Universidad de Buenos Aires
Facultad De Ingeniería
Año 2019 - 1 er. cuatrimestre

Algoritmos y Programación I (95.11)
Curso 01 – Ing. Cardozo

Trabajo práctico n° 1
″Visualización de mensajes en formato NMEA″

Fecha de entrega: 26 de junio de 2019

Integrantes:

- Lareo, Matıas Federico - 97916
  < matilareo@gmail.com >
- Tedesco, Mauro Axel - 102958
  < mauroaxelt@gmail.com >
- Wawryczuk, Laureano Agustin - 102400
  < laureanow1997@hotmail.com >

INFORME

Introducción

En el presente informe se expone el desarrollo de un aplicativo de consola con comandos en línea de órdenes para interpretar mensajes de un GPS en formato "NMEA". Estos, luego de procesados, serán exportados en formato CSV o KML. Esta última opción podrá ser interpretada por un programa gráfico ("Google Maps" en este caso) que muestre la ruta tomada por el GPS.

Primero se indican las bases tomadas para el desarrollo del programa y una sencilla explicación de los procesos que este realiza. En conjunto, se exponen las dificultades encontradas en el transcurso del desarrollo. Seguido, se ilustran el diagrama de flujo y de funciones del aplicativo. Luego de ello se explicitan las conclusiones obtenidas a partir de la realización del trabajo.

Al finalizar el informe, se encuentran los códigos fuente utilizados junto con el script de compilación del programa.

Desarrollo

La puesta en marcha del proyecto no fue fácil, inicialmente se estudió qué procesos serían necesarios para confeccionar el programa. A continuación se dividieron las tareas para acelerar el desarrollo, pero todo se retrasó al momento se ensamblar todas las partes. Después de resolver los problemas más graves, el programa comenzó a funcionar y arrojó los primeros errores importantes a corregir.
Fueron de suma utilidad los apuntes tomados en clase para definir los tipos de dato abstracto y sus primitivas.

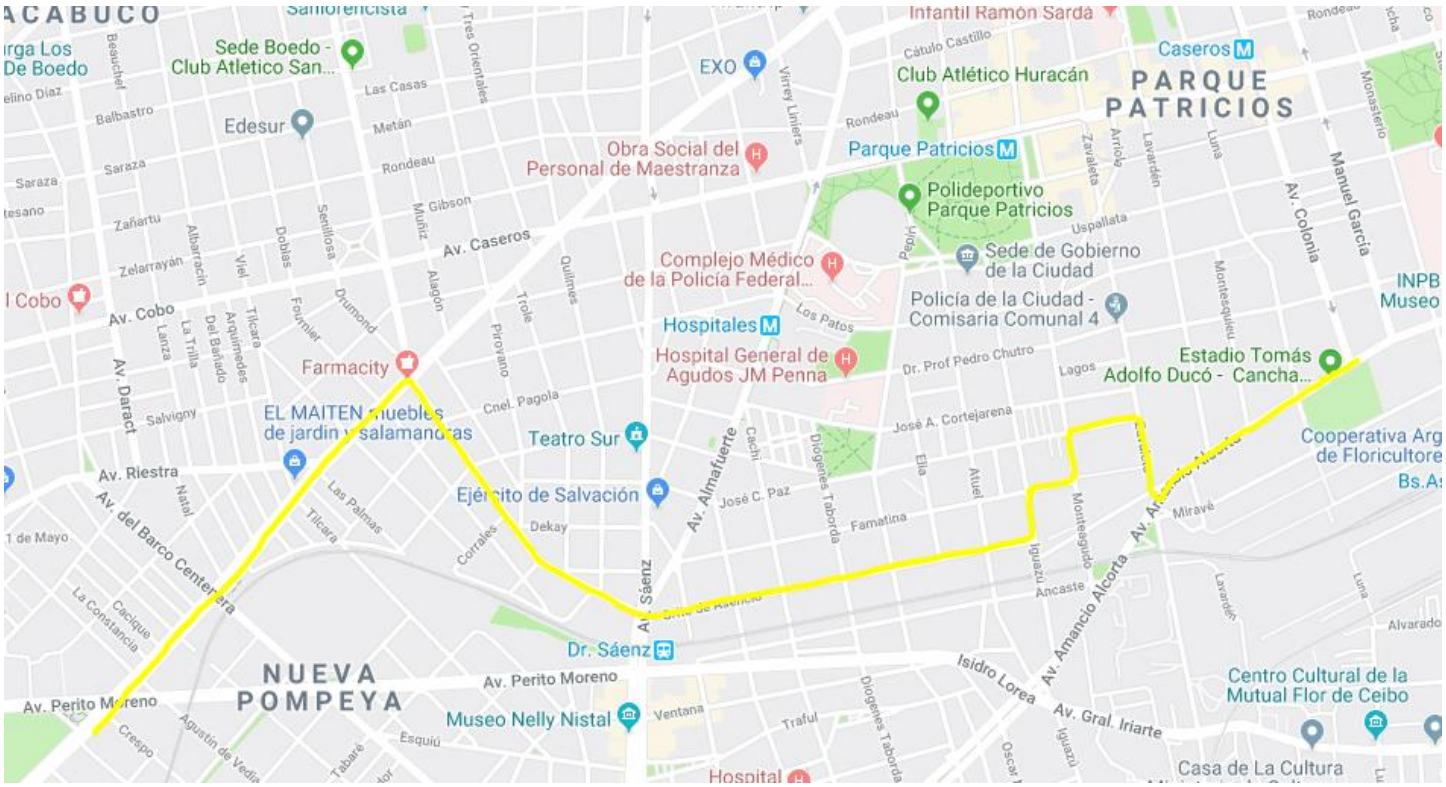
Para la ejecución del programa se supone que las rutas de los archivos de exportación (<salida>) y lectura (<entrada>) son válidos y que este último es de formato "NMEA". También, se determinó que la ejecución será de forma fija, es decir, no se debe alterar el orden de las banderas ni argumentos. Según las pautas impuestas, se suponen desordenados los registros obtenidos del archivo de lectura. Consecuentemente, se realizó un filtrado de registros, quedando solo los que contienen información de geolocalización. También, se realizó la verificación de todos los registro relevantes, para comprobar si fueron de alguna forma corrompidos. Posteriormente se ordenó cronológicamente la totalidad de los registros con el método de ordenamiento burbujeo (bubble sort). Cabe aclarar que los datos obtenidos del GPS fueron transformados para poder ser interpretados por el programa Google Maps.

