UNIVERSIDad DE BUENOS AIRES FACULTAD DE INGENIERÍA

Año 2019 - 1er cuatrimestre

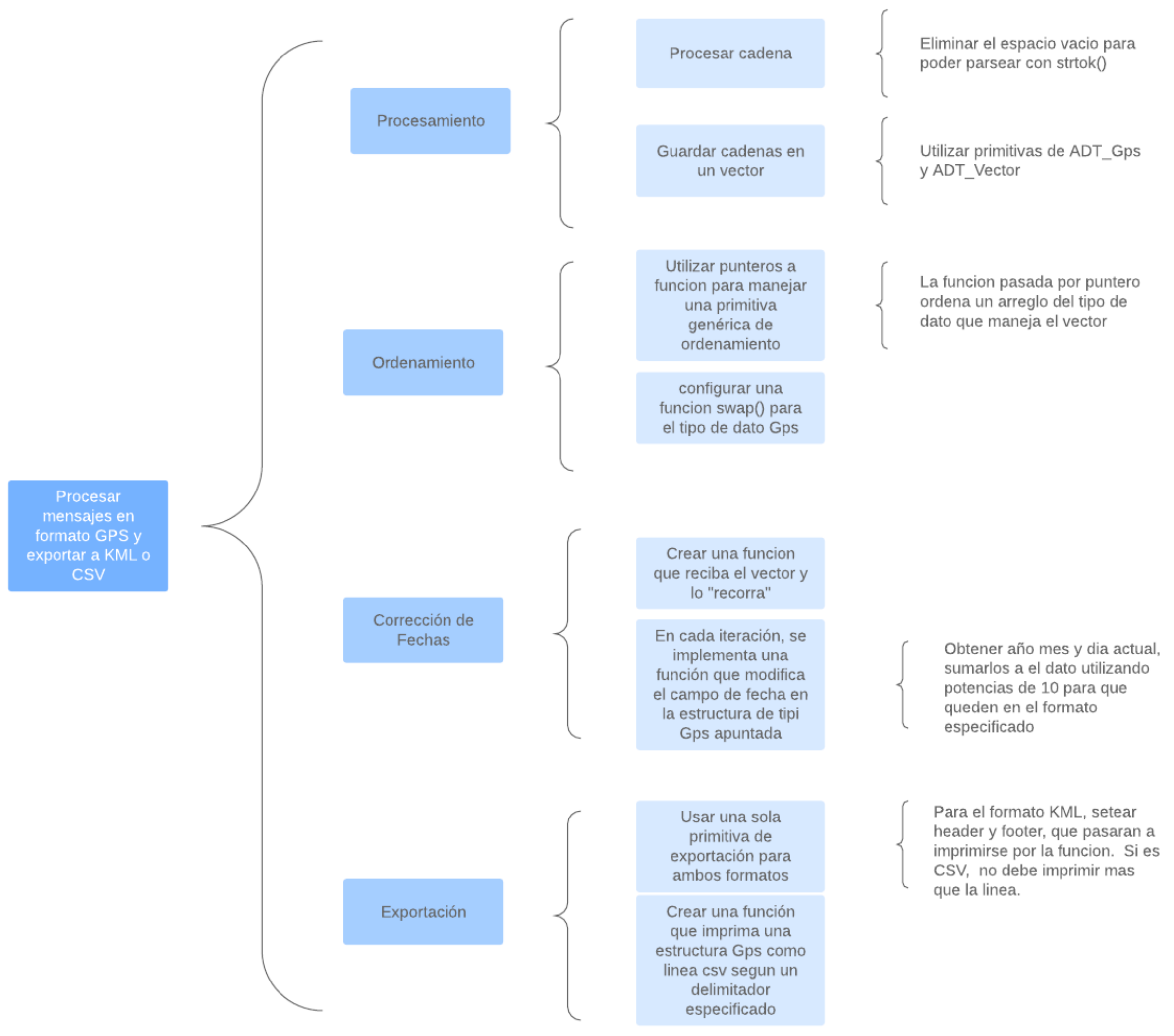
ALGORITMOS y PROGRAMACIÓN I (95.11)

Trabajo Práctico: Visualización de mensajes GPS en formato NMEA

Alumno: Lautaro De Lucia
Padrón:100203
Fecha: 26/06/19

Explicación de alternativas consideradas y estrategias adoptadas:

Se optó por una metodología **Top-down** para la resolución de este trabajo. En principio, se subdividió el problema en 4 sub-problemas principales. A su vez, cada sub-problema se subdividió en problemas más pequeños. Una vez establecido el esquema de trabajo, se empezó a codificar, empezando por el primer sub-problema, desde su módulo más pequeño, testeándolo y luego incorporándolo a un módulo superior, hasta tener completado el sub-problema. Se procedió de la misma forma para los siguientes 3 sub-problemas, concluyendo en la resolución del problema principal.

