

Programación Orientada a Objetos
Universidad Nacional de Cuyo - Facultad de
Ingeniería

TP2 – Actividad 1

TRABAJO PRACTICO N° 2 – Desarrollo
Orientado a Objetos

F. Barrios Retta

Septiembre 2025

Incluye código probado en hardware.

Contents

1	Consideraciones Generales	2
1.1	Alcance	2
1.2	Hipótesis y supuestos de diseño	2
1.3	Principios OO aplicados	2
2	Esquema general de la solución	3
2.1	Diseño de la solución	3
3	Interfaces de usuario	3
4	Recursos adicionales	6
4.1	¿Para qué sirven los <code>#include</code> ?	6
5	Manual de instrucciones de la aplicación	7
5.1	Uso (manual breve)	7
5.1.1	Compilación	7
5.1.2	Captura (modo escritura)	7
5.1.3	Lectura/serialización (modo lectura)	8
5.2	Flujo en modo escritura	8
5.3	Flujo en modo lectura	8
5.4	Serialización y robustez	8
5.5	Pruebas realizadas	8
6	Conclusiones	8
7	Referencias consultadas	9
8	Anexo	9
8.1	Definición de clases	9

1 Consideraciones Generales

Este informe documenta el desarrollo de una aplicación de línea de comandos para adquisición y gestión de datos desde un microcontrolador, utilizando C++17 y una arquitectura simple orientada a objetos. La herramienta fue probada con hardware real y contempla lectura/escritura de archivos, conversión entre formatos y comunicación serie.

1.1 Alcance

Se implementa una aplicación de consola orientada a objetos que integra tres ejes: comunicación (transporte serie), serialización/deserialización (CSV/JSON/XML) y persistencia en archivos locales. El objetivo es leer o emitir mensajes estructurados y registrarlos en disco con control básico de errores.

1.2 Hipótesis y supuestos de diseño

- **Transporte.** El puerto serie opera a un baudrate fijo (p.ej. 115200 8N1), lectura línea a línea con terminador `\n`, buffer suficiente para un mensaje completo, y timeout finito configurable. Se asume un solo productor de datos y encolado FIFO.
- **Formato de datos.** Cada mensaje llega completo por línea. Si el modo elegido es:
 - CSV: campos separados por comas/punto y coma, sin saltos de línea embebidos.
 - JSON: un objeto por línea, UTF-8 válido.
 - XML: elemento raíz por línea o bloque bien formado recuperable por acumulación.
- **Dominio.** `Types.h` define tipos como `Mode` (lectura/escritura/append), `IOFormat` (CSV/JSON/XML) y `Message` (código/ payload/metadata). El valor por defecto de code se interpreta como “OK” salvo que la deserialización indique lo contrario.
- **Errores.** Fallas de E/S o parseo se gestionan con excepciones y/o códigos de estado encapsulados; la app registra incidentes y continúa según política de tolerancia.
- **Persistencia.** La escritura es a FS local usando flujos C++ estándar con apertura segura y cierre garantizado. No se usa base de datos.

1.3 Principios OO aplicados

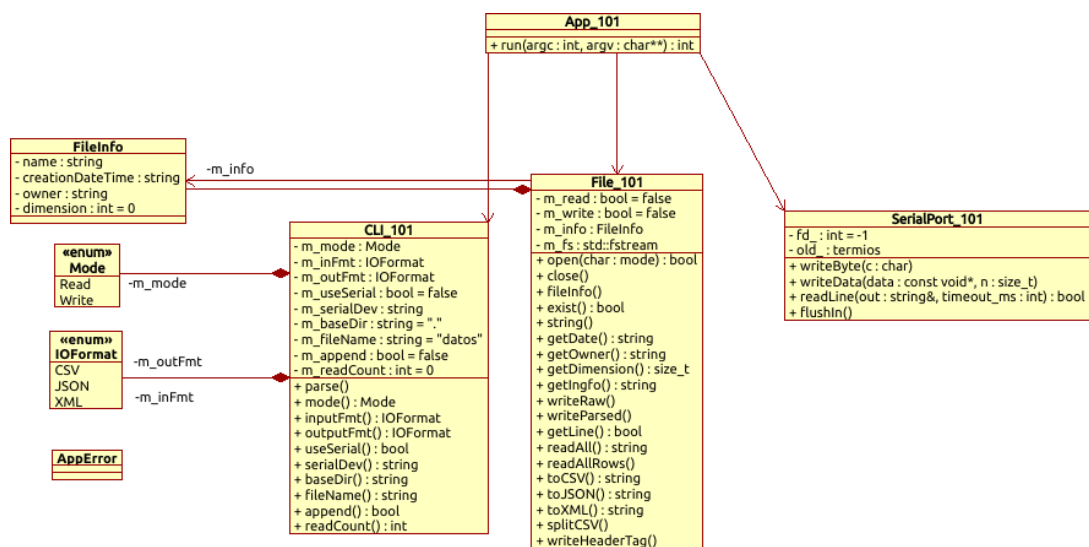
- **Encapsulamiento:** detalles de E/S serie y de formato se ocultan detrás de `ITransport` e `ISerializer`. La app solo “pide servicios”.
- **Modularidad y compilación separada:** headers/definitions por componente, enlazado posterior.
- **Bajo acoplamiento, alta cohesión:** cada clase tiene una única responsabilidad clara (GRASP “experto” y “creador”).
- **Relaciones UML:** asociaciones dirigidas `App`→`Transport/Serializer` y composición `App`→`Repo` de archivos.

