ESCUELA SUPERIOR DE INGENIERÍA Y TECONOLOGÍA

Trabajo Fin de Grado

DESARROLLO DE UN SOFTWARE PARA GESTIONAR UN CAPTADOR SECUENCIAL DE AEROSOL DESDE UN PC Y CONTROL REMOTO

Titulación: Grado en Ingeniería Electrónica Industrial y Automática

Autor: David Valdivieso López

Tutores: Santiago Torres Álvarez, Sergio Rodríguez González

Septiembre, 2016

ESCUELA SUPERIOR

DE INGENIERÍA Y TECONOLOGÍA

TITULACIÓN: Grado en Ingeniería Electrónica Industrial y Automática

Memoria

Trabajo Fin de Grado

TÍTULO

DESARROLLO DE UN SOFTWARE PARA GESTIONAR UN CAPTADOR SECUENCIAL DE AEROSOL DESDE UN PC Y CONTROL REMOTO

AUTOR

David Valdivieso López

TUTORES

Santiago Torres Álvarez

Sergio Rodríguez González

ÍNDICE

1 ABSTRACT

Currently at the Agencia Estatal de Metereología in Izaña (Tenerife), there is a project that takes part in the Global Atmospheric Watch program (GAW) which has the objective of studying de physical and chemical properties of atmospheric particles that directly impact in the quality of the air and weather. In order to get the data for the project, workers at Izaña use a range of system and methods to get the needed data.

For this project, we are focusing in one instrument (aerosol sequential volume sampler) by using different filters to do its labour. They use different samplers to get particles of different volumes stored into the machine's filters. Once the particles are inside the device, they transport the filters to a laboratory and save the data of the process (date of the process, environment volume aspirated, temperature, pressure…) in a file.

At the moment, the workers need to manipulate in person the device to get the data from its display, having to write the results in a notebook and transfer them later to their database in an excel document.

Our goal for this project will be to improve the efficiency for the workers who manipulate the device by creating a program that will allow to plan measuring dates in the device, extract the results directly from it to a PC and manipulate it remotely.

2 ASPECTOS GENERALES

2.1 Objeto

2.1.1 Objetivos técnicos

Desarrollar un programa capaz de comunicarse con los dispositivos captadores emplazados en la sede de la Agencia Estatal de Meteorología (AEMET) en Izaña (Tenerife), de manera

que se puedan extraer los datos del dispositivo e introducir comandos para regular la actividad de los mismos.

2.1.2 Objetivos administrativos

Aprobar la asignatura “Trabajo Fin de Grado” para la consecución del título de grado en Ingeniería Industrial Electrónica y Automática.

2.2 Antecedentes

En la AEMET en Izaña se realizan muestreos con captadores de aerosoles con el objetivo de realizar un seguimiento a largo plazo (décadas) de las propiedades fisicoquímicas de las partículas atmosféricas que afectan a la calidad del aire y al clima, en la troposfera libre del Atlántico Norte Subtropical. Para alcanzar este objetivo, se realiza una monitorización en continuo de la distribución de tamaño, la composición química y de las propiedades ópticas de las partículas. Este proyecto forma parte del programa de Vigilancia Atmosférica Global (siglas en ingles GAW = *Global Atmospheric Watch*) de la Organización Mundial de Meteorología. [1]

Figura 1.1 Captadores secuenciales de aerosoles con los que nos vamos a comunicar.

Para realizar dichas mediciones, se poseen diferentes instrumentos con los que realizar medidas a diferentes rangos de tamaño de partículas y metodologías. Para nuestro proyecto, vamos a trabajar con los captadores secuenciales de aerosoles que aparecen en la Figura1.1.

Estos dispositivos cuentan con unas cámaras cilíndricas donde se colocan los filtros sobre los que hacen los muestreos. Estos filtros deben estar, antes de ser introducidos en las cámaras, dos días en condiciones de temperatura y presión controlada para homogenizar humedad si la hubiera. A través de unos tubos que tienen colocados (que van desde fuera del edificio hasta la cámara de los filtros), acceden las partículas con tamaños de 2.5 micras y 10 micras(cada uno a su respectivo captador).

El muestreo se hace durante el tiempo en el que se programa el dispositivo, con un caudal y temperatura también prefijados. Una vez realizado el muestreo, se debe dejan los filtros reposar dos días y se llevan al laboratorio para calcular la concentración de la masa mediante gravimetría y a partir de los resultados obtenidos en el captador. Las muestras son finalmente llevadas a otro laboratorio en Barcelona donde se determinan las concentraciones de los diferentes aerosoles en los filtros (material mineral, sal marina, sulfato, nitrato, amonio, materia orgánica, *black* carbón y un amplio número de elementos traza).

2.3 Descripción y objetivo

Actualmente se realiza un trabajo monótono para extraer y anotar manualmente todos los datos obtenidos de los proceso de muestreo por parte de los captadores secuenciales de aerosoles. Es necesario estar presente el día anterior para poder programar el dispositivo a pesar de no ser necesario extraer las muestras hasta un número de muestreos determinados.

Por lo tanto, el presente proyecto tiene en sí dos partes beneficiosas para los manipuladores de los captadores de aerosoles:

- En primer lugar, a partir de la conexión serial entre un PC y el dispositivo se podría, según se muestra en la documentación aportada, extraer los datos de manera automática y programar el instrumento usando los comandos adecuados con el protocolo que se ha proporcionado. Para ello, se deberá de comprobar las conexiones necesarias que hay que realizar entre el PC y el dispositivo.

- En segundo lugar, en caso de conseguir que se pudiera comunicar remotamente con el programa o que este pudiese trabajar independientemente, podría evitarse que se subiera específicamente para la recolección de datos y preparación del instrumento cada vez que haya que realizar una prueba.

Figura 1.2 Dispositivo principal del Captador secuencial de aerosoles.

El proyecto que aquí se propone tiene como objetivo el desarrollo de un software para poder programar los captadores secuenciales de muestreos de aerosoles en Izaña desde un PC remoto usando un puerto de comunicaciones. Este software debe leer los datos de entrada desde un fichero, programar el captador y volcar los datos de salida (una vez terminado cada muestreo) en otro fichero. En un principio se ideó la realización de un software que se permitiese manipular remotamente el dispositivo.

Por problemas con la conectividad, la solución provisional será de prefijar las fechas a través del software leyendo un archivo concreto, permitiendo además la extracción datos y programación del dispositivo de manera *in situ*.

2.4 Motivación

Actualmente los sistemas de medición son cada vez más sofisticados y son muchos los detalles que tenemos en cuenta a la hora de utilizar un equipo u otro para realizar una medición. Si miramos al pasado, los métodos eran mucho más rudimentarios, toscos y lentos. A pesar de tener hoy en día herramientas cada vez más avanzadas, siempre existirán avances y mejoras que podamos hacer para recoger y analizar estas muestras de manera más rápida, cómoda y eficaz.

El uso de sistemas informatizados ha contribuido y beneficiado en gran parte al aumento del ritmo con el que se consiguen estas mejoras. La utilización de procesadores (como ordenadores) en estos procesos ha sido fundamental para la creación de bases de datos con las que trabajar junto a los instrumentos de medición. Es por ello por lo que ahora el software es una parte vital cuando hace falta una gran cantidad de mediciones y de cálculos avanzados.

Por ello, es importante conocer a la hora de realizar un estudio, con qué estamos trabajando, qué instrumentos tenemos a nuestra disposición para llevar a cabo dicho estudio y cómo vamos a gestionar estos resultados.

Con este proyecto pretendemos mejorar las condiciones y la eficiencia de los trabajadores que manipulan el captador de aerosoles de manera que, a partir de los conocimientos adquiridos en la carrera y ampliados gracias a la búsqueda de información durante el proyecto, se pueda contribuir a mejorar el desempeño con estos dispositivos en Izaña.

3 BASE DE LA COMUNICACIÓN SERIAL

3.1 Hardware para la comunicación serial

3.1.1 Introducción

Para el funcionamiento del proyecto, es fundamental conocer las partes tanto de hardware como de software que intervienen en él. Es importante saber que estamos trabajando con los elementos adecuados para que el mensaje llegue correctamente a su destino.

En este capítulo explicaremos la función de cada interfaz de comunicación y su integración en el sistema.

3.1.2 Comunicación serie

Cuando hablamos de *comunicación en serie* o *comunicación serial*, nos referimos a la forma de envío de datos uno detrás de otro o en serie. La comunicación en serie que utilizaremos es la basada en RS-232 que se basa en este fundamento.

RS-232 (*Recommended Standard 232*) es una interfaz que designa una norma para el intercambio serie de datos binarios entre un DTE (*Data Terminal Equipment*) y un DCE (*Data Communication Equipment*).

En particular, existen ocasiones en que interesa conectar otro tipo de equipamientos, como pueden ser computadores. Evidentemente, en el caso de interconexión entre los mismos, se requerirá la conexión de un DTE con otro DTE. Para ello se utiliza una conexión entre los dos DTE sin usar modem, por ello se llama: *null modem*.

3.1.3 Puerto serie RS-232

En la década de 1960, la actualmente conocida como la EIA (*Electronic Industries Association*), desarrolló un estándar de interfaz de comunicación para el intercambio de datos en equipos de comunicaciones. En ese momento, se pensó en el sentido del intercambio de datos digitales entre un ordenador central y una terminal de computadora a distancia, o de entre dos terminales sin necesidad de un maestro. Estos dispositivos fueron vinculados por líneas de teléfono, y por lo que requerían un módem en cada extremo para la

traducción de la señal. Existen muchas posibilidades de error en los datos que se producen durante la transmisión de datos a través de un canal analógico. Por ello, se pensó una norma que garantizara una comunicación fiable, y en segundo lugar, que permitiera la interconexión de los equipos fabricados por diferentes fabricantes, fomentando así los beneficios de la producción en masa y la competencia. A partir de estas ideas, nació el estándar RS-232.[2]

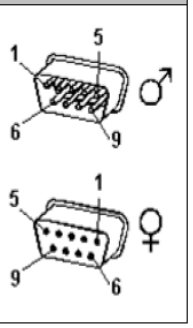
En 1969, la interfaz RS-232-C fue desarrollada por la EIA en cooperación con el sistema BELL y fabricantes independientes de computadores y módems. Desde entonces, se han publicado varias modificaciones, siendo la más reciente la norma EIA-232-F introducido en 1997,además de cambiar el nombre de RS-232 a EIA-232.

3.1.3.1 Especificaciones

Hablando de las especificaciones mecánicas, el conector RS-232 puede consistir o bien en un conector tipo de DB-25 (25 pines), o en nuestro caso y como es más normal de encontrar, en una versión de 9 pines (DB-9).

El conector RS-232 utiliza voltaje negativo para el 1 lógico (-3 a -25 voltios siendo receptor y de -5 a -25 voltios siendo emisor), positivo para el 0 lógico (de 3 a 25 voltios siendo receptor y de 5 a 25 voltios siendo emisor) y un pequeño margen no definido entre ambos límites (de -3 a 3 voltios en el receptor y de -5 a 5 para el emisor).

Las señales TXD (línea de transmisión de Datos), DTR (terminal de datos listo) y RTS(terminal de datos listo para enviar) son de salida, mientras que RXD(línea de recepción de datos), DSR(modem de datos listo) y CTS(modem de datos listo para recibir) son de entrada. La tierra de referencia para todas las señales es SG (Tierra de Señal). Finalmente, existen otras señales como RI (timbre telefónico).

#	Pin	E/S	Función	Conector DB 9
1			Tierra de Chasis	
2	RXD	E	Recibir Datos	
3	TXD	S	Transmitir Datos	
4	DTR	S	Terminal de Datos Listo	
5	SG		Tierra de señal	
6	DSR	E	Equipo de Datos Listo	
7	RTS	S	Solicitud de Envío	
8	CTS	E	Libre para Envío	
9	RI	S	Timbre Telefónico	

Las señales de cada pin tienen el siguiente uso en el DB-9 :

Pin 1 – Tierra de chasis: Se conecta internamente al chasis del dispositivo.

Pin 2 – RXD: Por este pin se reciben los datos del dispositivo externo.

Pin 3 – TXD: A través de este pin se envían los datos al dispositivo externo.

Pin 4 – DTR: Este pin realiza el control maestro del dispositivo externo. Cuando este pin está en 1 lógico, el dispositivo externo no transmite ni recibe datos.

Pin 5 – SG: Con la tierra de señal se referencian los voltajes para comprobar si los tenemos a un valor positivo o negativo.

Pin 6 – DSR: Por lo general, los dispositivos externos mantienen este pin con un valor de 0 lógico. Con ello se determina que el módem está listo.

Pin 7 - RTS: A través de este pin, el terminal indica que está listo para enviar datos y solicita el permiso, realizando lo que se conoce como *handshake*. Si el dispositivo externo acepta la petición, pone a 0 lógico el pin correspondiente al CTS.

Pin 8 – CTS : Corresponde a la otra parte que realiza el *handshake* como nombramos en el pin RTS. El dispositivo externo pone a 0 lógico este pin para iniciar la comunicación.

Pin 9 – RI: Timbre telefónico. Sólo se una cuando el terminal se conecta a un PLC (*Programable Logical Controller*).[3]

La conexión finalmente entre dos terminales (PC1 y PC2) se haría de la siguiente manera :

Terminal 1 Terminal 2

RxD ←==========→ TxD

TxD ←==========→ RxD

DTR ←==========→ DSR

DSR ←==========→ DTR

RTS ←==========→ CTS

CTS ←==========→ RTS

Tierra ←==========→ Tierra

3.1.3.2 Principio de funcionamiento

El puerto serie es un tipo de conexión muy extendido y ya sea por uno o dos puertos, con conector grande o pequeño, todos los equipos PC lo incorporan actualmente.

La comunicación realizada con el puerto serial es una comunicación asíncron*a*: para este tipo de comunicación no son necesarios los pulsos de reloj (como sí es necesario en la síncrona, donde la velocidad de transmisión se basa en ellos). En este caso, el emisor y el receptor establecen de forma predeterminada una velocidad para el envío y recepción de datos.

Para la sincronización de una comunicación se precisa siempre de un bit adicional a través del cual el emisor y el receptor intercambian la señal del pulso. Pero en la transmisión serial a través de un cable de dos líneas esto no es posible, ya que ambas están ocupadas por los datos y la tierra. Por este motivo se intercalan antes y después los datos de información de estado según el protocolo RS-232.

Esta información es determinada por el emisor y el receptor al estructurar la conexión mediante la correspondiente programación de sus puertos seriales. La estructura de la configuración que se usa es la siguiente:

- Bit de inicio: Cuando el receptor detecta el bit de inicio sabe que la transmisión ha comenzado y es a partir de entonces que debe leer la transmisión y las señales de la línea a distancias concretas de tiempo, en función de la velocidad determinada.

- Bits de datos: Indica el número de datos que se incluyen para un paquete de envío. Se pueden transmitir grupos de 5 a 8 bits de datos, pero suele ser lo más común una configuración de 7 u 8 bits en cada paquete.
- Bit de paridad: Con este bit se puede descubrir errores en la transmisión. Se puede dar paridad par o impar. En la paridad par, la palabra de datos a transmitir se completa con el bit de paridad de manera que el número de bits 1 enviados es par.
- Bit de parada: Indica la finalización de la transmisión de una palabra de datos. El protocolo de transmisión de datos permite 1, 1.5 y 2 bits de parada. [3]

3.1.3.3 Control del puerto Serial. Handshaking.

El ordenador controla el puerto serial mediante un circuito integrado específico llamado UART *(Universal Asynchronous Receptor Transmitter*). Para controlarlo, el CPU emplea direcciones de puertos de E/S, mediante los cuales se pueden intercambiar datos, y líneas de interrupción (IRQ), que producen una interrupción para indicar a la CPU que ha ocurrido un evento (por ejemplo, que ha llegado un dato o que ha cambiado el estado de algunas señales de entrada). La CPU debe responder a estas interrupciones lo más rápido posible para que dé tiempo a recoger el dato antes de que el siguiente se sobrescriba.

El puerto RS-232 puede transmitir los datos en grupos de 5 a 8 bits a unas velocidades determinadas (lo que conocemos por *Baud Rate*, que son el número de bits transmitidos por segundo). Normalmente, el protocolo de envío de datos utilizado es el denominado *8N1* (que significa 8 bits de datos, sin paridad y 1 bit de parada).

Una vez comenzada la transmisión de un dato, los bits tienen que llegar uno detrás de otro a una velocidad constante (previamente determinada como hemos dicho)

y en determinados instantes de tiempo. Es por esto por lo que denominamos el RS-232 como comunicación asíncrona.

La velocidad del puerto serial no tiene por qué ser la misma que la de transmisión de los datos, de hecho debe de ser superior. Por ejemplo, para transmisiones de 1200 baudios es recomendable usar 9600, y para 9600 se pueden usar desde 19200 hasta 38400.[2]

El envío de información se produce a través de algo que denominamos *handshaking*:

En principio, el método de comunicación usado por RS-232 utiliza sólo tres líneas: TxD, RxD y Tierra. Sin embargo, para que los datos puedan ser transmitidos correctamente ambos extremos deben estar sincronizados a la misma velocidad. Aunque este método es más que suficiente para la mayoría de las aplicaciones, es limitado por su respuesta a posibles problemas que puedan surgir durante la comunicación; por ejemplo, si el receptor se comienza a sobrecargar de información. Es en estos casos cuando el intercambio de pulsos de sincronización, o *handshaking*, es útil. Las tres formas más populares de *handshaking* con RS-232 son: handshaking por software, handshaking por hardware, y XModem. Nos centraremos en las dos primeras.

- *Handshaking* por software: Las líneas necesarias para la comunicación siguen siendo TxD, RxD, y GND, ya que los caracteres de control se envían a través de las líneas de transmisión como si fueran datos. Para establecer el flujo de control, el transmisor usa dos bytes predefinidos en los *caracteres ASCII* : XOFF y XON.

  La mayor desventaja a tener en cuenta de usar este método es : los números decimales 17 y 19 son límites de transición para este método de control de flujo. Cuando se transmite en *ASCII* no importa mucho, ya que estos valores no representan caracter alguno. Sin embargo, si la transmisión de datos es en binario, es probable que estos valores sean transmitidos como datos regulares y falle la comunicación.

- *Handshaking* por hardware: El segundo método de *handshaking* utiliza líneas de hardware. Tal y como hemos descrito en las especificaciones, las líneas TxD y RxD, las líneas RTS/CTS y DTR/DSR trabajan de manera conjunta siendo un par la entrada y el otro par la salida. Estas líneas permiten al puerto serie y módem indicarse mutuamente su estado. Como regla general, las líneas DTR/DSR se utilizan para indicar que el sistema está listo para la comunicación, mientras que las líneas RTS/CTS se utilizan para paquetes individuales de datos. [4]

Nosotros actualmente hemos utilizado como flujo de control *handshaking* por hardware (descrito en el proceso anterior) y flujo de control nulo. Es útil utilizar el proceso de *handshaking* para prevenir recibir datos y evitar el colapso cuando el PC no está listo y, de igual manera, para prevenir los mismos riesgos en el dispositivo con el que realizamos el *handshake*.

3.1.3.4 Convertidor USB a Serial

A falta de puerto serial en algunos PC modernos como el caso de los ordenadores portátiles, se ha visto imprescindible para la comunicación el utilizar un convertidor USB a serial con los respectivos usos actualizaciones y protocolos para crear la compatibilidad adecuada.

Como la mayoría de los cables de este tipo, el convertidor transforma la señal de USB a serial DB9 macho.

A pesar de tener que actualizar los drivers en el ordenador para el uso de este mecanismo de conexión, en ocasiones la conversión de la señal no se realiza satisfactoriamente y puede resultar a ser el origen de los problemas en la comunicación.

Figura 2.1 Cable DB-9 RS232 – USB que utilizamos para conectarnos.

3.2 Software en la comunicación por puerto serie.

3.2.1 Manejo del puerto serie a través de Windows.

En Windows la configuración de los puertos serie instalados en el PC puede realizarse en el Administrador de Dispositivos. A partir del Panel de Control, dentro del administrador de dispositivos, se muestra una lista con todos los puertos de comunicación del PC. Los puertos series son los que aparecen con denominación COM.

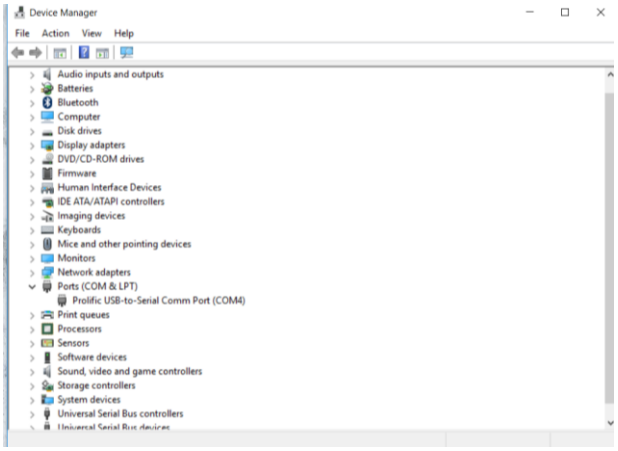


Figura 2.2 Administrador de dispositivos de Windows.

En las propiedades del puerto que estemos utilizando, accediendo a la pestaña de Configuración de puerto, aparecen los parámetros de comunicación del puerto serie:

La velocidad de transmisión (*Baud Rate*), Bits de datos en el mensaje, paridad, control de flujo, bits de parada y tipo de control de flujo.

