

Trabajo Práctico N°2

Procesador de datos de sistemas GNSS

Ezequiel Centofanti - 100941 - ezecentofanti@gmail.com

Kevin Michalewicz - 100978 - kmichalewicz@fi.uba.ar

Ulises Montenegro - 102921 - uli-m.98@live.com.ar

Fecha de entrega: 6 de diciembre de 2018

1. Estructura funcional del programa

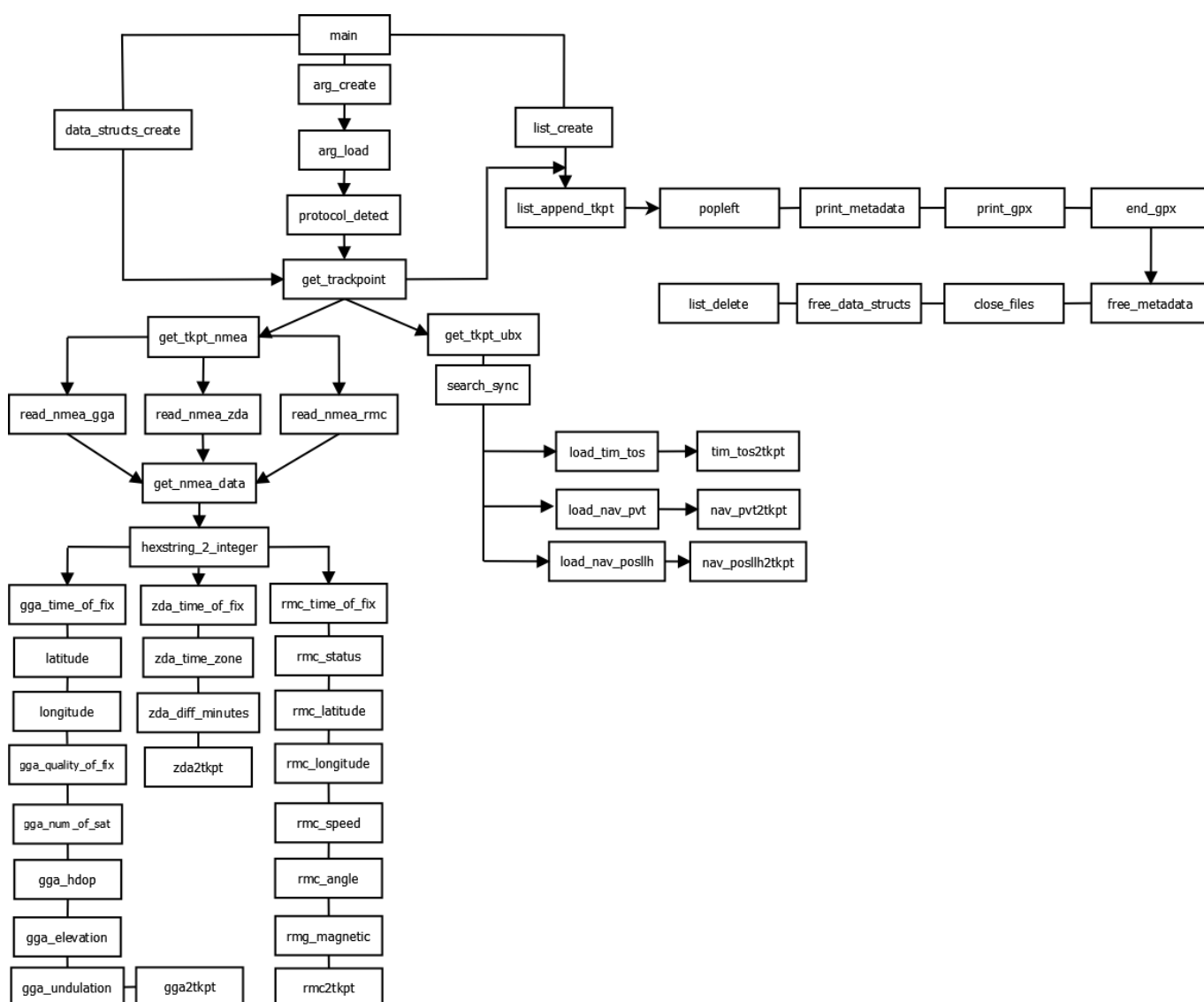


Figura 1: Estructura funcional del programa.