2 Esquema general de la solución

2.1 Diseño de la solución

Se adoptó una arquitectura simple orientada a objetos, respetando el diagrama de clases provisto en la consigna. El diagrama actualizado se ilustra a continuación:

- **App** coordina la ejecución general y delega en **CLI** la interpretación de argumentos.
- **CLI** encapsula la lectura de los parámetros y expone *getters* tipados (**Mode**, **IOFormat**).
- **File** maneja la persistencia en CSV, mantiene los metadatos (**FileInfo**) y ofrece conversores a JSON/XML.
- **SerialPort** abstrae la comunicación por puerto serie usando **termios** para trabajar en modo crudo a 115200 bauds.
- **Types** concentra tipos comunes (**AppError**, **Mode**, **IOFormat**) y las funciones utilitarias `parseLine` y `tryParseHeaderTag`.



3 Interfaces de usuario

La interfaz es puramente CLI. Los casos de uso principales son:

- Escritura (captura desde μC a CSV)

```

1 ./app -m w -i c -n 20 -s /dev/ttyACM0 -d logs -f sensor
2

```

Lectura en distintos formatos

- CSV:

```

1 ./app -m r -o c -d logs -f sensor
2

```

- JSON:

```

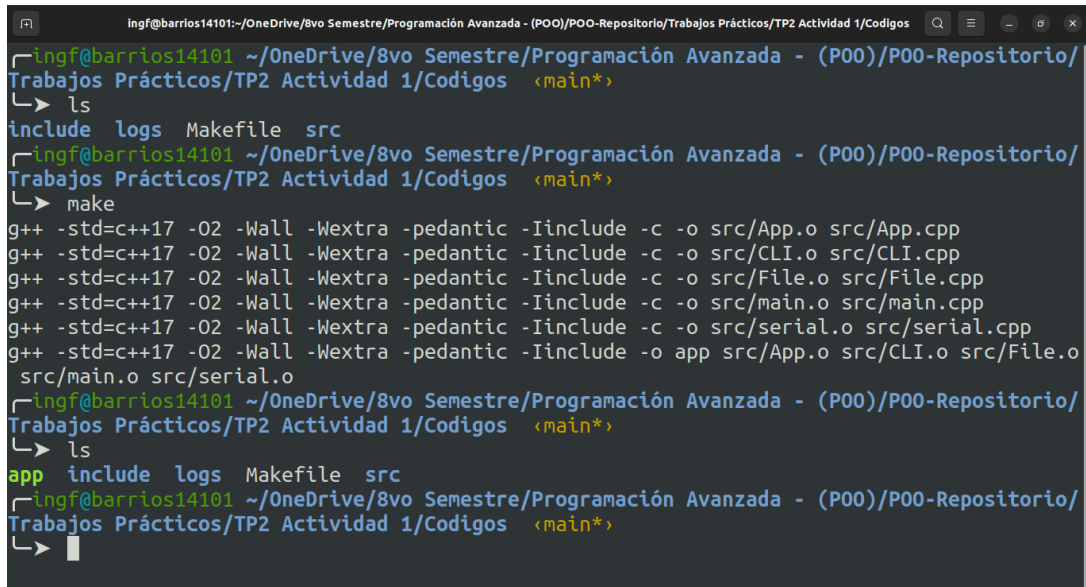
1 ./app -m r -o j -d logs -f sensor
2

