Universidad de Buenos Aires
Facultad de Ingeniería
Primer cuatrimestre de 2019
Algoritmos y programación I (95.11)
Curso 01 (Ing. Cardozo)

TRABAJO PRÁCTICO N.º 1 – Visualización de mensajes GPS en formato NMEA

Fecha de entrega: 22 de junio de 2019

Integrantes:
LÜTZELSCHWAB, Nahila - Padrón número: 100686
<nahilutz@gmail.com>
HIRSCHMANN, Juan Ignacio - Padrón número: 100755
<juanhirschmann@gmail.com>

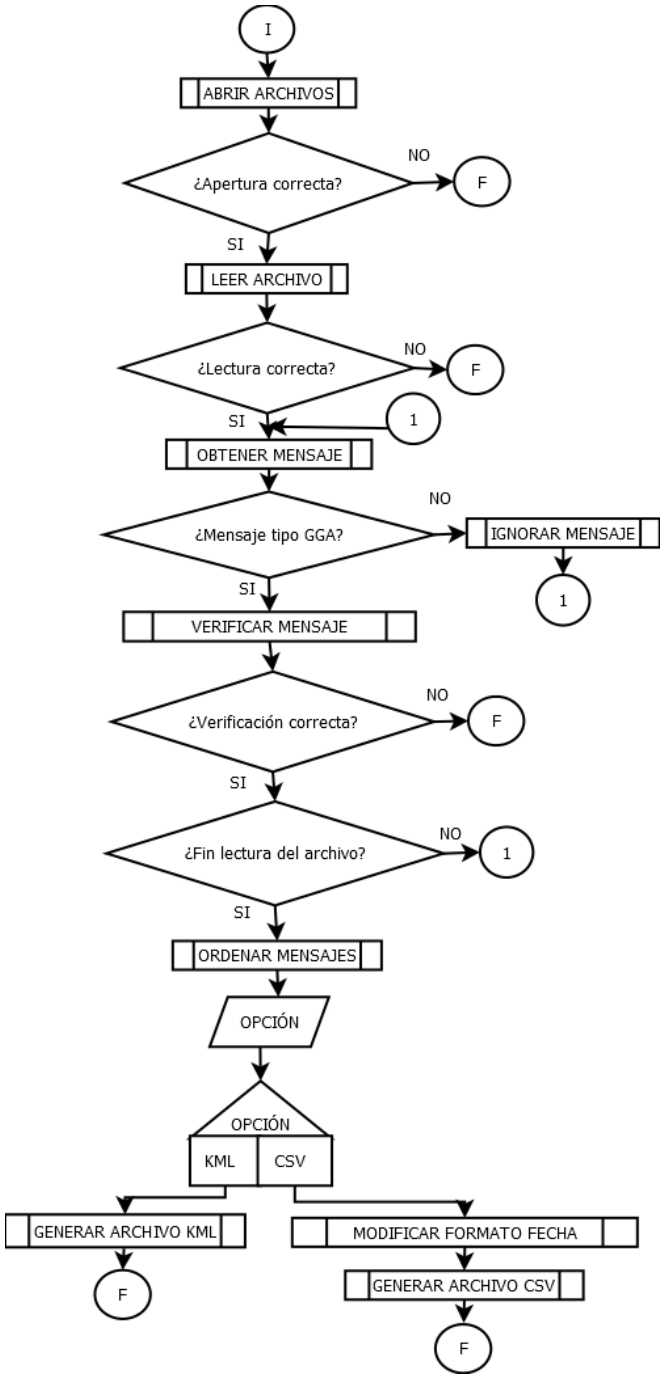
1-Introducción y objetivos

Se presenta el siguiente documento con el fin de documentar el desarrollo de un aplicativo de consola por medio de comandos en línea de órdenes, que permite interpretar mensajes GPS en formato NMEA a partir de un archivo ordenados por tiempo UTC y permite presentarlos en formato CSV y KML.