2. Alternativas consideradas y estrategias adoptadas

Antes de comenzar a escribir el código (y también durante el proceso) fue necesario tomar decisiones que permitieran considerar todos los casos posibles, ser ordenados y también eliminar ambigüedades.

En lo que respecta al ingreso de argumentos en línea de comandos, lo primero a ser revisado es si el usuario invocó la aparición del *help*. Caso contrario, se busca el argumento correspondiente al protocolo para definir en qué formato (de texto o binario) se abrirá el archivo de entrada. Nuevamente, se imprime el menú de *help* en caso de que este no sea ingresado o bien sea inválido. Si los argumentos *infile* y/o *outfile* tienen a continuación un guion medio, entonces por omisión se establecen respectivamente *stdin* y/o *stdout* como archivos de entrada/salida. Fue importante definir la estructura *data_structs_s* que poseía en su interior punteros a las estructuras más relevantes: *gga_s*, *rmc_s*, *zda_s*, *tim_tos_s*, *nav_pvt_s*, *nav_posllh_s* y *tkpt_s*. Esto permitió que los únicos argumentos pasados a las funciones *get_tkpt_nmea* y *get_tkpt_ubx* hayan sido un puntero a la estructura *arg_s* y otro a *data_structs_s*.

Para procesar el código *NMEA*, en primer lugar, se detectó para cada sentencia la presencia de *GGA*, *RMC* o bien *ZDA*. Detectar alguna de las dos últimas implicaba que se tuvieran datos temporales completos. Por ende, se definió un *time_flag* de tipo *bool_t* que tomaría el valor *TRUE*. En cambio, los primeros dos protocolos daban cuenta de información relacionada con un *trackpoint*; es así que se declaró una variable homónima. El hecho de que esta estuviera en *TRUE* daba la pauta de que la función *get_tkpt_nmea* estaba en condiciones de generar un *trackpoint*. Para llenar la estructura correspondiente se crearon funciones que tenían la utilidad de migrar datos de una estructura (*gga_s*, *rmc_s*, *zda_s*) a otra (*tkpt_s*). Los datos correspondientes a cada protocolo fueron extraídos de cada sentencia de la misma manera. Haciendo el cálculo de *checksum* en paralelo, se fueron reconociendo los distintos campos al leer hasta que apareciera el caracter delimitador (la coma, en este caso). De esta manera estos fueron alojados en un vector de cadenas, el cual sería utilizado luego para cargar las estructuras. El llamado a las demás funciones solo ocurriría si el *checksum* diera correctamente y, en todos los casos, al menos una de las cadenas del vector ya mencionado serían argumentos de las funciones. Para el cálculo de la suma de verificación se creó una función capaz de convertir los números en hexadecimal que figuran al final de cada sentencia en enteros en base 10. Esta cantidad luego se contrasta con una variable que resulta del XOR de todos los caracteres. Si no se cumpliera algún requisito, entonces la línea sería ignorada. Debido a que en muchos casos se tuvieron funciones llamadas dentro de otras funciones, se optó por que estas devolvieran por nombre el tipo enumerativo *status_t*. Esta serie de validaciones permitió que *get_tkpt_nmea* dé cuenta de la razón por la que una sentencia pudo no haber sido procesada correctamente, con lo cual al introducir los *logs* al programa se pudo hacer que el procedimiento *void* correspondiente fuese llamado únicamente desde aquella función.