```

- XML:

```
1 ./app -m r -o x -d logs -f sensor
2
```

Luego, tenemos las siguientes capturas de pantalla:



```
ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─ ls
include logs Makefile src
ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─ make
g++ -std=c++17 -O2 -Wall -Wextra -pedantic -Iinclude -c -o src/App.o src/App.cpp
g++ -std=c++17 -O2 -Wall -Wextra -pedantic -Iinclude -c -o src/CLI.o src/CLI.cpp
g++ -std=c++17 -O2 -Wall -Wextra -pedantic -Iinclude -c -o src/File.o src/File.cpp
g++ -std=c++17 -O2 -Wall -Wextra -pedantic -Iinclude -c -o src/main.o src/main.cpp
g++ -std=c++17 -O2 -Wall -Wextra -pedantic -Iinclude -c -o src/serial.o src/serial.cpp
g++ -std=c++17 -O2 -Wall -Wextra -pedantic -Iinclude -o app src/App.o src/CLI.o src/File.o src/main.o src/serial.o
ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─ ls
app include logs Makefile src
ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─
```



```
ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─ ./app -m w -i c -n 10 -s /dev/ttyACM0 -d logs -f sensor
Lecturas guardadas : 10
Nombre      : logs/sensor.csv
Creación    : 2025-09-21 23:16:59
Owner       : ingf
Dimensión   : 10 líneas
ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─ cat logs/sensor.csv
#h: Id,porcentaje_valvula,estado_nivel,caudal
2,27,medio,27.86
2,49,alo,98.23
2,10,lto,81.97
2,75,ato,31.74
2,84,alto,32.32
2,67,medio,26.18
2,26,alto,38.89
2,49,alto,68.20
2,23,medio,42.05
2,49,medio,40.11
```

Cabe resaltar que usar el comando `cat` me muestra los datos del archivo. Esto quiere decir que la **persistencia** se realiza en formato `.csv`.

```
ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─> ./app -m r -o c -d logs -f sensor
Nombre      : logs/sensor.csv
Creación    : 2025-09-21 23:17:08
Owner       : ingf
Dimensión   : 10 líneas
#h: Id,porcentaje_valvula,estado_nivel,caudal
2,27,medio,27.86
2,49,alo,98.23
2,10,lto,81.97
2,75,ato,31.74
2,84,alto,32.32
2,67,medio,26.18
2,26,alto,38.89
2,49,alto,68.20
2,23,medio,42.05
2,49,medio,40.11
└─>
```

```
ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─> ./app -m r -o j -d logs -f sensor
Nombre      : logs/sensor.csv
Creación    : 2025-09-21 23:17:08
Owner       : ingf
Dimensión   : 10 líneas
[
  {"Id": "2", "porcentaje_valvula": "27", "estado_nivel": "medio", "caudal": "27.86"},
  {"Id": "2", "porcentaje_valvula": "49", "estado_nivel": "alo", "caudal": "98.23"},
  {"Id": "2", "porcentaje_valvula": "10", "estado_nivel": "lto", "caudal": "81.97"},
  {"Id": "2", "porcentaje_valvula": "75", "estado_nivel": "ato", "caudal": "31.74"},
  {"Id": "2", "porcentaje_valvula": "84", "estado_nivel": "alto", "caudal": "32.32"},
  {"Id": "2", "porcentaje_valvula": "67", "estado_nivel": "medio", "caudal": "26.18"},
  {"Id": "2", "porcentaje_valvula": "26", "estado_nivel": "alto", "caudal": "38.89"},
  {"Id": "2", "porcentaje_valvula": "49", "estado_nivel": "alto", "caudal": "68.20"},
  {"Id": "2", "porcentaje_valvula": "23", "estado_nivel": "medio", "caudal": "42.05"},
  {"Id": "2", "porcentaje_valvula": "49", "estado_nivel": "medio", "caudal": "40.11"}
]
└─>
```

```

ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─ main*
└─ ./app -m r -o x -d logs -f sensor
Nombre      : logs/sensor.csv
Creación    : 2025-09-21 23:17:08
Owner       : ingf
Dimensión   : 10 líneas
<rows>
<row><Id>2</Id><porcentaje_valvula>27</porcentaje_valvula><estado_nivel>medio</estado_nivel><caudal>27.86</caudal></row>
<row><Id>2</Id><porcentaje_valvula>49</porcentaje_valvula><estado_nivel>alto</estado_nivel><caudal>98.23</caudal></row>
<row><Id>2</Id><porcentaje_valvula>10</porcentaje_valvula><estado_nivel>lto</estado_nivel><caudal>81.97</caudal></row>
<row><Id>2</Id><porcentaje_valvula>75</porcentaje_valvula><estado_nivel>ato</estado_nivel><caudal>31.74</caudal></row>
<row><Id>2</Id><porcentaje_valvula>84</porcentaje_valvula><estado_nivel>alto</estado_nivel><caudal>32.32</caudal></row>
<row><Id>2</Id><porcentaje_valvula>67</porcentaje_valvula><estado_nivel>medio</estado_nivel><caudal>26.18</caudal></row>
<row><Id>2</Id><porcentaje_valvula>26</porcentaje_valvula><estado_nivel>alto</estado_nivel><caudal>38.89</caudal></row>
<row><Id>2</Id><porcentaje_valvula>49</porcentaje_valvula><estado_nivel>alto</estado_nivel><caudal>68.20</caudal></row>
<row><Id>2</Id><porcentaje_valvula>23</porcentaje_valvula><estado_nivel>medio</estado_nivel><caudal>42.05</caudal></row>
<row><Id>2</Id><porcentaje_valvula>49</porcentaje_valvula><estado_nivel>medio</estado_nivel><caudal>40.11</caudal></row>
</rows>
ingf@barrios14101:~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos
└─ main*
└─

