```c
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  #include "vector.h"
 5
 6  extern char * errors_dictionary[MAX_ERRORS];
 7
 8
 9  /*Esta función crea un nuevo vector*/
10  status_t ADT_Vector_new(ADT_Vector_t ** v)
11  {
12      size_t i;
13
14      if (v == NULL) return ERROR_NULL_POINTER;
15
16      if ((*v =(ADT_Vector_t*)malloc(sizeof(ADT_Vector_t))) == NULL)
17          return ERROR_OUT_OF_MEMORY;
18
19      if (((*v)->elements = (void**)malloc(INIT_CHOP*sizeof(void*))) == NULL)
20      {
21          free(*v);
22          *v = NULL;
23          return ERROR_OUT_OF_MEMORY;
24      }
25
26      for(i=0; i<INIT_CHOP; i++)
27      {
28          (*v)->elements[i] = NULL;
29      }
30
31      (*v)->alloc_size = INIT_CHOP;
32
33      (*v)->size = 0;
34      (*v)->destructor = NULL;
35      (*v)->comparator = NULL;
36      (*v)->printer = NULL;
37
38      return OK;
39  }
40
41  /*Esta función destruye un vector*/
42  status_t ADT_Vector_delete (ADT_Vector_t ** v)
43  {
44      status_t st;
45      size_t i;
46      for(i=0; i<(*v)->size; i++)
47      {
48          st = ((*v)->destructor)((*v)->elements[i]);
49          if (st != OK)
50              return st;
51      }
52
53      free((*v)->elements);
54      (*v)->elements = NULL;
55      free(*v);
56      *v=NULL;
57      return OK;
58  }
59
60  /*Esta función obtiene un elemento de un vector*/
61  void * ADT_Vector_get_element (ADT_Vector_t * v, int position)
62  {
63      if (v == NULL) return NULL;
64
65      if (position < 0) return v->elements[v->size + position];
66      if (position > v->size) return NULL;
67
68      return v->elements[position];
69  }
70
71  /*Esta función se fija si un vector está vacío*/
72  bool_t ADT_Vector_is_empty (ADT_Vector_t * p)
73  {
74      return (p->size) ? FALSE:TRUE;
```

```
 75    }
 76
 77    /*Esta función establece una función de impresión de vector*/
 78    status_t ADT_Vector_set_printer(ADT_Vector_t * v, printer_t pf)
 79    {
 80        if(v==NULL) return ERROR_NULL_POINTER;
 81
 82        v->printer = pf;
 83        return OK;
 84    }
 85
 86    /*Esta función establece una función que compara elementos*/
 87    status_t ADT_Vector_set_comparator(ADT_Vector_t * v, comparator_t cf)
 88    {
 89        if(v==NULL) return ERROR_NULL_POINTER;
 90
 91        v->comparator = cf;
 92        return OK;
 93    }
 94
 95    /*Esta función establece una función que destruye elementos*/
 96    status_t ADT_Vector_set_destructor(ADT_Vector_t * v, destructor_t df)
 97    {
 98        if(v==NULL) return ERROR_NULL_POINTER;
 99
100        v->destructor = df;
101        return OK;
102    }
103
104    /*Esta función exporta un Vector*/
105    status_t ADT_Vector_export (ADT_Vector_t * v, const void * context, FILE * file, setup_t *
       setup)
106    {
107        size_t i;
108        status_t st;
109        char ** xml_contexts = NULL;
110
111        if (v == NULL || file == NULL)
112            return ERROR_NULL_POINTER;
113
114        if (setup->doc_type == FMT_XML)
115        {
116            xml_contexts = (char **)context;
117
118            if(fprintf(file, "%s\n", xml_contexts[XML_VERSION_LINE_INDEX]) < 0)
119                return ERROR_WRITING_TO_FILE;
120
121            if(fprintf(file, "%s%s%s\n", xml_contexts[XML_OPEN_INITIAL_BRACKET_INDEX],
       xml_contexts[XML_TRACKS_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
122                return ERROR_WRITING_TO_FILE;
123        }
124
125        for (i = 0; i < v->size; i++)
126        {
127            if ((st = (v->printer)(v->elements[i], context, file)) != OK)
128                return st;
129        }
130
131        if (setup->doc_type == FMT_XML)
132        {
133            if(fprintf(file, "%s%s%s\n", xml_contexts[XML_OPEN_FINISHER_BRACKET_INDEX],
       xml_contexts[XML_TRACKS_FLAG_INDEX], xml_contexts[XML_CLOSE_BRACKET_INDEX]) < 0)
134                return ERROR_WRITING_TO_FILE;
135        }
136
137        return OK;
138    }
139
140    /*Esta función establece un elemento*/
141    status_t ADT_Vector_set_element(ADT_Vector_t ** v, size_t position, void * new_element)
142    {
143        if(v==NULL)
144            return ERROR_NULL_POINTER;
145
146        if(position > (*v)->size)
147            return ERROR_OUT_OF_RANGE;
```

```c
148
149      if(position < 0)
150      {
151          (*v)->elements[(*v)->size + position] = new_element;
152          return OK;
153      }
154
155      (*v)->elements[position]=new_element;
156      return OK;
157  }
158
159  /*Esta función agrega un elemento a un vector*/
160  status_t ADT_Vector_append_element(ADT_Vector_t ** v, void * element)
161  {
162      size_t i;
163      void ** aux;
164
165      if(v == NULL || element == NULL)
166          return ERROR_NULL_POINTER;
167
168      i=(*v)->size;
169      if(i==(*v)->alloc_size)
170      {
171          if((aux = realloc((*v)->elements, ((*v)->alloc_size +
     ADT_VECTOR_CHOP_SIZE)*sizeof(void*))) == NULL)
172          {
173              return ERROR_OUT_OF_MEMORY;
174          }
175          (*v)->elements = aux;
176          (*v)->alloc_size += ADT_VECTOR_CHOP_SIZE;
177      }
178      (*v)->elements[i] = element;
179      ((*v)->size)++;
180
181      return OK;
182  }
183
184  /*Esta funcion intercambia ele lugar de dos elementos de un vector*/
185  status_t ADT_Vector_swap_elements (void ** element1, void ** element2)
186  {
187      void * aux;
188
189      if (element1 == NULL || element2 == NULL)
190          return ERROR_NULL_POINTER;
191
192      aux = *element1;
193      *element1 = *element2;
194      *element2 = aux;
195
196      return OK;
197  }
198
199  /*Esta función ordena los elementos de un vector*/
200  status_t  ADT_Vector_sort_elements (ADT_Vector_t ** vector, status_t (*elements_swapper)(void
     **, void **))
201  {
202      size_t i, j = 1;
203      status_t st;
204
205      if (vector == NULL)
206          return ERROR_NULL_POINTER;
207
208      while (j != 0)
209      {
210          j = 0;
211          for(i = 0; i < (*vector)->size - 1; i++)
212          {
213              if(((*vector)->comparator)((*vector)->elements[i], (*vector)->elements[i+1]) > 0)
214              {
215                  if ((st = elements_swapper(&((*vector)->elements[i]),
     &((*vector)->elements[i+1]))) != OK)
216                      return st;
217                  j++;
218              }
219          }
220      }
```

```
221
222        return OK;
223  }
```