Una vez realizados lo anterior y según lo indicado por comandos, se crea el archivo de exportación. Los formatos disponibles son CSV y KML.

Finalmente se subió el archivo KML al programa "Google Maps" y con él se obtuvo la trayectoria tomada por el GPS para corroborar el correcto funcionamiento del programa.

Lareo, Matıas Federico - 97916
Tedesco, Mauro Axel - 102958
Wawryczuk, Laureano Agustin - 102400

A continuación se ilustra una captura de pantalla de Google Maps con una ruta cargada, a modo de ejemplo, de un archivo KML generado con el programa.



Conclusión

Durante el desarrollo de los aplicativos se encontraron diversas dificultades que se superaron gracias a la bibliografía y los apuntes tomados en clase, siendo de suma utilidad para definir los tipos de dato abstracto y sus primitivas. Además, dado que se desconocía el tipo de archivo NMEA se debió buscar información acerca de este formato para poder trabajar con el archivo generado por el GPS. Lo mismo ocurrió para el archivo de salida KML.

A la hora de codificar, fragmentar el código y dividir los archivos por funcionalidad ayudó a facilitar la lectura del código, ya que este es extenso. gracias a esto, se detectaron rápidamente las fallas. En todos los códigos desarrollados para este trabajo práctico se tuvo en cuenta una serie de validaciones, las cuales logran que el programa sea robusto y funcione correctamente, aunque se pierda velocidad de ejecución al realizarlas.

Para finalizar, el trabajar con archivos estandarizados("NMEA") es de gran utilidad, ya que se puede reutilizar el código para nuevos proyectos que necesiten la funcionalidad de este programa.

SCRIPTS

Compilación

> make all

Ejecución

> ./gpsviewer -fmt <formato> -out <salida> <entrada>

CÓDIGOS FUENTE

MAKEFILE

```
CC = gcc

CFLAGS = -ansi -pedantic -Wall


all : gpsviewer clean


gpsviewer : main.o nmea_filter.o gps_processor.o ADT_vector.o ADT_GPS_record.o utils.o errors.o

	$(CC) $(CFLAGS) -o gpsviewer main.o nmea_filter.o gps_processor.o ADT_vector.o
ADT_GPS_record.o utils.o errors.o


main.o : main.c main.h

	$(CC) $(CFLAGS) -o main.o -c main.c


nmea_filter.o : nmea_filter.c nmea_filter.h

	$(CC) $(CFLAGS) -o nmea_filter.o -c nmea_filter.c


gps_processor.o : gps_processor.c gps_processor.h

	$(CC) $(CFLAGS) -o gps_processor.o -c gps_processor.c


ADT_GPS_record.o : ADT_GPS_record.c ADT_GPS_record.h

	$(CC) $(CFLAGS) -o ADT_GPS_record.o -c ADT_GPS_record.c


ADT_vector.o : ADT_vector.c ADT_vector.h

	$(CC) $(CFLAGS) -o ADT_vector.o -c ADT_vector.c


utils.o : utils.c utils.h

	$(CC) $(CFLAGS) -o utils.o -c utils.c
```

```
errors.o : errors.c errors.h
	$(CC) $(CFLAGS) -o errors.o -c errors.c


clean :
	rm *.o
```

MY_TYPES.H

```
#ifndef MY_TYPES__H
#define MY_TYPES__H

#include <stdio.h>

typedef char * string;
typedef unsigned char uchar;

typedef enum { TRUE, FALSE } bool_t;

typedef enum {FORMAT_CSV,FORMAT_KML} export_format_t;

typedef enum {
	OK,
	ERROR_NULL_POINTER,
	ERROR_FEW_ARGUMENTS,
	ERROR_TOO_MANY_ARGUMENTS,
	ERROR_INVOCATION,
```

```
        ERROR_MEMORY,
        ERROR_FORMAT_FILE,
        ERROR_OPEN_INPUT_FILE,
        ERROR_OPEN_OUTPUT_FILE,
        ERROR_OUT_OF_MEMORY,
        ERROR_CORRUPT_GPS_RECORD
}status_t;

typedef status_t (* destructor_t)(void *);
typedef status_t (* exporter_t)(const void *,const void *,FILE *);

#endif
```

ERRORS.H

```
#ifndef ERRORS__H
#define ERRORS__H

#include <stdio.h>
#include "my_types.h"

#define MAX_ERRORS 11

/*-------------------------------------------------PROTOTIPOS-----------------------------------------------------*/
status_t show_error(status_t st);

#endif
```

