

Ejercicio 1

No implementé el código de forma que el tamaño de la matriz fuera fijo, sino que permití al usuario elegirlo, al igual que en el ejercicio 2.

Modo de uso

Cuando se ejecuta el programa se muestra al usuario lo siguiente

```
C:\Users\Usuario\Desktop\P00\ejercicio1_tp1.exe
Ingrese tamaño de matriz...
Cantidad de filas:
```

Luego de ingresar la cantidad de filas, se pide la cantidad de columnas, ambos inputs se deben confirmar con la tecla “Enter”.

Se genera la matriz con caracteres aleatorios y se libera la memoria al finalizar el programa.

```
C:\Users\Usuario\Desktop\P00\ejercicio1_tp1.exe
Ingrese tamaño de matriz...
Cantidad de filas:4
Cantidad de columnas:3
I C ~
$ a l
V n r
p I 1
Matriz borrada con éxito

Process finished with exit code 0
```

El programa también comprueba si se ingresó una cantidad de filas o columnas válidas.

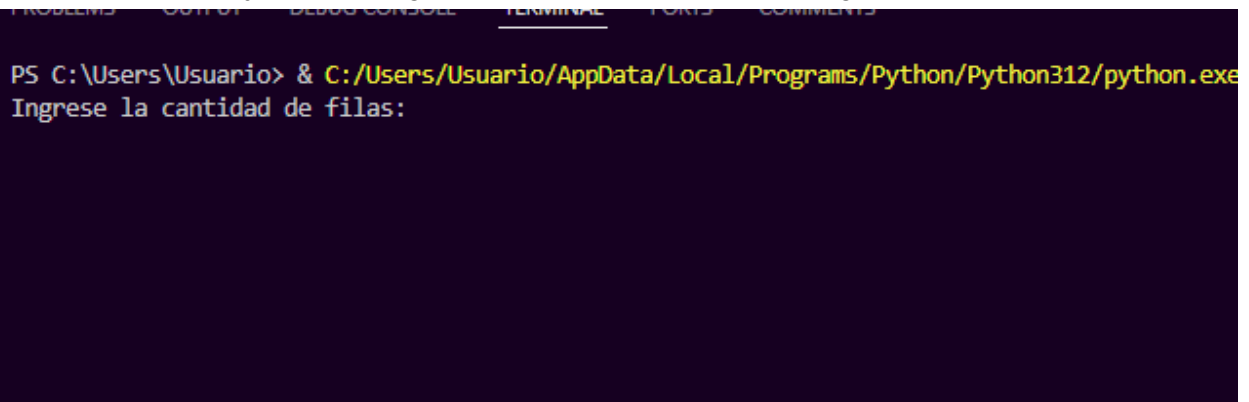
Recursos adicionales

Se utilizó la librería “windows.h” para setear la salida de la consola en formato UTF8, presentaba un error al mostrar caracteres en español tales como “ñ” o letras con tilde.