Para procesar los archivos binarios *UBX*, se leyeron los *bytes* de sincronismo (*SYNC 1*, *SYNC 2*). En caso de encontrarse el primero, se veía que el siguiente fuera el segundo; caso contrario, el proceso se repetía. A continuación se analizó la *class*, en donde se notó que las sentencias de tipo *NAV* (*posllh* y *pvt*) poseían la misma. Esta fue una forma de descartar al menos un protocolo. El *byte ID* definía de cuál se trataba, aunque se admitió la posibilidad de que este sea inválido. Para aquel caso simplemente se desplegó un *warning* en *logfile*. La desventaja de esto último tuvo que ver con el hecho de analizar el largo del *payload* para conocer el protocolo de la sentencia. En todos los casos se leyó de a un *uchar*, cargando el dato en donde correspondiera. En el caso de leer un *unsigned* de 4 *bytes*, por ejemplo, usando el formato *Little Endian*, se realizó el cálculo $LSB * 1 + UCHAR.2 * 256 + \dots$. En cambio, para *signed* de 4 *bytes*, pasándolos *uchar* a binario se obtuvo un número de 32 *bits*. Sin perder noción del formato empleado, se observó si el *MSB* era un 0 o 1, pues en este último caso es necesario invertir y sumar uno para conocer el módulo. Por último, es importante aclarar que para ciertos bloques de información fue necesario definir máscaras y *shifts*.

Conforme se generaban los *trackpoints*, estos eran cargados en una lista mediante *list_append_tkpt*. Para saber cuántos procesar se generó un número aleatorio: 0, 1, 2 o 3. Cada opción indicaba cuántas trayectorias imprimir, debido a que podría ocurrir que demasiadas sentencias llegaran al sistema y que no fuera posible trabajar con todas. La opción del número 3 estaba asociada a *ALL*, es decir, a imprimir todos los *trackpoints* alojados en la lista hasta el momento. Luego de incluir el encabezado del código *GPX* (i.e. *metadata*), fueron impresas en el archivo de salida las líneas de código correspondientes a cada *trackpoint*. Para ello se utilizó la función *popleft*. Con la función *end_gpx* se escribieron los *tags* de cierre del código. Finalmente, se liberó la memoria pedida, se cerraron los archivos abiertos y se destruyó la lista.

3. Resultados de la ejecución

Se ejecutó el programa con distintos archivos de prueba para visualizar su funcionamiento. Estos incluyeron condiciones normales e inesperadas de entrada. En primer lugar se analiza lo ocurrido con el archivo *fiuba.nmea*:

<pre> <?xml version="6.66" encoding="UTF-8"?> <gpx version="9.99" creator="El_Vago" xmlns="http:// www.trivago.com.ar"> <metadata> <name>Prueba001</name> <time>2018-12-06T05:03:10Z</time> </metadata> <trk> </trkseg> <trkpt lat="-34.617687" lon="-58.368472"> <ele>0.000000</ele> <time>2018-09-30T20:23:11.0Z</time> </trkpt> <trkpt lat="-34.617687" lon="-58.368472"> <ele>0.000000</ele> <time>2018-09-30T20:23:11.0Z</time> </trkpt> <trkpt lat="-34.617267" lon="-58.368559"> <ele>0.000000</ele> <time>2018-09-30T20:23:11.0Z</time> </trkpt> <trkpt lat="-34.617267" lon="-58.368559"> <ele>0.000000</ele> <time>2018-09-30T20:23:12.0Z</time> </trkpt> <trkpt lat="-34.617109" lon="-58.368752"> <ele>0.000000</ele> <time>2018-09-30T20:23:12.0Z</time> </trkpt> <trkpt lat="-34.617109" lon="-58.368752"> <ele>0.000000</ele> <time>2018-09-30T20:23:13.0Z</time> </trkpt> <trkpt lat="-34.617180" lon="-58.369364"> <ele>0.000000</ele> <time>2018-09-30T20:23:13.0Z</time> </pre>	<pre> The current time has been updated The current date has been updated The current time has been updated Trackpoint generated successfully Trackpoint printed successfully Ignored statement The current time has been updated The current date has been updated Trackpoint generated successfully Trackpoint printed successfully The current time has been updated The current date has been updated The current time has been updated Trackpoint generated successfully Ignored statement The current time has been updated The current date has been updated Trackpoint generated successfully Trackpoint printed successfully The current time has been updated The current date has been updated The current time has been updated Trackpoint generated successfully Trackpoint printed successfully The current time has been updated The current date has been updated Trackpoint generated successfully Trackpoint printed successfully Ignored statement The current time has been updated The current date has been updated Trackpoint generated successfully Trackpoint printed successfully The current time has been updated The current date has been updated Trackpoint generated successfully Trackpoint printed successfully Ignored statement The current time has been updated </pre>	<pre> GPX: 202318.400.A.3437.018.5.05822.046.W.234.1.265.1.300918.000.0.W*79 GPZDA: 202318.00.30.09.2018.-3.00*73 GPMSA: 202319.400.3437.025.5.05822.124.W.1.12.1.0.0.0.M.0.0.M.*60 GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPPRMC: 202319.400.A.3437.025.5.05822.124.W.234.1.265.1.300918.000.0.W*73 \$regional\$-FS-5736:19 \$info1 clear \$regional\$-FS-5736:19 \$info1 cat fiuba.nmea ./main -n Prueba001 -i -o outfile.txt -l logfile.txt -p nmea GPZDA: 202311.00.30.09.2018.-3.00*7A GPMSA: 202311.400.3437.060.5.05822.107.W.1.12.1.0.0.0.M.0.0.M.*60 GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPPRMC: 202311.400.A.3437.060.5.05822.107.W.096.9.352.5.300918.000.0.W*78 GPZDA: 202311.00.30.09.2018.-3.00*7A GPMSA: 202312.400.3437.033.5.05822.111.W.1.12.1.0.0.0.M.0.0.M.*6A GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPPRMC: 202312.400.A.3437.033.5.05822.111.W.049.9.303.2.300918.000.0.W*7B GPZDA: 202312.00.30.09.2018.-3.00*7B GPMSA: 202313.400.3437.024.5.05822.124.W.1.12.1.0.0.0.M.0.0.M.*6B GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPMSA: 202313.400.3437.024.5.05822.124.W.111.7.262.3.300918.000.0.W*7F GPPRMC: 202313.00.30.09.2018.-3.00*7B GPZDA: 202314.400.3437.029.5.05822.161.W.1.12.1.0.0.0.M.0.0.M.*60 GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPMSA: 202314.400.A.3437.029.5.05822.161.W.133.9.192.6.300918.000.0.W*73 GPZDA: 202314.00.30.09.2018.-3.00*7F GPMSA: 202315.400.3437.066.5.05822.169.W.1.12.1.0.0.0.M.0.0.M.*62 GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPMSA: 202315.400.A.3437.066.5.05822.169.W.140.6.157.6.300918.000.0.W*73 GPZDA: 202315.00.30.09.2018.-3.00*7E GPMSA: 202316.400.3437.103.5.05822.154.W.1.12.1.0.0.0.M.0.0.M.*6D GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPMSA: 202316.400.3437.103.5.05822.154.W.348.7.004.5.300918.000.0.W*7B GPZDA: 202316.00.30.09.2018.-3.00*7D GPMSA: 202317.400.3437.092.5.05822.037.W.1.12.1.0.0.0.M.0.0.M.*61 GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPMSA: 202317.400.A.3437.092.5.05822.037.W.267.0.353.4.300918.000.0.W*74 GPZDA: 202317.00.30.09.2018.-3.00*7C GPMSA: 202318.400.3437.018.5.05822.046.W.1.12.1.0.0.0.M.0.0.M.*6A GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPMSA: 202318.400.3437.018.5.05822.046.W.234.1.265.1.300918.000.0.W*79 GPZDA: 202318.00.30.09.2018.-3.00*73 GPMSA: 202319.400.3437.025.5.05822.124.W.1.12.1.0.0.0.M.0.0.M.*60 GPMSA: 3.01.02.03.04.05.06.07.08.09.10.11.12.1.0.1.0.1.0*30 GPMSA: 202319.400.A.3437.025.5.05822.124.W.234.1.265.1.300918.000.0.W*73 \$regional\$-FS-5736:19 \$info1 </pre>
---	--	--