Para controlar el intercambio de información entre terminal y modem recurrimos a la API de Windows (*Windows application programming interface*) (Win32 API), que consiste en un conjunto de funciones residentes en bibliotecas que permiten que una aplicación corra bajo un determinado sistema operativo. Nosotros en concreto, utilizaremos las funciones de la

API de Windows para, como hemos dicho, comunicarnos con un dispositivo externo (el captador en este caso), controlando así la entrada y salida de información de nuestro terminal.[7]

Para ello, vamos a valernos de la librería Windows. Este archivo se usa como cabecera cuando se programa en lenguaje C/C++ ya que contiene las declaraciones de todas las funciones de la biblioteca Windows API, todas las macros utilizadas por los programadores de aplicaciones Windows, y todas las estructuras de datos utilizadas en gran cantidad de funciones y subsistemas.

Podemos dividir en cuatro etapas la comunicación en puerto serie a la hora de trabajar con esta librería:

1. Etapa de apertura del puerto: Debemos abrir el puerto con el que vamos a establecer comunicación desde el terminal hasta el módem y configurar el tipo de comunicación que vamos a realizar a través del puerto.
2. Etapa Configuración de parámetros: Tras la apertura del puerto deberemos de establecer los parámetros de comunicación que utilizaremos para enviar y recibir información a través del puerto.
3. Etapa de intercambio de datos: Una vez el puerto ya está listo y configurado, usaremos dos funciones diferentes con sus respectivos *buffers* para enviar y almacenar la información recibida.
4. Etapa de cierre: Cuando acabemos el intercambio de información en su totalidad, deberemos cerrar el puerto.

A continuación, describiremos las funciones que utilizaremos para cada etapa.

- Etapa de apertura del puerto

Para abrir el puerto utilizamos la función *CreateFile( ),* que permite crear o abrir un archivo o un dispositivo de E/S. En nuestro caso lo utilizaremos para abrir el puerto serie que utilizaremos para comunicarnos con el captador como hemos comentado antes. La función devuelve un *handle* que se podrá usar para acceder al dispositivo a través de diferentes tipos de E/S dependiendo del propio dispositivo y de los atributos que especifiquemos.

En nuestro caso, estableceremos el puerto a través de una clase que contará con un atributo *handle* (*handle hSerial*). Contará con un puntero *this* para hacer referencia al propio objeto.

A través de la función, permitimos que se pueda leer y escribir a través del puerto serie, nos asegura de que exista la conexión cuando se crea y que el puerto tiene un uso individual para una comunicación serial.

```
//Nos tratamos de conectar al Puerto a través de CreateFile
this->hSerial = CreateFile(portName,
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL);
```

- Etapa Configuración de parámetros

A la hora de configurar los datos, la librería *Windows.h* cuenta con una estructura *DCB* que define los parámetros de control para la comunicación en serie con un dispositivo. Lo que haremos nosotros será inicializar todos los parámetros a cero (deshabilitándolos) y configurar los que nos interesen (es decir, los indispensables para la comunicación).

```
//Al conectar, inicializamos los parámetros
DCB dcbSerialParams = {0};

  //Definimos los parámetros de conexión
    dcbSerialParams.BaudRate=CBR_1200;
    dcbSerialParams.ByteSize=8;
    dcbSerialParams.StopBits=ONESTOPBIT;
    dcbSerialParams.Parity=NOPARITY;
    dcbSerialParams.fDtrControl = DTR_CONTROL_ENABLE;
```

- Etapa de intercambio de datos

Las funciones que usamos para enviar y recibir datos tiene un formato muy parecido. Tanto *WriteFile ( )* como *ReadFile ( )* utilizan como parámetros de entrada el puerto serie con el que estamos conectados (*handle hSerial*), un puntero a un *buffer* que contenga los datos de envío o recibidos y el número de bytes a escribir o a ser leídos (dependiendo de escritura o

lectura, son parámetros de entrada o salida). El último parámetro es un puntero a una estructura *overlapped* que, al haber indicado en la apertura del puerto que no lo vamos a utilizar (NULL), introducimos aquí el mismo valor.

La función para el envío de datos la utilizamos en el siguiente fragmento:

```
    if(!WriteFile(this->hSerial, (void *)buffer, nbChar, &bytesSend, 0))
    {
        //En caso de que no funcione, obtenenemos el error y devuelve falso
ClearCommError(this->hSerial, &this->errors, &this->status);

        return false;
    }
```

La función para la lectura de datos la utilizamos en el siguiente fragmento:

```
        if(ReadFile(this->hSerial, buffer, toRead, &bytesRead, NULL) )
        {
            return bytesRead;
        }
```

(Para la función lectura también tenemos la función para obtener el fallo en caso de que ocurra, sólo que no se ha incluido en el fragmento copiado).

- Etapa de cierre

Para cerrar el puerto serie, lo que hacemos es uso de la función *CloseHandle( )* que, como indica su nombre, cierra un objeto abierto de tipo *handle* (en nuestro caso, *handle hSerial*).

```
        CloseHandle(this->hSerial);
```

4 ENTORNO DE PROGRAMACIÓN

4.1 C++ como lenguaje de programación

Para desarrollar este proyecto, he decidido utilizar C++ como lenguaje para crear nuestro código.

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. El objetivo principal de C++ fue la inclusión de la creación y manipulación de objetos en el existente lenguaje C. En ese sentido, desde el punto de vista de lenguajes orientados a objetos, C++ es un lenguaje híbrido. El nombre de C++ fue propuesto por Rick

Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre de “C con clases”, y fue sustituido por el incremento de C, “C++”.

Posteriormente se añadieron facilidades para realizar programación genérica, que se sumaron a los paradigmas de programación estructurada y de programación orientada objetos. Por esto, se dice que C++ es un lenguaje de programación multiparadigma.

Gracias a ello, conseguimos un código fuente más ordenado, más fácil de entender y más eficiente.

Otra gran cualidad de C++, pese a que no la hemos aprovechado en el proyecto, es la capacidad de redefinir los operadores, creando nuevos tipos y cambiando su comportamiento. Además como ya mencionamos, nos permite la programación orientada a objetos, algo que ha sido fundamental para el desarrollo de nuestro programa.[5]

Hemos preferido utilizar C++ en vez de C por dos razones principalmente : Es un lenguaje que durante este año he manejado y utilizado más debido a la asignatura de Informática Industrial y la utilización de las clases, con lo que se consigue un código más fácil de modificar a la hora de implementar nuevas funciones y métodos en él.

Actualmente existe un estándar denominado ISO C++, ya que cuando hablamos de “versiones” de C++, nos referimos realmente a un estándar ISO al que se han adherido la mayoría de fabricantes de compiladores más modernos.

4.2 IDE para desarrollar el software

4.2.1 Introducción

Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado *Eclipse Modeling Project,* cubriendo casi todas las áreas de *Model Driven Engineering*.

En la página del software podemos comprobar además la gran cantidad de proyectos que tienen en marcha dada la gran cantidad de usuarios que participan en ellos actualmente. Uno de ellos es *WindowBuilder*, una herramienta para diseñar interfaces para los programas de manera sencilla.

Eclipse fue creado como el sucesor de la familia de herramientas para *Visual Age* de IBM. Actualmente está siendo desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse fue liberado originalmente bajo la *Common Public License*, aunque también después fue re-licenciado bajo la *Eclipse Public License*. [5]

4.2.2 Eclipse como IDE para este proyecto

La elección de Eclipse como entorno de desarrollo (IDE) es debido a que es un programa de desarrollo de software gratuito basado en GNU.

GNU es un sistema operativo de software libre patrocinado por la *Free Software Foundation*. Se respeta la libertad de usuarios y permite utilizar el programa sin coste alguno.

Eclipse se usa principalmente para desarrollar aplicaciones utilizando como lenguaje de programación Java. Sin embargo, este entorno permite también trabajar en C++ (como hemos hecho nosotros). La versión que nos recomendó la página es la del Eclipse Mars, la cual salió a la luz en Junio de 2015. [6]

Para poder utilizarla en nuestro equipo , hace falta tener instalado MinGW (*Minimalist GNU for Windows*), conocido también por MinGW32. MinGW es una implementación de los compiladores GCC para la plataforma Win32 que permite migrar la capacidad de este compilador al entorno de Windows. Es un *fork* de Cygwin en su versión 1.3.3. Además

MinGW incluye un conjunto de la API de Win32, permitiendo un desarrollo de aplicaciones nativas para esa plataforma, pudiendo generar ejecutables y bibliotecas usando la API de Windows.

Los beneficios de utilizar este software en este proyecto principalmente han sido:

- Comodidad y claridad en su entorno gráfico a la hora de programar.
- Cuenta con una gran cantidad de usuarios activos y soporte, lo que permite resolver rápidamente obstáculos cuando aparecen durante el proceso de trabajo.
- En un principio se contaba con *WindowBuilder*, un *plug-in* que permite desarrollar de manera sencilla una interfaz gráfica para el programa. Sin embargo actualmente sólo está disponible para Java (aunque próximamente estará para otros lenguajes como C++).

5 DISEÑO DEL PROGRAMA

A la hora de desarrollar el programa con el que nos comunicaremos con el captador secuencial de aerosoles, se ha establecido que la prioridad era conseguir conectarnos con el dispositivo para hacer las pruebas. En un principio, comencé consultando bibliografía específica sobre protocolo TCP/IP [8] para conseguir una conexión remota y explorando ejemplos y librerías (como las *Boost*) .

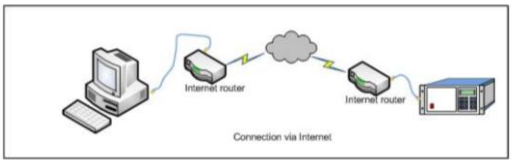
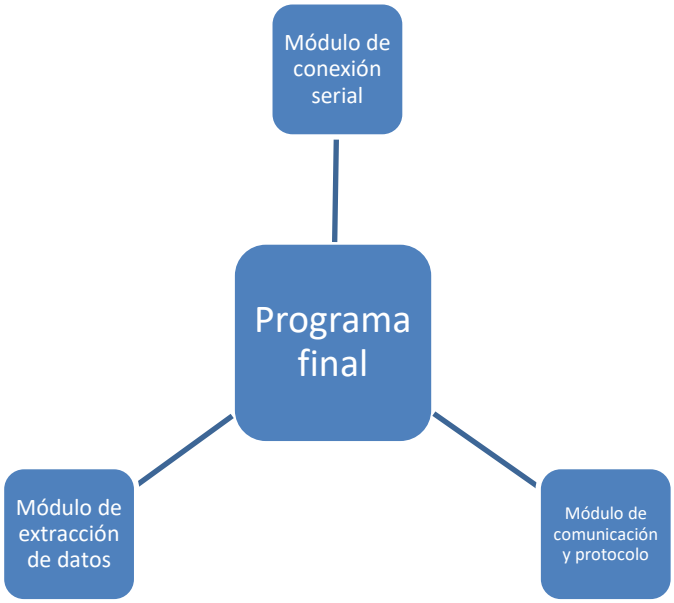


Figura 4.1 Modelo de conexión remota desde un PC hasta el captador

Sin embargo, rápidamente se optó por priorizar la conexión en serie con el dispositivo, y una vez conseguida dicha conectividad, ya se pasara a trabajar en la conexión remota.

Para realizar el software que pudiera cumplir los objetivos, se decidió trabajar primero en los bloques que componían el programa por separado para comprobar su correcto funcionamiento y, posteriormente, unirlos para poder trabajar y comprobar la eficiencia del

programa en Izaña. Estos bloques o módulos serían uno de conexión serial, otro módulo para la extracción de datos de un fichero de texto (Excel) y un tercer módulo de protocolo para comunicarnos con el dispositivo. El siguiente paso sería trabajar en una interfaz gráfica que hiciese al programa más "amigable" y más fácil de utilizar por el operario que trabajase con el dispositivo. Finalmente, se procedería a trabajar en la conexión remota.



Sin embargo, durante el proceso de diseño y creación del programa han existido algunos inconvenientes que han retrasado y dificultado el proceso y el avance del mismo.

A continuación expondremos los tres módulos y explicaremos las principales funciones de cada uno de ellos. Sólo utilizaremos los fragmentos más importantes en esta sección.

5.1 Módulo Serial

El desarrollo del módulo de conexión serial fue lento. La bibliografía que comprobé no contenía ningún apartado específico o que aclarase las dudas del mismo [9],[10],[11]. En las diversas páginas webs [12], [13] donde los programadores exponen sus dudas existen numerosos fragmentos de código para establecer una conexión por puerto serie. Sin embargo los métodos no acababan de ser del todo claros. A partir del código que expone Arduino para C++ en su web[14] y de *Microsoft Developer Network* [15], donde se explican

diversas funciones de las librerías dentro de la API de Windows, pude empezar a desarrollar y probar la conexión por puerto serie.

Para ponerlo a prueba, se utilizó una placa de Arduino Uno con la que se interactuó desde nuestro IDE con una pequeña aplicación en C++ y con el consecuente código implementado en dispositivo para comprobar que la comunicación se conseguía adecuadamente.

Además se probó con otro PC en los laboratorios de la universidad, comprobando que tanto recibía como enviaba mensajes simultáneamente entre el portátil con el que trabajamos y el PC.

Al igual que el módulo de extracción de datos de un fichero de texto exportado de Excel, hemos basado este módulo en una clase llamada "serial" por el cual dispone de unos atributos para guardar los parámetros de conexión y unos métodos para realizar la comunicación a través del puerto.

En el archivo *HPP* instanciamos los atributos y métodos de la clase. Son de particular interés los siguientes dentro de la parte *private* en la clase Serial.

```
        HANDLE hSerial;
```

```
hSerial es un atributo de tipo HANDLE ( tipo especifico propio de la API de
Windows) que nos permite realizar una connexion con el Puerto serial. Esta
connexion la haremos efectiva con el constructor de la clase más adelante.
```

```
        bool connected;
```

```
connected lo utilizaremos para conocer el estado de la function a través de
un método de la clase.
```

```
        COMMTIMEOUTS ComTimeOut;
```

Los *commTimeOuts* sirven para establecer ciertos parámetros de conexión para las funciones de escritura y lectura del puerto. Entre otras cosas, determinan el tiempo máximo de realización del proceso.

Las librerías que hemos utilizado para esta clase son:

- Windows.h , que es necesaria para utilizar las funciones y parámetros propias de la API de Windows para conectarnos al puerto serie.

- Stdlib.h o cstdlib, que significa “C Standard General Utilities” , nos permite usar funciones de uso general, como *atoi*.
- Iostream , que nos permite usar objetos estándar para hacer *streams* de entrada/salida, tales como *cin* o *cout*.
- String.h o cstring, que nos permite manipular *strings* y *arrays* con funciones como *strcat*( ).

A continuación, definiremos brevemente los métodos que haremos uso en esta clase. Todos ellos están instanciados en el archivo *HPP* y definidos en el *CPP*. Las funciones que tenemos en el archivo, al no ser métodos de la clase serial, sólo aparecerán en el *CPP* como tales.

- El constructor

En primer lugar, la clase Serial utiliza un constructor que recibe un puntero con el puerto al que nos vamos a conectar.

```
Serial::Serial(const char *portName)
```

Utilizamos el atributo anteriormente mencionado para establecer que la conexión aún no es efectiva, pero después del proceso, si no hay errores, la marcaremos como verdadera.

```
this->connected = false;
```

La función que permite obtener una conexión con el puerto es *CreateFile( ).* Como mencionamos anteriormente, esta es la función encargada de la *etapa de apertura del puerto.* Principalmente, necesita de un parámetro *handle* sobre el que realizar la conexión. A partir de esta función establecemos como queremos realizar la configuración del puerto:

Si establecer una comunicación solo de lectura, escritura o ambas, si realizar una comunicación síncrona, asíncrona u otras modalidades, si restringe el puerto a nuestra aplicación o permite a otras…

En nuestro caso, la programamos para realizar una comunicación serial a través de ella.

```
this->hSerial = CreateFile(portName,
        GENERIC_READ | GENERIC_WRITE,
        0,
```

```
            NULL,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            NULL);
```

En caso de no funcionar, utilizamos la función *GetLastError( )* que recoge el error (en caso de que lo hubiera) mandado por la última función utilizada.

Lo siguiente que hacemos en el constructor es inicializar los parámetros de la conexión a partir de la estructura DCB. Estos parámetros pueden variar según el protocolo que utilicemos para comunicarnos con el captador.

```
        DCB dcbSerialParams = {0};
//Definimos la conexión serial de los parametros para el intercambio de
datos.
            dcbSerialParams.BaudRate=CBR_1200;
            dcbSerialParams.ByteSize=8;
            dcbSerialParams.StopBits=ONESTOPBIT;
            dcbSerialParams.Parity=NOPARITY;
            dcbSerialParams.fDtrControl = DTR_CONTROL_ENABLE;
            dcbSerialParams.fRtsControl = RTS_CONTROL_ENABLE;
```

También se establecen los *TimeOuts*.

```
                    ComTimeOut.ReadIntervalTimeout=MAXDWORD;
                    ComTimeOut.ReadTotalTimeoutMultiplier=0;
                    ComTimeOut.ReadTotalTimeoutConstant=0;
                    ComTimeOut.WriteTotalTimeoutMultiplier=0;
                    ComTimeOut.WriteTotalTimeoutConstant=0;
```

Por último, se eliminan caracteres que pudieran quedar en el *buffer* de usos anteriores y cambiamos el parámetro *connected* a *true*.

```
                this->connected = true;
                PurgeComm(this->hSerial, PURGE_RXCLEAR | PURGE_TXCLEAR);
```

- El destructor

El destructor cuando es llamado elimina el objeto, cierra la conexión y ajusta el valor del parámetro booleano *connected*.

```
Serial::~Serial()
{
    //Antes de desconectar, probamos si estamos conectados.
    if(this->connected)
    {
        //Nos desconectamos
        this->connected = false;
```

```
        //Cerramos la conexión serial.
        CloseHandle(this->hSerial);
    }
}
```

- Método de lectura

Para leer los datos del *buffer* utilizamos la función *ReadData( ).* Para utilizar esta función, debemos pasarle como parámetros el puerto que estamos utilizando con un parámetro *handle*, un *buffer* donde depositar los caracteres leídos, la extensión de los datos que hemos recibido y los que hemos leído.

```
int Serial::ReadData(char *buffer, unsigned int nbChar)
```

Al método sin embargo sólo le pasamos dos parámetros: El *buffer* y una variable *int*. Esta variable (*nbChar*) se encarga de que, en caso de recibir un mensaje más grande del espacio disponible, recortar dicho mensaje y a su vez mandando un aviso.

```
if(this->status.cbInQue>nbChar)
        {
            toRead = nbChar;
            //mostramos por pantalla un aviso
            std::cout<<"Aviso : Tamaño del mensaje mayor de lo
esperado."<<std::endl;
        }
        else
        {
            toRead = this->status.cbInQue;
        }
```

La función guarda los datos en el *buffer* y le hacemos devolver el número de bytes leídos.

```
        if(ReadFile(this->hSerial, buffer, toRead, &bytesRead, NULL) )
        {
            return bytesRead;
        }
```

Para acceder después a la información, basta con recurrir al *buffer*.

Finalmente, en caso de que no se haya recibido nada, la función en lugar de devolver el número de bytes leídos devolverá un cero.

```
    return 0;
```

- Método de escritura

La función de escritura es análoga a la de lectura. *WriteData( )* necesita principalmente un *buffer* con el que mandar los datos y la longitud del mismo.

```
bool Serial::WriteData(char *buffer, unsigned int nbChar)
{
    DWORD bytesSend;

    //Tratamos de
    if(!WriteFile(this->hSerial, (void *)buffer, nbChar, &bytesSend, 0))
    {
        //En caso de no funcionar, obtenemos error com y devolvemos falso.
        ClearCommError(this->hSerial, &this->errors, &this->status);

        return false;
    }
    else
        return true;
}
```

- Método para comprobar la conexión

Este método simplemente comprueba el atributo *connected* y devuelve su estado.

```
bool Serial::IsConnected()
{
    //Devolvemos el estado de la conexión.
    return this->connected;
}
```

- Función para añadir BCC al mensaje

Para un determinado tipo de protocolo, es requerido añadir al mensaje un valor para comprobar que el mensaje no ha sufrido errores o pérdidas de información durante el envío. Por ello, a la función le pasamos del cual deseamos obtener el BCC y devolvemos el mensaje con dicho valor añadido al final.

```
char *GetBCC(char *Message){

            int length;
            int BCC1 ={0x00};
            int BCC2 ={0x00};
          length = strlen(Message);
          int MessageInt[length];
          char BCC1Out[1];
          char BCC2Out[1];

          for(int i=0;i<(length-2);i++){

         // Guardamos en valor actual de la iteración en otro array.
                MessageInt[i]=Message[i];
```

```
        // Hacemos el calculo de BCC utilizando el operador xor "^".
                BCC1 ^= MessageInt[i];
                i++;

                MessageInt[i]=Message[i];
                BCC2 ^= MessageInt[i];
         }

         itoa(BCC1,BCC1Out,16);
         itoa(BCC2,BCC2Out,16);

         // Lo concatenamos al final del mensaje y lo devolvemos.
         strcat(Message,BCC1Out);
         strcat(Message,BCC2Out);

         return Message;
```

Con la función *itoa( )* convertimos la variable *int* en un *string* y con *strcat( )* lo concatenamos al mensaje.