Ejercicio 2

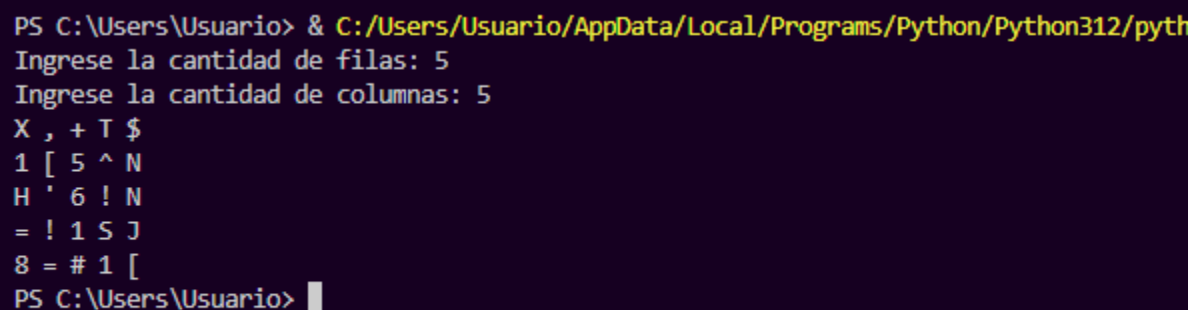
Modo de uso

Cuando se ejecuta el programa se muestra al usuario lo siguiente



```
PS C:\Users\Usuario> & C:/Users/Usuario/AppData/Local/Programs/Python/Python312/python.exe  
Ingrese la cantidad de filas:
```

Luego si el usuario procede con el ingreso de los valores, se obtiene la siguiente salida



```
PS C:\Users\Usuario> & C:/Users/Usuario/AppData/Local/Programs/Python/Python312/pyth  
Ingrese la cantidad de filas: 5  
Ingrese la cantidad de columnas: 5  
X , + T $  
1 [ 5 ^ N  
H ' 6 ! N  
= ! 1 5 J  
8 = # 1 [  
PS C:\Users\Usuario> |
```

Comentarios

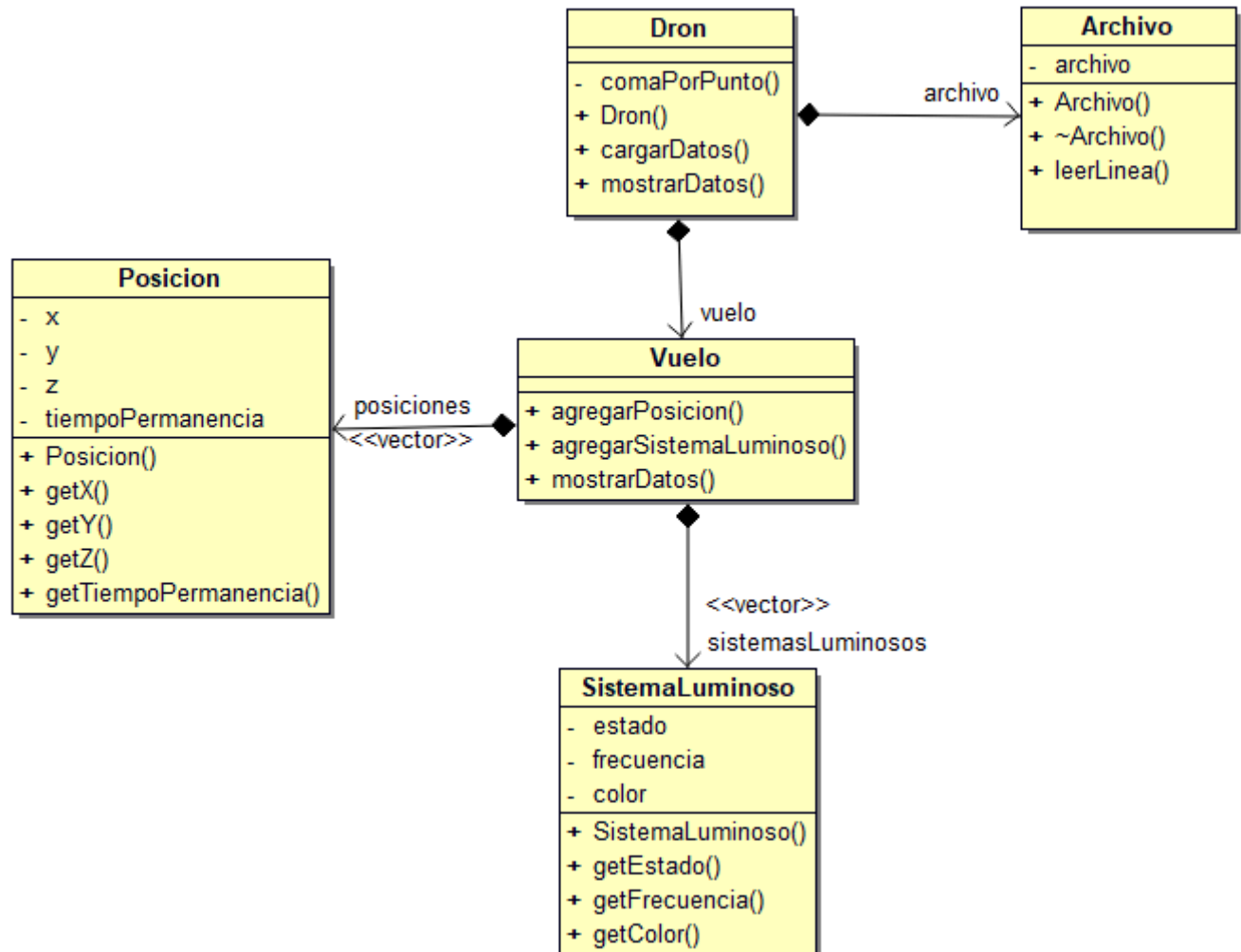
No fue necesaria la implementación de una librería simil “windows.h” como en el ejercicio 1, el formato de la consola ya estaba dado en UTF8, pero es posible que haya sido un problema de mi IDE.