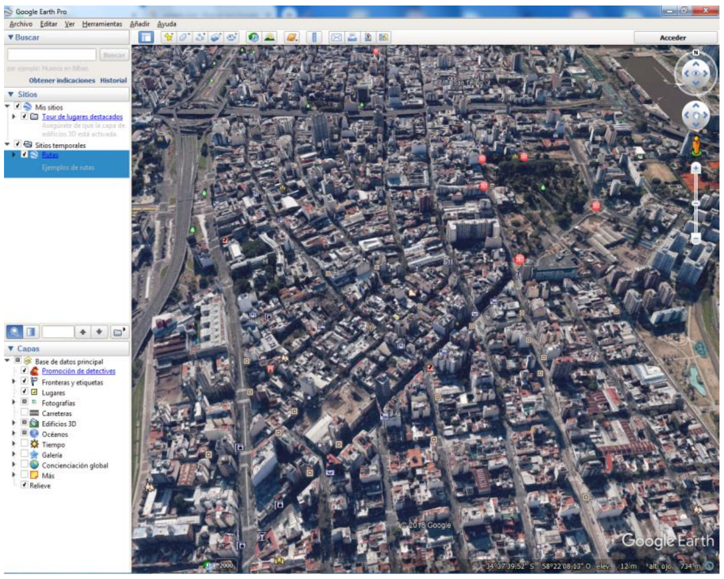

Procesar mensajes en formato GPS y exportar a KML o CSV
Procesamiento
Procesar cadena
Eliminar el espacio vacio para poder parsear con strtok()
Guardar cadenas en un vector
Utilizar primitivas de ADT_Gps y ADT_Vector
Ordenamiento
Utilizar punteros a funcion para manejar una primitiva genérica de ordenamiento
La funcion pasada por puntero ordena un arreglo del tipo de dato que maneja el vector
configurar una funcion swap() para el tipo de dato Gps
Corrección de Fechas
Crear una funcion que reciba el vector y lo "recorra"
En cada iteración, se implementa una función que modifica el campo de fecha en la estructura de tipi Gps apuntada
Obtener año mes y dia actual, sumarlos a el dato utilizando potencias de 10 para que queden en el formato especificado
Exportación
Usar una sola primitiva de exportación para ambos formatos
Para el formato KML, setear header y footer, que pasaran a imprimirse por la funcion. Si es CSV, no debe imprimir mas que la linea.
Crear una función que imprima una estructura Gps como linea csv segun un delimitador especificado

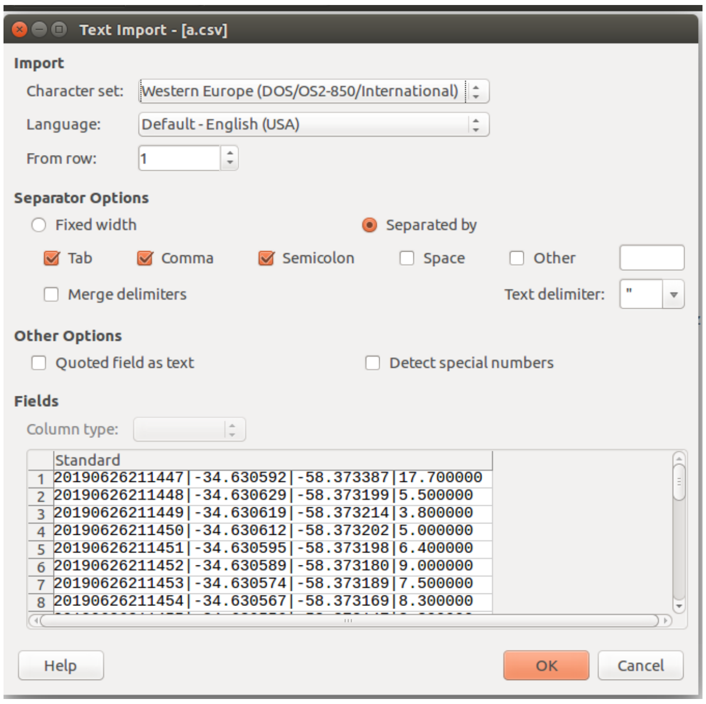
Se juzga que este método, a pesar de incrementar el volumen de trabajo en el corto plazo, termina reduciéndolo en el largo plazo. En efecto, en ningún momento de la codificación se produjeron errores inesperados que, de haberse producido, hubieran frenado el trabajo. Todos los errores advertidos por el compilador fueron fácilmente detectables y corregibles, y una vez corregidos la ejecución del programa se comportó de la forma esperada. Esto se debe fundamentalmente a que cada módulo había sido previamente testeado antes de incorporarse al programa principal.
Por otro lado, las horas que pueden perderse en plantear la esquematización y sub-división del problema se recuperan fácilmente. Una vez planteados todos los procedimientos a seguir, se trabaja en forma ordenada, continua y sin interrupciones.

Resultados de la ejecución en condiciones normales e inesperadas de entrada:

```
./gpsviewer –fmt csv –out a.csv trayectoria-A.txt
```

```
./gpsviewer –fmt kml –out a.kml trayectoria-A.txt
```





Text Import - [a.csv]

Import

Character set: Western Europe (DOS/OS2-850/International)
Language: Default - English (USA)
From row: 1

Separator Options

Fixed width [✓ Separated by]

[✓ Tab] [✓ Comma] [✓ Semicolon] Space Other

Merge delimiters Text delimiter: "

Other Options

Quoted field as text Detect special numbers

Fields

Column type:

```
  Standard
1 20190626211447|-34.630592|-58.373387|17.700000
2 20190626211448|-34.630629|-58.373199|5.500000
3 20190626211449|-34.630619|-58.373214|3.800000
4 20190626211450|-34.630612|-58.373202|5.000000
5 20190626211451|-34.630595|-58.373198|6.400000
6 20190626211452|-34.630589|-58.373180|9.000000
7 20190626211453|-34.630574|-58.373189|7.500000
8 20190626211454|-34.630567|-58.373169|8.300000
```

Help OK Cancel

```
./gpsviewer –fmt kml –out kml.txt trayectoria-G.txt
```

Error: Archivo de entrada

```
./gpsviewer –fmt txt –out kml.txt trayectoria-A.txt
```

Error: Formato inválido

```
./gpsviewer Rocky 4
```

Error: Invocación del programa

Conclusiones

Además de sintetizar ampliamente todos los contenidos vistos en el curso hasta la fecha, en este trabajo práctico se desarrollaron a fondo dos conceptos fundamentales de la etapa final del curso. Puntualmente, el uso de tipos de dato abstracto y el de punteros a funciones.