El informe se encuentra conformado por 5 partes: introducción, desarrollo, conclusiones, bibliografía consultada y anexo. En el desarrollo se presenta, mediante un diagrama de flujo, la representación del proceso del aplicativo. A su vez, se explican las estrategias que se adoptaron, las alternativas consideradas, se detallan las dificultades atravesadas y las soluciones adoptadas. En las conclusiones, se analiza la implicancia de las decisiones realizadas y se consignan sus ventajas y desventajas. Por último, en el anexo, se encuentran las entregas previas de este trabajo práctico junto con las consignas del mismo y se expone la solución hallada al problema.

2-Desarrollo

2.1 Diagrama de flujo



4.Anexo

4.1 main.h

```
#ifndef MAIN__H
#define MAIN__H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "types.h"
#include "format.h"
#include "errors.h"
#include "utils.h"
#include "gps.h"
#include "vector.h"
#define MAX_ARGS 6
#define FMT_FLAG "-fmt"
#define FMT_FLAG_POS 1
#define FMT_SPECIFIER_POS 2
#define OUT_FLAG "-out"
#define OUT_FLAG_POS 3
#define OUT_FILE_POS 4
#define IN_FILE_POS 5
#define CSV_FMT "CSV"
#define KML_FMT "KML"
typedef enum
{
      KML,
      CSV
}format_t;
#endif
```

4.2 main.c

```
#include "main.h"

status_t validate_args(int argc, char * argv[],format_t *fmt)
{
```

```
	if (argc!=MAX_ARGS)
		return ERROR_PROG_INVOCATION;
	if (strcmp(argv[OUT_FILE_POS],argv[IN_FILE_POS])==0)
		return ERROR_PROG_INVOCATION;

	if(fmt==NULL)
		return ERROR_NULL_POINTER;
	if (strcmp(argv[FMT_SPECIFIER_POS],KML_FMT)==0)
	{
		*fmt=KML;
		return OK;

	}
	if (strcmp(argv[FMT_SPECIFIER_POS],CSV_FMT)==0)
	{
		*fmt=CSV;
		return OK;

	}
	return ERROR_PROG_INVOCATION;
}
int main(int argc, char *argv[])
{
	status_t status;
	format_t fmt;
	FILE*fi;
	FILE*tmp,*tmp2;
	ADT_gps_record_t *record;
	ADT_vector_t * vector;

	if((status=validate_args(argc,argv,&fmt))!=OK)
	{
		print_error(status);
		return status;
	}
	fi=fopen("1.txt","rt");
	tmp=fopen("prueba.txt","rt");
	tmp2=fopen("ruta-A.txt","wt");
	filter_lines(fi,CRITERIA,tmp);
	ADT_vector_new(&vector);
	load_gps_record_vector(vector,&record,tmp);
	bubble_sort(vector);
	export_as_KML(vector,HEADER,FOOTER,tmp2);
	return OK;

}
```

4.3 Biblioteca <<utils.h>>

4.3.1 utils.h

```
#ifndef UTILS__H
#define UTILS__H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "types.h"
#include "gps.h"
#include "errors.h"
#include "vector.h"
#include "config.h"
#include "format.h"
#include "vector.h"
#include "vector.h"
#define WEST_INDICATOR 'W'
#define SOUTH_INDICATOR 'S'
#define DELIMITER ','
#define LF             '\n'
#define NUL            '\0'
#define INIT_CHOP100
#define CHOP_SIZE20
#define FILTER_MASK 0x7F

char * strdup(const char *s);
status_t destroy_strings(char***p,size_t length);
char ** split(const char * s, char  del, size_t * length);
status_t read_line (FILE *,char **,bool_t *);
status_t check_sum (char * string, bool_t * check );
status_t is_line_relevant(char * source,char * criteria,bool_t * selection);
status_t filter_lines(FILE * fi,char * criteria, FILE * tmp);
status_t read_record(FILE*fi, ADT_gps_record_t**record,char del ,bool_t*eof);
comparator_t is_greater(ADT_gps_record_t *a,ADT_gps_record_t *b);
status_t flag_consec_blank_fields(char*source,bool_t*flag);
/*status_t convert_UTCtime_CSV(float utc_time, char **utcTimeCsv);*/
status_t get_coordinates(ADT_gps_record_t*p,double*lat,double*longi);
status_t process_NMEA_file(FILE*input,format_t format,FILE*ouput);
status_t load_gps_record_vector(ADT_vector_t*v,ADT_gps_record_t**record,FILE*tmp);
status_t read_record(FILE*fi, ADT_gps_record_t**record,char del ,bool_t*eof);
status_t bubble_sort(ADT_vector_t*v);

#endif
```