ERRORS.C

```
#include <stdio.h>
#include "errors.h"
#include "my_types.h"

char * errors[MAX_ERRORS]={
        "EJECUCIÓN EXITOSA.",
        "PUNTERO NULO.",
        "POCOS ARGUMENTOS.",
        "MUCHOS ARGUMENTOS.",
        "FALLA DE INVOCACIÓN.",
        "NO HAY ESPACIO EN DISCO.",
        "FORMATO INCORRECTO.",
        "NO SE PUEDE ABRIR ARCHIVO ENTRADA.",
        "NO SE PUEDE ABRIR ARCHIVO SALIDA.",
        "FALLA DE MEMORIA.",
        "REGISTRO CORRUPTO."
};

/*-------------------------------------------------IMPRIMIR ERRORES-------------------------------------------------*/
status_t show_error(status_t st)
{
        fprintf(stderr,"%s\n\n",errors[st]);
        return OK;
}
```

MAIN.H

```
#ifndef MAIN__H
#define MAIN__H

#include <stdio.h>
#include "my_types.h"

#define MAX_CMD_ARGS                6           /*CANTIDAD MAXIMA DE ARGUMENTOS*/
#define CMD_FLAG_FMT_POS            1           /*ORDEN DE LA OPCIÓN FORMATO*/
#define CMD_FMT_OPTION_POS          2           /*ORDEN DEL FORMATO ELEGIDO*/
#define CMD_FLAG_OUT_POS            3           /*ORDEN DE LA OPCIÓN OUT*/
#define CMD_OUT_FILE_ROUTE          4           /*ORDEN DE LA RUTA DE SALIDA*/
#define CMD_IN_FILE_ROUTE           5           /*ORDEN DE LA RUTA DE ENTRADA*/
#define CMD_FLAG_FMT_DES            "-fmt"  /*DESCRIPCIÓN DE LA OPCION FORMATO*/
#define CMD_FLAG_OUT_DES            "-out"  /*DESCRIPCIÓN DE LA OPCION OUT*/

/*----------------------------------------------------PROTOTIPOS----------------------------------------------------*/
status_t validar_args( size_t l_arg, char * argv[] );

#endif
```

MAIN.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "my_types.h"
```

```
#include "main.h"
#include "gps_processor.h"
#include "errors.h"

int main ( int argc, char * argv[] )
{
	status_t st;
	FILE *input_file, *output_file;

/*-----------------------------------------VALIDAR ARGUMENTOS-----------------------------------------*/
	if ( ( st = validar_args( argc, argv ) ) != OK )
	{
		show_error( st );
		return st;
	}

/*-----------------------------------------ABRIR ARCHIVOS-----------------------------------------------*/
	if ( ( input_file = fopen(argv[CMD_IN_FILE_ROUTE], "rt" ) ) == NULL )
	{
		show_error( ERROR_OPEN_INPUT_FILE );
		return ERROR_OPEN_INPUT_FILE;
	}

	if ( ( output_file = fopen( argv[CMD_OUT_FILE_ROUTE], "wt" ) ) == NULL )
	{
		fclose( input_file );
		show_error( ERROR_OPEN_OUTPUT_FILE );
		return ERROR_OPEN_OUTPUT_FILE;
```

```
	}

/*--------------------------------PROCESAR REGISTROS DE GPS---------------------------------------*/
	if ( ( st = process_gps_records( input_file, output_file, argv[CMD_FMT_OPTION_POS] ) ) !=
OK )
	{
		show_error( st );
		return st;
	}

/*-----------------------------------------CERRAR ARCHIVOS-------------------------------------------*/
	fclose( input_file );
	if ( fclose( output_file ) == EOF )
	{
		show_error( ERROR_MEMORY );
		return ERROR_MEMORY;
	}

	return OK;
}


/*-----------------------------------------VALIDAR ARGUMENTOS-----------------------------------------*/
status_t validar_args( size_t l_arg, char * argv[] )
{
	if ( argv == NULL )
		return ERROR_NULL_POINTER;

	if ( l_arg < MAX_CMD_ARGS )
```

```
        return ERROR_FEW_ARGUMENTS;


    if ( l_arg > MAX_CMD_ARGS )
        return ERROR_TOO_MANY_ARGUMENTS;


    if ( ( strcmp( argv[CMD_FLAG_FMT_POS], CMD_FLAG_FMT_DES ) ) || ( strcmp(
argv[CMD_FLAG_OUT_POS], CMD_FLAG_OUT_DES ) ) )
        return ERROR_INVOCATION;


    if ( ( strcmp( argv[CMD_FMT_OPTION_POS], FMT_OPTION_KML ) ) && ( strcmp(
argv[CMD_FMT_OPTION_POS], FMT_OPTION_CSV ) ) )
        return ERROR_FORMAT_FILE;


    return OK;
}
```

ADT_VECTOR.H

```
#ifndef ADT_VECTOR__H
#define ADT_VECTOR__H

#include <stdio.h>
#include "my_types.h"

#define INIT_CHOP 5

#define KML_EXPORT_1 "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<kml
xmlns=\"http://www.opengis.net/kml/2.2\">\n\t<Document>\n\t\t<name>Rutas</name>\n\t\t<descriptio
n>Ejemplos de rutas</description>\n\t\t<Style
id=\"yellowLineGreenPoly\">\n\t\t\t<LineStyle>\n\t\t\t\t<color>7f00ffff</color>\n\t\t\t\t<width>4</width>\
n\t\t\t</LineStyle>\n\t\t\t<PolyStyle>\n\t\t\t\t<color>7f00ffff</color>\n\t\t\t</PolyStyle>\n\t\t</Style>\n"
```

```
#define KML_EXPORT_2 "\t\t<Placemark>\n\t\t\t<name>Relieve
absoluto</name>\n\t\t\t<description>Pared verde transparente con contornos
amarillos</description>\n\t\t\t<styleUrl>#yellowLineGreenPoly</styleUrl>\n\t\t\t<LineString>\n\t\t\t\t<ex
trude>1</extrude>\n\t\t\t\t<tessellate>1</tessellate>\n\t\t\t\t<altitudeMode>absolute</altitudeMode>\n\
t\t\t\t<coordinates>"