Una vez pensada la estructura troncal del trabajo, así como la de los módulos principales, las distintas primitivas de los tipos de dato TDA_Gps y TDA_Vector, definidas previa implementación, permitieron aproximar la etapa de codificación de forma intuitiva, haciendo mucho más ameno el traslado del concepto a su implementación en código. Además, el código resultante es muy legible así como más seguro.

A su vez, se complementó su uso con el de punteros a función. De esta forma, las primitivas de TDA_Vector pueden realizar las mismas operaciones (exportar, imprimir, ordenar, etc.) Independientemente del tipo de dato que este contenido en el vector. Esto hace que el código sea más reutilizable, ya que se independiza el concepto fundamental (“vector”) de cualquier aspecto específico de su implementación en el código.

```
CFLAGS = -ansi -pedantic -Wall
cc = gcc

all: gpsviewer

gpsviewer: main.o gps.o adtgps.o adtvector.o utilities.o errors.o
	$(cc) $(CFLAGS) -o gpsviewer main.o adtgps.o adtvector.o errors.o
utilities.o

main.o: main.c main.h gps.h adtgps.h adtvector.h setup.h errors.h
		$(cc) $(CFLAGS) -c main.c

gps.o: gps.c gps.h setup.h adtgps.h adtvector.h utilities.h
		$(cc) $(CFLAGS) -c gps.c

adtvector.o: adtvector.c adtvector.h setup.h utilities.h
		$(cc) $(CFLAGS) -c adtvector.c

adtgps.o: adtgps.c adtgps.h setup.h
		$(cc) $(CFLAGS) -c adtgps.c

utilities.o: utilities.c utilities.h setup.h
		$(cc) $(CFLAGS) -c utilities.c

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

```
#ifndef MAIN__H
#define MAIN__H

#include <stdio.h>

#include "setup.h"
#include "adtvector.h"
#include "adtgps.h"

#define FMT_CSV "csv"
#define FMT_KML "kml"

#define CSV_DELIMITER '|'
#define KML_DELIMITER ','
#define HEADER_FILE "kml_header.txt"
#define FOOTER_FILE "kml_footer.txt"

#define MAX_CMD_ARGS 6

#define CMD_ARG_FLAG_FMT 1
#define CMD_ARG_FORMAT 2
#define CMD_ARG_FLAG_OUT 3
#define CMD_ARG_OUTPUT 4
#define CMD_ARG_INPUT 5

status_t validate_arguments(int argc, char const *argv[], config_t * config);

#endif
```

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

#include "main.h"
#include "gps.h"
#include "adtvector.h"
#include "adtgps.h"
#include "setup.h"
#include "errors.h"

status_t validate_arguments(int argc, char const *argv[], config_t * config);

int main(int argc, char const *argv[]){

	FILE *file_gps;
	FILE *file_output;
	status_t st;
	ADT_Vector_t * v;
	config_t config;

	if((st = validate_arguments(argc,argv,&config)) != OK){
		print_error(st);
		return st;
	}

	if((file_gps = fopen(config.input_file,"rt")) == NULL){
		print_error(ERROR_INPUT_FILE);
		return ERROR_INPUT_FILE;
	}

	if((file_output = fopen(config.output_file,"wt")) == NULL){
		print_error(ERROR_OUTPUT_FILE);
		return ERROR_OUTPUT_FILE;
	}

	if ((st = ADT_Vector_new (&v)) != OK){
		print_error(st);
		return st;
	}

	if ((st = parse_nmea_file(file_gps, v )) != OK){
		print_error(st);
		return st;
	}

	fclose(file_gps);

	if (( st = ADT_Vector_sort( v , ADT_GPS_point_compare_by_date,
ADT_GPS_point_swap)) != OK){
		print_error(st);
		return st;
	}

	switch (config.format){

		case OUTPUT_FORMAT_CSV:
		if (( st = ADT_Vector_export( v , file_output , CSV_DELIMITER ,
Print_GPS_point_as_csv_line_date_lat_long_alt )) != OK){
			print_error(st);
			return st;
		}
		break;

		case OUTPUT_FORMAT_KML:
		if (( st = ADT_Vector_set_header_file( v , HEADER_FILE )) != OK){
```

```c
                        print_error(st);
                        return st;
                }
                if (( st = ADT_Vector_set_footer_file( v , FOOTER_FILE )) != OK){
                        print_error(st);
                        return st;
                }
                if (( st = ADT_Vector_export( v , file_output , KML_DELIMITER ,
Print_GPS_point_as_csv_line_long_lat_alt )) != OK){
                        print_error(st);
                        return st;
                }
                break;

        }

        fclose(file_output);

        return OK;

}

status_t validate_arguments(int argc, char const *argv[], config_t * config)
{
        if(argv == NULL || config == NULL)
                return ERROR_NULL_POINTER;

        if(argc != MAX_CMD_ARGS)
                return ERROR_PROGRAM_INVOCATION;

        if(strcmp(argv[CMD_ARG_FLAG_FMT],"-fmt"))
                return ERROR_PROGRAM_INVOCATION;

        if(strcmp(argv[CMD_ARG_FLAG_OUT],"-out"))
                return ERROR_PROGRAM_INVOCATION;

        if(!strcmp(argv[CMD_ARG_FORMAT],FMT_CSV))
                config->format = OUTPUT_FORMAT_CSV;
        else if(!strcmp(argv[CMD_ARG_FORMAT],FMT_KML))
                config->format = OUTPUT_FORMAT_KML;
        else
                return ERROR_INVALID_FORMAT;

        config->input_file = strdupl(argv[CMD_ARG_INPUT]);
        config->output_file = strdupl(argv[CMD_ARG_OUTPUT]);

        return OK;
}
```

```c
#ifndef GPS__H
#define GPS__H

#include <stdio.h>
#include "setup.h"
#include "adtgps.h"
#include "adtvector.h"
#include "utilities.h"

#define NMEA_ID "$GPGGA"
#define MASK_SUM 0x7F
#define MINUTES_IN_DEGREES 60
#define ALTITUDE_POSITION 9
#define PARSE_DELIMITER ','

status_t set_date_to_local_year_month_day( size_t * date );
bool_t validate_sum(char *str );
bool_t is_gpgga( char * str );
string remove_empty_fields_csv_line(const string str1, size_t * empty_fields, char
delimiter);
status_t get_nmea_line ( FILE *file_gps , char delimiter , char ** nmea_line ,
bool_t * eof);
status_t parse_nmea_file ( FILE *file_gps , ADT_Vector_t * v);
status_t load_GPS_point( ADT_GPS_point_t *pt , string* str);
status_t Print_GPS_point_as_csv_line_date_lat_long_alt (ADT_GPS_point_t* pt , char
delim , FILE* fo);
status_t Print_GPS_point_as_csv_line_long_lat_alt(ADT_GPS_point_t* pt , char
delim , FILE* fo);
double convert_longitude_str ( char* longitude_str , char* cardinal);
double convert_latitude_str ( char* latitude_str , char* cardinal);

#endif
```