```

4 Recursos adicionales

No se hace uso de ningún componente de software no estandar del lenguaje ni de la plataforma para codificar el programa. A continuación, se listan las librerías estandar de C++ y de Linux que se utilizan en los diferentes archivos .h y .cpp.

```

1 #include <string>
2 #include <vector>
3
4 #include <fstream>
5 #include <stdexcept>
6 #include <termios.h>
7 #include <chrono>
8 #include <ctime>
9 #include <sstream>
10 #include <iomanip>
11 #include <iostream>
12 #include <cstdlib>
13 #include <sys/stat.h>
14 #include <unistd.h>
15 #include <pwd.h>
16 #include <filesystem>
17 #include <system_error>
18 #include <algorithm>
19

```

4.1 ¿Para qué sirven los #include?

- <string>: std::string, cadenas decentes, no arrays C.
- <vector>: std::vector<T>, arrays dinámicos.
- <fstream>: std::ifstream/ofstream/fstream para leer/escribir archivos.
- <stdexcept>: excepciones estándar (std::runtime_error, std::invalid_argument...).
- <termios.h>: control POSIX de puertos/terminales. Configurar serial: baudrate, pari-

dad, flags.

- `<chrono>`: tiempo moderno C++: `steady_clock`, `milliseconds`, `sleep_for`.
- `<ctime>`: tiempo estilo C: `time_t`, `std::localtime`, `std::strftime-like`.
- `<sstream>`: `std::stringstream` para parsear/armar strings como si fueran streams.
- `<iomanip>`: formateo de streams: `std::setw`, `std::setprecision`, `std::put_time`.
- `<iostream>`: `std::cout`, `std::cin`, `std::cerr`. El trío inevitable.
- `<cstdlib>`: utilidades varias: `std::getenv`, `std::system`, `std::strtol`, `std::rand`.
- `<sys/stat.h>`: `stat`, permisos y `mkdir` con modos; funciones del Sistema Operativo.
- `<unistd.h>`: POSIX: `read`, `write`, `close`, `usleep`, `access`, `isatty`.
- `<pwd.h>`: info de usuario: `getpwuid` para home, etc.
- `<filesystem>`: C++17 paths y archivos: `std::filesystem::path`, `exists`, `create_directories`.
- `<system_error>`: `std::error_code` y `std::system_error` para reportes de errores del Sistema Operativo.
- `<algorithm>`: `std::sort`, `std::find`, `std::transform`, `std::accumulate...` herramientas útiles.

5 Manual de instrucciones de la aplicación

5.1 Uso (manual breve)

5.1.1 Compilación

Desde el directorio del código (./Codigos):

```
1 make
2
```

5.1.2 Captura (modo escritura)

Utilizandose como μC un ESP32 de placa de desarrollo DOIT Devkit v1, se ve el directorio `/dev/ttyACMX` para hacer lectura de la comunicación serial entre el C y la aplicación (equipo).