Figura 2: Archivo de prueba: *fiuba.nmea*. El archivo de entrada es *stdin*

En la imagen anterior se observa el ingreso de la opción correspondiente a un archivo de entrada por omisión; se lee de *stdin*. A su vez, se pueden observar los archivos de salida con el código *GPX* y los *logs*.

```

$ ./main -n Prueba001 -i fiuba.nmea -o - -l logfile.txt -p auto
run with -h or --help to view help file.
<?xml version="6.66" encoding="UTF-8"?>
<gpx version="9.99" creator="El_Vago" xmlns="http://www.trivago.com.ar">
  <metadata>
    <name>Prueba001</name>
    <time>2018-12-06T05:05:33Z</time>
  </metadata>
  <trk>
    <trkseg>
      <trkpt lat="-34.617668" lon="-58.368332">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:11.400Z</time>
      </trkpt>
      <trkpt lat="-34.617668" lon="-58.368332">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:11.400Z</time>
      </trkpt>
      <trkpt lat="-34.617168" lon="-58.368500">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:12.400Z</time>
      </trkpt>
      <trkpt lat="-34.617168" lon="-58.368500">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:12.400Z</time>
      </trkpt>
      <trkpt lat="-34.617001" lon="-58.368668">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:13.400Z</time>
      </trkpt>
      <trkpt lat="-34.617001" lon="-58.368668">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:13.400Z</time>
      </trkpt>
      <trkpt lat="-34.617001" lon="-58.369335">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:14.400Z</time>
      </trkpt>
      <trkpt lat="-34.617001" lon="-58.369335">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:14.400Z</time>
      </trkpt>
    </trkseg>
  </trk>
</gpx>

```

Figura 3: Archivo de prueba: *fiuba.nmea*. El archivo de salida es *stdout*.