Recursos adicionales

Se incluyó la librería “random”.

Ejercicio 3

Esquema general de la solución



Como puede verse en el diagrama, se crearon 5 clases, donde la clase principal es Dron, la cual se comunica unicamente con Vuelo y Archivo de forma directa.

Modo de uso

Para ejecutar el programa, dado que en `main()` se pasa como argumento el nombre del archivo, la forma de realizarlo es:

1. Colocar el archivo `.dat` en la misma carpeta donde el ejecutable.
2. Abrir terminal (Powershell o CMD) en el directorio donde se encuentra el ejecutable
3. Ejecutar el programa con el comando: `.\ejercicio3_tp1.exe c3-tray-01.dat`
4. En caso de que no funcionase el paso 3, debe incluirse como parámetro de ejecución en el IDE: `c3-tray-01.dat` (esto fué probado en CLion y funcionó).

5. La salida del programa debería mostrarse en consola.

Comentarios

Se diferencié a la clase Posición, de la clase Vuelo dado que la clase Dron podría llegar a necesitar otro tipo de datos de la clase Vuelo, como por ejemplo algún tipo de control PID para los motores, esta decisión fué fuertemente basada en el principio de Responsabilidad Única.

Se observa además el uso de la librería “vector.h” la cual facilita el tratamiento de las coordenadas en un array de 4 elementos.

Por el momento, la clase Dron sería la que se encarga de recibir la información, cargarla en memoria y mostrarla en por ejemplo, su pantalla integrada, luego evidentemente la clase Dron podría encargarse de enviar las consignas recibidas por Vuelo a cada motor.

Se optó por utilizar una librería para parsear archivos del tipo CSV, se incluyó el concepto de *token*.

Conclusiones

Puedo concluir que es clave el enfoque de la programación orientada a objetos para un sistema como este, dado que suponiendo un sistema más grande, podría llegar a necesitarse de muchos más archivos.

También encontré interesante la implementación de la librería *fstream* la cual me facilitó la etapa de la recopilación de datos.

