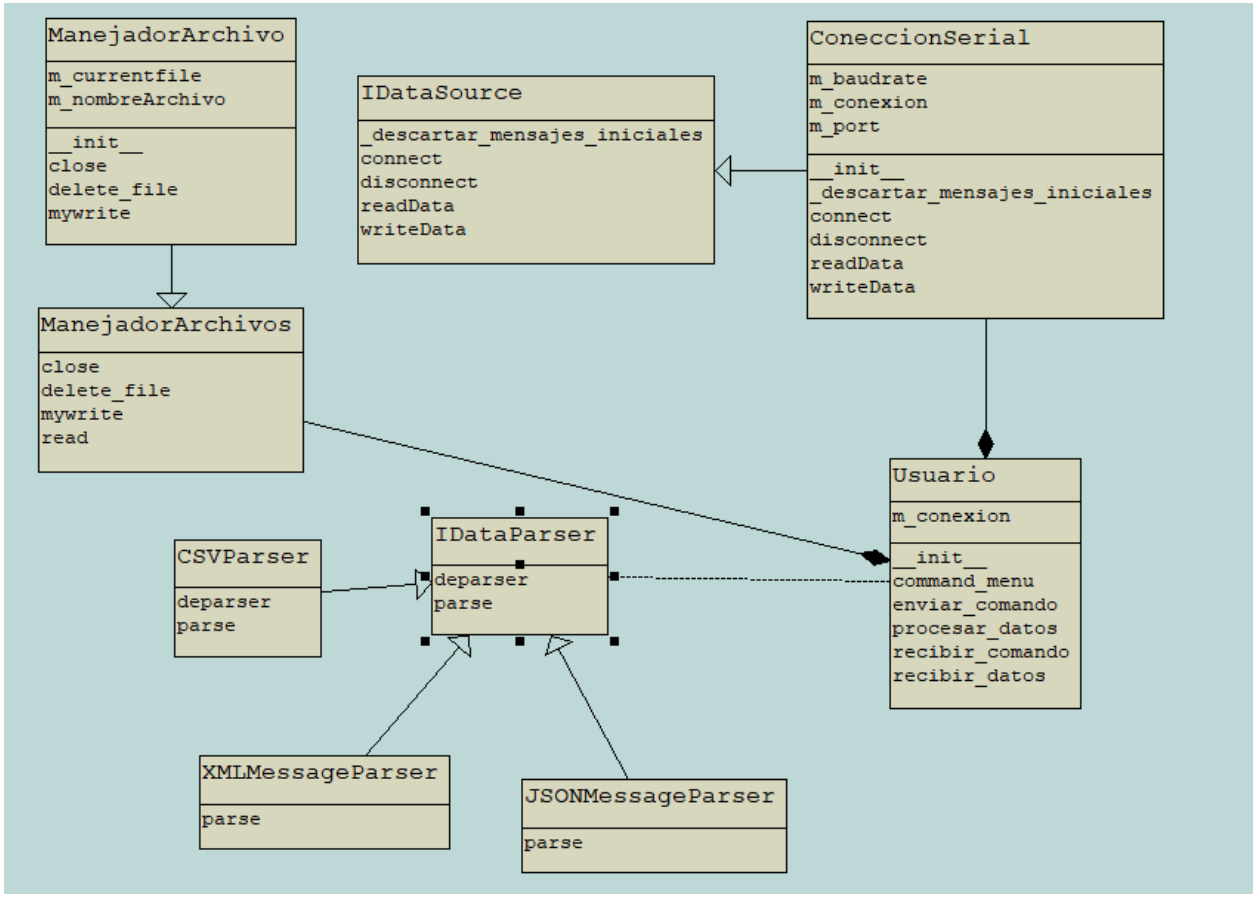
Ejercicio 1

Diagrama de clases general de la solución



Modo de uso

Para ejecutar el programa, solo se necesita una versión de Python 3.11 o superior y correr el script en IDE de preferencia.

```
Inicio de sketch: escucha/genera numero aleatorio
x: cadena XML
j: cadena Json
c: cadena CSV
Listo para recibir comandos del usuario.
1. Crear archivo nuevo
2. Escribir en archivo existente (enviar comandos)
3. Eliminar archivo
4. Salir
Ingrese una opción:
```

```
1. Crear archivo nuevo
2. Escribir en archivo existente (enviar comandos)
3. Eliminar archivo
4. Salir
Ingrese una opción: 1
Ingrese el nombre del archivo a crear (sin extensión):
prueba4
```