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

#include "gps.h"
#include "setup.h"
#include "adtgps.h"
#include "adtvector.h"
#include "utilities.h"

status_t set_date_to_local_year_month_day ( size_t * date ){

        time_t local_time;
        struct tm * Time_struct;
        size_t year;
        size_t month;
        size_t day;

        time (&local_time);

        Time_struct = localtime(&local_time);

        year = (Time_struct->tm_year+1900) * 10000000000;
        month = (Time_struct->tm_mon + 1) * 100000000;
        day = (Time_struct->tm_mday) * 1000000;

        *date = year + month + day + *date;

        return 0;

}

bool_t validate_sum(char * s){

        /*Precondicion: Puntero no nulo*/

        int sum = 0;
        int result = 0;
        char verify[2];
        char * aux;
        char * temp;

        aux = s;

        aux++;

        while(*aux != '*')
        {
                sum ^= (*aux);
                aux++;
        }

        sum &= MASK_SUM;

        aux++;

        verify[0] = *aux;
        aux++;
        verify[1] = *aux;
        verify[2] = '\0';

        result = strtoul(verify,&temp,16);

        if(sum != result)
                return FALSE;
        return TRUE;

}

bool_t is_gpgga( char * str ){

        /*Precondicion : Puntero no nulo*/

        char aux[6];

        memcpy(aux,str,6);

        aux[6] = '\0';

        if(!strcmp(aux,NMEA_ID))
                return TRUE;
        return FALSE;

}

string remove_empty_fields_csv_line(const string str , size_t * empty_fields , char
delimiter){

        /*Precondición: Puntero no nulo*/

        size_t i,j;
        size_t len;
        string aux;

        len = strlen(str);

        if((aux = (char *) malloc( sizeof(char) * (len + 1))) == NULL)
                return NULL;

        for ( i = 0 , j = 0 ; i < len ; i++ , j++ ){

                if(str[j] == delimiter && str[j+1] == delimiter)
                        j++;

                aux[i] = str[j];

        }

        *empty_fields = j - i;

        return aux;

}

status_t get_nmea_line ( FILE *file_gps , char delimiter , char ** nmea_line,
bool_t * eof) {

        status_t st;
        string aux;
        int dummy = 1;
        size_t empty_fields;

        if( file_gps == NULL || nmea_line == NULL || eof == NULL)
                return ERROR_NULL_POINTER;

        while (dummy){

                if ((st = (read_line(&aux,eof,file_gps))) != OK ){
                        return st;
                }
                if(*eof == TRUE)
                        break;

                if ((is_gpgga(aux)) != TRUE)
                        continue;

                if ((validate_sum(aux)) != TRUE)
                        continue;

                if (((*nmea_line) = remove_empty_fields_csv_line
(aux,&empty_fields,delimiter)) == NULL)
                        return ERROR_MEMORY;

                if(empty_fields > 1)
                        continue;

                return OK;

        }

        return OK;

}

status_t parse_nmea_file ( FILE *file_gps , ADT_Vector_t * v ){

        status_t st;
        bool_t eof;
        string nmea_line;
        char delimiter = PARSE_DELIMITER;
        ADT_GPS_point_t pt;
        ADT_GPS_point_t * new_point;
        size_t date;

        if (( st = ADT_GPS_point_new(&pt)) != OK)
                return st;

        if ((st = get_nmea_line (file_gps, delimiter , &nmea_line , &eof)) != OK){
                return st;
        }

        while (eof != TRUE){

                if (( new_point = ADT_GPS_clone_point( &pt )) == NULL){
                        ADT_Vector_delete(&v,ADT_GPS_point_delete);
                        return ERROR_MEMORY;
                }

                if (( st = ADT_GPS_point_from_nmea_line( new_point , nmea_line )) !
= OK){

                        ADT_Vector_delete(&v,ADT_GPS_point_delete);
                        return st;
                }

                date = ADT_GPS_point_get_date(new_point);

                if ((st = set_date_to_local_year_month_day(&date)) != OK)
                        return st;