Recursos adicionales

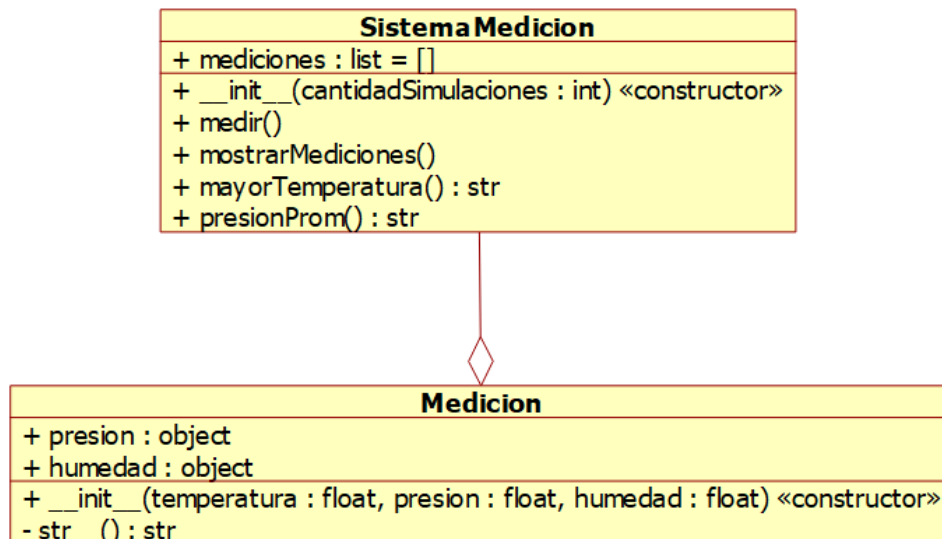
Bibliotecas:

- `iostream`
- `fstream`
- `vector`
- `string`
- `sstream`
- `iomanip`

Las principales funcionalidades de las clases incluidas son, en el caso de “*fstream*” que es la librería típica para la manipulación de archivos en C++, proporciona algunas clases para el manejo de estos, en este caso se utilizó solo la clase *ifstream* (input file stream, o sea para leer archivos). También se incluyó la librería “*sstream*”, la cual se utilizó para parsear archivos del tipo CSV; proporcionando el concepto de *token* para una fácil extracción de los datos, utilizando como delimitación el “,”.

Ejercicio 4

Esquema general de la solución



La clase *SistemaMedicion* presenta una relación de agregación con la clase *Medicion*

