Instituto Tecnológico de Estudios Superiores Monterrey
Campus Querétaro

Programación Orientada a Objetos

Situación Problema

Profesor:
Silvana de Gyvés Avila

Presenta:
Carlos Eduardo Velasco Elenes A01708634

Fecha de entrega: 11 de junio de 2022

Índice de Contenidos

Introducción

En los años recientes, servicios a bajo costo en demanda como Netflix, Disney o DC se han vuelto populares. Algunos de estos servicios están enfocados en el volumen de vídeos disponibles a los usuarios y algunos otros son retados de mostrar sólamente su contenido exclusivo. Una versión limitada para ayudar a un proveedor de contenido para este tipo de servicios se describe a continuación.

El proveedor trabajará con 2 tipos de videos: películas y series. Cada video tiene su ID, nombre, longitud, categoría, por ejemplo drama, acción, misterio.

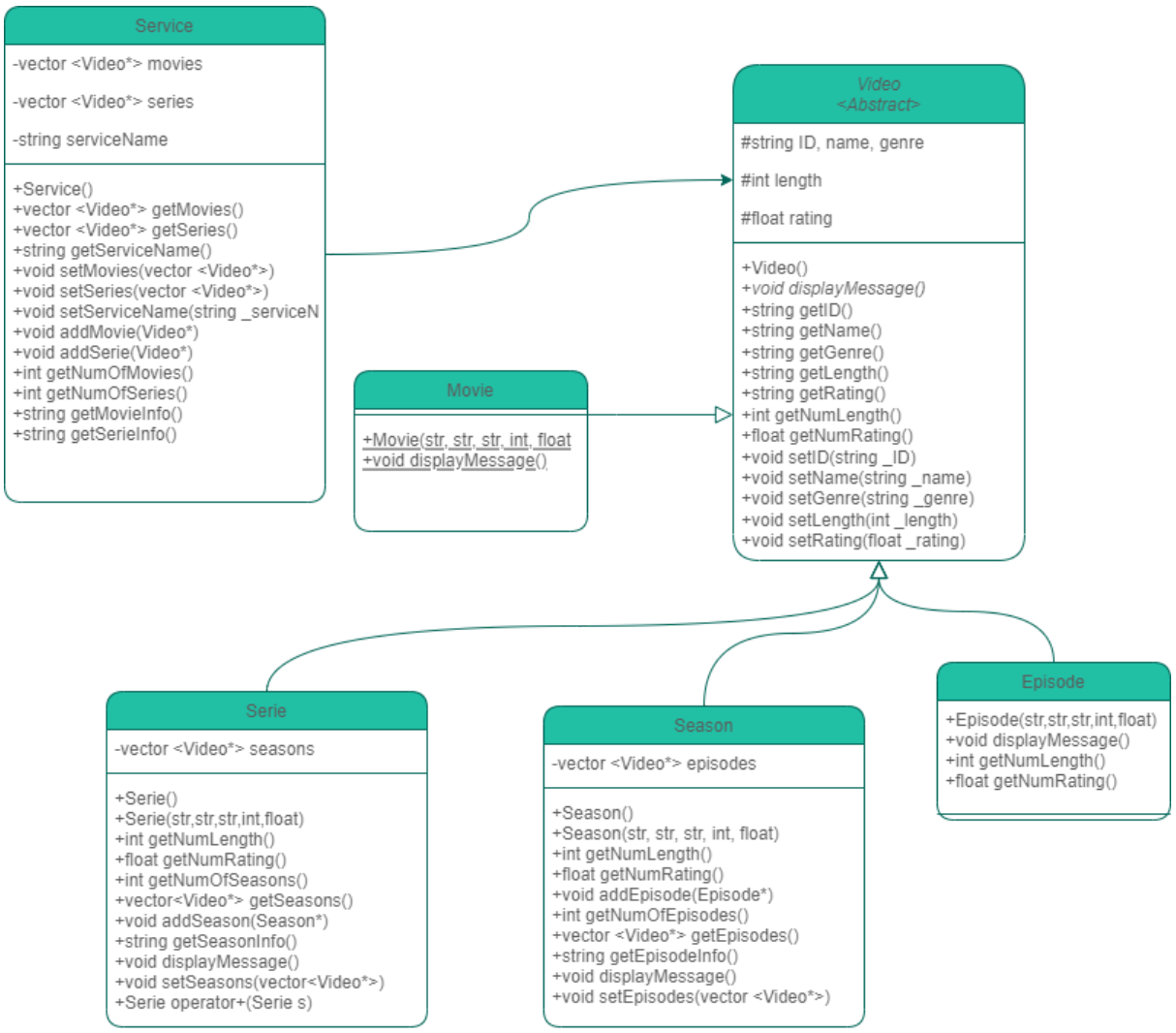
Las series tienen episodios y cada episodio tiene un título y una temporada a la que pertenece.