- Función para comprobar el BCC del mensaje

Esta función la utilizamos para que, cuando el dispositivo nos devuelva un mensaje, poder comprobar que su envío ha sido correcto. Para ello, obtenemos el BCC del mensaje y lo comparamos con el BCC que hemos recibido del mensaje original proveniente del dispositivo.

```
bool Serial::CheckBCC(char *InMessage){
```

InBCC corresponde al BCC calculado por nosotros e *InMessage* al del mensaje original. El valor de "I" corresponde al valor final del mensaje donde se haya el BCC.

```
                if(InBCC[i]!=InMessage[i]){
                        std::cout<<"BCC no coincide, error en el
mensaje"<<std::endl;
                        //En caso de que no sean iguales, devolvemos false.
                        return false;
                }

          }

          std::cout<<"BCC correcto"<<std::endl;

          //En caso de que sean iguales, devolvemos true.
          return true;
```

5.2 Módulo de extracción de datos.

A la hora de extraer los datos, lo primero que se pensó es: ¿Cómo conectar los ficheros que da soporte Excel con C++? A partir de la bibliografía consultada [9] donde explican cómo hacer un buen uso y como desarrollar en C++, a lo más que se aspira de manera básica es a trabajar con ficheros de texto.

Buscando en internet accesos a librerías que pudieran permitir trabajar con este tipo de ficheros, el resultado fue escaso. Un resultado interesante fue *LibXL*, una librería que permite trabajar con ficheros de Excel respetando las celdas y su formato. Sin embargo, es una librería de pago, por lo que de momento se ha descartado.[16]

Excel permite exportar formato de texto e importarlos también, luego esta podía ser una forma con la que poder trabajar con los datos necesarios de Excel y manipular el captador una vez tuviésemos los datos importados a nuestro programa.

Con los datos que se tuvo de cómo podría ser el archivo de Excel con el que se trabajaba en Izaña se ideó un prototipo a la espera de recibir el formato que realmente se utilizaba.

El proceso para obtener el fichero de texto es tan simple como guardar el archivo de Excel en formato texto (*MS-DOS*) y aceptar la pérdida de la estructura de celdas que posee el archivo, sustituyéndolo por espacios en blanco respectivamente.

A la hora de importar, el asistente nos indica que debemos realizar 3 pasos para abrir el archivo en Excel: En el primer paso, deberemos de elegir de ancho delimitado y de origen *MS-DOS* (PC-8). En el segundo paso, deberemos elegir como separadores la opción de tabulación. En el tercer y último paso, elegir el formato de cada columna. Se puede establecer como general.

Esto es una solución no final, pero que definitivamente ayudará al trabajo de los operarios pudiendo acceder y obtener los resultados del captador de manera más sencilla. Sería conveniente en versiones futuras del programa contar con una librería que permitiese trabajar con Excel de manera que se pudiese trabajar con el archivo original y poder modificar sólo las celdas que nos interesaran.

Para el primer ejemplo que ideamos, establecimos una línea con los nombres de los parámetros y a continuación varias filas de datos que traspasar al programa. Como podemos ver, a pesar de que los nombres de la primera fila estén descolocados, están separados ellos y los parámetros por tabulaciones que hace más fácil su lectura para el programa.

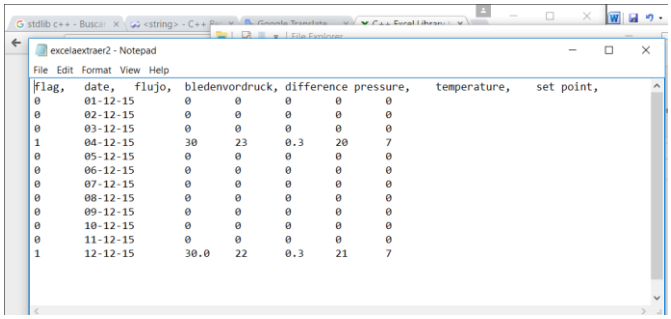


Figura 4.2 Fichero de texto exportado de una tabla de Excel

Para poder separar el nombre de los parámetros de los valores de cada columna, utilizábamos comas con el fin de encontrar estos parámetros en el fichero y separarlos de los otros.

Para el ejemplo que ideamos, es sencillo obtener la primera fila de nombres con la función *getline( ),* la cual substrae una línea entera con los nombres de los parámetros de los archivos.

A partir de comas o de espacios, mostrados en la figura 1, podemos obtener y guardar los nombres de los parámetros ordenadamente al igual que el de los datos a través de funciones propias de *string*, por el que compara un determinado carácter (como dijimos antes, comas o espacios) e ir separando una cadena larga obtenida por *getline( )* en diferentes partes de un *array* donde guardar los nombres de los parámetros.

Sin embargo, el fichero final resultó ser mucho más complejo al contar con 5 filas de texto mostrando diferente información.

Al querer conservar el documento original, resulta inservible a la hora de sacar los datos del captador al mismo, se conserven las líneas de texto. Es más cómodo a la hora de traspasar al documento original que, guardando los datos sacados del captador en el orden adecuado, se

importen al programa Excel y que desde ahí se copien y se peguen en el documento original , respetando así el formato.

Para ello, existen dos opciones: Se podría o bien guardar en un texto los valores sin el texto del Excel original o, por otro lado, se podría guardar el documento entero y desechar estas filas que no aportan utilidad para programar el captador.

En el documento contamos con la primera opción pero, en caso de que resulte mejor el segundo caso, está en texto la iteración que quitar las primeras líneas de texto usando, como hicimos antes, la función *getline( )* . Sin embargo, esta vez no guardamos el texto, sólo los parámetros útiles para el captador.

Al mostrar en pantalla desde el programa si podremos en cambio, ver a qué parámetro pertenece cada valor.

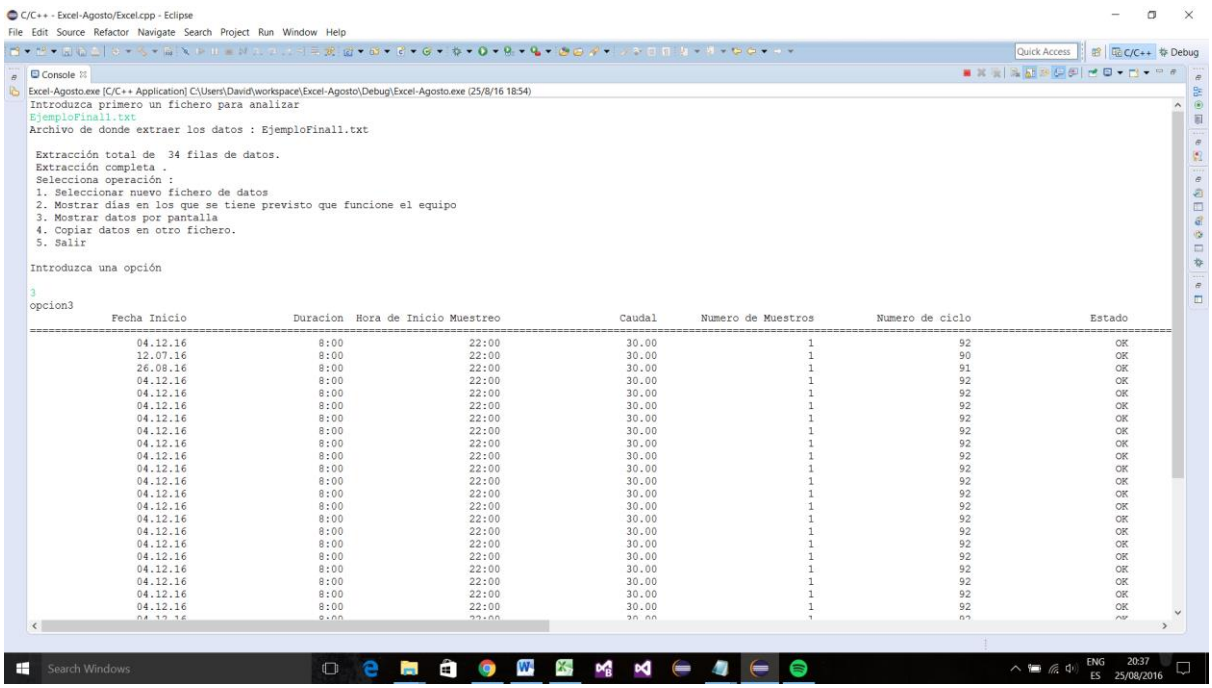


Figura 4.2 Salida de nuestro programa tras haber leído un fichero de texto y sacarlo por pantalla ordenadamente

Más adelante explicaremos la clase y los métodos que usamos para el módulo de extracción pero primero mencionaremos las librerías que hemos usado:

- Fstream, necesaria para poder extraer los datos de los ficheros de texto.

- Iostream, que utiliza ciertos objetos para trabajar con elementos de entrada/salida como *cin* o *cerr*.
- Iomanip, utilizada para mostrar ordenadamente los datos por pantalla.
- String, para trabajar con el tipo de datos *string*.
- Windows.h, para utilizar funciones de la API de Windows. No es fundamental, pero la utilizamos para la fluidez del programa con funciones como *pause( ).*

Una vez comentadas las librerías utilizadas, describiremos los métodos utilizados por la clase, instanciados en el archivo *HPP* y definidos en el *CPP*.
Los atributos de la clase son mayormente los valores que extraeremos del archivo de texto y todos estarán puestos en la parte pública de la clase para poder acceder a ellos en todo momento. Los tenemos creados como un *array* para que cuando se lea el archivo, se guarde fila por fila cada valor en el parámetro que le corresponda.

A parte de ellos, tenemos como atributos "*ExcelArchive*", un *string* donde guardaremos el nombre del archivo que introduzcamos, "*TotalDatos*", un entero donde guardaremos el número de filas leídas por el programa y dos atributos de tipo *bool* para controlar si se han extraído los datos correctamente para seguir trabajando ("*Extraccion*") y otro para controlar si existe un día de los leídos con *flag* a 1 para realizar un ciclo con el captador.
Los métodos son los siguientes

- El constructor y constructor de copia.

```
extractor::extractor()
extractor::extractor(string NewArchive)
```

Los constructores se encargan de inicializar los atributos a cero. Para crear el objeto, podemos hacerlo invocando el constructor predeterminado o, si tenemos el nombre de un archivo guardado de tipo *string*, pasarlo y crearlo como constructor de copia con dicho parámetro.

- Método de extracción de datos de un fichero de texto.

```
void extractor::GetData(){
```

Para poder sacar los datos del archivo de texto creamos un objeto fin que se encargará de establecer la conexión con el archivo. Utilizando la función **open**, podemos abrir el archivo. La terminación ".c_str ( )" es necesaria para poder realizar la operación.

```
ifstream fin;
fin.open(this->ExcelArchive.c_str());
```

En caso de no poder abrirse el archive, devolvemos un error.

```
if(!fin){// Error mostrado en caso de que no pueda abrirse el archivo.

        cerr << "No puede abrirse "<< this->ExcelArchive
        << " para extraer los datos."<< endl;
        return;
}
```

Como comentábamos antes, tenemos una iteración en texto para descartar las líneas que no son datos del inicio del documento. Sabemos que son 5 por el documento que nos han pasado desde el Instituto Meteorológico en Izaña.

A continuación, a partir de un bucle *for*, guardamos los datos en los parámetros que le corresponden. En caso de no tener archivos que sacar porque el documento ya no tenga más, paramos de sacar datos.

```
for(i=0; i < mes ; i++){

    if(fin != NULL){

        // Datos para enviar al captador.

        fin >> this->Flag[i];
        //cout << Flag[i]<<endl;

        fin >> this->EFechaMuestreo[i];
        //cout << EFechaMuestreo[i]<<endl;

        […]
```

Una vez no queden más datos que sacar, se cumplirá la condición para pasar al *else*, mostrando el total de filas extraídas y parará el bucle *for*.

```
else{
    cout<<endl;
    cout << " Extracción total de  " << i-1<< " filas de datos."
    << endl;
    break;
```

```
        }
    this->TotalDatos = i;
    }
```

Por último, podremos el parámetro booleano de extracción como *true* y cerraremos el archivo con la función *close.*

```
fin.close();
```

- Método para mostrar los datos

```
void extractor::ShowData()
```

Para mostrar los datos en pantalla no utilizaremos ninguna función característica pero si utilizaremos algunos manipuladores en las operaciones de salida al mostrar los valores que hemos guardado anteriormente.

Utilizaremos *setw( ),* que nos permitirá distribuir organizadamente los datos que queramos mostrar en pantalla*, setfill( )* para diferenciar los indicadores al principio de cada columna de los datos y *setprecision( )* para establecer la cantidad de números que mostraremos de los parámetros *float* tras la coma.

```
        for(i=0; i <= TotalDatos; i++){
/
        cout <<setw(AnchuraCol)<< EFechaMuestreo[i] ;
        cout <<setw(AnchuraCol)<< EDuracionMuestreo[i];
        cout <<setw(AnchuraCol)<< EHoraMuestreo[i];
        cout <<setprecision(Precision)<<setw(AnchuraCol)<< ECaudal[i];
        // […]
```

- Método para guardar los datos

```
void extractor::SaveData(string SaveArchive){
```

Para utilizar este método será necesario pasarle un parámetro *string* con el nombre del archivo que queremos guardar. Deberemos realizar la misma operación que hicimos con el método para sacar los datos.

```
ifstream fin(SaveArchive.c_str());
```

Esta vez, si se pudiera abrir el archivo significaría que ya existe. Por ello, se preguntará al usuario si quiere sobrescribir el archivo o no. En caso de que así se quiera, seguiremos.

Para exportar los datos al archivo deberemos recurrir a crear un objeto *ofstream* sobre el que poder guardar los archivos. Esta vez al abrirlo, estableceremos que el objeto servirá como salida y para guardar datos.

```
ofstream fout;
fout.open(SaveArchive.c_str(), ios::out | ios::trunc);
```

Con un bucle *for*, nos disponemos a guardar todos los datos en orden.

```
for(i=0; i <= TotalDatos; i++){
      //  Guardamos los datos que se introducen en el captador

      fout <<setw(AnchuraCol)<< Flag[i];

      fout <<setw(AnchuraCol)<< EFechaMuestreo[i] ;

      fout <<setw(AnchuraCol)<< EDuracionMuestreo[i];

      //[…]
```

Una vez acabado, cerramos el objeto. En caso de no haberse podido guardar los datos, mostraremos un error.

```
fout.close();
```

- Método para mostrar los días con flag activado

```
bool extractor::DateProcess()

      for(int i=0;i<this->TotalDatos;i++){
            if(this->Flag[i]==true){
                        this->ProcessDay =true;
                        cout<<this->EFechaMuestreo[i]<<endl;
            }
      }
```

```
//[…]

return this->ProcessDay;
```

Estos métodos sirven para conocer en que fechas de los datos que hemos extraído está preparado un ciclo para el captador. Para ello, comprobamos si algún parámetro del *array* de *flag* está a uno. En caso de que así sea, devolvemos la fecha del muestreo.

```
bool extractor::DateProcess(string ADate)
```

Con el otro método por el contrario, comparamos si los días con *flag* a 1 son iguales a la fecha introducida.

```
            if(this->Flag[i]==true){

                  if(ADate==this->EFechaMuestreo[i]){
                        cout<<"La fecha"<<this->EFechaMuestreo[i]<<"está
                        apuntada para el proceso."<<endl;
                        this->ProcessDay =true;

            }
```

Por último, utilizamos una última versión donde le pasamos un *array int* para sacar los días en los que hay *flag* a 1 en orden secuencial. De esta manera, se elige más fácilmente el día a utilizar para una operación, con el fin de enviar comandos al dispositivo.

```
bool extractor::DateProcess(int vector[]){

      this->ProcessDay = false;
      int b=0;
            cout<<"Las fechas para el uso del equipo son : "<<endl;
            for(int i=0;i<this->TotalDatos;i++){
                  if(this->Flag[i]==true){
                              this->ProcessDay =true;
                              cout<<this->EFechaMuestreo[i]<<endl;
                              vector[b]=i;
                              b++;
                  }
            }

            if(this->ProcessDay ==false)
                  cout<<"Ninguna fecha tiene prevista una puesta en
marcha."<<endl;

            // Se devuelve verdadero o falso en caso de tener una fecha( o
            más) o no tener ninguna.

            return this->ProcessDay;
```

```
}
```

- El destructor

```
extractor::~extractor()
```

El destructor se encarga de cerrar el objeto “extractor”. Se invocará cuando se cierre el programa.

5.3 Módulo de comunicación y protocolo

El último módulo con el que trabajamos es aquel con el que nos comunicamos con el captador secuencial de aerosoles, es decir, donde utilizamos los protocolos de comunicación que nos han otorgado para poder comunicarnos con el dispositivo y mandar comandos que conozca para poder intercambiar información. Éste módulo va, como no puede ser de otra forma, de la mano del módulo de comunicación serial. Cabe aclarar que no utilizamos clases propias para este apartado : Utilizamos las clases del archivo *CPP* y *HPP* del módulo serial y los utilizamos en un nuevo archivo *CPP* donde irá la parte ejecutable dentro del *main*.

El principal problema con este módulo ha sido la falta de información por parte de los proveedores del captador de aerosoles (MCZ *Umwelttechnik*). Los documentos obtenidos para desarrollar el protocolo no han sido del todo útiles para elaborar un código que estuviera listo para una respuesta, pues no sabemos realmente para el protocolo con el que estábamos trabajando que respuesta nos iba a dar el programa. Los otros protocolos con los que se han trabajado proporcionan una información de respuesta insuficiente para lo planteado en los objetivos del proyecto y tienen una capacidad de manipulación del dispositivo mínima. A esto, hay que añadirle que se nos comunicó que hacía falta un “tarjeta” (un cable específico) para comunicarnos con el captador y que hasta el mes de Agosto no ha estado disponible, haciendo imposible el trabajo conjunto al captador y los tutores y siendo un proceso por ende más dificultoso. A pesar de estos inconvenientes idearemos como sería el software final en el siguiente apartado.

Para llevar a cabo las pruebas, teniendo en mente el protocolo que vamos a utilizar, modificamos en el *CPP* de la clase serial los parámetros de la conexión necesarios para el protocolo que vayamos a usar. Generalmente sólo cambiamos la tasa de envío de datos (*baud rate*).

Dentro de la función *main*, vamos a utilizar un bucle *while* para poder elegir el protocolo que queramos usar (como hemos dicho con su previa configuración) y con el comando que queramos mandar.

Para los diferentes protocolos, hemos creado los mensajes en *arrays char* de manera preestablecida como en los siguientes ejemplos:

```
char StartProbe[81] = "Start Probe [PW=nnnnnn]* [Date=dd.mm.yy]
[Time=hh:mm] [Dura=hhh:mm] [Flow=nn.n] ";

 char GETALL[8]="GET;ALL";
```

Estos mensajes para no ser modificados los guardamos en un puntero y es el puntero el que modificamos.

En el caso de que se necesiten cambiar valores de los parámetros de los comandos, hacemos uso de la función *memmove* para reemplazar los números introducidos en los espacios que les corresponde.
En caso de necesitar añadir al mensaje bytes del código *ASCII* tal que CR, LF, STX o ETX utilizamos la función *strcat( )* para implementarlos al final de los mensajes.

Finalmente, utilizamos *strlen( )* para obtener la longitud del mensaje y le pasamos tanto el puntero con el mensaje como la longitud del puntero a la función de la clase serial *WriteData()* con la previa invocación de la misma a partir de un objeto que llamaremos SP.

```
if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);
```

Los tres protocolos que hemos incluido en las pruebas de comunicación con el captador son los siguientes:

- Protocolo MPNS

Originalmente, el protocolo MPNS se utiliza en un programa llamado MPNS-RemoteControl. Desde la interfaz del programa se puede programar los parámetros de los diferentes comandos existentes para ordenar al captador empezar una prueba, un ciclo o pararlos en caso contrario.

Desde la hoja informativa del documento, podemos ver que es necesario para conectarnos al captador un cable con interfaz RS-232 zero modem.

Contamos también con los comandos y los parámetros necesarios con los que debe contar. Además, se nos recuerda que los mensajes deben ir terminados con *carriage return* (CR) y *line feed* (LF).Para este protocolo, utilizamos una configuración de 9600 *Baud rate* y 8N1.

- Protocolo PFR USB DD-N4xx ASCII

Este protocolo era el que venía a modo de instrucciones con el "cable" que permitía la conexión entre el interior del captador (uno de los puertos disponibles) con el cable RS232-USB que teníamos que conectar a nuestro PC.

Sin embargo, leyendo los comandos que se utilizan en este protocolo acabamos imaginando que existe un fallo de coordinación y que podría ser para otro dispositivo, ya que sólo saca temperatura, caudal y presión. Debe ir acabado también por CR y LF.

Para este protocolo, utilizamos una configuración de 115200 *Baud rate* y 8N1.

- Protocolo Bavaria-Hessen

El protocolo B-H está diseñado para poder seguir una estructura de Polling donde el PC es el maestro y el captador (o los captadores) son los esclavos. Para ello, en la estructura del mensaje tiene un identificador para saber a cuál enviar el mensaje o opción de hacer un mensaje de difusión. La estructura del mensaje es la siguiente:

1 byte con STX (*Start of Text* del código *ASCII*), 1 byte con la ID del dispositivo al que se quiere enviar el mensaje, de 1 a 119 bytes de mensaje, 1 byte con STX *(End of Text* del código *ASCII*) y dos bytes de BCC.

El BCC (*Block check characcter*) es formado realizando una operación XOR de todo el mensaje (incluyendo los bytes STX y ETX). En el primer BCC (*Upper nibble*) se hará el *xor* de los bytes impares y en el segundo BCC (*Lower nibble*) el de los pares.

Para añadir y comparar el BCC que enviamos y el que recibiríamos del captador, hemos creado dos funciones y las hemos introducido en el archivo *CPP* del módulo serial por tener un mayor orden en los diferentes archivos.