Si se desea escribir sobre un archivo que ya tiene información:

```
1. Crear archivo nuevo
2. Escribir en archivo existente (enviar comandos)
3. Eliminar archivo
4. Salir
Ingrese una opción: 2
Ingrese el nombre del archivo a escribir (sin extensión):
Elija un archivo a escribir:
0 - prueba1.csv
1 - prueba3.csv
2 - prueba4.csv
3 - pruebaEj2.csv
```

```
0 - prueba1.csv
1 - prueba3.csv
2 - prueba4.csv
3 - pruebaEj2.csv
2
Ingrese el comando (x, j, c): x
Datos recibidos: ['2', '3', 'bajo', '40.39']
Ingrese el comando (x, j, c): c
Datos recibidos: ['2', '15', 'medio', '34.62']
Ingrese el comando (x, j, c): j
Datos recibidos: ['2', '95', 'alto', '59.51']
Ingrese el comando (x, j, c):
```

main.py | prueba4.csv | menu.py | archivoshandler.py

```
dispID;apertura;nivel;caudal
2;3;bajo;40.39
2;15;medio;34.62
2;95;alto;59.51
```

Si se desea dejar de enviar comandos y seleccionar otra opción, el comando “exit” vuelve al ciclo principal del menú de usuario:

```
Ingrese el comando (x, j, c): c
Datos recibidos: ['2', '15', 'medio', '34.62']
Ingrese el comando (x, j, c): j
Datos recibidos: ['2', '95', 'alto', '59.51']
Ingrese el comando (x, j, c): exit
1. Crear archivo nuevo
2. Escribir en archivo existente (enviar comandos)
3. Eliminar archivo
4. Salir
Ingrese una opción:
```

Comentarios

Se optó por modularizar el proyecto, y aunque esto haya traído algunas ventajas al momento de debuggear el código, se torna complejo mantener el hilo de trabajo y no perderse en lo que se desea realizar.

No se implementa correctamente una distinción de capas en lo que respecta al modelo MVC, pero se espera haber corregido el error en los próximos ejercicios.

Se eligió probar las librerías de Python dedicadas al manejo de archivos CSV y XML, las mismas me permitieron realizar el parseo de una manera rápida y eficiente, aunque las mismas merecen su debug dedicado ya que hay ciertos parámetros específicos, aun así no presentaron dificultades notables.

Se utilizó por primera vez el concepto de interfaz a modo de prueba, donde las mismas son:

- Iarchivos: se encargaría de dar las pautas para cualquier manejador de archivos.
- Idatasource: se encargaría de modelar cualquier fuente de datos, en este caso fue serial.
- Idataparser: pone las pautas para cualquier clase encargada de parsear/deparsear.

Breve conclusión

La utilización de librerías dedicadas fue una elección discutible, pero se concluye que la utilización de las mismas si se justifica completamente si se llegaran a manejar archivos considerablemente extensos.

La modularización del proyecto podría no estar justificada, pero la cantidad de código hace que no sea una mala opción.

El uso de interfaces no fue tan aprovechado, pero ayudó a comprender el concepto.

Recursos adicionales

Librerías / Paquetes :