4.3.2 utils.c

```
#include "utils.h"


char *strdup(const char *s)
{
    size_t size;
    char *p;

    size = strlen(s) + 1;
    if ((p=malloc(sizeof(char)*size))== NULL)
      return NULL;

    memcpy(p, s, size);
    return p;
}
status_t destroy_strings(char***p,size_t length)
{
      size_t i;

      if(p==NULL)
            return ERROR_NULL_POINTER;
      for (i = 0; i < length; i++)
      {
            free((*p)[i]);
            (*p)[i]=NULL;

            free(*p);
            (*p)=NULL;
      }
      return OK;
}
char ** split(const char * s, char  del, size_t * length)
{
      size_t i;
      char ** fields;
      char * str;
      char * q;
      char * p;

      if(s==NULL||length==NULL)
      {
            *length=0;
            return NULL;

      }
```

```
	if((str=strdup(s))==NULL)
	{
		*length=0;
		return NULL;
	}

	for(i=0,*length=0; str[i]!='\0' ;i++)
	{

		if(str[i]==del)
		{
			(*length)++;
		}

		(*length)++;
	}

	if((fields=(char **)malloc((*length)*sizeof(char*)))==NULL)
	{
		free(str);
		*length=0;
		return NULL;
	}
	for ( i=0, q=str;(p=strtok(q,&del))!=NULL;q=NULL,i++)
	{
		if((fields[i]=strdup(p))==NULL)
		{
			free(str);
			destroy_strings(&fields,*length);
			*length=0;
			 return NULL;
		}
	}

	return fields;
}
status_t read_line (FILE * fi, char ** 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 * sizeof(char))) == NULL)
```

```
            return ERROR_NO_MEMORY;
      alloc_size= INIT_CHOP;
      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_NO_MEMORY;
                  }
                  alloc_size += CHOP_SIZE;
                  *s = aux;
            }
            (*s)[used_size++] = c;
      }
      (*s)[used_size] = NUL;
      if (*s[0]=='\0')
      {
            *eof=TRUE;
            return OK;
      }
      *eof = ((c==EOF)?TRUE:FALSE);
      return OK;
}
status_t check_sum (char * string, bool_t * check )
{

      char * string_check;
      unsigned char check_sum_true_value;
      unsigned char check_sum_string_value;
      unsigned char check_partial_value;
      char*endptr;
      size_t i;
      if (string==NULL||check==NULL)
            return ERROR_NULL_POINTER;

      if (string[0]=='\0'||string[0]=='\n')
      {
            *check=FALSE;
            return OK;
      }
      if((string_check=(strchr(string,CHECK_SUM_CHARACTER)+1))==NULL)
            return ERROR_CHECK_SUM_NOT_FOUND;
```

```
	check_sum_true_value=strtoul(string_check,&endptr,16);

	if (endptr==NULL)
		return ERROR_CHECK_SUM_NOT_FOUND;

	check_sum_string_value=0;
	check_partial_value= *(strchr(string,MSG_ID_CHARACTER)+1);
	i=1;

	while((check_partial_value)!=CHECK_SUM_CHARACTER)
	{
		check_sum_string_value=check_partial_value^check_sum_string_value;
		i++;
		check_partial_value= *(strchr(string,MSG_ID_CHARACTER)+i);
	}