Estamos interesados en saber el rating promedio que cada uno de estos videos recibe. Este rating es en escala de 1 a 10, donde 10 es la mejor calificación.

El sistema computacional requerido para manejar videos debe ser capaz de hacer lo siguiente:

- Mostrar la colección de videos por título, con sus respectivos ratings.
- Mostrar los episodios de una serie dada, con sus respectivos ratings.
- Mostrar las películas, con sus respectivos ratings.

Diagrama UML



Para esta situación problema, se ideó un diagrama con 6 clases en mente:

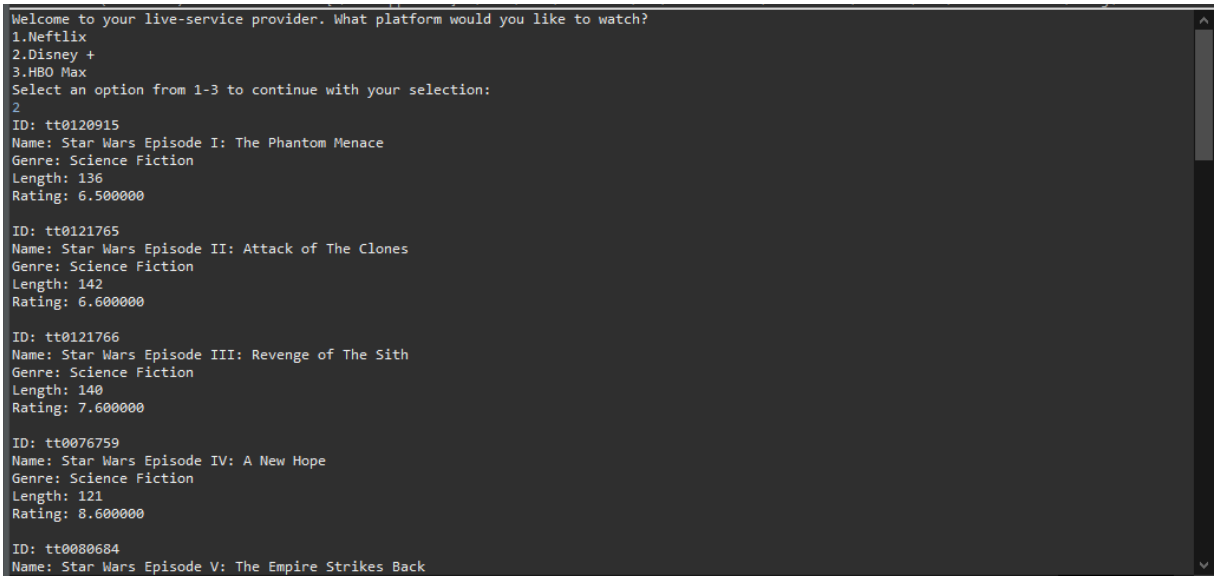
- Service: Clase que funciona del mismo modo que las plataformas digitales de streaming, la cual va a almacenar y mostrar al usuario la información superficial de cada película y serie que se tenga a disposición. Esto nos ayudará a tener una mayor organización de los datos, ya que todos las películas y series de Netflix van almacenadas en un lugar, las películas y series de Disney en otro, y así consecutivamente.
- Video: Nuestra única clase abstracta que tenemos en el programa, ya que en ningún momento vamos a necesitar instanciar para crear objetos. La razón es que en realidad no queremos tener objetos tipo 'Video', queremos tener únicamente objetos que sean subclases de esta clase padre, las cuales son películas, series, temporadas y episodios. A pesar de que es abstracta,