Se escribe en la carpeta donde se realizó el make:

```
1 ./app -m w -i c -n 10 -s /dev/ttyACM0 -d logs -f sensor
2
```

Donde: • `-m w`: modo escritura • `-i c|j|x`: formato de entrada esperado (CSV/JSON/XML “liviano”) • `-n`: cantidad de lecturas • `-s`: dispositivo serie • `-d` y `-f`: directorio y nombre base del archivo

5.1.3 Lectura/serialización (modo lectura)

```
1 ./app -m r -o c -d logs -f sensor # CSV
2 ./app -m r -o j -d logs -f sensor # JSON
3 ./app -m r -o x -d logs -f sensor # XM
4
```

5.2 Flujo en modo escritura

1. El operador define el formato esperado mediante `-i`.
2. `App` abre/crea el CSV en el directorio indicado y realiza, si es necesario, la cabecera `#h:`.
3. Para cada muestra se envía el byte de *handshake* (`c`, `j` o `x`) y se espera la respuesta del μC .
4. `parseLine` detecta el formato real, normaliza la estructura y devuelve un vector de campos.
5. `File::writeParsed` escapa los valores y los persiste siempre en CSV.
6. Se contabilizan las lecturas correctas y, al finalizar, se informa la tabla de metadatos.

5.3 Flujo en modo lectura

1. CLI determina el formato deseado (`-o`).
2. `File` abre el CSV, actualiza los metadatos y expone `getInfoTable()`.
3. En función del formato elegido se reutiliza el CSV original (`toCSV()`), se arma un arreglo JSON (`toJSON()`) o un documento XML (`toXML()`).
4. Cuando no existen encabezados explícitos se generan identificadores genéricos (`c1`, `c2`, ...).

5.4 Serialización y robustez

- Se configuran 115200 bauds, modo 8N1 sin control de flujo hardware.
- El `timeout` de lectura se fijó en 2000 ms para compensar jitter en la transmisión.
- Las líneas incompletas o vacías se descartan sin afectar el contador de lecturas exitosas.
- `File::countRowsOnDisk()` y `readAllRows()` ignoran comentarios y filas vacías para mantener una dimensión realista del dataset.

5.5 Pruebas realizadas

- Compilación con `make` (usar desde `Codigos/`).
- Captura de 20 muestras desde el μC con `./app -m w -i c -n 20 -s /dev/ttyACM0 -d logs -f sensor`.
- Visualización en los tres formatos disponibles: `./app -m r -o c`, `./app -m r -o j`, `./app -m r -o x`.
- Verificación manual de la consistencia del CSV resultante (`logs/sensor.csv`).

6 Conclusiones

La aplicación cumple con los objetivos del TP: separa responsabilidades, estandariza formatos heterogéneos y simplifica la interacción con el μC mediante un protocolo simple de handshakes. La arquitectura facilita extender nuevas fuentes/destinos de datos y admitir otros formatos.

7 Referencias consultadas

Documentación de la biblioteca estándar de C++ (containers, streams,).

Páginas de manual POSIX para términos, read/write/close, stat, getpwuid.

Apuntes de cátedra de POO (sección “Guía de Trabajos Prácticos”).

