

**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

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

## 1 Consideraciones Generales

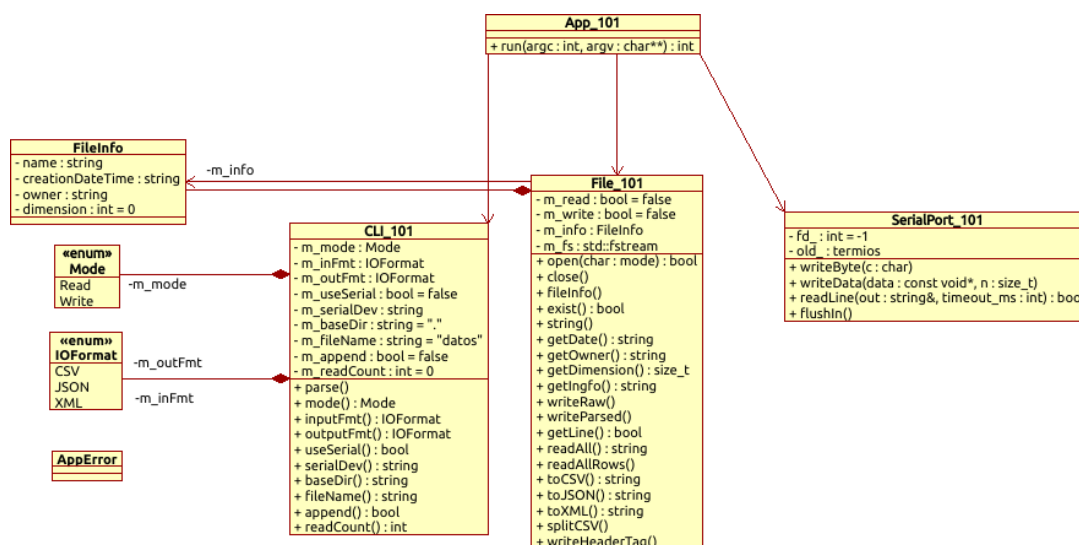
Este informe documenta el desarrollo de una aplicación de línea de comandos para adquisición y gestión de datos desde un microcontrolador, utilizando C++17 y una arquitectura simple orientada a objetos. La herramienta fue probada con hardware real y contempla lectura/escritura de archivos, conversión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

## 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
```

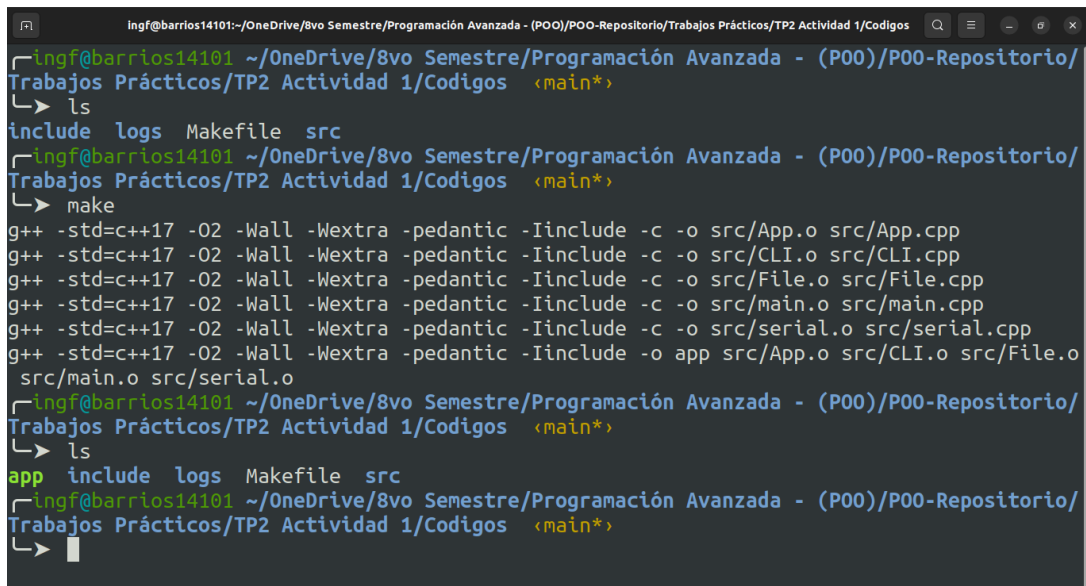
- JSON:

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

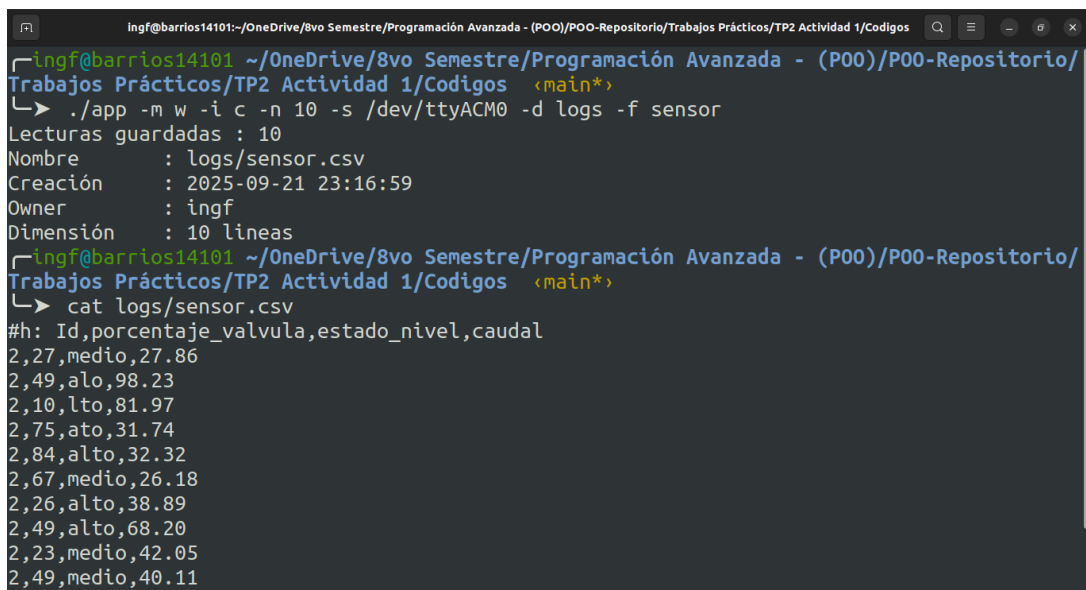
- XML:

```
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
└─ ingf@barrios14101 ~/OneDrive/8vo Semestre/Programación Avanzada - (P00)/P00-Repositorio/Trabajos Prácticos/TP2 Actividad 1/Codigos <main*>
└─ ./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 <main*>
└─
```

```
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 <main*>
└─ ./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*>
└─
```

```

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>68.28</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 estándar del lenguaje ni de la plataforma para codificar el programa. A continuación, se listan las librerías estándar 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, par-

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 C 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  $\mu$ C.
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  $\mu$ C 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  $\mu$ C 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 `termios`, `read/write/close`, `stat`, `getpwuid`.

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

## 8 Anexo

### 8.1 Esquema de clases

- **App** usa **CLI**, **File** y opcionalmente **SerialPort**
- **File** compone **FileInfo** y utiliza `std::fstream`.
- **CLI** agrupa `Mode` e `IOFormat`; **Types** concentra `AppError` y utilidades.