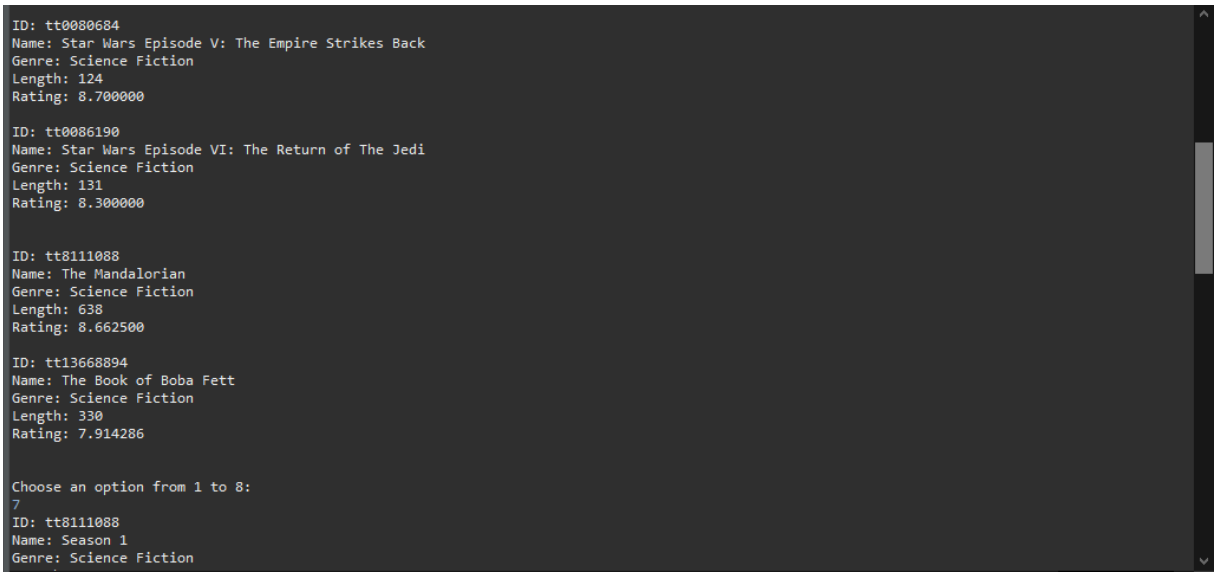
probablemente sea la clase clave de este diseño, ya que la mayoría de las funcionalidades de las 4 subclases previamente mencionadas son precisamente pertenecientes a la clase Video, así que si no existiera esta clase, no se pueden crear ni series ni películas ni nada adicional (esta es la clara justificación del uso de la herencia en el programa).

- Movie: Una clase bastante sencilla porque lo único que necesitamos de ella es la información base para poder declararla (como sabemos, las películas no tienen que lidiar con cuántas temporadas ni cuántos episodios tienen, por lo tanto, sólo necesitamos saber el ID, el nombre, la categoría, la duración y la calificación dada).
- Serie: Una clase fundamental para el desarrollo de este programa, ya que aquí se va a crear un objeto serie, y no solo eso, sino que también va a almacenar toda la información que conlleva este tipo de video: va a guardar un vector (una lista) de temporadas que tenga esa respectiva serie para luego juntar toda esa información y obtener los datos necesarios para declarar los valores correspondientes a la serie.
- Season: Una clase muy parecida a serie, sólo que en este caso va a almacenar en un vector episodios en lugar de temporadas, y como en la clase Serie, se va a llevar a cabo la recolección de datos de cada episodio para determinar atributos como la duración y el rating de todo un objeto temporada.
- Episode: Al igual que “Movie”, es una clase que sólo necesitamos los valores iniciales ya que su única función relevante es ser mostrado en pantalla con todos sus datos.