#define KML_EXPORT_3
"\t\t\t\t</coordinates>\n\t\t\t</LineString>\n\t\t</Placemark>\n\t</Document>\n</kml>"


typedef struct  {
        void ** elements;
        size_t size;
        destructor_t destructor;
        exporter_t exporter;
} ADT_vector_t;

/*-----------------------------------------------------PRIMITIVAS----------------------------------------------*/

status_t ADT_vector_create(ADT_vector_t ** p);
status_t ADT_vector_destroy(ADT_vector_t **p);
status_t ADT_vector_append(ADT_vector_t * p, void * n);
status_t ADT_vector_set_destructor(ADT_vector_t *p,destructor_t destructor);
status_t ADT_vector_set_exporter(ADT_vector_t *p,exporter_t exporter);
status_t ADT_vector_export_as_csv(const void * p, const void* delim, FILE * fo);
status_t ADT_vector_export_as_kml(const void * p, const void* delim, FILE * fo);
status_t ADT_vector_sort (ADT_vector_t **p);

#endif
```

ADT_VECTOR.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "my_types.h"
#include "ADT_vector.h"
#include "ADT_GPS_record.h"
#include "utils.h"

/*-----------------------------------------CREAR ADT_VECTOR-----------------------------------------*/
status_t ADT_vector_create ( ADT_vector_t ** p )
{
   if( p == NULL )
      return ERROR_NULL_POINTER;

   if( ( *p = (ADT_vector_t*) malloc( sizeof(ADT_vector_t) ) ) == NULL )
      return ERROR_MEMORY;

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

   (*p)->size = 0;
   (*p)->destructor = NULL;
   (*p)->exporter = NULL;
```

```
    return OK;
}

/*-----------------------------------------DESTRUIR ADT_VECTOR-----------------------------------------*/
status_t ADT_vector_destroy ( ADT_vector_t **p )
{
    size_t i;
    status_t st;

        if( p == NULL )
                return ERROR_NULL_POINTER;

        for( i=0 ; i < (*p)->size ; i++ )
    {
                if( ( st = ( *( (*p)->destructor ) ) ( (*p)->elements[i] ) ) != OK )
                        return st;

                (*p)->elements[i] = NULL;
        }
    free( (*p)->elements );
        (*p)->elements = NULL;
    free(*p);
    *p = NULL;
    return OK;
}

/*-------------------AGREGAR ELEMENTO AL FINAL DE ADT_VECTOR---------------------------------*/
```

```
status_t ADT_vector_append ( ADT_vector_t * p, void * n )
{
	void ** aux;

	if( p==NULL || n==NULL )
		return ERROR_NULL_POINTER;

	if( (aux = (void **) realloc(p->elements , (p->size+1) * sizeof(void*) ) ) == NULL )

		return ERROR_MEMORY;

	p->elements = aux;
	p->elements[p->size] = n;
	(p->size)++;
	return OK;
}

/*-------------------------------SETEAR DESTRUCTOR EN ADT_VECTOR---------------------------------*/
status_t ADT_vector_set_destructor ( ADT_vector_t *p, destructor_t destructor )
{
	if( p == NULL )
		return ERROR_NULL_POINTER;

	p->destructor = destructor;

	return OK;
}
```

```
/*--------------------------------SETEAR EXPORTADOR EN ADT_VECTOR---------------------------------*/

status_t ADT_vector_set_exporter ( ADT_vector_t *p, exporter_t exporter )

{

	if( p == NULL )

		return ERROR_NULL_POINTER;


	p->exporter = exporter;


	return OK;

}


/*-------------------EXPORTAR ELEMENTOS DE ADT_VECTOR A CSV---------------------------------*/

status_t ADT_vector_export_as_csv ( const void * p, const void * delim, FILE * fo )

{

	size_t i;

	status_t st;


	if( p == NULL )

		return ERROR_NULL_POINTER;


	for(i=0 ; i < ( (ADT_vector_t*) p )->size ; i++ )

		if( ( st = ( *( ( (ADT_vector_t*) p )->exporter ) ) ( (void*) ( ( (ADT_vector_t*) p )->elements[i] ), delim, fo) ) != OK )

			return st;


	return OK;

}


/*-------------------EXPORTAR ELEMENTOS DE ADT_VECTOR A KML---------------------------------*/
```

```
status_t ADT_vector_export_as_kml ( const void * p, const void * delim, FILE * fo )
{
	size_t i;
	status_t st;

	if( p == NULL )
		return ERROR_NULL_POINTER;

	fprintf(fo,"%s", KML_EXPORT_1);
	fprintf(fo,"%s", KML_EXPORT_2);

	for(i=0 ; i < ( (ADT_vector_t*) p )->size ; i++ )
		if( ( st = ( *( ( (ADT_vector_t*) p )->exporter ) ) ( (void*) ( ( (ADT_vector_t*) p )->elements[i] ), delim, fo ) ) != OK )
			return st;

	fprintf( fo, "%s\n", KML_EXPORT_3);
	return OK;
}

/*--------------------------------ORDENAR ELEMENTOS EN ADT_VECTOR---------------------------------*/
status_t ADT_vector_sort ( ADT_vector_t **p )
{
	size_t i, j;
	bool_t sorted;

	if( p == NULL )
		return ERROR_NULL_POINTER;
```

```
	for( i = 0 ; i < (*p)->size ; i++ )
	{
		sorted = TRUE;
		for( j = 0; j < ( (*p)->size - 1 - i ) ; j++ )
		{
			if( ( (ADT_GPS_record_t*) ( (ADT_vector_t*) *p )->elements[j] )->time > ( (ADT_GPS_record_t*) ( (ADT_vector_t*) *p )->elements[j+1] )->time )
			{
				swap( &( (*p)->elements[j] ) , &( (*p)->elements[j+1] ) );
				sorted = FALSE;
			}
		}
		if( sorted == TRUE )
			return OK;
	}
	return OK;
}
```