- serial
- os
- csv
- json
- xml.etree.ElementTree
- from io, StringIO

Ejercicio 2

Modo de uso

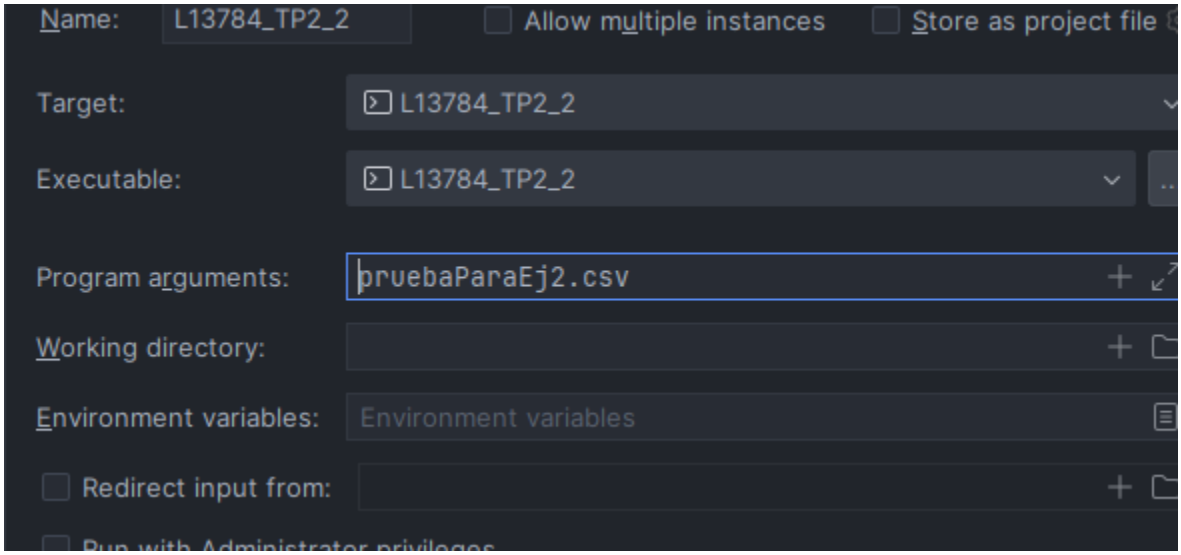
En terminal puede escribirse lo siguiente estando dentro de la carpeta “codigo”:

```
\L13784_TP2\L13784_TP2_2\codigo> .\L13784_TP2_2.exe ..\anexo\pruebaParaEj2.csv
\L13784_TP2\L13784_TP2_2\codigo>
```

El archivo resultante xml se creará dentro de la carpeta de anexos.

En el caso de que se desee recompilar y utilizar tal ejecutable, debe realizar la operación análoga pero con el ejecutable correspondiente.

También existe la opción de usar CMake directamente desde el IDE, para ello debe tomarse el ejecutable que se crea en la carpeta “cmake-build-debug” y copiarlo en la carpeta “código”, para luego realizar el procedimiento anterior. También puede agregarse el archivo .csv a la carpeta “cmake-build-debug” y agregar el nombre del archivo en los Argumentos del Programa del IDE correspondiente. (Ejemplo con CLion)



Comentarios

Este ejercicio fue interesante ya que no se utilizó ninguna librería específica por lo que se debió realizar el parser de forma propia mediante el conveniente método *getline* que lee línea por línea, y además tiene un parámetro que recibe el delimitador que debe usarse, en este caso se optó por el delimitador “;”.

Breve conclusión

Este ejercicio ayudó a comprender un poco el detrás de las librerías que se utilizaron en el ejercicio anterior.

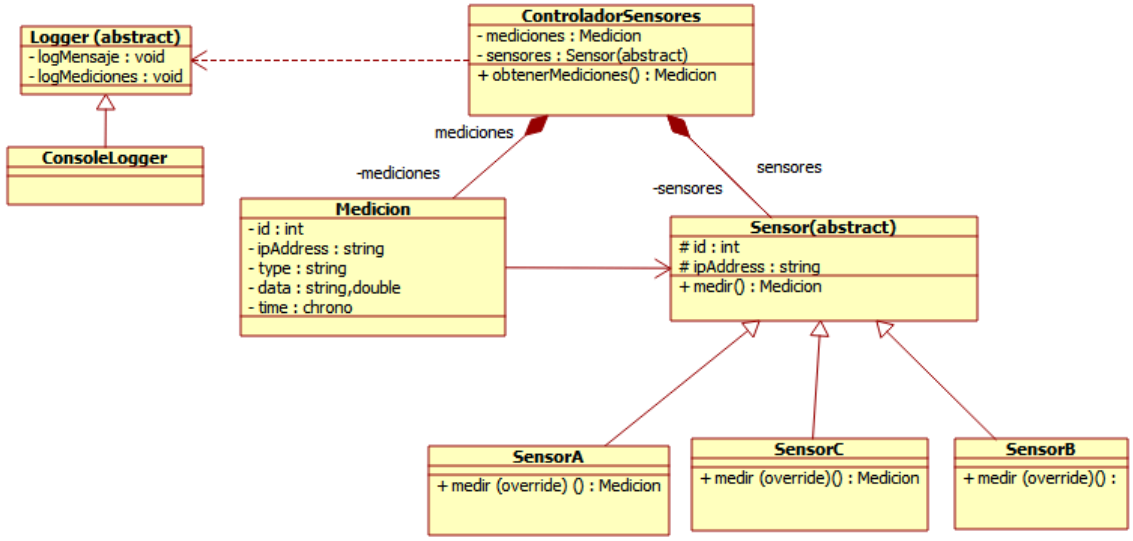
Los archivos csv pueden llegar a estar separados por **coma** y no por **punto y coma** por lo que debería haberse añadido algún verificador.

