06/28/18 02:37:29 /home/ivan/Desktop/Repositorio-Final/track.c

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

#include "types.h"
#include "track.h"
#include "errors.h"
#include "main.h"

extern char * errors_dictionary[MAX_ERRORS];


/*Diccionarios de formato, tipos de ordenes y generos*/

status_t (*format_output[MAX_FORMATS]) (void *, const void *, FILE *) =
{
    ADT_track_export_to_csv,
    ADT_track_export_to_xml
};

int (*sort_output[MAX_SORTS]) (void *, void *) =
{
    ADT_track_compare_by_title,
    ADT_track_compare_by_artist,
    ADT_track_compare_by_genre
};

char * genres_dictionary[MAX_GENRES] =
{
    "Blues", "Classic Rock", "Country", "Dance", "Disco", "Funk", "Grunge",
    "Hip-Hop", "Jazz", "Metal", "New Age", "Oldies", "Other", "Pop",
    "R&B", "Rap", "Reggae", "Rock", "Techno", "Industrial", "Alternative",
    "Ska", "Death Metal", "Pranks", "Soundtrack", "Euro-Techno", "Ambient", "Trip-Hop",
    "Vocal", "Jazz+Funk", "Fusion", "Trance", "Classical", "Instrumental", "Acid",
    "House", "Game", "Sound Clip", "Gospel", "Noise", "AlternRock", "Bass",
    "Soul", "Punk", "Space", "Meditative", "Instrumental Pop", "Instrumental Rock", "Ethnic",
    "Gothic", "Darkwave", "Techno-Industrial", "Electronic", "Pop-Folk", "Eurodance", "Dream",
    "Southern Rock", "Comedy", "Cult", "Gangsta", "Top 40", "Christian Rap", "Pop/Funk",
    "Jungle", "Native American", "Cabaret", "New Wave", "Psychadelic", "Rave", "Showtunes",
    "Trailer", "Lo-Fi", "Tribal", "Acid Punk", "Acid Jazz", "Polka", "Retro",
    "Musical", "Rock & Roll", "Hard Rock", "Folk", "Folk-Rock", "National Folk", "Swing",
    "Fast Fusion", "Bebob", "Latin", "Revival", "Celtic", "Bluegrass", "Avantgarde",
    "Gothic Rock", "Progressive Rock", "Psychedelic Rock", "Symphonic Rock", "Slow Rock", "Big
Band", "Chorus",
    "Easy Listening", "Acoustic", "Humour", "Speech", "Chanson", "Opera", "Chamber Music",
    "Sonata", "Symphony", "Booty Brass", "Primus", "Porn Groove", "Satire", "Slow Jam",
    "Club", "Tango", "Samba", "Folklore", "Ballad", "Poweer Ballad", "Rhytmic Soul",
    "Freestyle", "Duet", "Punk Rock", "Drum Solo", "A Capela", "Euro-House", "Dance Hall"
};

/*Esta función crea una nueva pista*/
status_t ADT_track_new (ADT_track_t ** track)
{
    if (track == NULL)
        return ERROR_NULL_POINTER;

    if ((*track =(ADT_track_t*)malloc(sizeof(ADT_track_t))) == NULL)
        return ERROR_OUT_OF_MEMORY;

    (*track)->tag[0] = '\0';
    (*track)->title[0] = '\0';
    (*track)->artist[0] = '\0';
    (*track)->album[0] = '\0';
    (*track)->year[0] = '\0';
    (*track)->comment[0] = '\0';
    (*track)->genre = 0;

    return OK;
}

/*Esta funcion destruye una pista*/
status_t ADT_track_delete (void * t)
{
    ADT_track_t * track;
```

```
 74
 75      track = (ADT_track_t *)t;
 76
 77      if (track == NULL)
 78          return ERROR_NULL_POINTER;
 79
 80      track->tag[0] = '\0';
 81      track->title[0] = '\0';
 82      track->artist[0] = '\0';
 83      track->album[0] = '\0';
 84      track->year[0] = '\0';
 85      track->comment[0] = '\0';
 86      track->genre = 0;
 87
 88      free(track);
 89      track = NULL;
 90
 91      return OK;
 92  }
 93
 94  /*Esta función establece una pista*/
 95  status_t ADT_track_set (char header[], ADT_track_t * track)
 96  {
 97      char aux[2];
 98
 99      if (header == NULL || track == NULL)
100         return ERROR_NULL_POINTER;
101
102      memcpy(track->tag,header+LEXEM_START_TAG,LEXEM_SPAN_TAG);
103      track->tag[LEXEM_SPAN_TAG] = '\0';
104
105      memcpy(track->title,header+LEXEM_START_TITLE,LEXEM_SPAN_TITLE);
106      track->title[LEXEM_SPAN_TITLE] = '\0';
107
108      memcpy(track->artist,header+LEXEM_START_ARTIST,LEXEM_SPAN_ARTIST);
109      track->artist[LEXEM_SPAN_ARTIST] = '\0';
110
111      memcpy(track->album,header+LEXEM_START_ALBUM,LEXEM_SPAN_ALBUM);
112      track->album[LEXEM_SPAN_ALBUM] = '\0';
113
114      memcpy(track->year,header+LEXEM_START_YEAR,LEXEM_SPAN_YEAR);
115      track->year[LEXEM_SPAN_YEAR] = '\0';
116
117      memcpy(track->comment,header+LEXEM_START_COMMENT,LEXEM_SPAN_COMMENT);
118      track->comment[LEXEM_SPAN_COMMENT] = '\0';
119
120      memcpy(aux,header+LEXEM_START_GENRE,LEXEM_SPAN_GENRE);
121      track->genre = aux[0];
122
123      return OK;
124  }
125
126  /*Esta función exporta una pista a un archivo csv*/
127  status_t ADT_track_export_to_csv (void * t, const void * context, FILE * file_out)
128  {
129      char del;
130      ADT_track_t * track;
131
132      del = *((char *)context);
133      track = (ADT_track_t *)t;
134
135      if(fprintf(file_out, "%s", track->title) < 0)
136          return ERROR_WRITING_TO_FILE;
137
138      if (fputc(del, file_out) == EOF)
139          return ERROR_WRITING_TO_FILE;
140
141      if(fprintf(file_out, "%s", track->artist) < 0)
142          return ERROR_WRITING_TO_FILE;
143
144      if(fputc(del, file_out) == EOF)
145          return ERROR_WRITING_TO_FILE;
146
147      if(fprintf(file_out, "%s\n", genres_dictionary[track->genre]) < 0)
148          return ERROR_WRITING_TO_FILE;
149
```