	if (check_sum_true_value==check_sum_string_value)
	{
		*check=TRUE;
		return OK;
	}
	else
	{
		*check=FALSE;
		return OK;
	}
}
status_t is_line_relevant(char * source,char * criteria,bool_t * selection)
{
	size_t length;
	length=strlen(criteria);
	if(source==NULL||criteria==NULL||selection==NULL)
		return ERROR_NULL_POINTER;
	if (source[0]=='\0'||source[0]=='\n')
	{
		*selection=FALSE;
		return OK;
	}
	if (strncmp(source,criteria,length)==0)
	{
		*selection=TRUE;
		return OK;
	}
	*selection=FALSE;
	return OK;
}
status_t flag_consec_blank_fields(char*source,bool_t*flag)
{
```

```
	char blank_fields[5]={DELIMITER,DELIMITER,DELIMITER,DELIMITER};
	if (source==NULL||flag==NULL)
	{
		return ERROR_NULL_POINTER;
	}

	if (strstr(source,blank_fields)!=NULL)
	{
		*flag=FALSE;
		return OK;
	}
	*flag=TRUE;
	return OK;
}
status_t filter_lines(FILE * fi, char * criteria, FILE * tmp)
{
	char source[MAX_LENGTH];
	bool_t check_sum_value;
	bool_t relevance;
	bool_t flag;
	char*pointer;
	status_t status;


	if (fi==NULL||tmp==NULL)
		return ERROR_NULL_POINTER;
	while((fgets(source,MAX_LENGTH+2,fi))!=NULL)
	{

		if((status=check_sum(source,&check_sum_value))!=OK)
			return status;

		if((status=is_line_relevant(source,criteria,&relevance))!=OK)
			return status;

		if((status=flag_consec_blank_fields(source,&flag))!=OK)
			return status;

		if (check_sum_value==TRUE && relevance==TRUE && flag==TRUE)
		{
			pointer=strchr(source,CHECK_SUM_CHARACTER);
			*pointer=DELIMITER;
			fprintf(tmp,"%s",source);
		}
	}
```

```
	return OK;
}
status_t read_record(FILE*fi, ADT_gps_record_t**record,char del ,bool_t*eof)
{
	char *file_line;
	char**string_array;
	char *end_ptr;
	status_t status;
	size_t length;
	char msg_id[ID_LENGTH];
	size_t utc_time;
	double latitude;
	char vertical_orientation;
	double longitude;
	char horizontal_orientation;
	int fixed_pos_indicator;
	size_t satellite_number;
	float hdop;
	float altitude;
	char altitude_unit;
	float geoid_separation;
	char geoid_separation_unit;
	size_t age_of_diff_corr;
	int diff_ref_station_id;
	char check_sum[CHECK_SUM_LENGTH];
	if (fi==NULL || record==NULL || eof==NULL)
		return ERROR_NULL_POINTER;

	if((status=read_line(fi,&file_line,eof))!=OK)
		return status;
	if (file_line[0]=='\0')
	{
		*eof=TRUE;
		return OK;
	}

	if((string_array=split(file_line,del,&length))==NULL)
		return ERROR_NULL_POINTER;

	if (file_line[0]=='\0')
	{
		*eof=TRUE;
		return OK;
	}

	strcpy(msg_id,string_array[MSG_ID_POS]);

	utc_time=strtod(string_array[UTC_TIME_POS],&end_ptr);
```

```
	if (end_ptr==NULL)
		return ERROR_REGISTRY_FORMAT;

	latitude=strtod(string_array[LATITUDE_POS],&end_ptr);


	if (end_ptr==NULL)
		return ERROR_REGISTRY_FORMAT;

	vertical_orientation =string_array[VERTICAL_ORIENTATION_POS][0];

	longitude=strtod(string_array[LONGITUDE_POS],&end_ptr);
	if (end_ptr==NULL)