                ADT_GPS_point_set_date(new_point, date);


                if (( st = ADT_Vector_append( v , new_point )) != OK){
                        ADT_Vector_delete(&v,ADT_GPS_point_delete);
                        return st;
                }

                if ((st = get_nmea_line (file_gps, delimiter, &nmea_line , &eof)) !
= OK){
                        ADT_Vector_delete(&v,ADT_GPS_point_delete);
                        return st;
                }

        }

        return OK;

}

status_t load_GPS_point( ADT_GPS_point_t *pt , string* str){

        size_t aux1;
        double aux2;
        char *temp;

        if ( pt == NULL || str == NULL)
                return ERROR_NULL_POINTER;

        aux1 = (size_t) strtod(str[1],&temp);
                        if(*temp && *temp != '\n')
                                return ERROR_INVALID_DATE;
        ADT_GPS_point_set_date( pt , aux1 );
        aux2 = convert_latitude_str(str[2],str[3]);
        ADT_GPS_point_set_latitude( pt , aux2 );
        aux2 = convert_longitude_str(str[4],str[5]);
        ADT_GPS_point_set_longitude( pt , aux2 );
        aux2 = strtod(str[ALTITUDE_POSITION],&temp);
                        if(*temp && *temp != '\n')
                                return ERROR_INVALID_ALTITUDE;
        ADT_GPS_point_set_altitude( pt , aux2 );

        return OK;

}

status_t Print_GPS_point_as_csv_line_date_lat_long_alt (ADT_GPS_point_t* pt , char
delim , FILE* fo){

        if(pt == NULL || fo == NULL)
                return ERROR_NULL_POINTER;

        fprintf(fo,"%lu%c%f%c%f%c%f\n",
                ADT_GPS_point_get_date(pt),delim,
                ADT_GPS_point_get_latitude(pt),delim,
                ADT_GPS_point_get_longitude(pt),delim,
                ADT_GPS_point_get_altitude(pt));

        return OK;

}

status_t Print_GPS_point_as_csv_line_long_lat_alt(ADT_GPS_point_t* pt , char
delim , FILE* fo){

        if(pt == NULL || fo == NULL)
                return ERROR_NULL_POINTER;

        fprintf(fo,"%f%c%f%c%f\n",
                ADT_GPS_point_get_longitude(pt),delim,
                ADT_GPS_point_get_latitude(pt),delim,
                ADT_GPS_point_get_altitude(pt));
```

```c
        return OK;

}

double convert_longitude_str ( char* longitude_str , char* cardinal){

        char deg_str[3];
        char min_str[9];
        double deg;
        double min;
        double total;
        char* temp;

        memcpy(deg_str,longitude_str,3);
        deg_str[3] = '\0';

        memcpy(min_str,&longitude_str[3],9);
        min_str[9] = '\0';

        deg = strtod(deg_str,&temp);

        min = strtod(min_str,&temp);

        if(!strcmp(cardinal,"W"))
                total = -(deg+min/MINUTES_IN_DEGREES);
        else
                total = deg +min/MINUTES_IN_DEGREES;

        return total;
}

double convert_latitude_str ( char* longitude_str , char* cardinal){

        char deg_str[2];
        char min_str[8];
        double deg;
        double min;
        double total;
        char* temp;

        memcpy(deg_str,longitude_str,2);
        deg_str[2] = '\0';

        memcpy(min_str,&longitude_str[2],8);
        min_str[8] = '\0';

        deg = strtod(deg_str,&temp);

        min = strtod(min_str,&temp);

        if(!strcmp(cardinal,"S"))
                total = -(deg+min/MINUTES_IN_DEGREES);
        else
                total = deg +min/MINUTES_IN_DEGREES;

        return total;
}
```

```
#ifndef ADTVECTOR__H
#define ADTVECTOR__H

#include <stdio.h>

#include "setup.h"
#include "utilities.h"

typedef struct {
        void ** elements;
        size_t size;
        string header_file;
        string footer_file;
} ADT_Vector_t;

status_t ADT_Vector_new (ADT_Vector_t **pv);
size_t ADT_Vector_get_size (const ADT_Vector_t *v);
void * ADT_Vector_get_element_at_pos( const ADT_Vector_t *v , size_t position );
status_t ADT_Vector_delete (ADT_Vector_t ** pv , status_t (*pf) (void*));
status_t ADT_Vector_export(const ADT_Vector_t *v , FILE * fo, char delim ,
status_t (*pf)(void* , char , FILE*));
status_t ADT_Vector_clone (ADT_Vector_t* v , ADT_Vector_t * pc , void * (*pf)
(const void*) , status_t (*pf2)(void*));
bool_t ADT_Vector_equals (const ADT_Vector_t *v1 , const ADT_Vector_t *v2 , bool_t
(*pf)(void* , void*));
status_t ADT_Vector_append( ADT_Vector_t * v , void * elem );
status_t ADT_Vector_print_on_terminal (ADT_Vector_t * v , status_t (*pf) (void *));
status_t ADT_Vector_set_header_file(ADT_Vector_t *v , char * str);
status_t ADT_Vector_set_footer_file(ADT_Vector_t *v , char * str);
status_t ADT_Vector_sort( ADT_Vector_t * v , bool_t (*compare) (void*,void*) , void
(*swap) (void*,void*));


