADT_Vector_t * vector;
    void * context;

    /*Se validan los argumentos*/
    if((st = validate_arguments(argc, argv, &setup, &out_index)) != OK)
    {
        print_errors(st);
        return st;
    }

    if (setup.doc_type == FMT_CSV)
    {
        context = &context_csv;
    }
    else
    {
        context = &context_xml;
    }

    /*Se crea el vector*/
    if((st = ADT_Vector_new(&vector)) != OK)
    {
        print_errors(st);
        return st;
    }

    /*Se establece una funcion para imprimir*/
    if((st = ADT_Vector_set_printer (vector, format_output[setup.doc_type])) != OK)
    {
        print_errors(st);
        ADT_Vector_delete(&vector);

        return st;
    }

    /*Se establece una funcion para comparar*/
    if((st = ADT_Vector_set_comparator (vector, sort_output[setup.sort_by])) != OK)
    {
        print_errors(st);
        ADT_Vector_delete(&vector);
        return st;
    }

    /*Se establece una funcion para destruir elementos*/
    if((st = ADT_Vector_set_destructor(vector, ADT_track_delete)) != OK)
    {
        print_errors(st);
        ADT_Vector_delete(&vector);
        return st;
    }
```

```
 89
 90      /*Se abre el archivo de salida*/
 91      if ((file_out = fopen(argv[out_index], "wt")) == NULL)
 92      {
 93          st = ERROR_INVALID_OUTPUT_FILE;
 94          print_errors(st);
 95          ADT_Vector_delete(&vector);
 96          return st;
 97      }
 98
 99
100
101
102      /*Aquí se abren los archivos mp3, se processan los datos y luego se cierran*/
103      for(i = 0; i < argc - INDEX_FIRST_MP3; i++ )
104      {
105
106          if((mp3_file = fopen(argv[INDEX_FIRST_MP3 + i], "rt")) == NULL)
107          {
108              st = ERROR_INVALID_MP3_FILE;
109              print_errors(st);
110              return st;
111          }
112
113          if((st = process_mp3_data(&setup, mp3_file, vector)) != OK)
114          {
115              ADT_Vector_delete(&vector);
116              fclose(mp3_file);
117              fclose(file_out);
118              print_errors(st);
119              return st;
120          }
121
122          if((fclose(mp3_file)) == EOF)
123          {
124              st = ERROR_CLOSING_FILE;
125              print_errors(st);
126              ADT_Vector_delete(&vector);
127              return st;
128          }
129      }
130
131
132      /*Se ordena el vector donde se ingresaron los datos de los archivos mp3*/
133      if((st = ADT_Vector_sort_elements(&vector, ADT_Vector_swap_elements)) != OK)
134      {
135          print_errors(st);
136          fclose(file_out);
137          ADT_Vector_delete(&vector);
138          return st;
139      }
140
141      /*Se imprimen los elementos en el orden y formato elegido*/
142      if((st = ADT_Vector_export(vector, context, file_out, setup)) != OK)
143      {
144          print_errors(st);
145          fclose(file_out);
146          ADT_Vector_delete(&vector);
147          return st;
148      }
149
150      /*Se destruye el vector utilizado*/
151      if((st = ADT_Vector_delete(&vector)) != OK)
152      {
153          fclose(file_out);
154          print_errors(st);
155          return st;
156      }
157
158      /*Se cierra el archivo de salida*/
159      if((fclose(file_out)) == EOF)
160      {
161          st = ERROR_CLOSING_FILE;
162          print_errors(st);
163          return st;
164      }
165
166      return OK;
167  }
168
169  /*Función que valida los argumentos de la invocación*/
170  status_t validate_arguments(int argc, char * argv[], setup_t * setup, size_t * index_out_file)
171  {
172      size_t i;
173      size_t fmt_flag = 0;
174      size_t sort_flag = 0;
175      size_t out_flag = 0;
176
177      if(argv == NULL || setup == NULL)
178          return ERROR_NULL_POINTER;
179
```

```
180     if(argc < MIN_ARGUMENTS)
181     {
182         return ERROR_INVOCATION;
183     }
184
185     for(i=0; i<argc; i++)
186     {
187         if(strcmp(argv[i], FORMAT_FLAG_TOKEN) == 0)
188             fmt_flag = i;
189         if(strcmp(argv[i], SORT_FLAG_TOKEN) == 0)
190             sort_flag = i;
191         if(strcmp(argv[i], OUT_FLAG_TOKEN) == 0)
192             out_flag = i;
193     }
194
195     if(!fmt_flag || !sort_flag || !out_flag)
196         return ERROR_INVOCATION;
197
198     for(i=0 ; i < MAX_FORMATS; i++)
199     {
200         if (!(strcmp(argv[fmt_flag + 1], format_dictionary[i])))
201         {
202             setup->doc_type = i;
203             break;
204         }
205     }
206
207     if(i == MAX_FORMATS)
208         return ERROR_INVOCATION;
209
210     for(i=0 ; i < MAX_SORTS ; i++)
211     {
212         if(!(strcmp(argv[sort_flag + 1], sort_dictionary[i])))
213         {
214             setup->sort_by = i;
215             break;
216         }
217     }
218
219     if(i == MAX_SORTS)
220         return ERROR_INVOCATION;
221
222     *index_out_file = out_flag + 1;
223
224     return OK;
225 }
```