Modo de uso

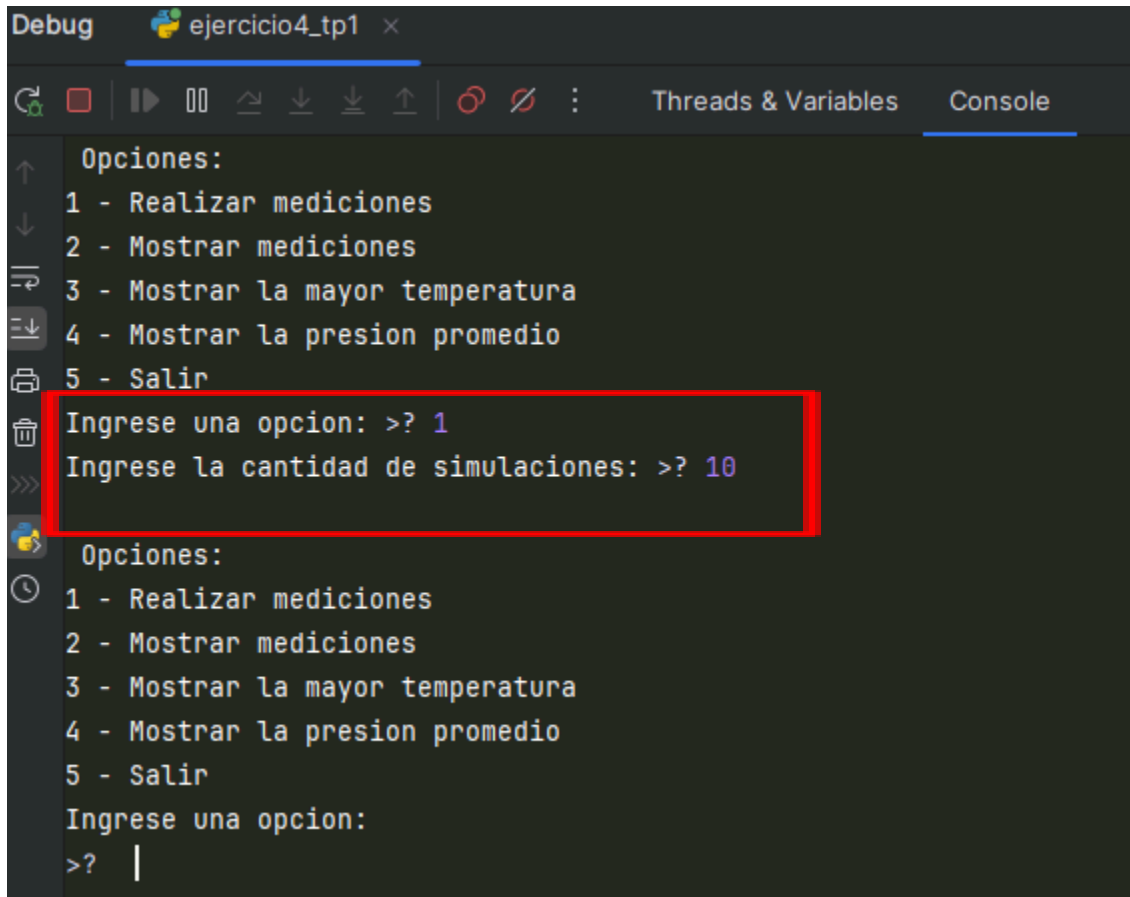
Cuando se corre el programa, se muestra al usuario en consola lo siguiente

```

/////Sistema de medicion/////
Simule las mediciones de un sistema, y realice operaciones sobre las mediciones
Cualquier operacion, sera realizada sobre las ultimas mediciones simuladas
Cada vez que quiera volver a realizar mediciones, se perderan las anteriores

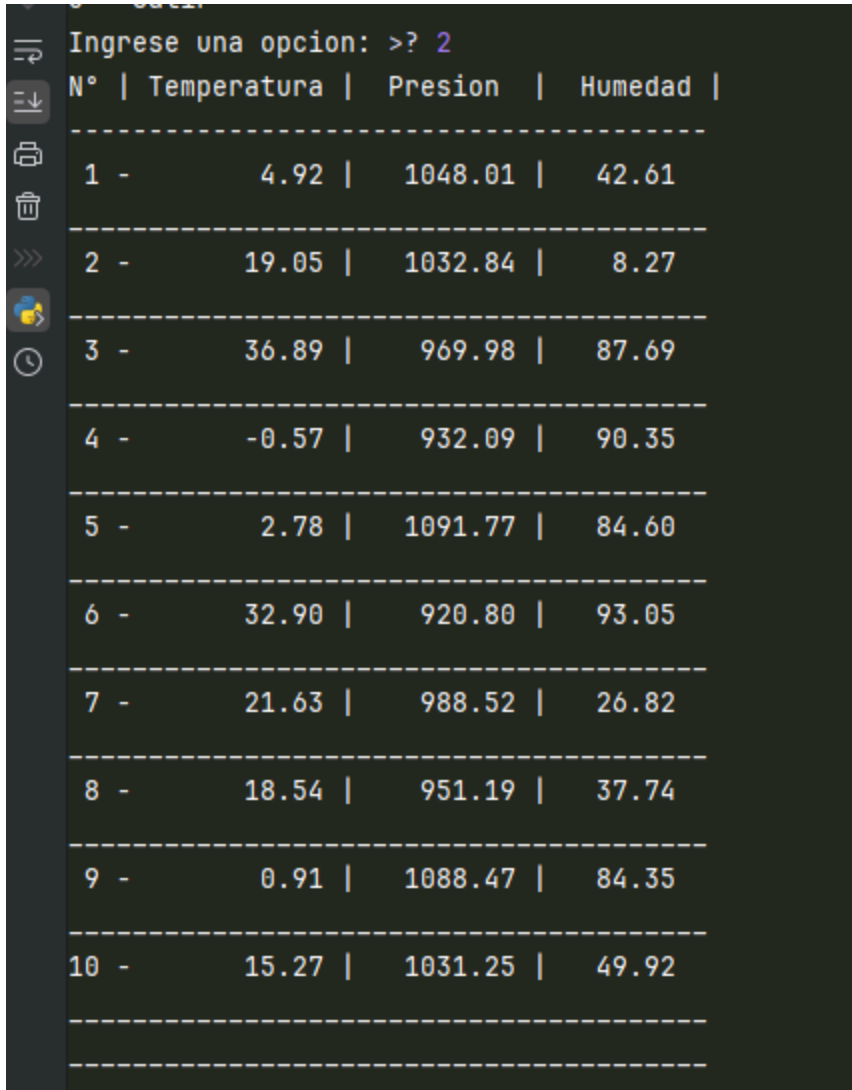
Opciones:
1 - Realizar mediciones
2 - Mostrar mediciones
3 - Mostrar la mayor temperatura
4 - Mostrar la presion promedio
5 - Salir
Ingrese una opcion:
>?
  