Ejemplo de Ejecución

A continuación se mostrará lo que aparece en consola al término de ejecución del programa:





```
Genre: Science Fiction
Length: 315
Rating: 8.474999

ID: tt8111088
Name: Season 2
Genre: Science Fiction
Length: 323
Rating: 8.850000


Choose an option from 1 to 2
1
ID: tt9095424
Name: Chapter 1: The Mandalorian
Genre: Science Fiction
Length: 39
Rating: 8.600000

ID: tt9121530
Name: Chapter 2: The Child
Genre: Science Fiction
Length: 31
Rating: 8.500000

ID: tt9121534
Name: Chapter 3: The Sinn
Genre: Science Fiction
Length: 37
Rating: 9.000000

ID: tt9121536
```

```
Rating: 9.000000

ID: tt9121536
Name: Chapter 4: Sanctuary
Genre: Science Fiction
Length: 41
Rating: 7.700000

ID: tt9121538
Name: Chapter 5: The Gunslinger
Genre: Science Fiction
Length: 35
Rating: 7.500000

ID: tt9121542
Name: Chapter 6: The Prisoner
Genre: Science Fiction
Length: 43
Rating: 8.300000

ID: tt9121544
Name: Chapter 7: The Reckoning
Genre: Science Fiction
Length: 41
Rating: 9.000000

ID: tt9121546
Name: Chapter 8: Redemption
Genre: Science Fiction
Length: 48
Rating: 9.200000
```

```
ID: tt9121538
Name: Chapter 5: The Gunslinger
Genre: Science Fiction
Length: 35
Rating: 7.500000

ID: tt9121542
Name: Chapter 6: The Prisoner
Genre: Science Fiction
Length: 43
Rating: 8.300000

ID: tt9121544
Name: Chapter 7: The Reckoning
Genre: Science Fiction
Length: 41
Rating: 9.000000

ID: tt9121546
Name: Chapter 8: Redemption
Genre: Science Fiction
Length: 48
Rating: 9.200000


Here are the selection of episodes! Enjoy!
Thank you for your visit! We hope to see you soon! If you didn't get to watch them all, here is a bit of info about the services that we have i
The total length of Netflix Series are: 2262. And the average Rating is: 8.571453
The total length of Disney Plus Series are: 968. And the average Rating is: 8.288393
The total length of HBO Series are: 1098. And the average Rating is: 8.545536

The total length of ALL Series are: 4328. And the average Rating is: 8.467300
```

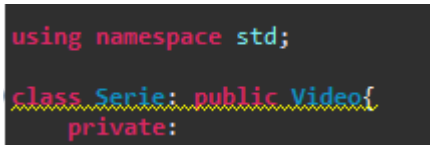
Argumentación de Implementación

A) Clases

Las 6 clases están declaradas de manera correcta. Cada una de ellas tiene su archivo.h y su archivo.cpp. Cada una de las clases tiene una participación en el main. Cada una es propiamente definida con los comandos de #ifndef, #define y #endif.

B) Herencia

Existe una relación entre todas las clases, excepto la clase Service. La clase padre viene siendo la clase “Video”, y eso se puede ver reflejado en alguno de los métodos, los cuales tienen un prefijo “virtual”, dejando claro que esta es la superclase a la cual se le está aplicando polimorfismo. Las subclases son: Movie, Serie, Season y Episode. Esto se confirma al ver que después de la declaración de la clase, está declarado que esa clase es hija de la clase pública “Video”. He aquí un ejemplo:



También al observar los métodos, vemos que algunos métodos que han sido declarados en Video, están presentes en las subclases, pero eso lo abordaremos en la sección de métodos de sobreescritura y polimorfismo.

C) Modificadores de acceso

El uso de los 3 modificadores de acceso está siendo utilizado de manera adecuada:

- Público: Todos los métodos de todas las clases están declarados como públicos, esto debido a que este va a ser nuestra forma de poder interactuar, desde el archivo main, con los atributos de los objetos que declaremos, sin necesidad de generar una desorganización durante las declaraciones de los métodos.
- Privado: Todas las clases que no son superclases de otras, tienen sus atributos marcados como privados, es decir, solamente los métodos de esa clase en particular pueden accederlos y modificarlos. Esto para generar un nivel superior de organización, y que a la hora de correr, el código no tenga confusiones de donde almacenar la información

- Protegido: La única clase que tiene este modificador de acceso dentro de su clase, es la única superclase que tenemos, y esa es Video. La razón por la que sus atributos están marcados de esta manera, es que para las subclases de Video puedan ver los atributos como públicos, pero sí una clase no es hija de esta superclase, entonces el acceso directo a los atributos es restringido, por lo tanto, si en el main queremos conocer el atributo de una variable protegida en video, tendríamos que llamar a un método, pero si queremos saber un atributo desde la clase Serie, este puede ser accesado directamente.

D) Métodos de sobreescritura

Ya se había rozado este tema en la parte de herencia, pero en efecto, en este programa también tenemos el uso de method overriding. Esto se puede encontrar en la clase Video, donde tenemos 3 métodos que son virtuales, los cuales significan que van a sufrir una modificación en implementación en las clases hijas.

El primer método que tiene overriding es muy sencillo, simplemente imprime un mensaje depende de en cuál clase estás. Un método simple, pero bastante útil para hacer pruebas en la etapa temprana del código (y de hecho, este método nos ayudará más adelante respecto a la abstracción de la clase Video).