Recursos adicionales

Librerías convencionales de la última versión del compilador g++.

Ejercicio 3

Diagrama de clases general de la solución



Modo de uso

Solo debe correrse el programa y mediante el archivo .cfg que es proporcionado en la carpeta “cmake-build-debug” dentro de la carpeta “codigo”.

Se envía junto al proyecto, un archivo adicional “*cfgFile.cpp”* que puede ser utilizado para sobreescribir el archivo .cfg existente y pueden cambiarse el vector **sensores**.

```
int main() {
    std::vector <SensorCfg> sensores = {
            {'A', 1, "192.168.1.10"},
            {'B', 2, "192.168.1.11"},
            {'C', 3, "192.168.1.12"},
            {'A', 4, "192.168.1.13"},
            {'B', 5, "192.168.1.14"},
            {'C', 6, "192.168.1.16"}
            };
```

Si se desea cambiar el archivo .cfg solo debe agregarse a dicha carpeta con el nombre "**sensores.cfg**".

Una vez en ejecución el programa debería mostrar:

```
C:\Users\Usuario\Desktop\RepoProgramacion\CASTEL_POO_2024\L13784_TP2\L13784_TP2_3\codigo\cmake-build-debug\L13784_TP2_3.exe
Sensores cargados
Mediciones recolectadas
Mediciones recolectadas
Mediciones recolectadas
```

Para luego mostrar:

Tipo	ID	IP	Temperatura	Humedad	Presion	Lumens	Radiacion UV	Viento
A	1	192.168.1.10	17.5721	68.3508	1040.78	0	0	0
B	2	192.168.1.11	0	0	0	92523	10	0
C	3	192.168.1.12	0	0	0	0	0	63.3992
A	4	192.168.1.13	21.5139	78.0694	1016.78	0	0	0
B	5	192.168.1.14	0	0	0	97161	1	0
C	6	192.168.1.16	0	0	0	0	0	74.8955
A	1	192.168.1.10	21.888	82.2138	1049.08	0	0	0
B	2	192.168.1.11	0	0	0	98910	2	0
C	3	192.168.1.12	0	0	0	0	0	36.4788
A	4	192.168.1.13	33.0755	52.2568	1032.12	0	0	0
B	5	192.168.1.14	0	0	0	93017	6	0
C	6	192.168.1.16	0	0	0	0	0	28.5409
A	1	192.168.1.10	28.4898	84.6522	1007.3	0	0	0
B	2	192.168.1.11	0	0	0	98466	11	0
C	3	192.168.1.12	0	0	0	0	0	46.9314
A	4	192.168.1.13	21.8044	77.3247	1026.24	0	0	0
B	5	192.168.1.14	0	0	0	94442	13	0
C	6	192.168.1.16	0	0	0	0	0	58.4704
A	1	192.168.1.10	19.5112	91.5677	1047.01	0	0	0
B	2	192.168.1.11	0	0	0	96269	9	0
C	3	192.168.1.12	0	0	0	0	0	48.1368
A	4	192.168.1.13	24.0982	92.0789	1022.34	0	0	0
B	5	192.168.1.14	0	0	0	95388	5	0
C	6	192.168.1.16	0	0	0	0	0	93.1394
A	1	192.168.1.10	21.7006	98.7243	1016.83	0	0	0
B	2	192.168.1.11	0	0	0	93560	2	0
C	3	192.168.1.12	0	0	0	0	0	96.231
A	4	192.168.1.13	29.8625	70.3497	1029.55	0	0	0
B	5	192.168.1.14	0	0	0	94007	1	0
C	6	192.168.1.16	0	0	0	0	0	74.749

Comentarios

La modularización en este proyecto fue crucial, además trabaje con interfaces y la verdad que la comprensión de las mismas ha quedado bastante asentada.