#endif
```

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

#include "setup.h"
#include "adtvector.h"
#include "utilities.h"

status_t ADT_Vector_new (ADT_Vector_t **pv)
{
        if (pv == NULL)
                return ERROR_NULL_POINTER;

        if((*pv = (ADT_Vector_t*) malloc (sizeof(ADT_Vector_t))) == NULL)
        {
                (*pv)->elements = NULL;
            (*pv)->size = 0;
            return ERROR_MEMORY;
    }

        (*pv)->size = 0;

        return OK;

}

size_t ADT_Vector_get_size (const ADT_Vector_t *v){
        return v->size;
}

void * ADT_Vector_get_element_at_pos( const ADT_Vector_t *v , size_t position ){
        return v->elements[position];
}

bool_t ADT_Vector_equals (const ADT_Vector_t *v1 , const ADT_Vector_t *v2 , bool_t
(*pf)(void* , void*))
{

    size_t i;

    if ( v1 == NULL || v2 == NULL)
        return FALSE;
    if ( v1->size != v2->size)
        return FALSE;

    for (i = 0 ; i<v1->size;i++)
        if(((*pf)(v1->elements[i],v2->elements[i]))!=0)
                return FALSE;

    return TRUE;
}

status_t ADT_Vector_delete (ADT_Vector_t ** pv , status_t (*pf) (void*)){

        size_t i;
        status_t st;

    if (pv == NULL || pf == NULL)
        return ERROR_NULL_POINTER;

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

    free((*pv)->elements);
    (*pv)->elements = NULL;
    free(*pv);
    *pv = NULL;

    return OK;
}

status_t ADT_Vector_clone ( ADT_Vector_t* v , ADT_Vector_t * pc , void * (*pf)
(const void*) , status_t (*pf2)(void*) )
{

        size_t i;
    size_t j;
    status_t st;
    ADT_Vector_t * aux;

    if (v == NULL || pc == NULL || pf == NULL)
        return ERROR_NULL_POINTER;

    if((st = ADT_Vector_new(pc)))
        return st;

    for ( i = 0 ; i < v->size ; i++ )
    {
        aux = (*pf)(v->elements[i]);
        if (aux == NULL){
                for( j = 0 ; j < pc-> size ; j++)
                (*pf2)(v->elements[j]);
        v->elements = NULL;
        }
    }

    return OK;
}

status_t ADT_Vector_append ( ADT_Vector_t * v , void * elem ){

        void ** aux;

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

        if (( aux = (void**) realloc( v-> elements, (v->size + 1) * sizeof
(void*))) == NULL)
                return ERROR_MEMORY;

        v->elements = aux;
        v->elements [v->size] = elem;
        v->size++;

        return OK;

}

status_t ADT_Vector_print_on_terminal (ADT_Vector_t * v, status_t (*pf) (void *)){

        size_t i;

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

        return OK;

}

status_t ADT_Vector_set_header_file(ADT_Vector_t *v , char * str){
```

```c
        if( v == NULL || str == NULL)
                return ERROR_NULL_POINTER;

        v->header_file = str;

        return OK;
}

status_t ADT_Vector_set_footer_file(ADT_Vector_t *v , char * str){

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

        v->footer_file = str;

        return OK;

}


status_t ADT_Vector_export(const ADT_Vector_t *v , FILE * fo, char delim ,
status_t (*pf)(void* , char , FILE*)){

        size_t i;
        status_t st;
        FILE *file_header;
        FILE *file_footer;
        char c;

        if ( v == NULL || fo == NULL || pf == NULL )
                return ERROR_NULL_POINTER;

        if (v->header_file != NULL)
                if((file_header = fopen(v->header_file,"rt")) == NULL)
                        return ERROR_HEADER_FILE;

        if (v->footer_file != NULL)
                if((file_footer = fopen(v->footer_file,"rt")) == NULL)
                        return ERROR_FOOTER_FILE;

        if (v->header_file != NULL){
        while((c = fgetc(file_header)) != EOF)
                fputc(c,fo);
        }

        for ( i = 0 ; i < v->size ; i++ ){
                if ((st = (*pf)(v->elements[i],delim, fo )) != OK)
                        return st;
        }

        if(v->footer_file != NULL){
        while((c = fgetc(file_footer)) != EOF)
                fputc(c,fo);
        }

        return OK;

}


status_t ADT_Vector_sort ( ADT_Vector_t * v , bool_t (*compare) (void*,void*) ,
void (*swap) (void*,void*)){

        size_t i,j,min;
        size_t size = v->size;
```

```c
		for ( i = 0 ; i < size - 1 ; i++ ){
			min = i;
			for ( j = i + 1 ; j < size ; j++ ){
				if((*compare)(v->elements[j],v->elements[min]))
					min = j;
			}
			(*swap)(v->elements[min],v->elements[i]);
		}

	return OK;
}
```

```c
#ifndef ADTGPS__H
#define ADTGPS__H

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

#include "setup.h"

typedef struct {
        size_t date;
        double latitude;
        double longitude;
        double altitude;
} ADT_GPS_point_t;

status_t ADT_GPS_point_new(ADT_GPS_point_t * p);
status_t ADT_GPS_point_delete(ADT_GPS_point_t * p);
status_t ADT_GPS_point_set_date(ADT_GPS_point_t * p, size_t d);
status_t ADT_GPS_point_set_latitude(ADT_GPS_point_t * p, double l);
status_t ADT_GPS_point_set_longitude(ADT_GPS_point_t * p, double l);
status_t ADT_GPS_point_set_altitude(ADT_GPS_point_t * p, double a);
status_t ADT_GPS_copy_point(ADT_GPS_point_t * p1 , const ADT_GPS_point_t * p2);
void ADT_GPS_point_swap( ADT_GPS_point_t * p1 , ADT_GPS_point_t * p2 );
bool_t ADT_GPS_point_compare_by_date ( ADT_GPS_point_t * p1 , ADT_GPS_point_t * p2
);
size_t ADT_GPS_point_get_date(ADT_GPS_point_t * p);
double ADT_GPS_point_get_latitude(ADT_GPS_point_t * p);
double ADT_GPS_point_get_longitude(ADT_GPS_point_t * p);
double ADT_GPS_point_get_altitude(ADT_GPS_point_t * p);
status_t ADT_GPS_sort_by_date( ADT_GPS_point_t * arr[] , size_t * size );
status_t ADT_GPS_point_print(ADT_GPS_point_t * p);
ADT_GPS_point_t * ADT_GPS_clone_point(const ADT_GPS_point_t * p);
status_t ADT_GPS_point_from_nmea_line ( ADT_GPS_point_t * p , char * str );

#endif
```