```
150     return OK;
151 }
152 
153 /*Esta función exporta una pista a un archivo csv*/
154 status_t ADT_track_export_to_xml (void * t, const void * context, FILE * file_out)
155 {
156     char ** xml_contexts;
157     ADT_track_t * track;
158 
159     xml_contexts = (char **)context;
160     track = (ADT_track_t *)t;
161 
162     if(fprintf(file_out, "\t%s%s%s\n", xml_contexts[XML_OPEN_INITIAL_BRACKET_INDEX],
    xml_contexts[XML_TRACK_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
163         return ERROR_WRITING_TO_FILE;
164 
165     if(fprintf(file_out, "\t\t%s%s%s", xml_contexts[XML_OPEN_INITIAL_BRACKET_INDEX],
    xml_contexts[XML_NAME_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
166         return ERROR_WRITING_TO_FILE;
167 
168     if(fprintf(file_out, "%s", track->title) < 0)
169         return ERROR_WRITING_TO_FILE;
170 
171     if(fprintf(file_out, "%s%s%s\n", xml_contexts[XML_OPEN_FINISHER_BRACKET_INDEX],
    xml_contexts[XML_NAME_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
172         return ERROR_WRITING_TO_FILE;
173 
174     if(fprintf(file_out, "\t\t%s%s%s", xml_contexts[XML_OPEN_INITIAL_BRACKET_INDEX],
    xml_contexts[XML_ARTIST_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
175         return ERROR_WRITING_TO_FILE;
176 
177     if(fprintf(file_out, "%s", track->artist) < 0)
178         return ERROR_WRITING_TO_FILE;
179 
180     if(fprintf(file_out, "%s%s%s\n", xml_contexts[XML_OPEN_FINISHER_BRACKET_INDEX],
    xml_contexts[XML_ARTIST_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
181         return ERROR_WRITING_TO_FILE;
182 
183     if(fprintf(file_out, "\t\t%s%s%s", xml_contexts[XML_OPEN_INITIAL_BRACKET_INDEX],
    xml_contexts[XML_GENRE_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
184         return ERROR_WRITING_TO_FILE;
185 
186     if(fprintf(file_out, "%s", genres_dictionary[track->genre]) < 0)
187         return ERROR_WRITING_TO_FILE;
188 
189     if(fprintf(file_out, "%s%s%s\n", xml_contexts[XML_OPEN_FINISHER_BRACKET_INDEX],
    xml_contexts[XML_GENRE_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
190         return ERROR_WRITING_TO_FILE;
191 
192     if(fprintf(file_out, "\t%s%s%s\n", xml_contexts[XML_OPEN_FINISHER_BRACKET_INDEX],
    xml_contexts[XML_TRACK_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
193         return ERROR_WRITING_TO_FILE;
194 
195     return OK;
196 }
197 
198 /*Esta función compara dos pistas según el artista*/
199 int ADT_track_compare_by_artist (void * t1, void * t2)
200 {
201     size_t i;
202     ADT_track_t *track1, *track2;
203 
204     track1 = (ADT_track_t *)t1;
205     track2 = (ADT_track_t *)t2;
206 
207     if (track1 == NULL || track2 == NULL)
208         return 0;
209 
210     for(i=0; track1->artist[i] && track2->artist[i]; i++)
211     {
212         if (track1->artist[i] != track2->artist[i])
213         {
214             return (track1->artist[i] - track2->artist[i]);
215         }
216     }
217     if (!track1->artist[i] && track2->artist[i])
```

```
218     {
219         return 1;
220     }
221     if (track1->artist[i] && !track2->artist[i])
222     {
223         return -1;
224     }
225     return 0;
226 }
227 
228 /*Esta función compara dos pistas segun el nombre*/
229 int ADT_track_compare_by_title (void * t1, void * t2)
230 {
231     size_t i;
232     ADT_track_t *track1, *track2;
233 
234     if (t1 == NULL || t2 == NULL)
235         return 0;
236 
237     track1 = (ADT_track_t *)t1;
238     track2 = (ADT_track_t *)t2;
239 
240 
241     for(i=0; track1->title[i] && track2->title[i]; i++)
242     {
243         if (track1->title[i] != track2->title[i])
244         {
245             return (track1->title[i] - track2->title[i]);
246         }
247     }
248     if (!track1->title[i] && track2->title[i])
249     {
250         return 1;
251     }
252     if (track1->title[i] && !track2->title[i])
253     {
254         return -1;
255     }
256     return 0;
257 }
258 
259 /*Esta función compara dos pistas segun el género*/
260 int ADT_track_compare_by_genre (void * t1, void * t2)
261 {
262     ADT_track_t *track1, *track2;
263 
264     if (t1 == NULL || t2 == NULL)
265         return 0;
266 
267     track1 = (ADT_track_t *)t1;
268     track2 = (ADT_track_t *)t2;
269 
270     return track1->genre - track2->genre;
271 }
```