Procesador parecido a un microcontrolador:

```
controlador.cargarSensores("sensores.cfg");
for (int i = 0; i < 5; ++i) {
    controlador.recolectarMedidas();
    std::this_thread::sleep_for(std::chrono::seconds(1));
}
```

En estas líneas de código parece que se está manejando cualquier microcontrolador al cual se le está imponiendo un delay en el ciclo, me pareció interesante destacarlo.

La implementación de un *logger* es muy interesante a mi parecer, ya que tiene todo el sentido del mundo separar correctamente las capas, en mi caso creo que logró su cometido, ya

que se podría haber realizado cualquier otro módulo dedicado a *loggear* y hubiera podido implementarse sin ningún problema.

Respecto a la creación dinámica de los sensores dados en el archivo .cfg, fue interesante porque pude aprender un poco sobre los ***smart pointers*** pero no debe olvidarse que cuando se trata de un microcontrolador (*o cualquier dispositivo sin una memoria RAM destacable)* el manejo dinámico de objetos o variables no es recomendable, ya que puede llegarse a una situación de overflow.

Breve conclusión

Este ejercicio fue realmente entretenido y exprime no sólo bastante más que los anteriores las ventajas del diseño y programación OO, sino que también las particularidades de C++. También se concluye la cantidad inmensa de aplicaciones que tienen las variables tipo ***vector*** y la versatilidad que presentan.

La implementación de una clase **Sensor** que sea abstracta creo que fue acertada, ya que permite fácilmente la agregación de cualquier otro tipo de sensor agregando el módulo correspondiente al mismo y heredando dicha clase, además de modificar

```
while(configFile >> tipoSensor >> id >> ipAddress) {
    Sensor* sensor = nullptr;
    switch (tipoSensor) {
        case 'A': sensor = new SensorA(id, ipAddress); break;
        case 'B': sensor = new SensorB(id, ipAddress); break;
        case 'C': sensor = new SensorC(id, ipAddress); break;
        default: m_logger->logMessage("Error: tipo de sensor desconocido"); break;
    }
    if(sensor) {
        sensores.push_back(sensor);
    }
}
```

y agregar el caso respectivo.

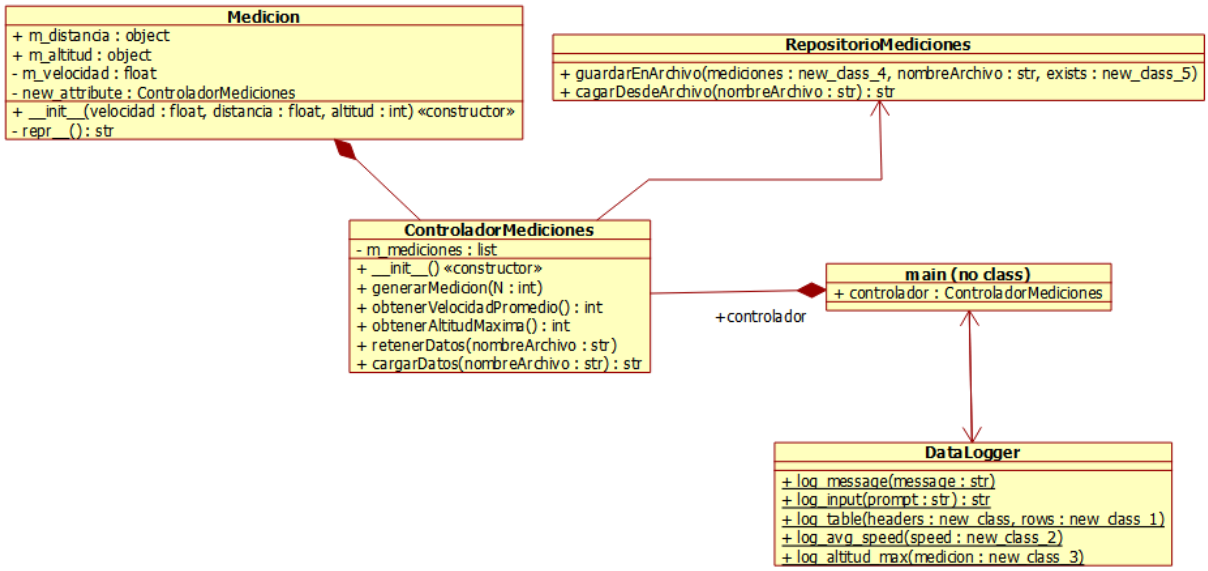
Recursos adicionales

Librerías (convencionales e instaladas por default):