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

#include "adtgps.h"
#include "utilities.h"
#include "gps.h"

status_t ADT_GPS_point_new(ADT_GPS_point_t * p)
{
	if(p == NULL)
		return ERROR_NULL_POINTER;

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

	p->date = 0.0;
	p->latitude = 0.0;
	p->longitude = 0.0;
	p->altitude = 0.0;

	return OK;
}

status_t ADT_GPS_point_delete(ADT_GPS_point_t * p)
{
	if(p == NULL)
		return ERROR_NULL_POINTER;

	free(p);

	p = NULL;

	return OK;
}

void ADT_GPS_point_swap( ADT_GPS_point_t * p1 , ADT_GPS_point_t * p2 ){

	ADT_GPS_point_t * aux;
	aux = ADT_GPS_clone_point( p1 );
	ADT_GPS_copy_point( p1 , p2 );
	ADT_GPS_copy_point( p2 , aux);

}

status_t ADT_GPS_point_set_date(ADT_GPS_point_t * p, size_t d)
{
	/*Precondición: Puntero no nulo*/

	p->date = d;

	return OK;
}


status_t ADT_GPS_point_set_latitude(ADT_GPS_point_t * p, double l)
{
	/*Precondición: Puntero no nulo*/

	p->latitude = l;

	return OK;
}

status_t ADT_GPS_point_set_longitude(ADT_GPS_point_t * p, double l)
{
	/*Precondición: Puntero no nulo*/
	p->longitude = l;

	return OK;
}

status_t ADT_GPS_point_set_altitude(ADT_GPS_point_t * p, double a)
{
	/*Precondición: Puntero no nulo*/

	p->altitude = a;

	return OK;
}

status_t ADT_GPS_point_print(ADT_GPS_point_t * p)
{
	if(p == NULL)
		return ERROR_NULL_POINTER;

	printf("%lu\n",p->date);
	printf("%f\n",p->latitude);
	printf("%f\n",p->longitude);
	printf("%f\n",p->altitude);

	return OK;
}

ADT_GPS_point_t * ADT_GPS_clone_point(const ADT_GPS_point_t * p)
{

	/*Precondición: Puntero no nulo*/

	ADT_GPS_point_t * c;

	if((c = (ADT_GPS_point_t *)malloc(sizeof(ADT_GPS_point_t))) == NULL)
		return NULL;

	c->date = p->date;
	c->latitude = p->latitude;
	c->longitude = p->longitude;
	c->altitude = p->altitude;

	return c;
}

size_t ADT_GPS_point_get_date(ADT_GPS_point_t * p)
{
	return p->date;
}

double ADT_GPS_point_get_latitude(ADT_GPS_point_t * p)
{
	return p->latitude;
}
double ADT_GPS_point_get_longitude(ADT_GPS_point_t * p)
{
	return p->longitude;
}
double ADT_GPS_point_get_altitude(ADT_GPS_point_t * p)
{
	return p->altitude;
}

status_t ADT_GPS_copy_point(ADT_GPS_point_t * p1 , const ADT_GPS_point_t * p2){
        if ( p1 == NULL || p2 == NULL )
                return ERROR_NULL_POINTER;

        p1->date = p2->date;
        p1->latitude = p2->latitude;
        p1->longitude = p2->longitude;
        p1->altitude = p2->altitude;

        return OK;

}

status_t ADT_GPS_point_from_nmea_line ( ADT_GPS_point_t * p , char * nmea_line ){

        size_t str_num;
        char **fields;
        status_t st;

        if ((st = split(nmea_line,',',&str_num,&fields)) != OK)
                return st;

        if (( st = load_GPS_point(p,fields)) != OK)
                return st;

        return OK;

}

bool_t ADT_GPS_point_compare_by_date ( ADT_GPS_point_t * p1 , ADT_GPS_point_t * p2
){

        /*Precondición: Puntero no nulo*/

        if (p1->date > p2->date)
                return TRUE;
        else
                return FALSE;

}
```

```
#ifndef UTILITIES__H
#define UTILITIES__H

#include <stdio.h>

#include "setup.h"

#define INIT_SIZE 10
#define CHOP_SIZE 10

status_t split(const string str, char delimiter, size_t * str_num, string **
fields );
status_t destroy_strings(char *** strings, size_t *l );
char * strdupl(const char *str);
status_t read_line(char ** str, bool_t *eof, FILE * file_str);

