06/27/18 07:38:55 /home/ivan/Desktop/Repositorio-Final/vector.c

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

#include "vector.h"

extern char * errors_dictionary[MAX_ERRORS];


/*Esta función crea un nuevo vector*/
status_t ADT_Vector_new(ADT_Vector_t ** v)
{
    size_t i;

    if (v == NULL) return ERROR_NULL_POINTER;

    if ((*v =(ADT_Vector_t*)malloc(sizeof(ADT_Vector_t))) == NULL)
        return ERROR_OUT_OF_MEMORY;

    if (((*v)->elements = (void**)malloc(INIT_CHOP*sizeof(void*))) == NULL)
    {
        free(*v);
        *v = NULL;
        return ERROR_OUT_OF_MEMORY;
    }

    for(i=0; i<INIT_CHOP; i++)
    {
        (*v)->elements[i] = NULL;
    }

    (*v)->alloc_size = INIT_CHOP;

    (*v)->size = 0;
    (*v)->destructor = NULL;
    (*v)->comparator = NULL;
    (*v)->printer = NULL;

    return OK;
}

/*Esta función destruye un vector*/
status_t ADT_Vector_delete (ADT_Vector_t ** v)
{
    status_t st;
    size_t i;
    for(i=0; i<(*v)->size; i++)
    {
        st = ((*v)->destructor)((*v)->elements[i]);
        if (st != OK)
            return st;
    }

    free((*v)->elements);
    (*v)->elements = NULL;
    free(*v);
    *v=NULL;
    return OK;
}

/*Esta función obtiene un elemento de un vector*/
void * ADT_Vector_get_element (ADT_Vector_t * v, int position)
{
    if (v == NULL) return NULL;

    if (position < 0) return v->elements[v->size + position];
    if (position > v->size) return NULL;

    return v->elements[position];
}

/*Esta función se fija si un vector está vacío*/
bool_t ADT_Vector_is_empty (ADT_Vector_t * p)
{
    return (p->size) ? FALSE:TRUE;
}

/*Esta función establece una función de impresión de vector*/
status_t ADT_Vector_set_printer(ADT_Vector_t * v, printer_t pf)
{
    if(v==NULL) return ERROR_NULL_POINTER;

    v->printer = pf;
    return OK;
}

/*Esta función establece una función que compara elementos*/
status_t ADT_Vector_set_comparator(ADT_Vector_t * v, comparator_t cf)
{
    if(v==NULL) return ERROR_NULL_POINTER;

    v->comparator = cf;
    return OK;
}

/*Esta función establece una función que destruye elementos*/
status_t ADT_Vector_set_destructor(ADT_Vector_t * v, destructor_t df)
{
    if(v==NULL) return ERROR_NULL_POINTER;

    v->destructor = df;
    return OK;
}

/*Esta función exporta un Vector*/
status_t ADT_Vector_export (ADT_Vector_t * v, const void * context, FILE * file, setup_t setup)
{
    size_t i;
    status_t st;
    char ** xml_contexts = NULL;

    if (v == NULL || file == NULL)
        return ERROR_NULL_POINTER;

    if (setup.doc_type == FMT_XML)
    {
        xml_contexts = (char **)context;

        if(fprintf(file, "%s\n", xml_contexts[0]) < 0)
            return ERROR_WRITING_TO_FILE;

        if(fprintf(file, "%s%s%s\n", xml_contexts[1], xml_contexts[4], xml_contexts[3]) < 0)
            return ERROR_WRITING_TO_FILE;
    }

    for (i = 0; i < v->size; i++)
    {
        if ((st = (v->printer)(v->elements[i], context, file)) != OK)
            return st;
    }

    if (setup.doc_type == FMT_XML)
    {
        if(fprintf(file, "%s%s%s\n", xml_contexts[2], xml_contexts[4], xml_contexts[3]) < 0)
            return ERROR_WRITING_TO_FILE;
    }

    return OK;
}

/*Esta función establece un elemento*/
status_t ADT_Vector_set_element(ADT_Vector_t ** v, size_t position, void * new_element)
{
    if(v==NULL)
        return ERROR_NULL_POINTER;

    if(position > (*v)->size)
        return ERROR_OUT_OF_RANGE;

    if(position < 0)
    {
        (*v)->elements[(*v)->size + position] = new_element;
        return OK;
    }

    (*v)->elements[position]=new_element;
    return OK;
}

/*Esta función agrega un elemento a un vector*/
status_t ADT_Vector_append_element(ADT_Vector_t ** v, void * element)
{
    size_t i;
    void ** aux;

    if(v == NULL || element == NULL)
        return ERROR_NULL_POINTER;

    i=(*v)->size;
    if(i==(*v)->alloc_size)
    {
        if((aux = realloc((*v)->elements, ((*v)->alloc_size + ADT_VECTOR_CHOP_SIZE)*sizeof(void*))) == NULL)
        {
            return ERROR_OUT_OF_MEMORY;
        }
        (*v)->elements = aux;
        (*v)->alloc_size += ADT_VECTOR_CHOP_SIZE;
    }
    (*v)->elements[i] = element;
    ((*v)->size)++;

    return OK;
}

/*Esta funcion intercambia ele lugar de dos elementos de un vector*/
status_t ADT_Vector_swap_elements (void ** element1, void ** element2)
{
    void * aux;

    if (element1 == NULL || element2 == NULL)
        return ERROR_NULL_POINTER;

    aux = *element1;
    *element1 = *element2;
    *element2 = aux;

    return OK;
}

/*Esta función ordena los elementos de un vector*/
status_t  ADT_Vector_sort_elements (ADT_Vector_t ** vector, status_t (*elements_swapper)(void **, void **))
{
    size_t i, j = 1;
    status_t st;

    if (vector == NULL)
        return ERROR_NULL_POINTER;

    while (j != 0)
    {
        j = 0;
        for(i = 0; i < (*vector)->size - 1; i++)
        {
            if(((*vector)->comparator)((*vector)->elements[i], (*vector)->elements[i+1]) > 0)
            {
                if ((st = elements_swapper(&((*vector)->elements[i]), &((*vector)->elements[i+1]))) != OK)
                    return st;
                j++;
            }
        }
    }

    return OK;
}
```