8 Anexo

8.1 Definición de clases

Se dejan los headers utilizados, los cuales dan una idea de cómo se hicieron las clases.

```
1  #ifndef TYPES_H
2  #define TYPES_H
3
4  #include <stdexcept>
5  #include <string>
6  #include <string>
7  #include <vector>
8
9
10 class AppError : public std::runtime_error {
11 public:
12     explicit AppError(const std::string& msg) : std::runtime_error(msg) {}
13 };
14
15 enum class Mode { Read, Write };
16 enum class IOFormat { CSV, JSON, XML };
17
18 inline IOFormat charToFormat(char c) {
19     switch (c) {
20         case 'c': case 'C': return IOFormat::CSV;
21         case 'j': case 'J': return IOFormat::JSON;
22         case 'x': case 'X': return IOFormat::XML;
23         default: throw std::invalid_argument("Formato inválido (use c|j|x).");
24     }
25 }
26 inline const char* fmtName(IOFormat f) {
27     switch (f) { case IOFormat::CSV: return "CSV"; case IOFormat::JSON: return "JSON"; d_
28     efault: return "XML"; }
29 }
30 // Convierte una línea recibida en formato c/j/x a un vector de strings.
31 // c => CSV; j => JSON plano tipo {"a":1,"b":2}; x => XML plano <row><a>1</a>...</row>
32 std::vector<std::string> parseLine(IOFormat inFmt, const std::string& line,
33                                   std::vector<std::string>* headersOpt = nullptr);
34
35 // Encabezados: para CSV se asumen si vienen en la primera fila con '#h: a,b,c'
36 // Para JSON/XML se infieren del primer objeto si no se proveen.
37 bool tryParseHeaderTag(const std::string& line, std::vector<std::string>& headers);
38
39
40 #endif
41
```

```
1  #ifndef FILE_H
2  #define FILE_H
3
4  #include "Types.h"
5  #include <string>
6  #include <vector>
7  #include <fstream>
8
9  struct FileInfo {
10     std::string name;
11     std::string creationDateTime;
12     std::string owner;
13     size_t      dimension = 0; // líneas de datos
14 };
15
16 class File {
17     bool      m_read  = false;
18     bool      m_write = false;
19     FileInfo  m_info;
20     std::fstream m_fs;
21
22 public:
23     File() = default;
24     explicit File(const std::string& path);
25
26     bool open(char mode); // 'r' o 'w' o 'a'
27     void close();
28
29     const FileInfo& info() const { return m_info; }
30     std::string      getInfoTable() const;
31
32     bool      exist() const;
33     std::string getNombre() const;
34     std::string getFecha() const;
35     std::string getPropietario() const;
36     size_t      getDimension() const;
37     std::string getInfo() const;
38
39     // Escritura
40     void writeRaw(const std::string& line); // escribe tal cual (CSV ya
formateado)
41     void writeParsed(const std::vector<std::string>& fields); // une por coma
42
43     // Lectura
44     bool  getLine(std::string& out);
45     std::string readAll(); // contenido CSV completo (sin cabecera especial)
46     std::vector<std::vector<std::string>> readAllRows();
47
48     // Presentación (a partir del CSV)
49     std::string toCSV(); // igual al archivo
50     std::string toJSON(const std::vector<std::string>& headers);
51     std::string toXML (const std::vector<std::string>& headers);
52
53     // Utilidad
54     static std::vector<std::string> splitCSV(const std::string& line);
55     void writeHeaderTag(const std::vector<std::string>& headers);
56
57 private:
58     size_t m_rows = 0; // filas de datos (excluye cabeceras #h)
```

```
59     size_t countRowsOnDisk() const;
60 };
61
62 #endif
63
```

```
1  #ifndef SERIAL_H
2  #define SERIAL_H
3
4  #include <string>
5  #include <termios.h>
6
7  class SerialPort {
8      int      fd_ = -1;
9      termios  old_{};
10 public:
11     // baud: B9600, B115200, etc.
12     SerialPort(const std::string& dev, int baud);
13     ~SerialPort();
14
15     void writeByte(char c);
16     void writeData(const void* data, size_t n);
17
18     // Lee hasta '\n' o '\r' con timeout en ms. Devuelve true si obtuvo una línea.
19     bool readLine(std::string& out, int timeout_ms);
20
21     // >>> NUEVO: limpia el buffer de entrada (evita "o>", restos, etc.)
22     void flushIn();
23
24     // no copiable
25     SerialPort(const SerialPort&) = delete;
26     SerialPort& operator=(const SerialPort&) = delete;
27 };
28
29
30 #endif
31
```

```
1  #ifndef CLI_H
2  #define CLI_H
3
4  #include "Types.h"
5  #include <string>
6
7  class CLI {
8      Mode      m_mode = Mode::Read;
9      IOFormat  m_inFmt = IOFormat::CSV;    // para -m w
10     IOFormat  m_outFmt = IOFormat::CSV;    // para -m r
11     bool      m_useSerial = false;
12     std::string m_serialDev;
13     std::string m_baseDir = ".";
14     std::string m_fileName = "datos";
15     bool      m_append = false;
16     int      m_readCount = 0;
17
18 public:
19     void parse(int argc, char** argv);

```

```
20
21     Mode        mode()        const { return m_mode; }
22     IOFormat    inputFmt()    const { return m_inFmt; }
23     IOFormat    outputFmt()    const { return m_outFmt; }
24     bool        useSerial()    const { return m_useSerial; }
25     std::string serialDev()    const { return m_serialDev; }
26     std::string baseDir()      const { return m_baseDir; }
27     std::string fileName()     const { return m_fileName; }
28     bool        append()       const { return m_append; }
29     int         readCount()     const { return m_readCount; }
30 };
31
32 #endif
33
```

```
1  #ifndef APP_H
2  #define APP_H
3
4  #include "CLI.h"
5
6  class App {
7  public:
8      int run(int argc, char** argv);
9  };
10
11 #endif
12
```