Este protocolo nos permitiría obtener hasta 4 valores de medidas pero no sabemos cuáles serían exactamente. Utilizamos una configuración de 1200 *Baud rate* y 7N1.

6 SOFTWARE FINAL

A pesar de no haber conseguido resultados estableciendo conexión entre el PC y el captador después de varios intentos, se han juntado los tres módulos en un solo programa y se ha reestructurado la función *main* de forma que ahora podemos extraer los datos de un archivo de texto e introducir los parámetros en los comandos del protocolo elegido (MPNS en este caso ya que era el más completo) y enviarlos vía serial.

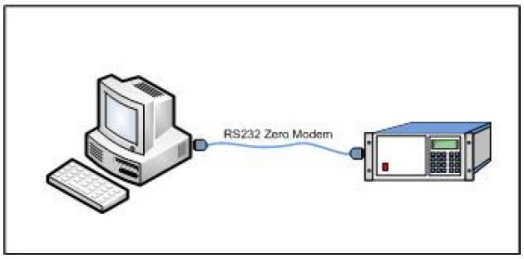


Figura 5.1 La única conexión que hemos podido intentar ha sido presencial.

El no conseguir entablar comunicación con el captador ha supuesto un problema porque, además de no conseguir el objetivo en sí del proyecto de poder trabajar con el desde un PC, no sabemos qué respuesta obtendríamos de los mensajes o que forma tendría de enviar los datos.

Para seguir adelante con la versión final por lo tanto, hemos tenido que suponer la respuesta del programa para poder tener al menos una base a la hora de tener un procedimiento para leer esos datos, guardarlos en el programa y pasarlos a un fichero.

A pesar de no ser el mismo tipo de conexión , para juntar todas las partes y comprobar que cada una realiza su función, se ha decidido utilizar una placa de Arduino conectada al PC por un cable tipo A/B – USB e introducirle dentro un mensaje que nos devolverá al ordenador una vez le llegue un mensaje vía serial. También se probó en Izaña la conexión al mismo PC para ver los mensajes por eco de hardware (devolviéndonos a nosotros mismo el mensaje).

Figura 5.2 Elementos para probar el eco por hardware para ver nuestro mensaje.

Un diagrama básico del funcionamiento del programa sería el siguiente.

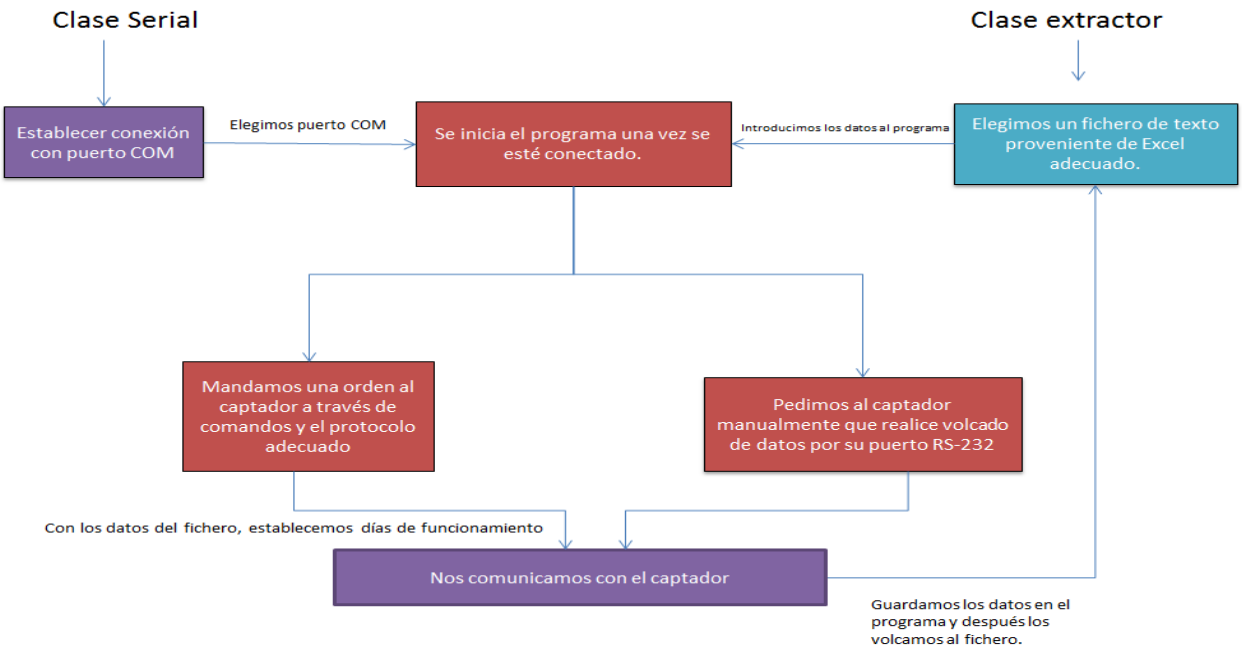


Figura 5.3 Diagrama del funcionamiento de nuestro programa

A continuación, expondré las diferentes opciones que proporcionan el programa y los pequeños cambios realizados para unificar todos los módulos.

Antes de entrar en un bucle *while* con las diferentes opciones dentro del *switch* que expondremos, crearemos un objeto de la clase Serial para poder conectarnos al puerto serie y dos clases de extractor para trabajar con el archivo de texto y obtener información del captador.

- La primera opción del programa nos permite elegir un archivo de texto con el que trabajar. Muchas de las opciones para mandar comandos MPNS necesitan de un archivo que tenga el parámetro flag a 1, lo que implica que ese día se debería de realizar una prueba.

Este archivo y la información que guardemos sobre el captador la guardaremos en los parámetros de primer objeto llamado *ArchivoPrincipal*.

Dentro de la primera opción se puede elegir que, en caso de que cuando se haga más adelante un volcado de datos del dispositivo y no se traspase correctamente, volver a guardar los datos en la primera línea de texto del archivo.

- La segunda opción del programa nos permitirá ejecutar la función del módulo de extracción de datos, lo que mostrará en pantalla los días que tenemos flag a 1 y con los que podremos mandar al captador a funcionar en dichos días o a parar los muestreos según el comando que se elija.

```
ArchivoPrincipal->DateProcess();
```

- La tercera opción mostará todos los datos guardados en *ArchivoPrincipal* en una tabla de forma ordenada. También se podrá elegir mostrar los datos guardados en *ArchivoSecundario*, los cuales explicaremos más adelantes cuales son.

- La cuarta opción del programa nos permitirá guardar los valores que tengamos en *ArchivoPrincipal* en el fichero que introduzcamos. En caso de existir, se preguntará si se desea sobrescribir.

- La quinta opción nos permite elegir que comando MPNS queremos mandar al captador. En caso de necesitar un día y datos del fichero, el programa se encargará de que, en caso de no haberlo introducido, volver atrás.

Los parámetros que introducir al captador se pasarán de dos formas:

Obteniéndolos de los parámetros de *ArchivoPrincipal* y pasándolos del formato (*string*, *int* o *float*) a *char* y utilizando *memmove()* para sustituirlos en el mensaje predeterminado o bien , en caso de que sean parámetros que no existan en el archivo de texto como, por ejemplo, cuando parar una prueba, deberá introducirse manualmente por teclado.

Para cada opción existen varias versiones del mensaje : Se puede por ejemplo parar una prueba instantáneamente o bien en un día o fecha determinado. Para ello, en el puntero en el que tengamos el mensaje que modifiquemos, sólo le pasaremos las partes del mensaje que deseemos utilizando nuevamente *memmove()* sobre otro atributo char y después pasándolo finalmente al puntero donde lo manipularemos.

Para el comando *get state* utilizaremos el *ArchivoSecundario*, ya que obtiene parámetros de la máquina pero no los que nos interesan para guardar en el archivo de texto finalmente.

- La sexta opción nos permitiría extraer los datos obtenidos en el *buffer* del ordenador una vez realizado el volcado de datos del captador. Los valores se recibirán y se separarán para guardarnos en sus respectivos parámetros en el programa y después se permitirá guardarlos en el mismo fichero txt.

Para realizar la asignación de valores obtenidos del captador a los parámetros del objeto *ArchivoPrincipal* de la clase extractor, hemos supuesto que el mensaje del captador tendría la siguiente forma

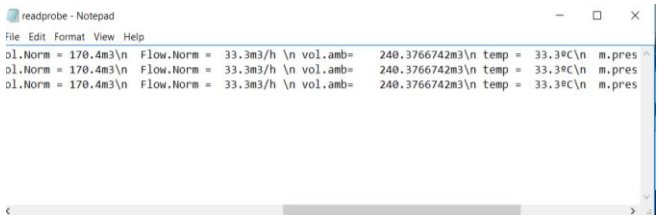


Figura 5.4 Respuesta ideal del captador para nuestro programa. Lo utilizamos para poner a prueba el método de lectura de la clase serial y guardado de archivos del extractor.

Los datos vendrían dados por el nombre indicativo del parámetro, acabaría con la correspondiente unidad de medida e irían separados por saltos de línea. Este es el formato que imaginamos que podría tener pero, en caso de que fuera otro distinto, habría que reescribirlo de nuevo.

En el proceso de extracción, utilizamos un objeto *istringstream* para obtener los datos del *buffer* y, posteriormente, utilizamos la función *getline()* sobre una variable *string* para extraer la información. A continuación, de forma iterativa, extraemos sólo el valor de la cadena a partir del número de espacios existentes en la línea (anterior y posterior del signo '=') y quitamos la unidad de medida.

Para realizar el proceso, copiamos directamente el del *string* resultante obtenido al parámetro adecuado del objeto o bien lo pasamos a *char* y a *float* posteriormente con la función *atof( ).*

Para los procesos de lecturas ha sido importante contar con la biblioteca *sstream*, la cual nos ha permitido recurrir a objetos *istringstream* que nos han facilitado el proceso del traspaso de datos del captador al programa, y del programa a los ficheros de texto.

- Por último, la séptima opción servirá para salir del programa y cerrar los objetos creados durante su ejecución.

Concluyendo, los principales cambios con los módulos han sido la implementación de algunos parámetros para mejorar su eficiencia (como líneas escritas para ir guardando los datos volcados del captador en líneas sucesivas) , corregido un error del módulo extractor por el cual no guardaba correctamente el valor de los datos en el fichero de texto o creando una opción de lectura como la que acabamos de explicar para obtener los datos de la máquina y guardarlos correctamente en base al modelo ideal que suponemos en el que nos vendrán los datos dados.

El programa pues, estaría disponible y listo si se consiguiera lograr la conectividad con el captador. En caso de que la respuesta fuera diferente, se debería de seguir el patrón de

respuesta del programa para aislar los valores de los datos y transmitirlos a un fichero al igual que hemos hecho en la versión final provisional. Sin embargo, debemos anotar que por problemas en la conexión no hemos conseguido hacer el programa totalmente funcional.

Otro paso para mejorar la calidad del programa indudablemente sería contar con una interfaz gráfica [17]. Hemos trabajado con el Microsoft Visual Studio que ofrece a partir de sus librerías MFC (*Microsoft Foundation Class*) la oportunidad de crear programas con una interfaz gráfica que pueda reconocer ficheros y trabajar con ellos, pudiendo gestionar una ventana de opciones que resultarían muy útiles para este proyecto. También ofrece de manera más sencilla el trabajar con una ventana de diálogo, la cual sería mejor para una primera toma de contacto. En primer lugar pensamos en recurrir a las opciones que brindaba eclipse para este tema, pero aún no están disponibles para C++. Por su parte, Microsoft Visual Studio trabaja con Visual C++, lo que complicó un poco a la hora de estudiar y desarrollar a tiempo esta solución.

7 PRESUPUESTO

Según lo estipulado por el Colegio de Ingenieros Técnicos Industriales de Tenerife (COITITF), los honorarios profesionales tipificados para trabajos por tiempo empleados ascienden a 45,10 euros la hora. El proyecto desarrollado corresponde a la asignatura de Trabajo de Fin de Grado la cual supone un total de 60 créditos, lo que corresponde a 120 horas.

Objeto	Horas	Precio/Hora (€)	Importe final(€)
Mano de obra	**120**	**45,10**	**5412,0**

Objeto	Importe (€)
Honorarios	**5412,0**
Cable adaptador USB-RS232 Zero Modem.	**29,08**
TOTAL	**5441,08**

8 CONCLUSIONS

This project tries to improve the efficiency of the operators who work with the aerosol sequential volume samplers by developing software which allow to program a range of dates

to start samplings by using a specific file used by the workers at the Agencia Estatal de Meteorología in Izaña (Tenerife).

Along the development of the project, we have encountered different problems that we have overcome in different ways. We have created a functional program that enable the operators to work with their excel files, by converting them in text files, and introducing them to our software to program the device and load the data from it.

The main problem of this project is that the connectivity failed despite testing various methods to get a response from the volume sampler. It implies also that we couldn't achieve the remote control of the device. However, we supposed an ideal response and introduced it into an Arduino Uno board to test our program and it turned out to be successful.

In conclusion, Our software is expecting to solve the connection problems to modify the real entry of data from the retainer to be ready to use by the operators in Izaña.

9 BIBLIOGRAFÍA

[1] Programa de investigación de aerosoles del Observatorio Atmosférico de Izaña. Disponible en: http://izana.aemet.es/index.php?option=com_content&view=article&id=93&Itemid=28&lang=es

[2] Referencia sobre la historia del conector RS-2-32 en ingeniatic.

Disponible en :http://ingeniatic.euitt.upm.es/index.php/tecnologias/item/581-rs-232

[3] O.E. Morales Hernández. 2003. Tesis sobre interfase gráfica, capítulo 3.

[4] National Instrument. Conceptos generales sobre la comunicación serial. Disponible en: http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1)

[5]C++. Wikipedia. Disponible en : https://es.wikipedia.org/wiki/C%2B%2B

[5] Eclipse. Wikipedia. Disponible en : https://es.wikipedia.org/wiki/Eclipse_(software)

[6]MinGW. Wikipedia. Disponible en :https://es.wikipedia.org/wiki/MinGW

[7]API de Windows. Wikipedia. Disponible en: https://es.wikipedia.org/wiki/API_de_Windows

[8] Feit,S.,1998. TCP/IP. McGrawHill.

[9] Cohoon, J.P., Davidson, J.W.,2000. Programación y diseño en C++. McGrawHill.

[10]Stroustrup, B., 2002. El lenguaje de programación C++.Addison Wesley.

[11]Cplusplus.com , donde hemos consultado diferentes librerías y funciones para utilizar en nuestro programa. Disponible en : http://www.cplusplus.com/
[12]Codeproject. Disponible en : http://www.codeproject.com/

[13]StackOverflow. Disponible en : http://es.stackoverflow.com/

[14] Arduino y C++. Disponible en : http://playground.arduino.cc/Interfacing/CPPWindows

[15] Microsoft Developer Network para el uso de funciones que se beneficien de la API de Windows. Disponible en : https://msdn.microsoft.com/es-es/
[16] LibXL . Disponible en : http://www.libxl.com/

[17] Horton, I., 2010. Ivor Horton's Beggining Visual C++ 2010.Wrox.

ESCUELA SUPERIOR

DE INGENIERÍA Y TECONOLOGÍA

TITULACIÓN: Grado en Ingeniería Electrónica Industrial y Automática

Anexo I: Código Fuente

Trabajo Fin de Grado

TÍTULO

DESARROLLO DE UN SOFTWARE PARA GESTIONAR UN CAPTADOR SECUENCIAL DE AEROSOL DESDE UN PC Y CONTROL REMOTO

AUTOR

David Valdivieso López

TUTORES

Santiago Torres Álvarez

Sergio Rodríguez González

ÍNDICE

1. INTRODUCCIÓN

En el presente Anexo se muestra el código de los distintos módulos que componen el programa de este proyecto. Se mostrará las clases Serial y Extractor y el archivo CPP con el que se han realizado las pruebas de los diferentes protocolos. Finalmente, se mostará el archivo CPP final que combina los tres módulos para el caso de ideal funcionamiento.

2. Módulo Serial. Clase Serial

2.1 Declaración de la clase (HPP)

```
/*
 * SerialClass.hpp
 *
 *  Created on: 21 de ago. de 2016
 *      Author: David
 */

#ifndef SERIALCLASS_HPP_
#define SERIALCLASS_HPP_


#include <windows.h>
#include <stdlib.h>
#include <iostream>
#include <cstring>

class Serial
{
    private:
        //Handler del Serial comm.
        HANDLE hSerial;
        //Estado de conexión.
        bool connected;
        //Obtiene diversa información de la conexión.
        COMSTAT status;
        //Sigue el rastro de los últimos errores.
        DWORD errors;
        // Parametros de la conexión serial
        COMMTIMEOUTS ComTimeOut;

    public:
        //Constructor de copia.Inicializa la comunicación serial con el
puerto comm que le pasemos como parámetro.
        Serial(const char *portName);

        //Destructor. Cierra la conexión.
        ~Serial();

        //Lee los datos del buffer. Si nbChar es mayor que el número máx de
bytes disponibles, devolverá
        //sólo los bytes disponibles. Devuelve -1 cuando no se pueden leer
los bytes que se han obtenido.
        int ReadData(char *buffer, unsigned int nbChar);

        //Escribe datos desde el buffer a través de la conexión serial.
Devuelve true si se consigue.
        bool WriteData(char *buffer, unsigned int nbChar);

        //Comprueba si estamos conextados realmente
        bool IsConnected();
```

```
};

//Funcions del modulo de protocolos pero que aprovechamos a declarar aquí.

//Calcula el BCC de un mensaje y lo añade al mensaje.
char *GetBCC(char *Message);

//Comprueba el BCC de un mensaje de leído.
bool CheckBCC(char *InMessage);

#endif /* SERIALCLASS_HPP_ */
```

2.2 Constructor

```
//Constructor de copia.Inicializa la comunicación serial con el puerto comm
que le pasemos como parámetro.

//Constructor de copia.Inicializa la comunicación serial con el puerto comm
que le pasemos como parámetro.

Serial::Serial(const char *portName)
{
//Aún no estamos conectados.
this->connected = false;

//Nos intentamos conectar a través de CreateFile. Se necesita un parámetro
HANDLE como hSerial.
this->hSerial = CreateFile(portName,
GENERIC_READ | GENERIC_WRITE,
0,
NULL,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL,
NULL);

//Comprobamos si la conexión se ha realizado correctamente.
if(this->hSerial==INVALID_HANDLE_VALUE)
{
//En caso de que no, mostramos en error por pantalla.
if(GetLastError()==ERROR_FILE_NOT_FOUND){

//Mostramos el error en caso de que sea necesario.
std::cout<<"ERROR: No se ha conseguido la conexión. Motivo: "<< portName<<"
no disponible."<<std::endl;

}
else
{
errors = GetLastError();
std::cout<<"ERROR!!! "<<errors<<std::endl;
}
}
else
{
//Si estamos conectados, inicializamos los parámetros de la conexión.
DCB dcbSerialParams = {0};
```

```
//Tratamos de obtener los parámetros actuales.
if (!GetCommState(this->hSerial, &dcbSerialParams))
{
//Si no es posible, error.
std::cout<<"failed to get current serial parameters!"<<std::endl;
}
else
{
//Definimos la conexión serial de los parametros para el intercambio de
datos.
dcbSerialParams.BaudRate=CBR_9600;

// Estos son las diferentes velocidades necesarias para los protocolos que
estamos probando.
// Dependiendo de cual probemos, cambiamos la configuración.

//     1. Protocolo MPNS.// 9600 Bd rate
//     2. Protocolo PFR USB DD-N4xx ASCII.//115200 Bd rate
//     3. Protocolo Bavaria.//1200 Bd rate

dcbSerialParams.ByteSize=8;
dcbSerialParams.StopBits=ONESTOPBIT;
dcbSerialParams.Parity=NOPARITY;
// Con Control de DTR y RTS habilitado nos aseguramos que la comunicación
se reinicia con la conexión del puerto.
dcbSerialParams.fDtrControl = DTR_CONTROL_ENABLE;
dcbSerialParams.fRtsControl = RTS_CONTROL_ENABLE;

//            dcbSerialParams.fOutX = TRUE;
//            dcbSerialParams.fInX = TRUE;


//Guardamos los parámetros y comprobamos si se han aplicado correctamente.
if(!SetCommState(hSerial, &dcbSerialParams))
{
std::cout<<"ALERTA: No se han podido establecer los parámetros para la
conexión serial."<<std::endl;
}
else
{


ComTimeOut.ReadIntervalTimeout=MAXDWORD; //
ComTimeOut.ReadTotalTimeoutMultiplier=0;
ComTimeOut.ReadTotalTimeoutConstant=0;
ComTimeOut.WriteTotalTimeoutMultiplier=0;
ComTimeOut.WriteTotalTimeoutConstant=0;

SetCommTimeouts(hSerial,&ComTimeOut);
if (!SetCommTimeouts(hSerial,&ComTimeOut)){
std::cout<<"ALERTA: no se han podido guardar los TimeOuts"<<std::endl;
}

else{
//Si todo está correcto, mostramos que estamos conectados.
this->connected = true;
//Limpiamos el buffer de cualquier rasto que pudiese haber.
PurgeComm(this->hSerial, PURGE_RXCLEAR | PURGE_TXCLEAR);
//Esperamos dos segundos.
```

```
Sleep(2000);
}
}
}
}

}
```

2.3 Destructor

```
//Constructor de copia.Inicializa la comunicación serial con el puerto comm
que le pasemos como parámetro.

Serial::~Serial()
{
//Antes de desconectar, probamos si estamos conectados.
if(this->connected)
{
//Nos desconectamos
this->connected = false;
//Cerramos la conexión serial.
CloseHandle(this->hSerial);
}
}
```