```

El usuario puede ingresar distintas opciones, en el caso de que ingrese la opción 1, se le volverá a pedir una entrada, en este caso la cantidad de simulaciones que desea realizar.



```
Debug ejercicio4_tp1 x
Threads & Variables Console
Opciones:
1 - Realizar mediciones
2 - Mostrar mediciones
3 - Mostrar la mayor temperatura
4 - Mostrar la presion promedio
5 - Salir
Ingrese una opcion: >? 1
Ingrese la cantidad de simulaciones: >? 10
Opciones:
1 - Realizar mediciones
2 - Mostrar mediciones
3 - Mostrar la mayor temperatura
4 - Mostrar la presion promedio
5 - Salir
Ingrese una opcion:
>? |
```

Luego, eligiendo la opción 2, se le mostrarían las últimas mediciones realizadas.



```
Ingrese una opcion: >? 2
```

N°	Temperatura	Presion	Humedad
1 -	4.92	1048.01	42.61
2 -	19.05	1032.84	8.27
3 -	36.89	969.98	87.69
4 -	-0.57	932.09	90.35
5 -	2.78	1091.77	84.60
6 -	32.90	920.80	93.05
7 -	21.63	988.52	26.82
8 -	18.54	951.19	37.74
9 -	0.91	1088.47	84.35
10 -	15.27	1031.25	49.92

Manejo de errores

Se implementó un humilde manejo de errores para este programa, el cual contempla la mala utilización del programa por parte del usuario.

Supongamos que el usuario desea mostrar mediciones y no se ha generado ninguna todavía.

```
-----
Opciones:
1 - Realizar mediciones
2 - Mostrar mediciones
3 - Mostrar la mayor temperatura
4 - Mostrar la presion promedio
5 - Salir
Ingrese una opcion: >? 2
Error: No hay mediciones para mostrar

/////Sistema de medicion/////
Simule las mediciones de un sistema, y realice operaciones sobre las mediciones
Cualquier operacion, sera realizada sobre las ultimas mediciones simuladas
Cada vez que quiera volver a realizar mediciones, se perderan las anteriores

-----
Opciones:
1 - Realizar mediciones
2 - Mostrar mediciones
3 - Mostrar la mayor temperatura
4 - Mostrar la presion promedio
5 - Salir
Ingrese una opcion:
>? |
```

El programa muestra el mensaje de error y vuelve a relanzarse, para permitirle al usuario corregir su error, procede de la misma manera con cualquier otra opción mal elegida.

Comentarios

No hay comentarios adicionales.

Conclusiones

En estos programas de tipo preguntar al usuario y a partir de ahí tomar acción, el manejo de excepciones es primordial, en mi caso no pude implementarlo de la mejor forma dado que no pude contemplar casos en los que el usuario ingresara algún caracter raro en la consola.

Recursos adicionales

Librería "random".