- chrono
- map
- string_view
- ctime
- cstdlib

Ejercicio 4

Diagrama de clases general de la solución



Modo de uso

Para ejecutar el programa, solo se necesita una versión de Python 3.11 o superior y correr el script en IDE de preferencia.

Lo primero que muestra el programa es:

```
----------------------------
1. Generar mediciones
2. Obtener velocidad promedio
3. Obtener altitud máxima
4. Persistir datos en archivo
5. Cargar datos desde archivo
6. Mostrar datos en tabla
7. Salir
Ingrese una opción:
```

Donde las únicas dos opciones que podemos elegir sin tener mediciones previas cargadas son la 1 y la 5.

```
1. Generar mediciones
2. Obtener velocidad promedio
3. Obtener altitud máxima
4. Persistir datos en archivo
5. Cargar datos desde archivo
6. Mostrar datos en tabla
7. Salir
Ingrese una opción: 2
------------------------------
ERROR: No hay mediciones
------------------------------
```

Luego si se quieren generar mediciones:

```
1. Generar mediciones
2. Obtener velocidad promedio
3. Obtener altitud máxima
4. Persistir datos en archivo
5. Cargar datos desde archivo
6. Mostrar datos en tabla
7. Salir
Ingrese una opción: 1
------------------------------
Ingrese la cantidad de mediciones a generar: 4
4 Mediciones generadas
```

Y se desea por ejemplo obtener la velocidad promedio:

```
Ingrese una opción: 2
------------------------------
La velocidad promedio es: 55.49 m/s
```

O mostrar los datos en una tabla:

```
Ingrese una opción: 6
------------------------------
Velocidad (m/s)   Distancia (m)     Altitud (m)
-----------------------------------------------
4.00              411.36            9925.00
37.81             412.13            4158.00
94.41             684.38            7798.00
85.73             215.49            2048.00
```

También se presenta el siguiente mensaje si se desean generar nuevamente mediciones cuando ya existe al menos una cargada en memoria:

```
Ingrese una opción: 1
-----------------------------
Hay mediciones cargadas, si continua se agregarán a las existentes
Ingrese 's' para continuar,'b' crear nuevas o 'n' para cancelar:
```

Cabe aclarar que crear nuevas mediciones no borrará las ya persistidas en el archivo (si lo estaban), son situaciones completamente independientes.

Comentarios

Se utilizó un esquema de persistencia sencillo mediante el uso de la librería ***pickle*** donde mediante el siguiente fragmento de código se le dice al programa que cualquier objeto dela clase Medición será representado de esa forma, luego la librería pickle realizará la correspondiente serialización en binario:

```
def __repr__(self):  fcastel2002 *
    """
    Devuelve una representación en cadena del objeto Medicion.

    Returns:
        str: Una cadena que describe el objeto Medicion.
    """
    return f"{self.m_velocidad},{self.m_distancia},{self.m_altitud}"
```

El manejo de excepciones no es el mejor pero creo que puedo decir que cubre varias posibilidades de interacción errónea con el usuario.

El uso del logger se ve bastante claro al momento de mostrar el menú y la utilidad que tiene un módulo aparte que se encargue de ello, ya que permite la traducción a cualquier otra interfaz solamente modificando los tipos de log que se han implementado. Nuevamente el diseño OO provee muchísima flexibilidad al proyecto.

Breve conclusión

Trabajando por primera vez con la persistencia de objetos me da una idea de que es una herramienta muy útil para proyectos más grandes y que, al menos en Python no es compleja su implementación.

El uso del logger fue clave, pero se podría haber implementado el concepto de **tipo de mensaje** o **tipo de log** , entonces cada log desde el programa principal se envía con una etiqueta la cual informa si es información, advertencia, o directamente error.

Recursos adicionales

Librería **pickle** y **random**.