2.4 Leer por serial

```
//Lee los datos del buffer. Si nbChar es mayor que el número máx de bytes
disponibles, devolverá
//sólo los bytes disponibles. Devuelve -1 cuando no se pueden leer los
bytes que se han obtenido.

int Serial::ReadData(char *buffer, unsigned int nbChar)
{
//Número de bytes que vamos a leer.
DWORD bytesRead;
//Número de bytes que vamos a querer leer.
unsigned int toRead;

//Usamos la función ClearCommError para obtener el estado del puerto
Serial.
ClearCommError(this->hSerial, &this->errors, &this->status);

//Comprobamos si hay algo que leer.
if(this->status.cbInQue>0)
{
//Si hay algo, comprobamos si hay suficientes datos para leer el numero
requerido de caracteres
//En caso de que no, leeremos sólo los caracteres dispinibles para prevenir
el bloqueo del programa.

if(this->status.cbInQue>nbChar)
{
toRead = nbChar;
```

```
//mostramos por pantalla un aviso
std::cout<<"Aviso : Tamaño del mensaje mayor de lo esperado."<<std::endl;
}
else
{
toRead = this->status.cbInQue;
}

// Tratamos de leer el numero requerido de caracteres, devolviendo el
número de caracteres leidos correctamente.

if(ReadFile(this->hSerial, buffer, toRead, &bytesRead, NULL) )
{
return bytesRead;
}

}

//Si no se ha leido nada o ha existido un error, devolvemos 0.
return 0;

}
```

2.5 Escribir serial

```
//Escribe datos desde el buffer a través de la conexión serial. Devuelve
true si se consigue.

bool Serial::WriteData(char *buffer, unsigned int nbChar)
{
DWORD bytesSend;

//Tratamos de
if(!WriteFile(this->hSerial, (void *)buffer, nbChar, &bytesSend, 0))
{
std::cout<<"error al enviar datos."<<std::endl;
//En caso de no funcionar, obtenemos error com y devolvemos falso.
ClearCommError(this->hSerial, &this->errors, &this->status);

return false;
}
else
return true;
}
```

2.6 Comprobación de conexión

```
//Comprueba si estamos conextados realmente

bool Serial::IsConnected()
{
//Devolvemos el estado de la conexión.
return this->connected;
}
```

3. Módulo de extracción de datos. Clase extractor

```
// Parámetros globales para el CPP

using namespace std;

int i = 0;
char tab = '\t';

string ENombresParametros[5]={"Fecha Inicio","Duracion","Hora de Inicio
Muestreo","Caudal","Numero de Muestros"};
string SNombresParametros[10]={"Numero de ciclo","Estado","Fecha
Inicio","Fecha fin","Volumen Normalizado","Caudal Medio","Volumen
Ambiente","Temperatura media","Presion Media","Caudal muestreado"};
```

3.1 Declaración de la clase

```
/*
 * ExcelClass.hpp
 *
 *  Created on: 21 de ago. de 2016
 *      Author: David
 */

#ifndef EXCELCLASS_HPP_
#define EXCELCLASS_HPP_



#define mes 31
#define AnchuraCol 25
#define Precision 2

#include<fstream>
#include<iostream>
#include<stdlib.h>
#include<string>
#include<iomanip>
```

```
#include<windows.h>
#include<ctype.h>


class extractor

{

private:


public:


	std::string ExcelArchive;

	//Datos para introducir al captador.

	std::string EFechaMuestreo[mes];
	std::string EDuracionMuestreo[mes];
	std::string EHoraMuestreo[mes];
	float ECaudal[mes];
	int ENumeroMuestras[mes];

	//Datos que leemos del captador.

	int Flag[mes];
	int SNumeroCiclos[mes];
	std::string SEstado[mes];
	std::string SFechaInicioMuestreo[mes];
	std::string SHoraInicioMuestreo[mes];
	std::string SFechaFinMuestro[mes];
	std::string SHoraFinMuestreo[mes];
	float SVolumenNormalizado[mes];
	float SCaudalNormalizado[mes];
	float SVolumenAmbiente[mes];
	std::string SVolumenAmbPrint[mes];// No se consigue para guardar en
el fichero un buen resultado con float y fout.
	float STemperaturaMedia[mes];
	float SPresionMedia[mes];
	float SCaudalPorgramado[mes];
	std::string SDuracion[mes]; // Aparece con GetState

	int TotalDatos;
	bool Extraccion;
	bool ProcessDay;
	bool Principal; //Para diferenciar al secundario.
	int LineasEscritas; // Para conocer el número de lineas escritas.



	// METHODS

	extractor();

	extractor(std::string NewArchive);
```

```
        // Obtiene la informacion del archivo txt.
        void GetData();

        // Enseña información en pantalla

        void ShowData();

        // Guarda los datos en un archivo.
        void SaveData(std::string SaveArchive);


        // Métodos para obtener los días de proceso con flag a 1.

        bool DateProcess();

        bool DateProcess(std::string ADate);

        bool DateProcess(int vector[]);

        ~extractor();


};


#endif /* EXCELCLASS_HPP_ */
```

3.2 Constructor

```
// Inicializamos los vectores que contendran al información entrante del
archivo.

// Constructor predeterminado.

extractor::extractor()
{
// Constructor predeterminado


// Inicializamos los vectores que contendran al información entrante del
archivo


//Datos para introducir al captador.

EFechaMuestreo[mes-1]={"0"};
EDuracionMuestreo[mes-1]={"0"};
EHoraMuestreo[mes-1]={"0"};
ECaudal[mes-1]={};
ENumeroMuestras[mes-1]={};


//Datos que leemos del captador.
```

```
Flag[mes-1]={};
SNumeroCiclos[mes-1]={};
SEstado[mes-1]={"0"};
SFechaInicioMuestreo[mes-1]={"0"};
SHoraInicioMuestreo[mes-1]={"0"};
SFechaFinMuestro[mes-1]={"0"};
SHoraFinMuestreo[mes-1]={"0"};
SVolumenNormalizado[mes-1]={};
SCaudalNormalizado[mes-1]={};
SVolumenAmbiente[mes-1]={};
STemperaturaMedia[mes-1]={};
SPresionMedia[mes-1]={};
SCaudalPorgramado[mes-1]={};

SVolumenAmbPrint[mes-1]={"0"};

// Otros parametros

TotalDatos=0; // Total de datos extraídos del archivo
Extraccion = false; // Parametro booleano que se activa en GetData() para
que , en caso de no tener valores , no se puedan realizaar los otros
métodos.
ProcessDay = false; // Parámetro que nos sirve para en DateProcess() saber
si hay que poner algún día al captador en funcionamiento.
Principal = true;
LineasEscritas = 0;

};
```

3.3 Constructor de copia

```
// Constructor de copia.


extractor::extractor(string NewArchive)
{

this->ExcelArchive = NewArchive;

//Datos para introducir al captador.

EFechaMuestreo[mes-1]={"0"};
EDuracionMuestreo[mes-1]={"0"};
EHoraMuestreo[mes-1]={"0"};
ECaudal[mes-1]={};
ENumeroMuestras[mes-1]={};

//Datos que leemos del captador.

Flag[mes-1]={};
SNumeroCiclos[mes-1]={};
SEstado[mes-1]={"0"};
SFechaInicioMuestreo[mes-1]={"0"};
SHoraInicioMuestreo[mes-1]={"0"};
SFechaFinMuestro[mes-1]={"0"};
```

```
SHoraFinMuestreo[mes-1]={"0"};
SVolumenNormalizado[mes-1]={};
SCaudalNormalizado[mes-1]={};
SVolumenAmbiente[mes-1]={};
STemperaturaMedia[mes-1]={};
SPresionMedia[mes-1]={};
SCaudalPorgramado[mes-1]={};

SVolumenAmbPrint[mes-1]={"0"};


// Otros parametros

TotalDatos=0; // Total de datos extraídos del archivo
Extraccion = false; // Parametro booleano que se activa en GetData() para
que , en caso de no tener valores , no se puedan realizaar los otros
métodos.
ProcessDay = false; // Parámetro que nos sirve para en DateProcess() saber
si hay que poner algún día al captador en funcionamiento.
Principal = true;
LineasEscritas = 0;

};
```

3.4 Extracción de datos de un fichero .txt

```
// Método que sirve para extraer los datos de un fichero de texto.
// El orden debe mantenerse siempre para que se realize la extracción con
éxito.

void extractor::GetData(){

cout<< "Archivo de donde extraer los datos : "<< ExcelArchive<<endl;

// Proceso de extracción. Extraemos el archivo.

//Creamos un objeto "fin" para poder extraer el archivo.
ifstream fin;
fin.open(this->ExcelArchive.c_str());
if(!fin){

// Error mostrado en caso de que no pueda abrirse el archivo.

cerr << "No puede abrirse "<< this->ExcelArchive
<< " para extraer los datos."<< endl;
return;
}

//Este código en texto permite desechar tantas filas [5] como sean
necesarias de
//parámetros que no sean valores a recibir.

//                string trash[5];
//                for(i=0;i<5;i++){
//                      getline(fin,trash[i]);
//                }
```

```
// Obtenemos las variables en el orden concreto mostrado en el bucle for.
// Los valores se pueden mostrar si decidimos usar cout debajo de cada cin.


for(i=0; i < mes ; i++){


//    cout << i <<" ";

// En caso de que el fichero siga teniendo contenido que sacar, seguimos
iterando.

if(fin != NULL){


// Guardando en la memoria de las variables del programa los datos del
archivo.

// Datos para enviar al captador.

fin >> this->Flag[i];
//cout << Flag[i]<<endl;

fin >> this->EFechaMuestreo[i];
//cout << EFechaMuestreo[i]<<endl;

fin >> this->EDuracionMuestreo[i];
//cout << EDuracionMuestreo[i]<<endl;

fin >> this->EHoraMuestreo[i];
//cout << EHoraMuestreo[i]<<endl;

fin >> this->ECaudal[i];
//cout << ECaudal[i]<<endl;

fin >> this->ENumeroMuestras[i];
//cout << ENumeroMuestras[i]<<endl;


// Datos que recibiremos del captador.

fin >> this->SNumeroCiclos[i];
//cout << SNumeroCiclos[i]<<endl;

fin >> this->SEstado[i];
//cout << SEstado[i]<<endl;

fin >> this->SFechaInicioMuestreo[i];
//cout <<"Inicio"<< SFechaInicioMuestreo[i]<<endl;

fin >> this->SHoraInicioMuestreo[i];
//cout << SHoraInicioMuestreo[i]<<endl;

fin >> this->SFechaFinMuestro[i];
//cout <<"Fin"<< SFechaFinMuestro[i]<<endl;

fin >> this->SHoraFinMuestreo[i];
//cout << SHoraFinMuestreo[i]<<endl;
```

```
fin >> this->SVolumenNormalizado[i];
//cout << SVolumenNormalizado[i]<<endl;

fin >> this->SCaudalNormalizado[i];
//cout << SCaudalNormalizado[i]<<endl;

fin >> this->SVolumenAmbiente[i];
//cout << SVolumenAmbiente[i]<<endl;

fin >> this->STemperaturaMedia[i];
//cout << STemperaturaMedia[i]<<endl;

fin >> this->SPresionMedia[i];
//cout << SPresionMedia[i]<<endl;

fin >> this->SCaudalPorgramado[i];
//cout << SCaudalPorgramado[i]<<endl;



}

else{
cout<<endl;
cout << " Extracción total de  " << i-1<< " filas de datos."<< endl;
break;
}
this->TotalDatos = i;
}


cout << " Extracción completa .";
Extraccion = true;

// Cerramos el vínculo con el fichero cerrando el objeto.

fin.close();


};
```

3.5 Mostrar datos por pantalla

```
// Método para mostrar datos por pantalla.

void extractor::ShowData() {

// Mostramos los Nombres de los parametros

if (Principal==true){ // Solo se mostraran los parametros de entrada en el
principal
for(i=0;i<5;i++){
cout<< setw(AnchuraCol)<<ENombresParametros[i];
}
}
for(i=0;i<10;i++){
```

```
cout<< setw(AnchuraCol)<<SNombresParametros[i];
}

cout<<setw(AnchuraCol)<<endl;

// Dibujamos una barra para diferenciar

cout<< setfill('=')<<setw(8*AnchuraCol*2)<<"="<<endl;
cout<<setfill(' ')<<fixed ;

// Mostramos los valores correspondientes


for(i=0; i <= TotalDatos; i++){

//    cout << i <<" ";

// Mostramos datos que se introducen en el captador


// Puede hacer falta "setprecision(Precision)"
if(Principal==true){

cout <<setw(AnchuraCol)<< EFechaMuestreo[i] ;

cout <<setw(AnchuraCol)<< EDuracionMuestreo[i];

cout <<setw(AnchuraCol)<< EHoraMuestreo[i];

cout <<setprecision(Precision)<<setw(AnchuraCol)<< ECaudal[i];

cout <<setw(AnchuraCol)<< ENumeroMuestras[i];
}

// Mostramos datos que se extraen del captador

cout <<setw(AnchuraCol)<< SNumeroCiclos[i];

cout<<setw(AnchuraCol)<< SEstado[i];

cout<<setw(AnchuraCol)<< SFechaInicioMuestreo[i]+ ' '+SHoraFinMuestreo[i];

cout<<setw(AnchuraCol)<< SFechaFinMuestro[i]+' '+ SHoraFinMuestreo[i];

cout<<setprecision(Precision)<<setw(AnchuraCol)<< SVolumenNormalizado[i];

cout<<setprecision(Precision)<<setw(AnchuraCol)<< SCaudalNormalizado[i];

cout<<setprecision(10)<<setw(AnchuraCol)<< SVolumenAmbiente[i];

cout<<setprecision(Precision)<<setw(AnchuraCol)<< STemperaturaMedia[i];

cout<<setprecision(Precision)<<setw(AnchuraCol)<< SPresionMedia[i];

cout<<setprecision(Precision)<<setw(AnchuraCol)<< SCaudalPorgramado[i];


cout<<endl;
}
```

```
cout << "Total de días en este archivo : "<< TotalDatos<< endl;


};
```

3.6 Guardar datos en un fichero .txt

```
// Método para guardar los datos en un fichero. Es necesario pasar el
nombre del fichero.

void extractor::SaveData(string SaveArchive){

// Comprobamos si el fichero existe ya.
string YesOrNo = "0";

ifstream fin(SaveArchive.c_str());
if(fin){
cerr << "El archivo "<< SaveArchive
<< " ya existe."<< endl;
cerr << "¿ Seguro que desea sobreescribirlo? "<<endl;
Sleep(200);
cout <<" Yes / No"<<endl;
cin >> YesOrNo;

if(YesOrNo == "No" || YesOrNo == "N"|| YesOrNo == "no"|| YesOrNo == "n" ||
YesOrNo =="0"){

cout <<" Vuelta al menu..."<<endl;

system("PAUSE");
return;
}

else if(YesOrNo == "Yes" || YesOrNo == "Y"|| YesOrNo == "yes"|| YesOrNo ==
"y" || YesOrNo =="1"){
}

}

// Creamos un objeto "fout" para poder guardar los datos.

ofstream fout;
fout.open(SaveArchive.c_str(), ios::out | ios::trunc);

if (fout.is_open()){
cout<<endl;
cout<< "Se procede a introducir los datos ; "<<endl;

// Podemos mostrar los datos antes si queremos quitando las líneas de
anotaciones a continuación.

//this->ShowData();


cout<<"Dentro y escribiendo"<<endl;
```

```
// Se guardan los datos en el archivo.

//Usamos las mismas funciones que en  ShowData(), pero fout se usa para
introducir los datos en el archivo.

// No ponemos los nombres de los parámetros por que es más incómodo para
los trabajadores que estén pasando después los datos a otro archivo.

for(i=0; i <= TotalDatos; i++){
//  Guardamos los datos que se introducen en el captador

fout <<tab<<this->Flag[i];


fout <<tab<< this->EFechaMuestreo[i] ;

fout <<tab<< this->EDuracionMuestreo[i];

fout <<tab<< this->EHoraMuestreo[i];

fout <<tab<< this->ECaudal[i];

fout <<tab<< this->ENumeroMuestras[i];

// Guardamos los datos que se extraen del captador

fout <<tab<< this->SNumeroCiclos[i];

fout<<tab<< this->SEstado[i];

fout<<tab<< this->SFechaInicioMuestreo[i]<<" "<< this-
>SHoraInicioMuestreo[i];

fout<<tab<< this->SFechaFinMuestro[i]<<" "<< this->SHoraFinMuestreo[i];

fout<<tab<< this->SVolumenNormalizado[i];

fout<<tab<< this->SCaudalNormalizado[i];

//fout<<tab<<setprecision(10)<<SVolumenAmbiente[i];  //Float no da un buen
resultado. probamos con string.
fout<<tab<<this->SVolumenAmbPrint[i];

fout<<tab<<this->STemperaturaMedia[i];

fout<<tab<< this->SPresionMedia[i];

fout<<tab<< this->SCaudalPorgramado[i];


fout<<endl;

}

cout << "Total de días en este archivo : "<< TotalDatos<< endl;


fout.close();
```

```
}else cerr << "failed to open file\n"<<endl;

};
```

3.7 Métodos para contrastar los días de iniciación de proceso de medida

3.7.1 Comprobar los días con flag a 1

```
// Método para comprobar los días con flag a 1.
bool extractor::DateProcess(){

this->ProcessDay = false;

cout<<"Las fechas para el uso del equipo son : "<<endl;
for(int i=0;i<this->TotalDatos;i++){
if(this->Flag[i]==true){
this->ProcessDay =true;
cout<<this->EFechaMuestreo[i]<<endl;
}
}

if(this->ProcessDay ==false)
cout<<"Ninguna fecha tiene prevista una puesta en marcha."<<endl;

// Se devuelve verdadero o falso en caso de tener una fecha( o más) o no
tener ninguna.

return this->ProcessDay;


};
```

3.7.2 Comprobar si un día tiene el flag a 1.

```
// Método para comprobar si un día tiene flag a 1.
bool extractor::DateProcess(string ADate){

this->ProcessDay = false;

cout<<"Las fechas para el uso del equipo son : "<<endl;
for(int i=0;i<this->TotalDatos;i++){
if(this->Flag[i]==true){

if(ADate==this->EFechaMuestreo[i]){
cout<<"La fecha"<<this->EFechaMuestreo[i]<<"está apuntada para el
proceso."<<endl;
this->ProcessDay =true;
}
}
}

if(this->ProcessDay ==false)
cout<<"La fecha no tiene prevista una puesta en marcha."<<endl;

// Se devuelve verdadero o falso en caso de tener una fecha( o más) o no
tener ninguna.
```

```
return this->ProcessDay;
};
```

3.7.3 Comprobación de número de días con flag a 1.

```
bool extractor::DateProcess(int vector[]){

this->ProcessDay = false;
int b=0;
cout<<"Las fechas para el uso del equipo son : "<<endl;
for(int i=0;i<this->TotalDatos;i++){
if(this->Flag[i]==true){
this->ProcessDay =true;
cout<<this->EFechaMuestreo[i]<<endl;
vector[b]=i;
b++;
}
}

if(this->ProcessDay ==false)
cout<<"Ninguna fecha tiene prevista una puesta en marcha."<<endl;

// Se devuelve verdadero o falso en caso de tener una fecha( o más) o no
tener ninguna.

return this->ProcessDay;

}
```

3.8 Destructor

```
// Destructor.
extractor::~extractor(){

};
```

4. Módulo de protocolo y comunicaciones. Versión de prueba con diferentes protocolos.

4.1 Función para añadir BCC al final de un mensaje

```
char *GetBCC(char *Message){

int length;
int BCC1 ={0x00};
int BCC2 ={0x00};
length = strlen(Message);
int MessageInt[length];
char BCC1Out[1];
char BCC2Out[1];



for(int i=0;i<(length-2);i++){

// Guardamos en valor actual de la iteración en otro array.
```

```
MessageInt[i]=Message[i];
// Hacemos el calculo de BCC utilizando el operador xor "^".
BCC1 ^= MessageInt[i];
i++;

MessageInt[i]=Message[i];
BCC2 ^= MessageInt[i];
}

// Convertimos la variable int a un string.

itoa(BCC1,BCC1Out,16);
itoa(BCC2,BCC2Out,16);

//        std::cout<<"BCC1Out ["<<BCC1Out<<"]"<<std::endl;
//        std::cout<<"BCC2Out ["<<BCC2Out<<"]"<<std::endl;

// Lo concatenamos al final del mensaje y lo devolvemos.
strcat(Message,BCC1Out);
strcat(Message,BCC2Out);

return Message;

}
```