ADT_GPS_RECORD.H

```
#ifndef ADT_GPS_RECORD__H
#define ADT_GPS_RECORD__H

#include <stdio.h>
#include "my_types.h"

typedef struct  {
	float time;
	float latitude;
	float longitude;
```

```
	float height;
} ADT_GPS_record_t;

/*-----------------------------------------------PRIMITIVAS-----------------------------------------------------------*/
status_t ADT_GPS_record_create(ADT_GPS_record_t ** gps_record);
status_t ADT_GPS_record_delete(ADT_GPS_record_t ** gps_record);
status_t ADT_GPS_record_set_coord(ADT_GPS_record_t * gps_record, double time, double latitude, double longitude, double height);
status_t ADT_GPS_record_export_as_csv(const void * gps_record, const void * delim, FILE * fo);
status_t ADT_GPS_record_export_as_kml(const void * gps_record, const void * delim, FILE * fo);


#endif
```

ADT_GPS_RECORD.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "my_types.h"
#include "ADT_GPS_record.h"
#include <time.h>

/*-------------------------------------CREAR ADT_GPS_RECORD-------------------------------------------------*/
status_t ADT_GPS_record_create(ADT_GPS_record_t ** p)
{
	if(p==NULL)
		return ERROR_NULL_POINTER;

	if((*p=(ADT_GPS_record_t *)calloc(1,sizeof(ADT_GPS_record_t)))==NULL)
```

```
		return ERROR_MEMORY;


	return OK;
}

/*----------------------------------------ELIMINAR ADT_GPS_RECORD-----------------------------------*/
status_t ADT_GPS_record_delete(ADT_GPS_record_t ** gps_record){
	if(gps_record==NULL)
		return ERROR_NULL_POINTER;


	free(*gps_record);
	*gps_record=NULL;
	return OK;
}

/*------------------SETEAR COORDENADAS EN ADT_GPS_RECORD---------------------------------*/
status_t ADT_GPS_record_set_coord(ADT_GPS_record_t * gps_record, double time, double
latitude, double longitude, double height)
{
	if(gps_record==NULL)
		return ERROR_NULL_POINTER;


	gps_record->time=time;
	gps_record->latitude=latitude;
	gps_record->longitude=longitude;
	gps_record->height=height;
	return OK;
}
```

```
/*----------------------------EXPORTAR ADT_GPS_RECORD COMO CSV---------------------------------*/

status_t ADT_GPS_record_export_as_csv(const void * gps_record, const void * delim, FILE * fo)

{

	time_t t;

	struct tm *tm;


	if(gps_record==NULL || fo==NULL)

		return ERROR_NULL_POINTER;


	t=time(NULL);

	tm=localtime(&t);


	fprintf(fo,"%02d%02d%02d%.0f%c%.14f%c%.14f%c%.1f\n", 1900+tm->tm_year, 1+tm-
>tm_mon, tm->tm_mday,


	((ADT_GPS_record_t*)gps_record)->time,*((char*)delim),


	((ADT_GPS_record_t*)gps_record)->latitude,*((char*)delim),


	((ADT_GPS_record_t*)gps_record)->longitude,*((char*)delim),


	((ADT_GPS_record_t*)gps_record)->height);

	return OK;

}


/*-------------------------EXPORTAR ADT_GPS_RECORD COMO KML---------------------------------*/

status_t ADT_GPS_record_export_as_kml(const void * gps_record, const void * delim, FILE * fo)

{

	if(gps_record==NULL || fo==NULL)

		return ERROR_NULL_POINTER;
```

```
        fprintf(fo,"%.14f%c%.14f%c%.0f\n",      ((ADT_GPS_record_t*)gps_record)-
>longitude,*((char*)delim),

        ((ADT_GPS_record_t*)gps_record)->latitude,*((char*)delim),

        ((ADT_GPS_record_t*)gps_record)->height);
        return OK;
}
```

GPS_PROCESSOR.H

```
#ifndef GPS_PROCESSOR__H
#define GPS_PROCESSOR__H

#include <stdio.h>
#include "my_types.h"
#include "ADT_vector.h"
#include "ADT_GPS_record.h"
#include "utils.h"


#define GPS_RECORD_LENGTH                       72

#define MAX_SIZE_COORDINATES            9
#define GPS_RECORD_TIME_POS             1
#define GPS_RECORD_LATITUDE_POS         2
#define GPS_RECORD_LONGITUDE_POS        4
#define GPS_RECORD_N_S_POS              3
#define GPS_RECORD_E_W_POS              5
```

```
#define GPS_RECORD_HEIGHT_POS                   9
#define INDICATOR_CHAR_SOUTH                    'S'
#define INDICATOR_CHAR_WEST                     'W'
#define COORD_LATITUDE_DEGREES_POS                      0
#define COORD_LATITUDE_DEGREES_LENGTH           2
#define COORD_LATITUDE_MINUTES_POS              2
#define COORD_LATITUDE_MINUTES_LENGTH           9
#define COORD_LONGITUDE_DEGREES_POS             0
#define COORD_LONGITUDE_DEGREES_LENGTH          3
#define COORD_LONGITUDE_MINUTES_POS             3
#define COORD_LONGITUDE_MINUTES_LENGTH          9

#define MAX_FORMATS                             2
#define CSV_DELIM_CHAR                          '|'
#define KML_DELIM_CHAR                          ','
#define FMT_OPTION_CSV                          "csv"   /*FORMATO DE SALIDA*/
#define FMT_OPTION_KML                          "kml"   /*FORMATO DE SALIDA*/