Los otros dos métodos son los que nos interesan con más importancia. getNumRating() y getNumLength(). ¿Por qué existen estos métodos si tenemos getters de Rating y Length? Porque si nos damos cuenta, los getters de Rating y Length regresan strings, esto debido a que para imprimir la información completa de una serie necesitábamos usar strings, por lo tanto, en los getters de estos dos atributos, las variables son transformadas a strings. Así que necesitábamos encontrar una manera de poder obtener la longitud y rating de una serie y temporada por otra manera, y decidimos hacer overriding de dos nuevos métodos, los cuales están declarados en las 3 clases hijas (aunque tendrían el verdadero uso en las clases Serie y Season), los cuales iteran en el vector privado de la clase respectiva, y obtienen los atributos de length y rating y los van sumando, así, nos da la duración y el rating total de una temporada o una serie, dependiendo de los episodios y temporadas respectivamente.

E) Polimorfismo

El polimorfismo está aplicado correctamente en sus dos variantes: compile time (que lo veremos más adelante en la sección de sobrecarga de operadores) y run time (que es la sección de overriding del punto anterior).

Hay que dejar en claro que se siguieron las buenas prácticas y para overriding se utilizaron las palabras reservadas de “virtual” y “overriding” de manera correcta en las clases padres e hijas.

F) Clases Abstractas

Como se mencionó en el diagrama UML y en la sección de métodos de sobreescritura, nuestra clase padre está declarada como una clase abstracta. Esto se debe a que en ningún momento vamos a necesitar a un objeto declarado solamente como tipo Video, por lo tanto, y para tener una mejor optimización, se decidió declarar esta super clase como abstracta. ¿Y cómo podemos hacerlo si todos los métodos son utilizados por las subclases? Justamente el método simple pero útil de displayMessage(), nos ayudará a dejar a esta clase como abstracta, ya que si este método lo igualamos a 0, esto sucederá. Eso es lo que se ve en la implementación del código, y gracias a esto, el código cumple con la declaración de mínimo una clase abstracta.

G) Sobrecarga de operadores

La sobrecarga de operadores fue de suma importancia para el éxito de este código. Lo que se hizo fue implementar un método que utilizara sobrecarga, para poder llevar a cabo una suma de todos los atributos de las series que se tienen disponibles en el proveedor. Como los únicos datos que se pueden sumar y darnos información de valor son la duración y el rating, solo se tomaron esos en cuenta para este método. La declaración de este operator+() se puede ver en la clase Serie, y la implementación de este método se ve al final de nuestro código cuando sumamos cada una de las series de los respectivos servicios de streaming, para al final hacer una sumatoria de duración y un promedio de rating para todas las series sin importar la plataforma. Los resultados se ven imprimidos en la consola, dándonos información vital sobre los datos que se tienen en este proveedor de streaming.

H) ¿Por qué es la mejor solución?

Esta es la mejor solución porque una gran cantidad de los datos son obtenidos con métodos dentro del código. No es necesario llevar a cabo cálculos manuales para poder obtener la duración de una temporada o una serie, ni tampoco es necesario

para saber cuánto es el rating de una temporada o serie. Porque el código tiene implementada toda esa metodología, y lo único que tienes que hacer es llamar al método adecuado.
También es un código que está muy bien organizado, y gracias a ello, se entiende perfectamente dónde van los objetos y qué se hace con ellos, de la misma forma que se intuye muy bien cómo la información es transmitida al usuario.
Un punto adicional es que en el método while no es posible dar un mal input, al menos que utilices símbolos o letras que en algunos casos sí se puede ver un error de compilación, pero en temas numéricos, el programa puede funcionar perfectamente con cualquier input que el usuario ingrese.

Identificación de los casos

Claramente para la realización de este programa se consideró la siguiente pregunta: "¿En qué sección el programa no puede funcionar?". Una pregunta muy común a la hora de desarrollar un código. Primeramente se pensó en lo que puede ingresar el usuario que puede hacer que el programa no funcione, y eso se resolvió rápidamente. Al usuario se le pide específicamente un rango específico en cada rango que ingrese, y si este no cumple con los límites, o si es un carácter no-numérico, saldrá un mensaje en pantalla el cual dirá que lo que ingresó el usuario es incorrecto y que por favor ingrese otro valor el cual cumpla con los parámetros del rango. Esto fue posible gracias a la implementación de condicionales dentro del programa.
Fuera de lo que puede ingresar el usuario, errores del programador pueden suceder como los siguientes:

- Dentro del ciclo while que despliega el menú, el no poner a la condición del ciclo como falsa al terminar la selección del usuario, podría frustrar al usuario ya que lo llevaría de nueva cuenta al inicio del programa.
- A la hora de instanciar una película, serie, temporada o episodio, no colocar de manera correcta los datos de la siguiente manera (ID, name, genre, length, rating). Si se colocan de otra manera, hay problemas en temas de organización de datos e incluso de funcionamiento en la creación del objeto.
- El no declarar a las clases Episodio, Temporada, Serie y Películas como subclases de Video.