		return ERROR_REGISTRY_FORMAT;

	horizontal_orientation=string_array[HORIZONATAL_ORIENTATION_POS][0];
	fixed_pos_indicator=atoi(string_array[FIXED_INDICATOR_POS]);

	satellite_number=strtoul(string_array[SATELLITE_NUMBER_POS],&end_ptr,10);
	if (end_ptr==NULL)
		return ERROR_REGISTRY_FORMAT;

	hdop=strtod(string_array[HDOP_POS],&end_ptr);
	if (end_ptr==NULL)
		return ERROR_REGISTRY_FORMAT;

	altitude=strtod(string_array[ALTITUDE_POS],&end_ptr);
	if (end_ptr==NULL)
		return ERROR_REGISTRY_FORMAT;

	altitude_unit=string_array[ALTITUDE_UNIT_POS][0];

	geoid_separation=strtod(string_array[GEOID_SEPARATION_POS],&end_ptr);
	if (end_ptr==NULL)
		return ERROR_REGISTRY_FORMAT;

	geoid_separation_unit=string_array[GEOID_SEPARATION_UNIT][0];

	age_of_diff_corr=strtoul(string_array[AGE_OF_DIFF_CORR_POS],&end_ptr,10);
	if (end_ptr==NULL)
		return ERROR_REGISTRY_FORMAT;
	if (string_array[DIFF_REF_STATION_ID_POS]==NULL)
	{

string_array[DIFF_REF_STATION_ID_POS]=string_array[AGE_OF_DIFF_CORR_POS];
		string_array[AGE_OF_DIFF_CORR_POS]='\0';
	}
	diff_ref_station_id=atoi(string_array[DIFF_REF_STATION_ID_POS]);
```

```
ADT_gps_record_new_from_values(msg_id,utc_time,latitude,vertical_orientation,longitu
de,horizontal_orientation,fixed_pos_indicator,satellite_number,hdop,altitude,altitud
e_unit,geoid_separation,geoid_separation_unit,age_of_diff_corr,diff_ref_station_id,c
heck_sum,record);

      return OK;
}
status_t load_gps_record_vector(ADT_vector_t*v,ADT_gps_record_t**record,FILE*tmp)
{
      status_t status;
      bool_t eof;

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

      if((status=read_record(tmp,record,DELIMITER,&eof))!=OK)
            return status;

      while(eof!=TRUE)
      {
            if((status=ADT_vector_append(v,*record))!=OK)
                  return status;

            if((status=read_record(tmp,record,DELIMITER,&eof))!=OK)
                  return status;
      }

      return OK;
}
comparator_t is_greater(ADT_gps_record_t *a,ADT_gps_record_t *b)
{
      if (a->utc_time > b->utc_time)
            return A_IS_GREATER;

      if (a->utc_time < b->utc_time)
            return A_IS_LESSER;

      else {
            return A_EQUALS;
      }
}
status_t bubble_sort(ADT_vector_t*v)
{
      size_t i;
      size_t j;
      status_t status;
```

```
	comparator_t comparison;
	if (v==NULL)
		return ERROR_NULL_POINTER;
	for (i = 0; i < ((v->size)-1); ++i)
	{
		for (j = 0; j <((v->size)-1-i); ++j)
		{
			if
((comparison=is_greater(v->elements[j],v->elements[j+1]))==A_IS_GREATER)
			{
				if((status=swap((v->elements[j]),(v->elements[j+1]) ))!=OK)
					return status;
			}
		}
	}
	return OK;
}
status_t swap(ADT_gps_record_t*a,ADT_gps_record_t*b)
{
	ADT_gps_record_t aux;
	if (a==NULL||b==NULL)
		return ERROR_NULL_POINTER;
	aux=*a;
	*a=*b;
	*b=aux;
	return OK;
}
```