La figura anterior presenta el mismo contenido, solo que ahora se ha establecido por omisión a *stdout* como archivo de salida, en tanto que el archivo original *fiuba.nmea* es ingresado luego del indicador *i*. A continuación se tiene el mismo caso pero para un archivo binario *UBX*. Es importante notar que el campo de protocolo está en *auto*, lo que significa que el programa deberá reconocer si se tienen sentencias *NMEA* o bien *UBX*.

```

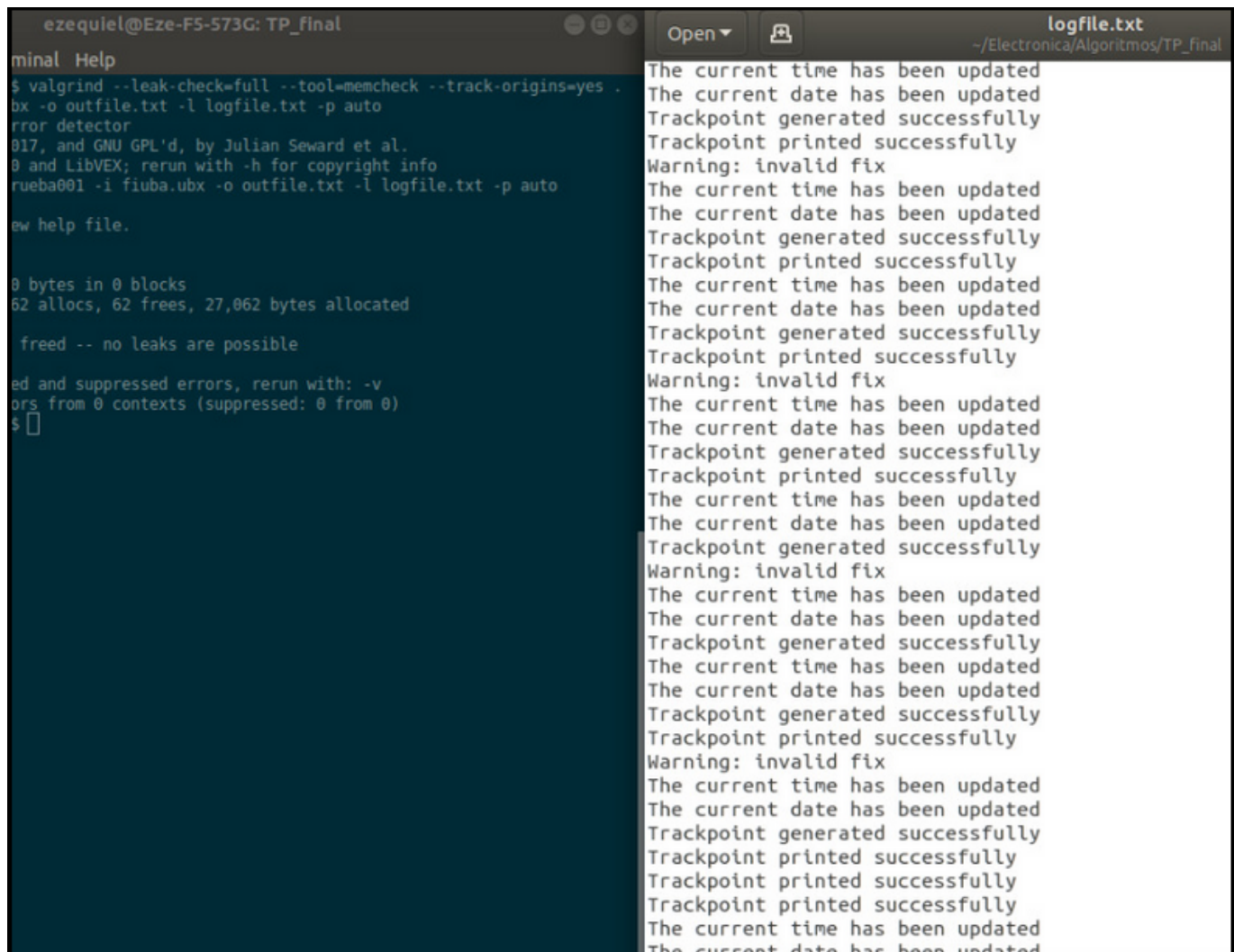
ezequiel@eze-F5-5736:TP_final$ ./main -n Prueba001 -i fiuba.ubx -o - -l logfile.txt -p auto
rerun with -h or --help to view help file.
0
<?xml version="6.66" encoding="UTF-8">

<gpx version="9.99" creator="El_Vago" xmlns="http://www.trivago.com.ar">
  <metadata>
    <name>Prueba001</name>
    <time>2018-12-06T05:05:08Z</time>
  </metadata>
  <trk>
    <trkseg>
      <trkpt lat="-34.617687" lon="-58.368472">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:11.0Z</time>
      </trkpt>
      <trkpt lat="-34.617687" lon="-58.368472">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:11.0Z</time>
      </trkpt>
      <trkpt lat="-34.617267" lon="-58.368559">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:11.0Z</time>
      </trkpt>
      <trkpt lat="-34.617267" lon="-58.368559">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:12.0Z</time>
      </trkpt>
      <trkpt lat="-34.617109" lon="-58.368752">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:12.0Z</time>
      </trkpt>
      <trkpt lat="-34.617109" lon="-58.368752">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:13.0Z</time>
      </trkpt>
      <trkpt lat="-34.617180" lon="-58.369364">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:13.0Z</time>
      </trkpt>
      <trkpt lat="-34.617180" lon="-58.369364">
        <ele>0.000000</ele>
        <time>2018-09-30T20:23:14.0Z</time>
      </trkpt>
    </trkseg>
  </trk>
</gpx>

```

Figura 4: Archivo de prueba: *fiuba.ubx*.

En la figura siguiente se hace foco en los *logs* de otro archivo *UBX*.



```
ezequiel@Eze-F5-573G: TP_final
minial Help
$ valgrind --leak-check=full --tool=memcheck --track-origins=yes .
bx -o outfile.txt -l logfile.txt -p auto
rror detector
017, and GNU GPL'd, by Julian Seward et al.
0 and LibVEX; rerun with -h for copyright info
rueba001 -i fiuba.ubx -o outfile.txt -l logfile.txt -p auto

ew help file.

0 bytes in 0 blocks
62 allocs, 62 frees, 27,062 bytes allocated

freed -- no leaks are possible

ed and suppressed errors, rerun with: -v
ors from 0 contexts (suppressed: 0 from 0)
$
```

```
logfile.txt
~/Electronica/Algoritmos/TP_final
The current time has been updated
The current date has been updated
Trackpoint generated successfully
Trackpoint printed successfully
Warning: invalid fix
The current time has been updated
The current date has been updated
Trackpoint generated successfully
Trackpoint printed successfully
The current time has been updated
The current date has been updated
Trackpoint generated successfully
Trackpoint printed successfully
Warning: invalid fix
The current time has been updated
The current date has been updated
Trackpoint generated successfully
Trackpoint printed successfully
The current time has been updated
The current date has been updated
Trackpoint generated successfully
Trackpoint printed successfully
Warning: invalid fix
The current time has been updated
The current date has been updated
Trackpoint generated successfully
Trackpoint printed successfully
Trackpoint printed successfully
Trackpoint printed successfully
The current time has been updated
The current date has been updated
```

Figura 5: Archivo de *logs*: Se obtiene un seguimiento de lo que ocurre en la ejecución de un archivo *UBX*.