- No incluir los headers correspondientes en las clases que trabajan unas con otras.
- No “setear” los atributos de una temporada o de una serie, ya que estos no se declaran al crear un objeto tipo temporada o serie.
- No poner la palabra reservada “endl” (este termina la línea de texto e inicia otra)al final de cada impresión en la consola, ya que esto hace que el texto aparezca desorganizado, y por consecuencia, no legible al usuario.
- No declarar que los métodos van a ser sobreescritos con las palabras clave “virtual” para el método de la clase padre, y “override” para el método de la clase hija.
- Declarar un método abstracto (el cual hace a una clase abstracta) cuando las subclases utilizan ese método para crearse o para aplicar una funcionalidad fuera de la clase superclase.
- Entre otros…

Hay muchísimas otras posibilidades en los que el código no pudo funcionar, y de hecho si llegaron a suceder, pero son temas más de sintaxis o que con pura lógica instantánea se pudieron resolver.

Conclusión

Me llevo una gran experiencia después de todo el proceso que este proyecto involucró. Le invertí bastantes horas para la realización, y al final, el éxito de este código. Tengo que admitir que al principio pensaba que era bastante sencillo, pero con el paso de las horas que le invertí, me di cuenta que esto fue un verdadero reto, más que nada me percaté de esto ya que después de varias horas, decidí darle click a “Build Project” y mi cara de susto cuando vi que tenía en realidad más de 70 errores, realmente me congeló la sangre. Después de debuggear todos los errores y que cada vez salieran más y más, pude generar mis primeras “Stable Builds” y con eso fui dejando atrás el estrés y adquiriendo más confianza, y desde ese punto, fue cuando las cosas fluyeron con más naturalidad.

Los resultados sufrieron grandes cambios desde mi visión inicial, pero todo este camino de aprendizaje me lo llevo con mucha ilusión, ya que esto no solo mejora mis habilidades como programador, sino que también me enseña a que respecto a estas cosas, tengo que aprender a ser más paciente, ya que la única razón detrás

de ese estrés y frustración, era porque las cosas no estaban saliendo como quería. Afortunadamente, pude salir adelante de eso, y terminé con un proyecto del que de verdad estoy muy orgulloso.

Referencias

- A. (2007, 5 octubre). 14.7 — Overloading the comparison operators. Learn C++. Recuperado 7 de junio de 2022, de https://www.learncpp.com/cpp-tutorial/overloading-the-comparison-operators/
- A. (2016, septiembre). Eclipse Community Forums: C / C++ IDE (CDT) » «could not be resolved» errors, but Go To Definition works! | The Eclipse Foundation. Eclipse Foundation. Recuperado 7 de junio de 2022, de https://www.eclipse.org/forums/index.php/t/1086391/
- E. (2013, 18 noviembre). Eclipse CDT error: «Symbol “XXX” could not be resolved». Stack Overflow. Recuperado 7 de junio de 2022, de https://stackoverflow.com/questions/20051550/eclipse-cdt-error-symbol-xxx-could-not-be-resolved
- GeeksforGeeks. (2021, 17 mayo). Types of Operator Overloading in C++. Recuperado 7 de junio de 2022, de https://www.geeksforgeeks.org/types-of-operator-overloading-in-c/
- GeeksforGeeks. (2022, 21 mayo). Operator Overloading in C++. Recuperado 7 de junio de 2022, de https://www.geeksforgeeks.org/operator-overloading-c/
- Programiz. (s. f.). C++ Operator Overloading (With Examples). Recuperado 7 de junio de 2022, de https://www.programiz.com/cpp-programming/operator-overloading
- User3281388. (2015, 31 marzo). Remove Object from C++ list. Stack Overflow. Recuperado 7 de junio de 2022, de https://stackoverflow.com/questions/29378849/remove-object-from-c-list
- X. (2010, 17 mayo). C++ Vector of pointers. Stack Overflow. Recuperado 7 de junio de 2022, de https://stackoverflow.com/questions/2853438/c-vector-of-pointers