/*-----------------------------------------------PROTOTIPOS-----------------------------------------------------*/
status_t process_gps_records( FILE * fi, FILE * fo, string fmt );
status_t load_gps_records( FILE *input_file, ADT_vector_t ** records );
status_t export_gps_records( ADT_vector_t * records, FILE * fo, export_format_t export_format );
status_t build_gps_record( string line,ADT_GPS_record_t ** gps_record );
status_t get_gps_record_coordinates( char ** str_array, double *time, double * latitude,double *
longitude,double * height );
export_format_t set_format( string fmt );

#endif
```

GPS_PROCESSOR.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "my_types.h"
#include "gps_processor.h"
#include "ADT_vector.h"
#include "ADT_GPS_record.h"
#include "utils.h"
#include "nmea_filter.h"

/*-----------------------------------PROCESAR REGISTROS DE GPS-----------------------------------*/
status_t process_gps_records ( FILE * fi, FILE * fo, string fmt )
{
	ADT_vector_t * records;
	status_t st;
	export_format_t export_format;

	if( fi == NULL )
		return ERROR_NULL_POINTER;

	if( ( st = load_gps_records( fi, &records ) ) != OK )
		return st;

	if( ( st = ADT_vector_sort( &records ) ) != OK )
		return st;

	export_format = set_format( fmt );
```

```
	if( ( st = export_gps_records( records, fo, export_format ) ) != OK )
		return st;

	return OK;
}

/*-------------------------------------CARGAR REGISTROS DE GPS-----------------------------------------*/
status_t load_gps_records( FILE * fi, ADT_vector_t ** records )
{
	status_t st;
	ADT_GPS_record_t *gps_record;
	bool_t eof;
	string line;


	if ( fi == NULL || records == NULL )
		return ERROR_NULL_POINTER;

	if( ( st = ADT_vector_create( records ) ) != OK)
		return st;

	if ( ( st = ADT_vector_set_destructor( *records, (destructor_t)ADT_GPS_record_delete ) ) !=
OK )
		return st;


	eof = FALSE;
	while ( eof != TRUE )
```

```
	{
		if ( (st = read_line( fi, &line, &eof ) ) != OK )
			return st;

		if ( ( st = nmea_filter( line ) ) != OK )
			continue;

		if ( ( st = build_gps_record( line, &gps_record ) ) != OK )
		{
			ADT_vector_destroy( records );
			return st;
		}
		if ( ( st = ADT_vector_append( *records, (void*)gps_record ) ) != OK )
		{
			ADT_vector_destroy( records );
			return st;
		}
	}
	return OK;
}

/*-------------------------------------CONSTRUIR REGISTRO DE GPS-------------------------------------*/
status_t build_gps_record ( string line, ADT_GPS_record_t ** gps_record )
{
	status_t st;
	string * str_array;
	size_t array_len;
	double time, latitude, longitude, height;
```

```
	if( gps_record == NULL || line == NULL )
		return ERROR_NULL_POINTER;

/*-------------------------------------------------PARSEAR CADENA-------------------------------------------------*/
	if ( ( st = split( line, &array_len, &str_array ) ) != OK )
		return st;

/*-----------------------------------------CREAR GPS RECORD-----------------------------------------------------*/
	if ( ( st = ADT_GPS_record_create( gps_record ) ) != OK )
		return st;

/*---------------------------------------OBTENER COORDENADAS----------------------------------------------------*/
	if ( ( st = get_gps_record_coordinates( str_array, &time, &latitude, &longitude, &height ) ) !=
OK )
		return st;

/*-------------------------------------CARGAR COORD A GPS RECORD------------------------------------------*/
	if ( ( st = ADT_GPS_record_set_coord( *gps_record, time, latitude, longitude, height ) ) != OK )
		return st;

	return OK;
}

/*-------------OBTENER HORA Y INFO GEOLOCALIZACIÓN DE REGISTRO GPS--------------------*/
status_t get_gps_record_coordinates( string *str_array, double *time, double *latitude, double
*longitude, double *height )
{
```

```
	char aux_string[MAX_SIZE_COORDINATES+1], * temp, deg;
	double min;

	if ( str_array == NULL || latitude == NULL || longitude == NULL || height == NULL )
		return ERROR_NULL_POINTER;

/*-----------------------------------------OBTENER HORA-----------------------------------------------*/
	if ( str_array[GPS_RECORD_TIME_POS] )
	{
		*time = strtod( str_array[GPS_RECORD_TIME_POS], &temp );

		if( *temp )