#endif
```

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

#include "utilities.h"
#include "setup.h"

status_t split(const string str, char delimiter, size_t * str_num, string **
fields ){

        size_t i;
        char *aux, *q, *p;
        char delim[2];

        if (str == NULL || str_num == NULL)
                return ERROR_NULL_POINTER;

        if( (aux = strdupl(str)) == NULL)
        {
                *str_num = 0;
                return ERROR_MEMORY;
        }

        for(i = 0, *str_num = 0; str[i]; i++)
        {
                if( str[i] == delimiter)
                        (*str_num)++;
        }

        (*str_num)++;

        if((*fields = (string *)malloc((*str_num)* sizeof(string))) == NULL)
        {
                *str_num = 0;
                free(aux);
                return ERROR_MEMORY;
        }

        delim[0] = delimiter;
        delim[1] = '\0';

        for(i = 0, q = aux; (p = strtok (q, delim)) != NULL; q = NULL, i++)
        {
                if(( (*fields)[i] = strdupl(p)) == NULL)
                {
                        free(aux);
                        destroy_strings( fields, str_num);
                        *str_num = 0;
                        return ERROR_MEMORY;
                }

        }

        free(aux);

        return OK;

}

status_t read_line(char ** str, bool_t *eof, FILE * file_str){

        size_t alloc_size;
        char c;
        char *aux;
        size_t used_size;
        if (str == NULL || eof == NULL || file_str == NULL)
                return ERROR_NULL_POINTER;

        if((*str = (char *)malloc(INIT_SIZE * sizeof(char))) == NULL)
                return ERROR_MEMORY;

        alloc_size = INIT_SIZE;

        used_size = 0;
        while ((c = fgetc(file_str)) != '\n' && c != EOF)
        {
                if(used_size == alloc_size - 1)
                {
                        if((aux = (char *)realloc(*str, (alloc_size + CHOP_SIZE) * 
sizeof(char))) == NULL)
                        {
                                free(str);
                                return ERROR_MEMORY;
                        }
                        *str = aux;
                        alloc_size += CHOP_SIZE;

                }

                (*str)[ used_size++ ] = c;
        }

        (*str)[ used_size ] = '\0';

        *eof = ( c == EOF )? TRUE : FALSE;

        return OK;

}

status_t destroy_strings(char *** strings, size_t *l ){

        size_t i;

        if ( strings == NULL || l == NULL)
                return ERROR_NULL_POINTER;

        for (i = 0; i < *l; i++)
        {
                free((*strings)[i]);
                (*strings)[i] = NULL;
        }

        free ( *strings);

        *strings = NULL;
        *l = 0;

        return OK;

}

char * strdupl(const char *str){

        char * aux;
        size_t len, i;

        if (str == NULL)
                return NULL;

        len = strlen(str);
        if((aux = (char *) malloc( sizeof(char) * (len + 1))) == NULL)
                return NULL;

        for( i = 0; (aux[i] = str [i]); i++ );

        return aux;
}
```

```c
#ifndef SETUP__H
#define SETUP__H

typedef char* string;

typedef enum{
        OUTPUT_FORMAT_CSV,
        OUTPUT_FORMAT_KML
} output_format_t;

typedef struct{
        output_format_t format;
        char* input_file;
        char* output_file;
} config_t;

typedef enum {TRUE,FALSE} bool_t;

typedef enum {
        OK,
        ERROR_NULL_POINTER,
        ERROR_MEMORY,
        ERROR_INPUT_FILE,
        ERROR_OUTPUT_FILE,
        ERROR_INVALID_DATE,
        ERROR_INVALID_LONGITUDE,
        ERROR_INVALID_LATITUDE,
        ERROR_INVALID_ALTITUDE,
        ERROR_HEADER_FILE,
        ERROR_FOOTER_FILE,
        ERROR_INVALID_FORMAT,
        ERROR_PROGRAM_INVOCATION,
        ERROR_TIME
} status_t;

#endif
```

```c
#ifndef ERRORS__H
#define ERRORS__H

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

#define MAX_ERRORS 20

status_t print_error (status_t error);

#endif
```

```c
#include <stdio.h>
#include "errors.h"
#include "setup.h"

char * error_dictionary[MAX_ERRORS] = {
        "Ok",
        "Error: Puntero nulo",
        "Error: Memoria insuficiente",
        "Error: Archivo de entrada",
        "Error: Archivo de salida",
        "Error(parseo): Valor de fecha inválido",
        "Error(parseo): Valor de longitud inválido",
        "Error(parseo): Valor de latitud inválido",
        "Error(parseo): Valor de altitud inválido",
        "Error: Archivo de encabezado",
        "Error: Archivo de pie de página",
        "Error: Formato inválido",
        "Error: Invocación del programa",
        "Error: Tiempo"
};

status_t print_error (status_t error){
        fprintf(stderr, "%s\n", error_dictionary[error]);
        return OK;
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
	<Document>
		<name>Rutas</name>
		<description>Ejemplos de rutas</description>
		<Style id="yellowLineGreenPoly">
			<LineStyle>
				<color>7f00ffff</color>
				<width>4</width>
			</LineStyle>
			<PolyStyle>
				<color>7f00ff00</color>
			</PolyStyle>
		</Style>
		<Placemark>
			<name>Relieve absoluto</name>
			<description>Pared verde transparente con contornos 
amarillos</description>
			<styleUrl>#yellowLineGreenPoly</styleUrl>
			<LineString>
				<extrude>1</extrude>
				<tessellate>1</tessellate>
				<altitudeMode>absolute</altitudeMode>
				<coordinates>
 </coordinates>
                       </LineString>
               </Placemark>
       </Document>
</kml>
```