Ya habiendo analizado ejecuciones exitosas para cada caso, también existe la posibilidad de que el usuario se equivoque al ingresar los argumentos por línea de comandos. En ese caso la siguiente pantalla aparecerá:

```
LIST OF COMMAND-LINE VALID COMMANDS
-h -n -p -i -o -l -m
--help --name --protocol --infile --outfile --logfile --maxlen

DESCRIPTION OF ARGUMENTS
[-h, --help]
Shows help.
[-n, --name]
Indicates metadata name.
[-p, --protocol]
Indicates the protocol. It can be a NMEA, UBX or AUTO.
[-i, --infile]
Indicates input file name. If argument is only '-', input file is stdin. In case this argument is not entered, program returns error.
[-o, --outfile]
Indicates output file name. If argument is only '.', output file is stdout. In case this argument is not entered, program returns error.
[-l, --logfile]
Indicates file name which will be used for logs. If argument is only '-', logfile is stderr. In case this argument is not entered, program returns error.
[-m, --maxlen]
Indicates maximum messages amount that can be stored in a list.
```

Figura 6: Archivo de prueba: *help.txt*. Se imprime el menu de ayuda.

Para que no tenga que probar muchos argumentos y adivinar cuáles hacen que el programa funcione, el usuario también tendrá a su disposición una lista de ejemplos válidos e inválidos de los mismos.

```
IMPLEMENTATION

Valid Examples:
-h
--help
-n monkey
--name monkey
-p nmea
--protocol ubx
-i fiuba.ubx
--infile jungle.nmea
-o -
--outfile output_file.gpx
-l logs.txt
--logfile notifications.txt
-m 100
--maxlen 50

Invalid Examples:
-help
--h
-n monkey%%#
--name
-p
--protocol monkey
-infile safari.ubx
--infile
-outfile
--o of.ubx
-logfile
--l logs.txt
-m a12
--maxlen 100Q
```

Figura 7: Ejemplos de argumentos válidos e inválidos.

A continuación se ve el caso de un error por archivo inexistente. Automáticamente se imprime el *help*.

```

ezequiel@Eze-P5-5736:TP_final$ gcc -std=c99 -Wall -pedantic main.c -o main -g
ezequiel@Eze-P5-5736:TP_final$ ./main -i fiuba_inexistente.ubx -o outfile.txt -l - -p auto
rerun with -h or --help to view help file.

LIST OF COMMAND-LINE VALID COMMANDS
-h -n -p -i -l -l -m
--help --name --protocol --infile --logfile --logfile --maxlen

DESCRIPTION OF ARGUMENTS
[-h, --help]
Shows help.
[-n, --name]
Indicates metadata name.
[-p, --protocol]
Indicates the protocol. It can be a NMEA, UBX or AUTO.
[-i, --infile]
Indicates input file name. If argument is only '-', input file is stdin. In case this argument is not entered, program returns error.
[-l, --logfile]
Indicates output file name. If argument is only '-', output file is stdout. In case this argument is not entered, program returns error.
[-l, --logfile]
Indicates file name which will be used for logs. If argument is only '-', logfile is stderr. In case this argument is not entered, program returns error.
[-m, --maxlen]
Indicates maximum messages amount that can be stored in a list.

IMPLEMENTATION

Valid Examples:
-h
--help
-n monkey
--name monkey
-p nmea
--protocol ubx
-i fiuba.ubx
--infile jungle.nmea
-o -
--outfile output_file.gpx
-l logs.txt
--logfile notifications.txt
-m 100
--maxlen 50

Invalid Examples:
-help
-h

```

Figura 8: Archivo de prueba inexistente. Se obtiene un error.