			return ERROR_CORRUPT_GPS_RECORD;
	}

/*-----------------------------------------OBTENER LATITUD-----------------------------------------------*/
	if ( str_array[GPS_RECORD_LATITUDE_POS] )
	{
		if ( memcpy( aux_string, str_array[GPS_RECORD_LATITUDE_POS] +
COORD_LATITUDE_DEGREES_POS, COORD_LATITUDE_DEGREES_LENGTH * sizeof( char ) )
== NULL )
			return ERROR_OUT_OF_MEMORY;

		aux_string[COORD_LATITUDE_DEGREES_LENGTH] = '\0';
		deg = (char) strtol( aux_string, &temp, 10 );

		if ( *temp )
			return ERROR_CORRUPT_GPS_RECORD;
```

```
		if ( memcpy( aux_string, str_array[GPS_RECORD_LATITUDE_POS] +
COORD_LATITUDE_MINUTES_POS, COORD_LATITUDE_MINUTES_LENGTH * sizeof(char) ) ==
NULL )
			return ERROR_OUT_OF_MEMORY;

		aux_string[COORD_LATITUDE_MINUTES_LENGTH] = '\0';
		min = strtod( aux_string, &temp );

		if ( *temp )
		return ERROR_CORRUPT_GPS_RECORD;

		*latitude = ( double ) deg + ( min / 60 );

		if ( str_array[GPS_RECORD_N_S_POS] && str_array[GPS_RECORD_N_S_POS][0]
== INDICATOR_CHAR_SOUTH )
			*latitude = -( *latitude );
	}

/*-----------------------------------------OBTENER LONGITUD-----------------------------------------------*/
	if(str_array[GPS_RECORD_LONGITUDE_POS])
	{
		if ( memcpy( aux_string, str_array[GPS_RECORD_LONGITUDE_POS] +
COORD_LONGITUDE_DEGREES_POS, COORD_LONGITUDE_DEGREES_LENGTH * sizeof(
char ) ) == NULL )
			return ERROR_OUT_OF_MEMORY;

		aux_string[COORD_LONGITUDE_DEGREES_LENGTH] = '\0';
		deg = (char) strtol( aux_string, &temp, 10 );

		if ( *temp )
```

```
                return ERROR_CORRUPT_GPS_RECORD;


        if ( memcpy( aux_string, str_array[GPS_RECORD_LONGITUDE_POS] +
COORD_LONGITUDE_MINUTES_POS, COORD_LONGITUDE_MINUTES_LENGTH * sizeof( char
) ) == NULL )
                return ERROR_OUT_OF_MEMORY;


        aux_string[COORD_LONGITUDE_MINUTES_LENGTH] = '\0';
        min = strtod( aux_string, &temp );


        if ( *temp )
                return ERROR_CORRUPT_GPS_RECORD;


        *longitude = ( double ) deg + ( min / 60 );


        if ( str_array[GPS_RECORD_E_W_POS] && str_array[GPS_RECORD_E_W_POS][0]
== INDICATOR_CHAR_WEST )
                *longitude = -(*longitude);
    }


/*-------------------------------------------------OBTENER ALTURA-------------------------------------------------*/
    if ( str_array[GPS_RECORD_HEIGHT_POS] )
    {
        *height = strtod( str_array[GPS_RECORD_HEIGHT_POS], &temp );


        if ( *temp )
                return ERROR_CORRUPT_GPS_RECORD;
    }


    return OK;
```

```
}

/*----------------------------------SETEAR FORMATO DE EXPORTACIÓN----------------------------------*/

export_format_t set_format( string fmt )

{

	if ( !strcmp( fmt, FMT_OPTION_CSV ) )

		return FORMAT_CSV;


	else

		return FORMAT_KML;

}


/*-------------------------------------------EXPORTAR REGISTROS-------------------------------------------*/

status_t export_gps_records( ADT_vector_t * records, FILE * fo, export_format_t export_format )

{

	status_t st;

	exporter_t exporter;

	exporter_t gps_record_exporter;

	char delim[MAX_FORMATS] = { CSV_DELIM_CHAR, KML_DELIM_CHAR };


	if ( records == NULL || fo == NULL )

		return ERROR_NULL_POINTER;


	if ( export_format == FORMAT_CSV )  /*ESTO ES LO QUE VOS PASAS POR LINEA DE ORDENES*/

	{

		exporter = (exporter_t)ADT_vector_export_as_csv;

		gps_record_exporter = (exporter_t)ADT_GPS_record_export_as_csv;
```

```
            if ( ( st = ADT_vector_set_exporter( records, (exporter_t)gps_record_exporter ) ) != OK
)
                return st;

            if ( ( st = exporter( (void*) records, (void*) ( &delim[FORMAT_CSV]), fo ) ) != OK )
                return st;

            return OK;
        }
        else /*ESTE ES EL CASO DE KML*/
        {
            exporter = (exporter_t)ADT_vector_export_as_kml;
            gps_record_exporter = (exporter_t)ADT_GPS_record_export_as_kml;

            if ( ( st = ADT_vector_set_exporter( records, (exporter_t)gps_record_exporter ) ) != OK
)
                return st;

            if ( ( st = exporter( ( void * ) records, ( void * ) ( &delim[FORMAT_KML] ), fo ) ) != OK )
                return st;

            return OK;
        }
}
```