4.9 Biblioteca <<export.h>>

4.9.1 time.h

4.9.2 time.c

4.9 Biblioteca <<format.h>>

4.9.1 format.h

```
#ifndef FORMAT__H
#define FORMAT__H
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "types.h"
#include "vector.h"
#include "utils.h"
#include "config.h"

#define HEADER "\<\?xml version=\"1.0\" encoding=\"UTF-8\"\?>\n<kml
xmlns=\"http://www.opengis.net/kml/2.2\">\n<Document>\n<name>Rutas</name>\n<descript
ion>Ejemplos de rutas</description>\n<Style
```

```
id=\"yellowLineGreenPoly\">\n<LineStyle>\n<color>7f00ffff</color>\n<width>4</width>\
n</LineStyle>\n<PolyStyle>\n<color>7f00ff00</color>\n</PolyStyle>\n</Style>\n<Placem
ark>\n<name>Relieve absoluto</name>\n<description>Pared verde transparente con
contornos\namarillos</description>\n<styleUrl>#yellowLineGreenPoly</styleUrl>\n<Line
String>\n<extrude>1</extrude>\n<tessellate>1</tessellate>\n<altitudeMode>absolute</a
ltitudeMode>"
#define FOOTER "\</coordinates>\n</LineString>\n</Placemark>\n</Document>\n</kml>"
status_t export_as_KML(ADT_vector_t*p,char*header,char*footer,FILE*out);
/*status_t export_as_CSV(ADT_vector_t *vec,FILE *fo);*/

#endif
```

4.9.2 format.c

```
#include "format.h"

status_t export_as_KML(ADT_vector_t*p,char*header,char*footer,FILE*out)
{
      size_t i;
      double latitude;
      double longitude;
      status_t status;

      if (p==NULL||footer==NULL||out==NULL)
            return ERROR_NULL_POINTER;
      if((status=ADT_vector_set_header(p,header))!=OK)
            return status;
      fprintf(out,"%s\n",p->header);

      for (i = 0; i < p->size; ++i)
      {
            if((status=(get_coordinates(p->elements[i],&latitude,&longitude)))!=OK)
                  return status;
            fprintf(out, "%f,%f,%f\n",latitude,longitude,p->elements[i]->altitude);
      }
      if((status=ADT_vector_set_footer(p,footer))!=OK)
            return status;
      fprintf(out,"%s\n",p->footer);
      return OK;
}
/*status_t export_as_CSV(ADT_vector_t *vec,FILE *fo)
{

      status_t st;
      double latitude,longitude;
      char *utc_timeCsv;
      size_t i;
```

```
      if(vec==NULL|| fo==NULL)
            return ERROR_NULL_POINTER_MSG;

      for(i=0; vec->elements[i];i++)
      {
            if((st=convert_UTCtime_CSV((vec->elements[i])->utc_time,
&utc_timeCsv))!=OK)
            return st;

            if((st=get_coordinates(vec->elements[i],&latitude,&longitude)) != OK)
            return st;

            fprintf(fo, "%s %c %lf %c %lf %c %i
\n",utc_timeCsv,CSV_DELIMITER,latitude,CSV_DELIMITER,longitude,CSV_DELIMITER,vec->el
ements[i]->altitude);
      }

      return OK;
}*/
```

4.10 Biblioteca <<gps.h>>

4.10.1 gps.h

```
#ifndef GPS__H
#define GPS__H
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "types.h"
#define ID_LENGTH 7
#define CHECK_SUM_LENGTH 4
#define MSG_ID_POS 0
#define UTC_TIME_POS 1
#define LATITUDE_POS 2
#define VERTICAL_ORIENTATION_POS 3
#define LONGITUDE_POS 4
#define HORIZONATAL_ORIENTATION_POS 5
#define FIXED_INDICATOR_POS 6
#define SATELLITE_NUMBER_POS 7
#define HDOP_POS 8
#define ALTITUDE_POS 9
#define ALTITUDE_UNIT_POS 10
#define GEOID_SEPARATION_POS 11
#define GEOID_SEPARATION_UNIT 12
```

```
#define AGE_OF_DIFF_CORR_POS 13
#define DIFF_REF_STATION_ID_POS 14
#define CHECK_SUM_POS 15

typedef struct
{
      char msg_id[ID_LENGTH];
      float utc_time;
      double latitude;
      char vertical_orientation;
      double longitude;
      char horizontal_orientation;
      int fixed_pos_indicator;
      size_t satellite_number;
      float hdop;
      float altitude;
      char altitude_unit;
      float geoid_separation;
      char geoid_separation_unit;
      size_t age_of_diff_corr;
      int diff_ref_station_id;
      char check_sum[CHECK_SUM_LENGTH];
}ADT_gps_record_t;
status_t ADT_gps_record_new_from_values(char * msg_id,size_t utc_time,double
latitude,char vertical_orientation,double longitude,char horizontal_orientation,int
fixed_pos_indicator,size_t satellite_number,float hdop,float altitude,char
altitude_unit,float geoid_separation,char geoid_separation_unit,size_t
age_of_diff_corr,int diff_ref_station_id,char * check_sum,ADT_gps_record_t**p);
status_t ADT_gps_record_delete(ADT_gps_record_t**p);
status_t swap(ADT_gps_record_t*a,ADT_gps_record_t*b);

#endif
```