4. Problemas encontrados y sus soluciones

Un primer problema encontrado fue el de ordenar los *logs* de la manera más prolija y entendible posible. Fue un tema con una sencilla solución pues, al incluirse *logfile* en la estructura *arg_s*, se pudo acceder mediante la notación en forma de flecha. Por otro lado, hubo problemas a la hora de analizar bloques de datos de 4 *bytes* en las sentencias *UBX*; la razón por la que esto fue dificultoso se detalla en la sección 2. Esta complicación no existió para los archivos de texto (*NMEA*). En ese caso se tuvo que la manera de procesar una cantidad diferente de campos (según el protocolo) y, además, independizarse (por una cuestión de simplicidad) de las longitudes de cada uno de ellos, era incompatible con la estrategia adoptada en el TP1. Fue necesario leer la documentación de la estructura *tm* pues la forma de contar los meses y años no es trivial. Se obtuvieron datos incorrectos de aquellos parámetros hasta que se revisaron las características de *tm_year* y *tm_mon*. Las variables *south_flag* y *west_flag* fueron útiles para multiplicar por menos uno a la latitud y longitud respectivamente (si correspondiera). Antes de eso se crearon funciones solo para realizar dicha multiplicación pero fueron eliminadas por su poca eficiencia. Por último, en ocasiones se obtuvo fuga de memoria que fue subsanada con el uso del programa *valgrind*. En resumen, modularizar el programa, unificar criterios y emplear los conocimientos adquiridos de una manera exitosa fue el mayor desafío de este TPF.

5. Fugas de memoria y *valgrind*

La aplicación *valgrind* es capaz de advertir las fugas de memoria del programa. Para que este detalle dónde se producen fugas de memoria, es necesario compilar nuestra aplicación en modo *debug*.


```

ezequiel@eze-F5-5736:TP_final$ valgrind --leak-check=full --tool=memcheck --track-origins=yes ./prueba_de_func -n Prueba001 -p auto -i travesia.nmea -o outfile.txt -l logfile.txt
==5134== Memcheck, a memory error detector
==5134== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5134== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5134== Command: ./prueba_de_func -n Prueba001 -p auto -i travesia.nmea -o outfile.txt -l logfile.txt
==5134==
==5134== HEAP SUMMARY:
==5134==   in use at exit: 0 bytes in 0 blocks
==5134==   total heap usage: 8,081 allocs, 8,081 frees, 475,202 bytes allocated
==5134==
==5134== All heap blocks were freed -- no leaks are possible
==5134==
==5134== For counts of detected and suppressed errors, rerun with: -v
==5134== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 9: Resultado de emplear valgrind para el ejemplo travesía.nmea

```

ezequiel@eze-F5-5736:TP_final$ valgrind --leak-check=full --tool=memcheck --track-origins=yes ./prueba_de_func -n Prueba001 -p auto -i travesia.ubx -o outfile.txt -l logfile.txt
==5056== Memcheck, a memory error detector
==5056== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==5056== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==5056== Command: ./prueba_de_func -n Prueba001 -p auto -i travesia.ubx -o outfile.txt -l logfile.txt
==5056==
==5056== HEAP SUMMARY:
==5056==   in use at exit: 0 bytes in 0 blocks
==5056==   total heap usage: 8,081 allocs, 8,081 frees, 475,161 bytes allocated
==5056==
==5056== All heap blocks were freed -- no leaks are possible
==5056==
==5056== For counts of detected and suppressed errors, rerun with: -v
==5056== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)

```

Figura 10: Resultado de emplear valgrind para el ejemplo travesía.ubx

El módulo principal compila sin ningún *warning* y sin fugas de memoria. Para compilar dicho archivo es necesario enlazarlo con los modulos restantes.

6. Bibliografía

- [1] P. Deitel y H. Deitel. C How to Program. 7ma Edición. Pearson Education, 2012. ISBN 9780133061567.
- [2] B.W. Kernighan y D.N. Ritchie The C Programming Language. 2da Edición. Prentice Hall software series. Prentice Hall, 1988. ISBN 9780131103627.