NMEA_FILTER.H

```
#ifndef NMEA_FILTER__H
#define NMEA_FILTER__H
```

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "my_types.h"


#define ID_MENSAJE                "$GPGGA"          /*COMIENZO DE LINEAS DE
INTERES*/

#define L_MIN_LINEA               72          /*LARGO MINIMO PARA CADA LINEA UTIL
(CAMPOS COMPLETOS)*/


#define MASK_SUM_VER              0x80        /*MASCARA PARA VERIFICACIÓN DE
SUMA*/



/*-----------------------------------------PROTOTIPOS-------------------------------------------------------*/

status_t nmea_filter (string line);

status_t check_record_sum (char * cadena);


#endif
```

NMEA_FILTER.C

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "my_types.h"

#include "nmea_filter.h"

#include "utils.h"
```

```
/*---------------------------------FILTRAR REGISTRO DE ARCHIVO NMEA-------------------------------*/
status_t nmea_filter( string line )
{
	status_t st;

/*--------------------SALTEAR LOS QUE NO COMIENCEN CON $GPGGA----------------------------*/
		if ( ( strncmp( ID_MENSAJE, line, 5 ) ) )
			return ERROR_CORRUPT_GPS_RECORD;

/*----------------SALTEAR LOS QUE NO TENGAN TODOS LOS DATOS-------------------------------*/
		if ( strlen( line ) < L_MIN_LINEA )
			return ERROR_CORRUPT_GPS_RECORD;

/*----------------------------------------SUMA DE VERIFICACIÓN-----------------------------------------*/
		if ( ( st = check_record_sum( line ) ) != OK )
			return ERROR_CORRUPT_GPS_RECORD;

	return OK;
}

/*----------COMPROBAR SUMA DE VERIFICACIÓN DE REGISTRO DE ARCHIVO NMEA-------------*/
status_t check_record_sum( char * cadena )
{
	uchar suma = 0x00, verif;
	size_t i, largo;
	char veri[2], *aux;

	if( cadena == NULL )
```

```
        return ERROR_NULL_POINTER;

    for ( i=1 ; cadena[i] != '*' ; i++ )
        suma = suma ^ (cadena[i] & (~MASK_SUM_VER));

    largo = strlen( cadena );
    veri[0] = cadena[largo-3];
    veri[1] = cadena[largo-2];
    verif = strtol( veri, &aux, 16 );

    if( suma != verif )
        return ERROR_CORRUPT_GPS_RECORD;

    return OK;
}
```

UTILS.H

```
#ifndef UTILS__H
#define UTILS__H

#include <stdio.h>
#include "my_types.h"

#define LF                      '\n'
#define NUL                     '\0'
#define INIT_CHOP_STR           72
#define CHOP_SIZE               2
#define GPS_RECORD_DELIM_STRING ","
```

```
#define GPS_RECORD_DELIM_CHAR           ','

/*-----------------------------------------------PROTOTIPOS-----------------------------------------------*/
status_t read_line( FILE * archivo_entrada, string * str, bool_t * eof );
status_t split( const string s, size_t *l, string **fields );
void destroy_strings( string **fields, size_t l );
char * strdupl( string cadena );
status_t swap( void ** a, void ** b );

#endif
```

UTILS.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "my_types.h"
#include "utils.h"

/*---------------------------------------LEER LÍNEA---------------------------------------------------*/
status_t read_line( FILE * fi, string * s, bool_t * eof )
{
        int c;
        size_t alloc_size;
        size_t used_size;
        char * aux;

        if ( fi == NULL || s == NULL || eof == NULL )
                return ERROR_NULL_POINTER;
```

```
	if ( ( *s = malloc(INIT_CHOP_STR * sizeof(char) ) ) == NULL )
		return ERROR_MEMORY;
	alloc_size = INIT_CHOP_STR;
	used_size = 0;

	while ( ( c = fgetc(fi)) != LF && c != EOF )
	{
		if ( used_size == alloc_size - 1 )
		{
			if ( ( aux = realloc( *s, ( alloc_size + CHOP_SIZE ) * sizeof( char ) ) ) == NULL )
			{
				free( *s );
				*s = NULL;
				return ERROR_MEMORY;
			}
			alloc_size += CHOP_SIZE;
			*s = aux;
		}
		(*s)[used_size++] = c;
	}
	(*s)[used_size] = NUL;
	*eof = ( c == EOF ) ? TRUE : FALSE;

	return OK;
}

/*----------------------------------------PARSEAR CADENA-----------------------------------------------*/
```

```
status_t split ( const string s, size_t *l, string **fields )
{
	string str, q, p;
	size_t i;

	if ( s == NULL || l == NULL || fields == NULL )
		return ERROR_NULL_POINTER;



	if ( ( str = strdupl( s ) ) == NULL ){
		*l = 0;
		return ERROR_MEMORY;
	}

	for ( i = 0, *l = 0 ; str[i] ; i++ )
		if ( str[i] == GPS_RECORD_DELIM_CHAR )
			(*l)++;
	(*l)++;

	if((*fields = (string **)malloc((*l)*sizeof(string *))) == NULL){
		free(str);
		*l = 0;
		fprintf(stderr, "%s\n", "no hay memoria");
		return ERROR_MEMORY;
	}

	for ( i = 0, q=str ; ( p = strtok( q, GPS_RECORD_DELIM_STRING ) ) != NULL ; q = NULL, i++)
	{
```

```
            if ( ( ( *fields ) [i] = strdupl( p ) ) == NULL )
            {
                free( str );
                destroy_strings( fields, *l );
                *l = 0;
                return ERROR_MEMORY;
            }
        }

        free( str );

        return OK;
}

/*----------------------------------DESTRUIR CADENAS DEL PARSEO-----------------------------------*/
void destroy_strings( string ** fields, size_t l )
{
        size_t i;
        for ( i = 0 ; i < l ; i++ )
        {
            free( ( *fields ) [i] );
            ( *fields ) [i] = NULL;
        }
        free( *fields );
        *fields = NULL;
}

/*-------------------------------------------DUPLICAR CADENAS-------------------------------------------*/
```

```
char * strdupl( string cadena ){
	string copia;
	if ( ( copia = (string) malloc( ( strlen( cadena ) + 1 ) * sizeof( char ) ) ) == NULL )
		return NULL;
	strcpy( copia, cadena );

	return copia;
}

/*-----------------------------------INTERCAMBIAR POSICIONES-----------------------------------*/
status_t swap( void ** a, void ** b )
{
	void * aux;

	if( a == NULL || b == NULL )
		return ERROR_NULL_POINTER;

	aux = *a;
	*a = *b;
	*b = aux;

	return OK;
}
```

Bibliografía

Google developer.*KML Reference*. https://developers.google.com/kml/documentation/kmlreference?hl=es.

Glenn Baddeley. *GPS - NMEA sentence information*. http://aprs.gids.nl/nmea/.

Brian W. Kernigham y Dennis M. Ritchie: "*El lenguaje de programación C*". Segunda edición. 1991. Prentice-Hall.

Harvey M. Deitel y Paul J. Deitel: "C How to program". Sexta edición. 2010. Prentice-Hall.