4.10.2 gps.c

```
#include "gps.h"

status_t ADT_gps_record_new_from_values(char * msg_id,size_t utc_time,double
latitude,char vertical_orientation,double longitude,char horizontal_orientation,int
fixed_pos_indicator,size_t satellite_number,float hdop,float altitude,char
altitude_unit,float geoid_separation,char geoid_separation_unit,size_t
age_of_diff_corr,int diff_ref_station_id,char * check_sum,ADT_gps_record_t**p)
{
      if(p==NULL||msg_id==NULL||check_sum==NULL)
            return ERROR_NULL_POINTER;
      if (((*p)=(ADT_gps_record_t*)malloc(sizeof(ADT_gps_record_t)))==NULL)
            return ERROR_NO_MEMORY;
```

```
	strcpy((*p)->msg_id,msg_id);
	(*p)->utc_time=utc_time;
	(*p)->latitude=latitude;
	(*p)->vertical_orientation=vertical_orientation;
	(*p)->longitude=longitude;
	(*p)->horizontal_orientation=horizontal_orientation;
	(*p)->fixed_pos_indicator=fixed_pos_indicator;
	(*p)->satellite_number=satellite_number;
	(*p)->hdop=hdop;
	(*p)->altitude=altitude;
	(*p)->altitude_unit=altitude_unit;
	(*p)->geoid_separation=geoid_separation;
	(*p)->geoid_separation_unit=geoid_separation_unit;
	(*p)-> age_of_diff_corr=age_of_diff_corr;
	(*p)-> diff_ref_station_id=diff_ref_station_id;
	strcpy((*p)->check_sum,check_sum);
	return OK;
}
status_t ADT_gps_record_delete(ADT_gps_record_t**p)
{
	if (p==NULL)
		return ERROR_NULL_POINTER;
	free(*p);
	*p=NULL;
	return OK;
}
status_t swap(ADT_gps_record_t*a,ADT_gps_record_t*b)
{
	ADT_gps_record_t aux;
	if (a==NULL||b==NULL)
		return ERROR_NULL_POINTER;
	aux=*a;
	*a=*b;
	*b=aux;
	return OK;
}
```

4.11 Biblioteca <<errors.h>>

4.11.1 errors.h

4.11.2 errors.c

4.12 Biblioteca <<types.h>>

4.13 Biblioteca <<constants.h>>

4.14 Biblioteca <<config.h>>

4.15 Makefile

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

all: gpsviewer

gpsviewer: main.o errors.o utils.o gps.o vector.o format.o
	$(CC) $(CFLAGS) -o MODIFICACIONES main_modificaciones.o errors.o utils.o gps.o
vector.o

main.o: main.c main.h errors.h utils.h gps.h vector.h
	$(CC) $(CFLAGS) -o main.o -c main.c

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

utils.o: utils.c utils.h
	$(CC) $(CFLAGS) -o utils.o -c utils.c

gps.o: gps.c gps.h
	$(CC) $(CFLAGS) -o gps.o -c gps.c

vector.o: vector.c vector.h
	$(CC) $(CFLAGS) -o vector.o -c vector.c

format.o: format.c format.h
	$(CC) $(CFLAGS) -o format.o -c format.c

clear:
	-rm *.o
```

5.Bibliografía consultada

Ghezzi,C.,Jazayeri,M.,Mandrioli,D.,”Fundamentals of software engineering”, Prentice-Hall,1991.
Klein,I.,”El taller del escritor universitario”, Prometeo libros, 2007.
Deitel,P.,Deitel,H., “C How to Program ”, Pearson / Prentice-Hall, 2010