4 . 2 Función para comprobar BCC de un mensaje extraído

```
// ----------- ASCII CHARACTERS FOR TUNNELING
char STX [1]= {02};
char ETX [1]= {03};
char CR[1]= {0x0D};
char LF[1]= {0x0A};
int ArrayLength = 0;

//------ POINTER USED AS A BUFFER TO SEND THE MESSAGE
char *sendmessage;

// VARIABLE USED FOR SWITCH
int pick=0;
int pick2=0;
int pick3=0;
//------------COMMANDS FOR MPNS

char StartProbe[81] = "Start Probe [PW=nnnnnn]* [Date=dd.mm.yy]
[Time=hh:mm] [Dura=hhh:mm] [Flow=nn.n] ";
char StopProbe[40] = "Stop Probe [Date=dd.mm.yy] [Time=hh:mm]";
char StartCycle[104] = "Start Cycle [PW=nnnnnn][Probes=n] [Date=dd.mm.yy]
[Time=hh:mm] [Dura=hhh:mm] [Cycle=hhh:mm] [Flow=nn.n]";
char StopCycle[55] = "Stop Cycle [Date=dd.mm.yy] [Time=hh:mm]
[AfterProbe=n]";
char GetState[10]= "Get State";


char *PStartProbe, *PStopProbe, *PStartCycle, *PStopCycle, *PGetState;

//--------- VARIABLES FOR MPNS

char day[2],month[2],year[2];
```

```
char hour[2],minute[2];
char PW[5];
char Flow[2],Flow2[1];
char Probes[1];
char Dhour[3],Dminute[2];
char Chour[3],Cminute[2];
char AfterProbe[1];

//-------- COMMANDS FOR PFR USB DD-N4xx ASCII


// GET COMMANDS : GET;Command

char GETALL[8]="GET;ALL";
char GETFLOW[9]="GET;FLOW";

//UNUSED COMMANDS ON GREEN

// char GETTEMPERATURE[16]="GET;TEMPERATURE";
// char GETPREASSURE[14]="GET;PREASSURE";
// char GETID[7]="GET;ID";
// SET COMMANDS : SET;Command;Value<CR>

char SETFLOW[10]="SET;FLOW;";


void lecturahand(Serial* SP);


int main()
{
char COMPORT[5];

cout<<"Seleccione puerto COM..."<<endl;
cout<<"Introduzca COM seguido del número del puerto. Ej: COM4 "<<endl;


Serial* SP = new Serial("COM3");  // COM PORT SELECTED



// -------------NOS CONECTAMOS POR PUERTO SERIE--------------------//
if (SP->IsConnected()){
printf("We're connected");
cout<<endl;
}

else{

printf("Connection failed");
return 0;
}

//    while (SP->IsConnected()){                    // ------------- SI
ESTAMOS CONECTADOS... EMPEZAMOS EL PROGRAMA ----------//



// -------------    PRIMERA LECTURA               ----------------------//
```

```
char incomingData [256]= "";



int dataLength = 255;
int readResult = 0;


if(SP->IsConnected()&& Stop==false)
{
cout<<" read test 1 "<<endl;
readResult = SP->ReadData(incomingData,dataLength);
printf("Bytes read: (0 means no data available)\n %i\n",readResult);
incomingData[readResult] = 0;
if(readResult!=0)
cout<<"Incoming Data..."<<readResult<<" . . ."<<incomingData<<endl;

Stop = true;


}



// -------------ELEGIMOS UN MENSAJE PARA ENVIAR ----------------------//


//-------------------------------------------------------------------------
Eligiendo un mensaje a enviar.

while(1){

cout<<" Elegir protocolo para enviar mensajes "<<endl;
cout<<" 1. Protocolo MPNS."<<endl; // 9600 Bd rate
cout<<" 2. Protocolo PFR USB DD-N4xx ASCII"<<endl; //115200 Bd rate
cout<<" 3. Protocolo Bavaria."<<endl; //1200 Bd rate

int protocol =0;

cin>>protocol;

switch(protocol)

{
// Inicio del switch (protocol)
```

4.3 Protocolo MPNS

```
case 1:

{
// Inicio del caso de mensajes MPNS

// 1 - MPNS MESSAGES

cout<<" Choose a message to send... "<<endl;
cout<<"1. Command MPNS: Start Probe"<<endl;
```

```
cout<<"2. Command MPNS: Stop Probe"<<endl;
cout<<"3. Command MPNS: Start Cycle"<<endl;
cout<<"4. Command MPNS: Stop Cycle"<<endl;
cout<<"5. Command MPNS: Get state"<<endl;
cout<<"6. Close Serial Port"<<endl;



cin>>pick;
cout<<endl;

switch(pick){

//--------------------------------------------------------------------------
--------OP1 : START PROBE

case 1:
{

//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

//Start Probe [PW=nnnnnn]* [Date=dd.mm.yy] [Time=hh:mm] [Dura=hhh:mm]
[Flow=nn.n]

// SIZE =  Message(81) + CR + LF;
char StartProbeMessage[83] ;


// Utilizamos un puntero para hacer sobre él las modificaciones del
mensaje. Esto es, cambiar los valores
// que vamos introduciendo al captador.


PStartProbe = StartProbe;

cout<<"Start Probe [PW=nnnnnn]* [Date=dd.mm.yy] [Time=hh:mm] [Dura=hhh:mm]
[Flow=nn.n] "<<endl;

// (17)[PW=nnnnnn]*



cout<<"[PW=nnnnnn]"<<endl;

int x=0;

for(int i=6;i>0;i--){

cout<<"Introduzca"<<i<<" digitos para PW"<<endl;

cin>>PW[x];

x++;

}

memmove(PStartProbe+16, PW, 6);

//(32)(35)(38)[Date=dd.mm.yy]
```

```
cout<<"[Date=dd.mm.yy]"<<endl;


for(int i=0;i<2;i++){
cout<<"Introduzca dia (2 digitos)"<<endl;
cin>>day[i];
}
memmove(PStartProbe+31, day, 2);

for(int i=0;i<2;i++){
cout<<"Introduzca mes (2 digitos)"<<endl;
cin>>month[i];
}
memmove(PStartProbe+34, month, 2);
for(int i=0;i<2;i++){
cout<<"Introduzca año (2 digitos)"<<endl;
cin>>year[i];
}
memmove(PStartProbe+37, year, 2);
//(48)(51)[Time=hh:mm]
cout<<"[Time=hh:mm]"<<endl;


for(int i=0;i<2;i++){
cout<<"Introduzca Hora (2 digitos)"<<endl;
cin>>hour[i];
}
memmove(PStartProbe+47, hour, 2);
for(int i=0;i<2;i++){
cout<<"Introduzca Minuto (2 digitos)"<<endl;
cin>>minute[i];
}
memmove(PStartProbe+50, minute, 2);
```

```
//(61)(65)[Dura=hhh:mm]

cout<<"[Dura=hhh:mm]"<<endl;


string Archivo = "005:30";
char StringToChar[10];
stringstream ss (stringstream::in | stringstream::out);

ss << Archivo;

string test = ss.str();

strncpy(StringToChar, test.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';
cout<<"antes de copy"<<endl;
cout<<StringToChar<<endl;
memmove(PStartProbe+60, StringToChar, 6);



//(75)(78)  [Flow=nn.n]

cout<<"[Flow=nn.n]"<<endl;



for(int i=0;i<2;i++){

cout<<"Introduzca Flujo (2 digitos)"<<endl;

cin>>Flow[i];

}


memmove(PStartProbe+74, Flow, 2);


cout<<"Introduzca Flow (1 digito)"<<endl;

cin>>Flow2;


memmove(PStartProbe+77, Flow2,1);


strcpy(StartProbeMessage,PStartProbe);

strcat(StartProbeMessage,CR);

strcat(StartProbeMessage,LF);

sendmessage = StartProbeMessage;



cout<<"Sending.. :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);
```

```
cout<<sendmessage<<endl;

cout<<"end of message"<<endl;
// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ----------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);


Sleep(500);

cout<<" reading incoming data .."<<endl;
SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Receiving : "<<incomingData<<endl;

cout<<"-------------------"<<endl;

break;

}
//-------------------------------------------------------------------------
-----------OP2 : STOP PROBE



case 2:
{

//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

// char StopProbe[40] = "Stop Probe [Date=dd.mm.yy] [Time=hh:mm]";

// SIZE (42) = Message(40) + CR + LF;

char StopProbeMessage[42] ;

// Utilizamos un puntero para hacer sobre él las modificaciones del
mensaje. Esto es, cambiar los valores
// que vamos introduciendo al captador.


PStopProbe = StopProbe;


cout<<"Stop Probe [Date=dd.mm.yy] [Time=hh:mm]"<<endl;

// (17)(19)(21)[Date=dd.mm.yy]*


cout<<"[Date=dd.mm.yy]"<<endl;



for(int i=0;i<2;i++){

cout<<"Introduzca dia (2 digitos)"<<endl;

cin>>day[i];
```

```
}
memmove(PStopProbe+17, day, 2);

for(int i=0;i<2;i++){
cout<<"Introduzca mes (2 digitos)"<<endl;
cin>>month[i];
}
memmove(PStopProbe+19, month, 2);
for(int i=0;i<2;i++){
cout<<"Introduzca año (2 digitos)"<<endl;
cin>>year[i];
}
memmove(PStopProbe+21, year, 2);


//(31)(33)[Time=hh:mm]
cout<<"[Time=hh:mm]"<<endl;

for(int i=0;i<2;i++){
cout<<"Introduzca Hora (2 digitos)"<<endl;
cin>>hour[i];
}
memmove(PStopProbe+31, hour, 2);
for(int i=0;i<2;i++){
cout<<"Introduzca Minuto (2 digitos)"<<endl;
cin>>minute[i];
}
memmove(PStopProbe+33, minute, 2);
strcpy(StopProbeMessage,PStopProbe);
strcat(StopProbeMessage,CR);
strcat(StopProbeMessage,LF);
```

```
sendmessage = StopProbeMessage;

cout<<"Sending.. :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<sendmessage<<endl;

cout<<"end of message"<<endl;

// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ----------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" reading incoming data .."<<endl;
SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Receiving : "<<incomingData<<endl;

cout<<"-------------------"<<endl;

break;
}
//-----------------------------------------------------------------OP3 : START
PROBE

case 3:
{

//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

//     char StartCycle[104] = "Start Cycle [PW=nnnnnn][Probes=n]
[Date=dd.mm.yy] [Time=hh:mm] [Dura=hhh:mm] [Cycle=hhh:mm] [Flow=nn.n]";

// SIZE (107) = Message(105) CR + LF ;
char StartCycleMessage[110] ;

cout<<"Start Cycle [PW=nnnnnnn][Probes=n] [Date=dd.mm.yy] [Time=hh:mm]
[Dura=hhh:mm] [Cycle=hhh:mm] [Flow=nn.n]"<<endl;



// Utilizamos un puntero para hacer sobre él las modificaciones del
mensaje. Esto es, cambiar los valores
// que vamos introduciendo al captador.

PStartCycle = StartCycle;


// (16)[PW=nnnnnn]

cout<<"[PW=nnnnnn]"<<endl;

int x=0;

for(int i=6;i>0;i--){
```

```
cout<<"Introduzca"<<i<<" digitos para PW"<<endl;
cin>>PW[x];
x++;
}
memmove(PStartCycle+16, PW, 6);

// (32)[Probes=n]
cout<<"[Probes=n]"<<endl;
cout<<"Introduzca Número de pruenbas (1 digito)"<<endl;
cin>> Probes;

memmove(PStartCycle+31, Probes, 1);
// (41)(44)(47)   [Date=dd.mm.yy]

cout<<"[Date=dd.mm.yy]"<<endl;
for(int i=0;i<2;i++){
cout<<"Introduzca dia (2 digitos)"<<endl;
cin>>day[i];
}
memmove(PStartCycle+40, day, 2);
for(int i=0;i<2;i++){
cout<<"Introduzca mes (2 digitos)"<<endl;
cin>>month[i];
}
memmove(PStartCycle+43, month, 2);
for(int i=0;i<2;i++){
cout<<"Introduzca año (2 digitos)"<<endl;
cin>>year[i];
}
memmove(PStartCycle+46, year, 2);
```

```
//(57)(60)[Time=hh:mm]
cout<<"[Time=hh:mm]"<<endl;
for(int i=0;i<2;i++){
cout<<"Introduzca Hora (2 digitos)"<<endl;
cin>>hour[i];
}
memmove(PStartCycle+56, hour, 2);
for(int i=0;i<2;i++){
cout<<"Introduzca Minuto (2 digitos)"<<endl;
cin>>minute[i];
}
memmove(PStartCycle+59, minute, 2);



//(70)(74)[Dura=hhh:mm]
cout<<"[Dura=hhh:mm]"<<endl;

for(int i=0;i<3;i++){
cout<<"Introduzca horas de duración (3 digitos)"<<endl;
cin>>Dhour[i];
}

memmove(PStartCycle+69, Dhour, 3);
for(int i=0;i<2;i++){
cout<<"Introduzca minutos de duración (2 digitos)"<<endl;
cin>>Dminute[i];
}
memmove(PStartCycle+73, Dminute, 2);
//(85)(89)[Cycle=hhh:mm]

cout<<"[Cycle=hhh:mm]"<<endl;
```

```
for(int i=0;i<3;i++){

cout<<"Introduzca horas de duración del Ciclo(3 digitos)"<<endl;

cin>>Chour[i];

}

memmove(PStartCycle+84, Chour, 3);

for(int i=0;i<2;i++){

cout<<"Introduzca minutos de duración del Ciclo (2 digitos)"<<endl;

cin>>Cminute[i];

}

memmove(PStartCycle+88, Cminute, 2);
//(99)(100) [Flow=nn.n]

cout<<"[Flow=nn.n]"<<endl;


for(int i=0;i<2;i++){

cout<<"Introduzca Flujo (2 digitos)"<<endl;

cin>>Flow[i];

}

memmove(PStartCycle+98, Flow, 2);

cout<<"Introduzca Flow (1 digito)"<<endl;

cin>>Flow2;

memmove(PStartCycle+101, Flow2,1);


strcpy(StartCycleMessage,PStartCycle);

strcat(StartCycleMessage,CR);

strcat(StartCycleMessage,LF);

sendmessage = StartCycleMessage;

cout<<"Sending.. :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);
```

```
cout<<sendmessage<<endl;

cout<<"end of message"<<endl;


// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ----------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" reading incoming data .."<<endl;
SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Receiving : "<<incomingData<<endl;

cout<<"-------------------"<<endl;

break;
}



//------------------------------------------------------------------------
------- OP 4 . STOP CYCLE




case 4:
{

//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

//  char StoptCycle[55] = "Stop Cycle [Date=dd.mm.yy] [Time=hh:mm]
[AfterProbe=n]";

// SIZE (57) =  Message(55) + CR + LF;

char StopCycleMessage[61] ;

PStopCycle = StopCycle;

cout<<"Stop Cycle [Date=dd.mm.yy] [Time=hh:mm] [AfterProbe=n]"<<endl;

// (17)(20)(23)[Date=dd.mm.yy]*


cout<<"[Date=dd.mm.yy]"<<endl;


for(int i=0;i<2;i++){

cout<<"Introduzca dia (2 digitos)"<<endl;

cin>>day[i];

}
```

```
memmove(PStopCycle+17, day, 2);

for(int i=0;i<2;i++){
cout<<"Introduzca mes (2 digitos)"<<endl;
cin>>month[i];
}
memmove(PStopCycle+20, month, 2);
for(int i=0;i<2;i++){
cout<<"Introduzca año (2 digitos)"<<endl;
cin>>year[i];
}
memmove(PStopCycle+23, year, 2);

//(33)(36)[Time=hh:mm]
cout<<"[Time=hh:mm]"<<endl;

for(int i=0;i<2;i++){
cout<<"Introduzca Hora (2 digitos)"<<endl;
cin>>hour[i];
}
memmove(PStopCycle+33, hour, 2);
for(int i=0;i<2;i++){
cout<<"Introduzca Minuto (2 digitos)"<<endl;
cin>>minute[i];
}
memmove(PStopCycle+36, minute, 2);

//                      [AfterProbe=n]
cout<<"[AfterProbe=n]"<<endl;
cout<<"Introduzca After Probe (1 digito)"<<endl;
cin>>AfterProbe;
```

```
memmove(PStopCycle+52,AfterProbe,1);

strcpy(StopCycleMessage,PStopCycle);

strcat(StopCycleMessage,CR);

strcat(StopCycleMessage,LF);

sendmessage = StopCycleMessage;

// Mandando el mensaje por puerto serial.


cout<<"Sending.. :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<sendmessage<<endl;

cout<<"end of message"<<endl;

// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ---------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" reading incoming data .."<<endl;

SP->ReadData(incomingData,sizeof(incomingData)-1);

cout<<"Receiving : "<<incomingData<<endl;

cout<<"-------------------"<<endl;

break;
}



//------------------------------------------------------------------------
------- OP5.GET STATE

case 5:
{

//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

//  char GetState[10]= "Get State";

// SIZE (12) =  Message(10) + CR + LF;

char GetStateMessage[12] ;

strcpy(GetStateMessage,GetState);
```

```
strcat(GetStateMessage,CR);

//                              strcat(GetStateMessage,LF);

sendmessage=GetStateMessage;
//                              sendmessage = GetBCC(GetStateMessage);


cout<<"Sending.. :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<sendmessage<<endl;

cout<<"end of message"<<endl;

// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE --------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" reading incoming data .."<<endl;

SP->ReadData(incomingData,sizeof(incomingData)-1);

cout<<"Receiving : "<<incomingData<<endl;

cout<<"--------------------"<<endl;

Sleep(10000);

cout<<" reading incoming data .."<<endl;

SP->ReadData(incomingData,sizeof(incomingData)-1);

cout<<"Receiving : "<<incomingData<<endl;

cout<<"--------------------"<<endl;
break;
}


case 6:
{
cout<<"cerrando puerto serie..."<<endl;
SP->~Serial();

return 0;
break;
}


default:
{
cout<<"No se reconoce la elección"<<endl;
break;
}

break;
```

```
// FIN SWITCH PICK1
}

//FIN CASE '1' PROTOCOL
break;
}
```

4.4 Protocolo PFR USB

```
case 2:

{


// 2 -PFR USB MESSAGES

cout<<" Choose a message to send..."<<endl;
cout<<"1. GET ..."<<endl;
cout<<"2. SET FLOW"<<endl;

cin>>pick2;


switch(pick2){


case 1:{

cout<<"1. GET ALL"<<endl;
cout<<"2. GET FLOW"<<endl;


cin>>pick3;

switch(pick3){


case 1:{

// GET ALL

// char GETALL[8]="GET;ALL"

//SIZE (9)= Message (8) + CR

char GetAllMessage[9];

strcpy(GetAllMessage,GETALL);

strcat(GetAllMessage,CR);

sendmessage = GetAllMessage;

ArrayLength = strlen(sendmessage);

cout<<sendmessage<<endl;

cout<<"end of message"<<endl;
```

```
// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ----------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

if(SP->IsConnected()){
cout<<" reading incoming data .."<<endl;
SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Receiving : "<<incomingData<<endl;

cout<<"-------------------"<<endl;


}

break;

//FIN CASE 1 GET ALL
}

case 2:{

// GET FLOW

// char GETFLOW[9]="GET;FLOW";

//SIZE (10)= Message (9) + CR

char GetFlowMessage[10];


strcpy(GetFlowMessage,GETFLOW);

strcat(GetFlowMessage,CR);

sendmessage = GetFlowMessage;

ArrayLength = strlen(sendmessage);

cout<<sendmessage<<endl;

cout<<"end of message"<<endl;

// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ----------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" reading incoming data .."<<endl;
SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Receiving : "<<incomingData<<endl;

cout<<"-------------------"<<endl;


break;
```

```
//FIN CASE 2 GET FLOW

}

default:
{
cout<<"No se reconoce la elección"<<endl;
break;
}
//FIN SWITCH PICK 3
break;
}

break;

// FIN CASE 1 GET..
}



case 2:{


// SET FLOW

// char SETFLOW[10]="GET;FLOW";

//SIZE (14)= Message (9)+ INPUT(3) + CR

char GetFlowMessage[9];

strcpy(GetFlowMessage,SETFLOW);

cout<<"Introduce FLOW (parte entera)( 2 DIGITOS)"<<endl;

cin>>Flow;

cout<<"Introduce FLOW (parte decimal)( 1 DIGITO)"<<endl;

cin>>Flow2;

strcat(GetFlowMessage,Flow);

strcat(GetFlowMessage,Flow2);

strcat(GetFlowMessage,CR);

sendmessage = GetFlowMessage;

ArrayLength = strlen(sendmessage);

cout<<sendmessage<<endl;

cout<<"end of message"<<endl;

// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ----------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);
```

```
Sleep(500);

cout<<" reading incoming data .."<<endl;
SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Receiving : "<<incomingData<<endl;

cout<<"--------------------"<<endl;


break;

//FIN CASE 2 SET FLOW
}

default:
{
cout<<"No se reconoce la elección"<<endl;
break;
}



break;

//FIN SWITCH PICK 2
}


break;
// FIN CASE '2' PROTOCOL
}
```

4.5 Protocolo Bavaria-Hessen

```
case 3:
{
// Inicio mensaje Bavaria


cout<<"Mensaje sencillo. Consulta de datos de la medición. "<<endl;

cin>>pick;
cout<<endl;

switch(pick){



//-----------------------------------------------OP1 : MENSAJE SENCILLO

case 1:

{
char SimpleMessage[8];
char DA[1];

strcpy(SimpleMessage,STX);

cout<<"Introduce el mensaje. 2 Bytes máximo"<<endl;
```

```
cin>>DA;

strcat(SimpleMessage,DA);

strcat(SimpleMessage,ETX);

// función BCC que devuelve el mensaje con el BCC añadido a la cola.

sendmessage = GetBCC(SimpleMessage);

cout<<"Sending.. :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<sendmessage<<endl;

cout<<"end of message"<<endl;

// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ----------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" reading incoming data .."<<endl;

// -------------LECTURA DEL MENSAJE PUERTO SERIE---------------------//

SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Receiving : "<<incomingData<<endl;

cout<<"--------------------"<<endl;

Sleep(500);

break;

}

default:
{
cout<<"No se reconoce la elección"<<endl;
break;
}

// FIN DEL SWITCH pick
}

break;
// FIN CASE '3' PROTOCOL
}



// FIN SWITCH PROTOCOL

break;
}
```

```
// FIN BUCLE WHILE
}



endl;
Sleep(500);
Stop= false;
}


cout<<"Fin de programa"<<endl;

// ------------- SI ESTAMOS CONECTADOS... AQUI ACABAMOS EL PROGRAMA -------
---//
//    }


// ------------- CIERRE PUERTO SERIE ---------------------//


cout<<"Closing serial port..."<<endl;
SP->~Serial();

return 0;


}
```

5. Programa final . Uso único de MPNS como protocolo.

```
/*
* Mainfile.cpp
*
*  Created on: 21 de ago. de 2016
*      Author: David
*/

#include "ExcelClass.hpp"
#include "SerialClass.hpp"
#include <sstream>



using namespace std;

//------------Parámetros para interacturar con los ficheros de texto y
datos del captador.

string Archive;// String para pasar el nombre del archivo de texto
```

```
string traspaso; // Estos parámetros los utilizaremos de intermediario para sacar líneas de datos de forma ordenada.
string complemento;
string substraer;
int iteracion=0;
int longitud;
bool repeticion = false;

char StringToChar[1024];// Lo utilizaremos para pasar parámetros de string a char y de char a int o float.
char StringToCharlLigth[10];

bool unidades = false;// Con estos dos parametros bool y el array char quitaremos las unidades de las medidas
bool primeraprueba = true;
char uds[5] ={'\t','m','°','h','/'};
string mystring;

bool Stop = false;

// ----------- ASCII CHARACTERS FOR TUNNELING
char STX [1]= {02};
char ETX [1]= {03};
char CR[1]= {0x0D};
char LF[1]= {0x0A};
int ArrayLength = 0;

//------ Puntero usado como buffer para enviar el mensaje.

char *sendmessage = " "; // Iniciamos este puntero de esta forma por que de otra forma, aparecen mensajes
// no intencionados en la consola ( se imprime el puerto com sin tener esa intención).


//------------Comunicacion serial.

char COMPORT[5]= "COM3"; // Parámetro para escribir el nombre del puerto.
char incomingData [1024]= "";             // Inicializamos el array para obtener la información.
int readResult = 0;           // Número de bytes leidos que nos da el return de ReadData.

//------------Comandos utilizados para MPNS.

char StartProbe[81] = "Start Probe [PW=nnnnnn]* [Date=dd.mm.yy] [Time=hh:mm] [Dura=hhh:mm] [Flow=nn.n] ";
char StopProbe[40] = "Stop Probe [Date=dd.mm.yy] [Time=hh:mm]";
char StartCycle[104] = "Start Cycle [PW=nnnnnn][Probes=n] [Date=dd.mm.yy] [Time=hh:mm] [Dura=hhh:mm] [Cycle=hhh:mm] [Flow=nn.n]";
char StopCycle[55] = "Stop Cycle [Date=dd.mm.yy] [Time=hh:mm]";
char GetState[10]= "Get State";

char *PStartProbe, *PStopProbe, *PStartCycle, *PStopCycle, *PGetState;

//--------- Variables que se pueden utilizar para MPNS.

char day[2],month[2],year[2];
char hour[2],minute[2];
```

```
char PW[5];
char Flow[4],Flow2[1];
char Probes[1];
char Dhour[3],Dminute[2];
char Chour[3],Cminute[2];
char AfterProbeCin[1];

int DiaEjecucion[50];

//----------Variablles para switch

int pick,pick2,pick3,pick4;
int LongitudFor =0;









int main()
{


cout<<"Seleccione puerto COM..."<<endl;
cout<<"Introduzca COM seguido del número del puerto. Ej: COM4 "<<endl;


cin>> COMPORT;

Serial* SP = new Serial(COMPORT);  // COM PORT SELECTED


// -------------NOS CONECTAMOS POR PUERTO SERIE---------------------//

if (SP->IsConnected()){
cout<<"Conexión realizada."<<endl;
cout<<endl;
}

else{

cout<<"Conexión fallida."<<endl;
return 0;
}



// Creamos el objeto donde vamos a guardar todos los datos de las medidas y
el archivo .
extractor* ArchivoPrincipal = new extractor();
```

```
// De manera provisional, creamos este objeto para guardar información
secundaria y no completa
// como la que obtenemos del comando MPNS Get State.
extractor* ArchivoSecundario = new extractor();
ArchivoSecundario->LineasEscritas = 0;
ArchivoSecundario->Principal = false;

// ------------ELEGIMOS UN MENSAJE PARA ENVIAR --------------------//


//---------------------------------------------------------------------
CHOSING A MESSAGE TO SEND

while(1){

for(int i=0;i<10;i++)
cout<<endl;

cout<< " Selecciona operación : " << endl;
cout<< " 1. Seleccionar nuevo fichero de datos "<< endl;
cout<< " 2. Mostrar días en los que se tiene previsto que funcione el
equipo"<<endl;
cout<< " 3. Mostrar datos por pantalla " << endl;
cout<< " 4. Copiar datos en otro fichero. "<<endl;
cout<< " 5. Utilizar comandos del switchoptionso MPNS."<<endl; // 9600 Bd
rate
cout<< " 6. Lectura de datos por puerto serie."<<endl;
cout<< " 7. Salir"<<endl; //115200 Bd rate



int switchoptions =0;

cin>>switchoptions;

switch(switchoptions) // Variable para los casos del switch

{
// Inicio del switch (switchoptions)

case 1:

{

//1. Seleccionar nuevo fichero de datos o reiniciar lineasaescritas

cout<<"1. Seleccionar nuevo fichero de datos."<<endl;
cout<<"2. Reiniciar escritura en línea 0."<<endl;
cout<<"3. atrás."<<endl;

pick2 = 0;

cin>>pick2;

switch(pick2){

case 1:{
cout<<"Introduzca el nombre del fichero que desea leer."<<endl;
```

```
cin>> Archive;

ArchivoPrincipal->ExcelArchive = Archive ;

ArchivoPrincipal->GetData();
system("PAUSE");

break;
}// FIN CASE 1 PICK2

case 2:{

ArchivoPrincipal->LineasEscritas=0;
cout<<"Líneas escritas reiniciadas. Se empezará a guardar de nuevo en la
primera línea."<<endl;
system("PAUSE");

break;
} // FIN CASE 2 PICK 2

default:
break;

case 3:
break;

break;}//fin switch pick 2



// FIN CASE '1' switchoptions
break;}


case 2:

{
//2. Mostrar días en los que se tiene previsto que funcione el equipo
cout<<"opcion2"<<endl;

if ( ArchivoPrincipal->Extraccion==false){
cout<< "Introduzca primero un fichero para analizar"<<endl;
break;
}
else{

ArchivoPrincipal->DateProcess();
system("PAUSE");
break;
}

// FIN CASE '2' switchoptions
}


case 3:
{

int pick2 = 0;
//3. Mostrar datos por pantalla
```

```
cout<<"opcion3"<<endl;

cout<<" 1. Mostrar datos del fichero de texto cargado al programa."<<endl;
cout<<" 2. Mostrar datos guardados en el programa desde funciones
MPNS."<<endl;
cout<<" 3. atrás."<<endl;

cin>>pick2;

switch(pick2)
{

case 1 :
{
if ( ArchivoPrincipal->Extraccion==false){
cout<< "Introduzca primero un fichero para analizar"<<endl;
break;
}
ArchivoPrincipal->ShowData();

system("PAUSE");
break;

//FIN SWITCH CASE '1' PICK2

}

case 2 :
{
if ( ArchivoSecundario->Extraccion==false){
cout<< "No existen datos extraidos"<<endl;
break;
}
ArchivoSecundario->ShowData();

system("PAUSE");
break;

//FIN SWITCH CASE '1' PICK2

}
//FIN SWITCH PICK2
}
break;
// FIN CASE '3' switchoptions
}

case 4:
{

//4. Cargar valores para importar a otro programa.
cout<<"opcion4"<<endl;

if ( ArchivoPrincipal->Extraccion==false){
cout<< "Introduzca primero un fichero para analizar"<<endl;
break;
}
```

```
cout << "Introduce el nombre del archivo donde quieres guardar los
datos.:"<<endl;
cin >> Archive;

ArchivoPrincipal->SaveData(Archive);
cout<<"Proceso finalizado."<<endl;
system("PAUSE");

break;
// FIN CASE '4' switchoptions
}


case 5:

{


//5. Inicio del caso de mensajes MPNS

// 1 - MPNS MESSAGES

cout<<" Elegir un comando MPNS... "<<endl;
cout<<"1. Command MPNS: Start Probe."<<endl;
cout<<"2. Command MPNS: Stop Probe."<<endl;
cout<<"3. Command MPNS: Start Cycle."<<endl;
cout<<"4. Command MPNS: Stop Cycle."<<endl;
cout<<"5. Command MPNS: Get state."<<endl;
cout<<"6. Atrás "<<endl;


pick = 0;
cin>>pick;
cout<<endl;

switch(pick){


case 1:     // START PROBE
{

//----- SELECIÓN DE DÍA PARA  EMPEZAR LA PRUEBA---//
if ( ArchivoPrincipal->Extraccion==false){
cout<< "Introduzca primero un fichero para analizar"<<endl;
break;
system("PAUSE");
}
else{
ArchivoPrincipal->DateProcess(DiaEjecucion);

cout<<"Seleccione día de ejecución : "<<endl;


for(int i=0;i<sizeof(DiaEjecucion);i++){
if(i>1 && DiaEjecucion[i]==0)
break;
cout<<(DiaEjecucion[i]+1)<<"\t"<<ArchivoPrincipal->EFechaMuestreo[i]<<"
"<<ArchivoPrincipal->EHoraMuestreo[i]<<endl;
//(DiaEjecucion[i]+1) de esta manera aparece la cuenta desde 0. Mas
adelante, corregimos la eleccion con EleccionLinea -= 1;
```

```
LongitudFor=i;
}
int EleccionLinea=0;
cin>>EleccionLinea;


if(DiaEjecucion[EleccionLinea]== 0 && (EleccionLinea == 0 || EleccionLinea
> LongitudFor)){
cout<<"La opción escogida "<<EleccionLinea<<" no es válida"<<endl;
cout<<" Volviendo atrás..."<<endl;
system("PAUSE");
break;
}

EleccionLinea -= 1;

//Creamos el mensaje a partir de los datos obtenidos de la línea
correspondiente.


//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

//Start Probe [PW=nnnnnn]* [Date=dd.mm.yy] [Time=hh:mm] [Dura=hhh:mm]
[Flow=nn.n]

// SIZE(83) =  Message(81) + CR + LF;

char StartProbeMessage[83] ;

// Utilizamos un puntero para hacer sobre él las modificaciones del
mensaje. Esto es, cambiar los valores
// que vamos introduciendo al captador.

PStartProbe = StartProbe;
//PStartProbe = StartProbe;


cout<<"Start Probe [PW=nnnnnn]* [Date=dd.mm.yy] [Time=hh:mm] [Dura=hhh:mm]
[Flow=nn.n] "<<endl;

// valor de posición = (valor-1) []

// (17)[PW=nnnnnn]*
// Es el único parámetro que deberemos introducir manualmente.



cout<<" Password ( 6 digitos) [PW=nnnnnn]"<<endl;

int x=0;

for(int i=6;i>0;i--){

cout<<"Introduzca"<<i<<" digitos para PW"<<endl;

cin>>PW[x];

x++;
```

```
}

memmove(PStartProbe+16, PW, 6);

//(31)(34)(37)[Date=dd.mm.yy]


// Pasamos de string a char
strncpy(StringToChar, ArchivoPrincipal-
>EFechaMuestreo[EleccionLinea].c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

memmove(PStartProbe+31, StringToChar, 8); // 8 son los valores que debemos
meter "dd.mm.yy"

cout<<"PStartProbe : "<<PStartProbe<<endl;

//(47)(50)[Time=hh:mm]


// Pasamos de string a char
strncpy(StringToChar, ArchivoPrincipal-
>EHoraMuestreo[EleccionLinea].c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

memmove(PStartProbe+47, StringToChar, 5);

cout<<"PStartProbe : "<<PStartProbe<<endl;

//(60)(64)[Dura=hhh:mm]


// Pasamos de string a char


strncpy(StringToCharlLigth,  ArchivoPrincipal-
>EDuracionMuestreo[EleccionLinea].c_str(), sizeof(StringToCharlLigth));
StringToCharlLigth[sizeof(StringToCharlLigth) - 1] = '\0';
ArrayLength = strlen(StringToCharlLigth);

if(ArrayLength == 4){
memmove(PStartProbe+60,"00",2);
memmove(PStartProbe+62, StringToCharlLigth,ArrayLength);
cout<<"PStartProbe : "<<PStartProbe<<endl;
}
if(ArrayLength == 5){
memmove(PStartProbe+60,"0",1);
memmove(PStartProbe+61, StringToCharlLigth,ArrayLength);
cout<<"PStartProbe : "<<PStartProbe<<endl;
}

//(74)(77)  [Flow=nn.n]

// Pasamos de float a string

stringstream ss (stringstream::in | stringstream::out);

//ostringstream ss2;
ss << ArchivoPrincipal->ECaudal[EleccionLinea];
```

```
string Flow3 = ss.str();

// Pasamos de string a char

strncpy(StringToCharlLigth, Flow3.c_str(), sizeof(StringToCharlLigth));
StringToCharlLigth[sizeof(StringToCharlLigth) - 1] = '\0';
ArrayLength = strlen(StringToCharlLigth);
cout<<"ArrayLength "<<ArrayLength<<endl;

if(ArrayLength == 2){
memmove(PStartProbe+74, StringToCharlLigth,ArrayLength);
memmove(PStartProbe+76, ".0",2);
ss.clear();
}

if(ArrayLength == 4){
memmove(PStartProbe+74, StringToCharlLigth,ArrayLength);
cout<<"PStartProbe : "<<PStartProbe<<endl;
ss.clear();
}


// Ahora, copiamos el puntero al mensaje y añadimos CR y LF.

Sleep(20);

strcpy(StartProbeMessage,PStartProbe);

strcat(StartProbeMessage,CR);

strcat(StartProbeMessage,LF);

sendmessage = StartProbeMessage;

cout<<"Enviando ... :"<<endl<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<"----------------------------"<<endl;
// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ---------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);


Sleep(500);

cout<<" Realizando lectura... "<<endl;
readResult = SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Bytes leidos: ( 0 en caso de que no haya datos
disponibles)."<<endl<<readResult<<endl;
incomingData[readResult] = 0;

if(readResult!=0)
cout<<"Recibiendo datos :  "<<readResult<<endl<<incomingData<<endl;
cout<<"-------------------"<<endl;

system("PAUSE");
break;
}//fin else
```

```
} //fin case1



case 2: // STOP PROBE
{

//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

//Seleccionamos si quieren parar la prueba actual u otra a una hora
determinada.

cout<<"1. Parar prueba en ejecución."<<endl;
cout<<"2. Parar prueba a un día y fecha determinado. "<<endl;

pick2 = 0;

cin>>pick2;

switch(pick2){

case 1:
{
// char StopProbe[10] = "Stop Probe"
// SIZE (12) = Message(10) + CR + LF;


char StopProbeMessage[12] ;

cout<<"Stop Probe"<<endl;

memmove(PStopProbe, StopProbe, 10);

strcpy(StopProbeMessage,PStopProbe);

strcat(StopProbeMessage,CR);

strcat(StopProbeMessage,LF);

sendmessage = StopProbeMessage;

cout<<"Enviando.. :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<sendmessage<<endl;

cout<<"Fin del mensaje."<<endl;

// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ----------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" Realizando lectura... "<<endl;
readResult = SP->ReadData(incomingData,sizeof(incomingData)-1);
```

```
cout<<"Bytes leidos: ( 0 en caso de que no haya datos
disponibles)."<<endl<<readResult<<endl;
incomingData[readResult] = 0;

if(readResult!=0)
cout<<"Recibiendo datos :  "<<readResult<<endl<<incomingData<<endl;
cout<<"--------------------"<<endl;

system("PAUSE");
break;

}


case 2:
{
//----- SELECIÓN DE DÍA PARA PARAR LA PRUEBA---//
if ( ArchivoPrincipal->Extraccion==false){
cout<< "Introduzca primero un fichero para analizar"<<endl;
break;
}
else{
ArchivoPrincipal->DateProcess(DiaEjecucion);

cout<<"Seleccione día de ejecución : "<<endl;


for(int i=0;i<sizeof(DiaEjecucion);i++){
if(i>1 && DiaEjecucion[i]==0)
break;
cout<<(DiaEjecucion[i]+1)<<"\t"<<ArchivoPrincipal->EFechaMuestreo[i]<<"
"<<ArchivoPrincipal->EHoraMuestreo[i]<<endl;
//(DiaEjecucion[i]+1) de esta manera aparece la cuenta desde 0. Mas
adelante, corregimos la eleccion con EleccionLinea -= 1;
LongitudFor=i;
}
int EleccionLinea=0;
cin>>EleccionLinea;


if(DiaEjecucion[EleccionLinea]== 0 && (EleccionLinea == 0 || EleccionLinea
> LongitudFor)){
cout<<"La opción escogida "<<EleccionLinea<<" no es válida"<<endl;
cout<<" Volviendo atrás..."<<endl;
system("PAUSE");
break;
}
EleccionLinea -= 1;

// A partir de la línea escogida, después introduciremos automáticamente el
día.

// char StopProbe[40] = "Stop Probe [Date=dd.mm.yy] [Time=hh:mm]";

// SIZE (42) = Message(40) + CR + LF;

char StopProbeMessage[42] ;
```

```
// Utilizamos un puntero para hacer sobre él las modificaciones del
mensaje. Esto es, cambiar los valores
// que vamos introduciendo al captador.


PStopProbe = StopProbe;


cout<<"Stop Probe [Date=dd.mm.yy] [Time=hh:mm]"<<endl;

// (17)(19)(21)[Date=dd.mm.yy]*

// Pasamos de string a char
strncpy(StringToChar, ArchivoPrincipal-
>EFechaMuestreo[EleccionLinea].c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

memmove(PStopProbe+17, StringToChar, 8); // 8 son los valores que debemos
meter "dd.mm.yy"


//(33)(36)[Time=hh:mm]

cout<<"[Time=hh:mm]"<<endl;

//En este caso, pediremos que se introduzca manualmente
//ya que se considera que este parámetro no está en el archivo.

for(int i=0;i<2;i++){

cout<<"Introduzca Hora (2 digitos)"<<endl;

cin>>hour[i];

}

memmove(PStopProbe+33, hour, 2);

for(int i=0;i<2;i++){

cout<<"Introduzca Minuto (2 digitos)"<<endl;

cin>>minute[i];

}

memmove(PStopProbe+36, minute, 2);

strcpy(StopProbeMessage,PStopProbe);

strcat(StopProbeMessage,CR);

strcat(StopProbeMessage,LF);

sendmessage = StopProbeMessage;
ArrayLength = strlen(sendmessage);

cout<<"Enviando ... :"<<sendmessage<<endl;
```

```
cout<<"----------------------------"<<endl;
// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE --------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);


Sleep(500);

cout<<" Realizando lectura... "<<endl;
readResult = SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Bytes leidos: ( 0 en caso de que no haya datos
disponibles)."<<endl<<readResult<<endl;
incomingData[readResult] = 0;

if(readResult!=0)
cout<<"Recibiendo datos :  "<<readResult<<endl<<incomingData<<endl;
cout<<"-------------------"<<endl;

system("PAUSE");
break;
}//FIN ELSE...
break;}//FIN CASE '2' PICK2
break;}//FIN SWITCH PICK2
break;}//FIN CASE '2'PICK -- STOP PROBE


case 3: // START CYCLE
{

//----- SELECIÓN DE DÍA PARA  EMPEZAR LA PRUEBA---//
if ( ArchivoPrincipal->Extraccion==false){
cout<< "Introduzca primero un fichero para analizar"<<endl;
break;
}
else{
ArchivoPrincipal->DateProcess(DiaEjecucion);

cout<<"Seleccione día de ejecución : "<<endl;


for(int i=0;i<sizeof(DiaEjecucion);i++){
if(i>1 && DiaEjecucion[i]==0)
break;
cout<<(DiaEjecucion[i]+1)<<"\t"<<ArchivoPrincipal->EFechaMuestreo[i]<<"
"<<ArchivoPrincipal->EHoraMuestreo[i]<<endl;
//(DiaEjecucion[i]+1) de esta manera aparece la cuenta desde 0. Mas
adelante, corregimos la eleccion con EleccionLinea -= 1;
LongitudFor=i;
}
int EleccionLinea=0;
cin>>EleccionLinea;


if(DiaEjecucion[EleccionLinea]== 0 && (EleccionLinea == 0 || EleccionLinea
> LongitudFor)){
cout<<"La opción escogida "<<EleccionLinea<<" no es válida"<<endl;
cout<<" Volviendo atrás..."<<endl;
system("PAUSE");
break;
```

```
}
EleccionLinea -= 1;



//Creamos el mensaje a partir de los datos obtenidos de la línea
correspondiente.


//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

//     char StartCycle[104] = "Start Cycle [PW=nnnnnn][Probes=n]
[Date=dd.mm.yy] [Time=hh:mm] [Dura=hhh:mm] [Cycle=hhh:mm] [Flow=nn.n]";

// SIZE (107) = Message(105) CR + LF ;
char StartCycleMessage[110] ;

cout<<"Start Cycle [PW=nnnnnnn][Probes=n] [Date=dd.mm.yy] [Time=hh:mm]
[Dura=hhh:mm] [Cycle=hhh:mm] [Flow=nn.n]"<<endl;



// Utilizamos un puntero para hacer sobre él las modificaciones del
mensaje. Esto es, cambiar los valores
// que vamos introduciendo al captador.

PStartCycle = StartCycle;


// (16)[PW=nnnnnn]

cout<<"[PW=nnnnnn]"<<endl;

int x=0;

for(int i=6;i>0;i--){

cout<<"Introduzca"<<i<<" digitos para PW"<<endl;

cin>>PW[x];

x++;

}

memmove(PStartCycle+16, PW, 6);


// (32)[Probes=n]
// Pasamos de int a char

itoa(ArchivoPrincipal->ENumeroMuestras[EleccionLinea],Probes,10);

//Probes debería ser un sólo digito.

memmove(PStartCycle+31, Probes, 1);

// (41)(44)(47)    [Date=dd.mm.yy]
```

```
// Pasamos de string a char

strncpy(StringToChar, ArchivoPrincipal-
>EFechaMuestreo[EleccionLinea].c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

memmove(PStartCycle+40, StringToChar, 8); // 8 son los valores que debemos
meter "dd.mm.yy"


//(57)(60)[Time=hh:mm]

// Pasamos de string a char
strncpy(StringToChar, ArchivoPrincipal-
>EHoraMuestreo[EleccionLinea].c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

memmove(PStartCycle+56, StringToChar, 5);

cout<<PStartCycle<<endl;


//(70)(74)[Dura=hhh:mm]

strncpy(StringToCharlLigth,  ArchivoPrincipal-
>EDuracionMuestreo[EleccionLinea].c_str(), sizeof(StringToCharlLigth));
//                          StringToCharlLigth[sizeof(StringToCharlLigth)
- 1] = '\0';
ArrayLength = strlen(StringToCharlLigth);

if(ArrayLength == 4){
memmove(PStartCycle+69,"00",2);
memmove(PStartCycle+71, StringToCharlLigth,ArrayLength);

}
if(ArrayLength == 5){
memmove(PStartCycle+69,"0",1);
memmove(PStartCycle+70, StringToCharlLigth,ArrayLength);

}

//(84)(88)[Cycle=hhh:mm]

cout<<"[Cycle=hhh:mm]"<<endl;


for(int i=0;i<3;i++){

cout<<"Introduzca horas de duración del Ciclo(3 digitos)"<<endl;

cin>>Chour[i];

}


memmove(PStartCycle+84, Chour, 3);

for(int i=0;i<2;i++){
```

```
cout<<"Introduzca minutos de duración del Ciclo (2 digitos)"<<endl;

cin>>Cminute[i];

}

memmove(PStartCycle+88, Cminute, 2);


//(98)(100) [Flow=nn.n]


// Pasamos de float a string

stringstream ss (stringstream::in | stringstream::out);

//ostringstream ss2;
ss << ArchivoPrincipal->ECaudal[EleccionLinea];
string Flow3 = ss.str();

// Pasamos de string a char

strncpy(StringToCharlLigth, Flow3.c_str(), sizeof(StringToCharlLigth));
StringToCharlLigth[sizeof(StringToCharlLigth) - 1] = '\0';
ArrayLength = strlen(StringToCharlLigth);
cout<<"ArrayLength "<<ArrayLength<<endl;

if(ArrayLength == 2){
memmove(PStartCycle+98, StringToCharlLigth,ArrayLength);
memmove(PStartCycle+199, ".0",2);
ss.clear();
}

if(ArrayLength == 4){
memmove(PStartCycle+98, StringToCharlLigth,ArrayLength);

ss.clear();
}



strcpy(StartCycleMessage,PStartCycle);

strcat(StartCycleMessage,CR);

strcat(StartCycleMessage,LF);

sendmessage = StartCycleMessage;

cout<<"Enviando ... :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<"----------------------------"<<endl;

// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ---------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);
```

```
cout<<" Realizando lectura... "<<endl;
readResult = SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Bytes leidos: ( 0 en caso de que no haya datos
disponibles)."<<endl<<readResult<<endl;
incomingData[readResult] = 0;

if(readResult!=0)
cout<<"Recibiendo datos :  "<<readResult<<endl<<incomingData<<endl;
cout<<"--------------------"<<endl;


system("PAUSE");
break;
}//else
break;} // FIN CASE'3' PICK -- START CYCLE



case 4: // STOP CYCLE

{
cout<<"1. Parar ciclo en ejecución."<<endl;
cout<<"2. Parar ciclo en ejecución tras una prueba determinada. "<<endl;
cout<<"3. Parar prueba a un día , fecha y prueba determinado. "<<endl;

pick2=0;

cin>>pick2;

switch(pick2){

case 1:
{
//  char StoptCycle[11] = "Stop Cycle";

// SIZE (12) =  Message(10) + CR + LF;

char StopCycleMessage1[13] ;

char StopCycleDivide [11];

memmove(StopCycleDivide, StopCycle, 10);

PStopCycle = StopCycleDivide;

strcpy(StopCycleMessage1,PStopCycle);

cout<<StopCycleMessage1<<endl;

strcat(StopCycleMessage1,CR);

strcat(StopCycleMessage1,LF);

sendmessage = StopCycleMessage1;

cout<<"Enviando ... :"<<sendmessage;

ArrayLength = strlen(sendmessage);
```

```
cout<<"----------------------------"<<endl;


// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ---------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" Realizando lectura... "<<endl;
readResult = SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Bytes leidos: ( 0 en caso de que no haya datos
disponibles)."<<endl<<readResult<<endl;
incomingData[readResult] = 0;

if(readResult!=0)
cout<<"Recibiendo datos :  "<<readResult<<endl<<incomingData<<endl;
cout<<"-------------------"<<endl;

system("PAUSE");
break;

}//FIN CASE'1'PICK2

case 2:
{

//  char StoptCycle[26] = "Stop Cycle [AfterProbe=n]";

// SIZE (28) =  Message(26) + CR + LF;

char StopCycleMessage2[29] ;
char StopCycleDivide [11];

memmove(StopCycleDivide, StopCycle, 10);

PStopCycle = StopCycleDivide; //Stop Cycle


char AfterProbearray[16]=" [AfterProbe=n]";


//    [AfterProbe=n]

cout<<"[AfterProbe=n]"<<endl;

cout<<"Introduzca After Probe (1 digito)"<<endl;

cin>>AfterProbeCin;

memmove(AfterProbearray+13,AfterProbeCin,1);

strcat(PStopCycle,AfterProbearray);

strcpy(StopCycleMessage2,PStopCycle);

strcat(StopCycleMessage2,CR);
```

```
strcat(StopCycleMessage2,LF);

sendmessage = StopCycleMessage2;

// Mandando el mensaje por puerto serial.

cout<<"Enviando ... :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<"----------------------------"<<endl;


// ------------ ENVIO DEL MENSAJE POR PUERTO SERIE --------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" Realizando lectura... "<<endl;
readResult = SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Bytes leidos: ( 0 en caso de que no haya datos
disponibles)."<<endl<<readResult<<endl;
incomingData[readResult] = 0;

if(readResult!=0)
cout<<"Recibiendo datos :  "<<readResult<<endl<<incomingData<<endl;
cout<<"-------------------"<<endl;

system("PAUSE");
break;

}//FIN CASE'2'PICK2


case 3:

{

//----- SELECIÓN DE DÍA PARA  EMPEZAR LA PRUEBA---//

if ( ArchivoPrincipal->Extraccion==false){
cout<< "Introduzca primero un fichero para analizar"<<endl;
break;
}
else{
ArchivoPrincipal->DateProcess(DiaEjecucion);

cout<<"Seleccione día de ejecución : "<<endl;


for(int i=0;i<sizeof(DiaEjecucion);i++){
if(i>1 && DiaEjecucion[i]==0)
break;
cout<<(DiaEjecucion[i]+1)<<"\t"<<ArchivoPrincipal->EFechaMuestreo[i]<<"
"<<ArchivoPrincipal->EHoraMuestreo[i]<<endl;
//(DiaEjecucion[i]+1) de esta manera aparece la cuenta desde 0. Mas
adelante, corregimos la eleccion con EleccionLinea -= 1;
LongitudFor=i;
```

```
}
int EleccionLinea=0;
cin>>EleccionLinea;


if(DiaEjecucion[EleccionLinea]== 0 && (EleccionLinea == 0 || EleccionLinea
> LongitudFor)){
cout<<"La opción escogida "<<EleccionLinea<<" no es válida"<<endl;
cout<<" Volviendo atrás..."<<endl;
system("PAUSE");
break;
}
EleccionLinea -= 1;



//Creamos el mensaje a partir de los datos obtenidos de la línea
correspondiente.

//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

//  char StoptCycle[40] = "Stop Cycle [Date=dd.mm.yy] [Time=hh:mm]";

// SIZE (42) =  Message(40) + CR + LF;

char StopCycleMessage3[42] ;

PStopCycle = StopCycle;

cout<<"Stop Cycle [Date=dd.mm.yy] [Time=hh:mm]"<<endl;

// (17)(20)(23)[Date=dd.mm.yy]*


// Pasamos de string a char

strncpy(StringToChar, ArchivoPrincipal-
>EFechaMuestreo[EleccionLinea].c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

memmove(PStopCycle+17, StringToChar, 8); // 8 son los valores que debemos
meter "dd.mm.yy"


//(33)(36)[Time=hh:mm]

// Hay que introducir la hora a la que se quiere parar el muesrtreo.

cout<<"[Time=hh:mm]"<<endl;


for(int i=0;i<2;i++){

cout<<"Introduzca Hora (2 digitos)"<<endl;

cin>>hour[i];

}
```

```
memmove(PStopCycle+33, hour, 2);

for(int i=0;i<2;i++){

cout<<"Introduzca Minuto (2 digitos)"<<endl;

cin>>minute[i];

}

memmove(PStopCycle+36, minute, 2);


strcpy(StopCycleMessage3,PStopCycle);

strcat(StopCycleMessage3,CR);

strcat(StopCycleMessage3,LF);

sendmessage = StopCycleMessage3;

// Mandando el mensaje por puerto serial.


cout<<"Enviando ... :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<"----------------------------"<<endl;


// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ---------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(500);

cout<<" Realizando lectura... "<<endl;
readResult = SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Bytes leidos: ( 0 en caso de que no haya datos
disponibles)."<<endl<<readResult<<endl;
incomingData[readResult] = 0;

if(readResult!=0)
cout<<"Recibiendo datos :  "<<readResult<<endl<<incomingData<<endl;
cout<<"-------------------"<<endl;

system("PAUSE");
break;

}//fin else

break;}// fin case'3' pick2

break;}//fin switch pick2

break;}// fin case '4'
```

```
case 5: //GET STATE
{

//-------- COMPOSICIÓN DEL MENSAJE Y UBICACIÓN DE LOS VALORES QUE DEBEREMOS
CAMBIAR EN EL MENSAJE---//

//  char GetState[10]= "Get State";

// SIZE (12) =  Message(10) + CR + LF;

char GetStateMessage[12] ;

strcpy(GetStateMessage,GetState);

strcat(GetStateMessage,CR);

//                            strcat(GetStateMessage,LF);

sendmessage=GetStateMessage;
//                            sendmessage = GetBCC(GetStateMessage);


cout<<"Enviando.. :"<<sendmessage<<endl;

ArrayLength = strlen(sendmessage);

cout<<sendmessage<<endl;

cout<<"Fin del mensaje."<<endl;

// ------------- ENVIO DEL MENSAJE POR PUERTO SERIE ----------------------//

if(SP->IsConnected())SP->WriteData(sendmessage,ArrayLength);

Sleep(1000);

cout<<" Realizando lectura... "<<endl;
readResult = SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Bytes leidos: ( 0 en caso de que no haya datos
disponibles)."<<endl<<readResult<<endl;
incomingData[readResult] = 0;

if(readResult!=0){

// Transferimos los datos recibidos a un fichero.

// clase para guardar los datos extraidos en el fichero
istringstream ISS(incomingData);

//Ponemos a cero el valor de iteracion en caso de que se haya usado
anteriormente.
iteracion = 0;

while(getline(ISS,traspaso)){

//ISS.clear();  // Para limpiar istringstream.

//Mostramos la línea obtenida
```

```
//cout<<traspaso<<endl;

if(iteracion>0){ // La primera línea no la sacamos ya que no contiene datos
de interés.

size_t found = traspaso.find(uds[0]); // buscamos donde haya una
tabulación, después del signo =.
longitud = traspaso.length();

if (found!=string::npos){

substraer = traspaso.substr(found,(longitud-1));

}

//ahora quitamos la unidad de medida

for(int i=1;i<6;i++){


found=substraer.find(uds[i]);
if (found!=std::string::npos && (unidades == false || primeraprueba ==
true)){

// Una vez encuentre uno de los tres caracteres que define a las unidades
// deja de buscar unidades y guarda la información pura en el string
substraer.

substraer = substraer.substr(0,found);
//cout <<"substraer "<< substraer <<endl<<endl;
unidades =true;

}
primeraprueba = false;
if(iteracion<2){
//Guardamos los parámetros según como nos vienen en la salida

ArchivoSecundario->SFechaInicioMuestreo[ArchivoSecundario->LineasEscritas]=
substraer;

}

else if(iteracion<3){

// Pasamos de string a char y de char a int
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';
ArchivoSecundario->SNumeroCiclos[ArchivoSecundario->LineasEscritas] =
atoi(StringToChar);

}
else if(iteracion<4){

// Pasamos de string a char y de char a float
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

ArchivoSecundario->SCaudalPorgramado[ArchivoSecundario->LineasEscritas] =
atof(StringToChar);
```

```
}
else if(iteracion<5){

// Pasamos de string a char y de char a float
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

ArchivoSecundario->SVolumenNormalizado[ArchivoSecundario->LineasEscritas] =
atof(StringToChar);

}
else if(iteracion<6){

ArchivoSecundario->SDuracion[ArchivoSecundario->LineasEscritas] =
substraer;

}




}

//Dejamos los parámetros como estaban inicialmente
primeraprueba = true;
unidades = false;


}

iteracion++;
}

//Una vez acabado, dejamos constancia de los datos que hemos guardado.

ArchivoSecundario->LineasEscritas++;
ArchivoSecundario->Extraccion = true;
ArchivoSecundario->TotalDatos = ArchivoSecundario->LineasEscritas;
}

else{
cout<<"No se han obtenido datos"<<endl;
}

cout<<"--------------------"<<endl;

system("PAUSE");
break;
}

case 6:
break;


default:
{
cout<<"No se reconoce la elección"<<endl;
system("PAUSE");
break;
```

```
}

break;
// FIN SWITCH PICK1
}

//FIN CASE '5' switchoptions
break;
}


case 6:
{

//      6. Lectura de datos por puerto serie.
cout<<" Lectura a partir de volcado de datos del dispositivo."<<endl;


cout<<"Para empezar, pulse cualquier tecla."<<endl;
system("pause");


if(SP->IsConnected())
{

cout<<" Realizando lectura... "<<endl;
readResult = SP->ReadData(incomingData,sizeof(incomingData)-1);
cout<<"Bytes leidos: ( 0 en caso de que no haya datos
disponibles)."<<endl<<readResult<<endl;
incomingData[readResult] = 0;
if(readResult!=0)
cout<<"Recibiendo datos :  "<<endl<<incomingData<<endl;


// Transferimos los datos recibidos a un fichero.

// clase para guardar los datos extraidos en el fichero
char* recibido = incomingData;
cout<<"recibido es "<<recibido;
istringstream ISS(recibido);

//Condiciones iniciales para el bucle while.
iteracion = 1;
repeticion = false;

while(getline(ISS,traspaso)){



//Mostramos la línea obtenida
//cout<<traspaso<<endl;

if(repeticion == true){
iteracion++;//saltamos directamente a la iteracion 1 para guardar
SNumeroCiclos.
repeticion = false; //Una vez cambiado el valor incial de iteracion, vuelve
a ser falso
// para que no esté incrementando el valor descontroladamente.
}
```

```
if(iteracion>0){ // La primera línea no la sacamos ya que no contiene datos
de interés.


// size_t found = traspaso.find(uds[0]); // buscamos donde está la primera
tabulacion, después del signo =.
size_t found = traspaso.find(' ');
longitud = traspaso.length();

if (found!=string::npos){
cout<<"substrayendo espacio..."<<endl;
substraer = traspaso.substr(found+1,(longitud-1)); // Desde después de la
tabulacion hasta el final de la linea
cout<<substraer<<endl;
}
}
//ahora quitamos la unidad de medida ( en caso de que la tenga)

for(int i=1;i<6;i++){


size_t      found=substraer.find(uds[i]);
if (found!=string::npos && (unidades == false || primeraprueba == true)){

// Una vez encuentre uno de los tres caracteres que define a las unidades
// deja de buscar unidades y guarda la información pura en el string
substraer.

substraer = substraer.substr(0,found);
cout <<"substraer "<< substraer <<endl<<endl;
unidades =true;

}
primeraprueba = false;

//Guardamos los parámetros según como nos vienen en la salida
if(iteracion<2){


// Pasamos de string a char y de char a int
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';
ArchivoPrincipal->SNumeroCiclos[ArchivoPrincipal->LineasEscritas] =
atoi(StringToChar);

}

else if(iteracion<3){

ArchivoPrincipal->SEstado[ArchivoPrincipal->LineasEscritas] = substraer;

}
else if(iteracion<4){

cout <<"substraer "<< substraer <<endl<<endl;

//Para sacar Fecha y hora, las guardaremos por separado.
```

```
size_t found = substraer.find(uds[0]); // buscamos donde haya una
tabulación, entre fecha y hora.
longitud = substraer.length();
if (found!=string::npos){


ArchivoPrincipal->SFechaInicioMuestreo[ArchivoPrincipal->LineasEscritas]=
substraer.substr(0,found);
cout<<ArchivoPrincipal->SFechaInicioMuestreo[ArchivoPrincipal-
>LineasEscritas]<<endl;
ArchivoPrincipal->SHoraInicioMuestreo[ArchivoPrincipal->LineasEscritas]=
substraer.substr(found+1,longitud);
cout<<ArchivoPrincipal->SHoraInicioMuestreo[ArchivoPrincipal-
>LineasEscritas]<<endl;
}



}
else if(iteracion<5){

cout <<"substraer "<< substraer <<endl<<endl;

//Para sacar Fecha y hora, las guardaremos por separado.
//El método de extracción con found no funciona bien si esta en diferentes
instrucciones



size_t found = substraer.find(uds[0]); // buscamos donde haya una
tabulación, entre fecha y hora.
longitud = substraer.length();
if (found!=string::npos){

ArchivoPrincipal->SFechaFinMuestro[ArchivoPrincipal->LineasEscritas]=
substraer.substr(0,found);
cout<<ArchivoPrincipal->SFechaFinMuestro[ArchivoPrincipal-
>LineasEscritas]<<endl;
ArchivoPrincipal->SHoraFinMuestreo[ArchivoPrincipal->LineasEscritas]=
substraer.substr(found+1,longitud);
cout<<ArchivoPrincipal->SHoraFinMuestreo[ArchivoPrincipal-
>LineasEscritas]<<endl;

}


}
else if(iteracion<6){

// Pasamos de string a char y de char a float
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

ArchivoPrincipal->SVolumenNormalizado[ArchivoPrincipal->LineasEscritas] =
atof(StringToChar);

}
```

```
else
if(iteracion<7){

// Pasamos de string a char y de char a float
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

ArchivoPrincipal->SCaudalNormalizado[ArchivoPrincipal->LineasEscritas] =
atof(StringToChar);

}
else if(iteracion<8){


// Pasamos de string a char y de char a float
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';
ArchivoPrincipal->SVolumenAmbiente[ArchivoPrincipal->LineasEscritas] =
atof(StringToChar);
ArchivoPrincipal->SVolumenAmbPrint[ArchivoPrincipal->LineasEscritas]=
substraer;


}
else if(iteracion<9){

// Pasamos de string a char y de char a int
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';
ArchivoPrincipal->STemperaturaMedia[ArchivoPrincipal->LineasEscritas] =
atof(StringToChar);

}
else if(iteracion<10){

// Pasamos de string a char y de char a int
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';
ArchivoPrincipal->SPresionMedia[ArchivoPrincipal->LineasEscritas] =
atof(StringToChar);

}
else if(iteracion<11){


// Pasamos de string a char y de char a float
strncpy(StringToChar, substraer.c_str(), sizeof(StringToChar));
StringToChar[sizeof(StringToChar) - 1] = '\0';

ArchivoPrincipal->SCaudalPorgramado[ArchivoPrincipal->LineasEscritas] =
atof(StringToChar);


}


// fin bucle for
}

//Dejamos los parámetros como estaban inicialmente
```

```
//Para poder quitar las unidades al siguiente dato.

primeraprueba = true;
unidades = false;

//fin if(iteracion>0)
}

if(iteracion<10)
iteracion++;
else{
iteracion = 0;
repeticion = true;
ArchivoPrincipal->LineasEscritas++;
}

//fin bucle while
}


//Lo pasamos al fichero en el orden que nos han dado

cout<<" Lineas de datos traspasadas al fichero : "<<ArchivoPrincipal-
>LineasEscritas<<endl;


}
else{

cout<<"conexión perdida..."<<endl;
cout<<"volviendo atrás."<<endl;
break;
}

system("PAUSE");
break;
} // FIN CASE '6'SWITCH OPTIONS




case 7:
{
cout<<"cerrando puerto serie..."<<endl;
SP->~Serial();

cout<<"Saliendo..."<<endl;

ArchivoPrincipal->~extractor();
ArchivoSecundario->~extractor();
return 0;
break;
}
// FIN SWITCH switchoptions
break;
}
```

```
// FIN BUCLE WHILE
}
return 0;

}//FIN MAIN
```