Tecnológico de Monterrey

Campus Santa Fe

“Proyecto Integrador: Plataforma de Streaming”

Gabriel Rodríguez De Los Reyes, A01027384

Programación Orientada a Objetos

Jesús Leopoldo Llano García

Fecha de entrega

06 de Mayo, 2022

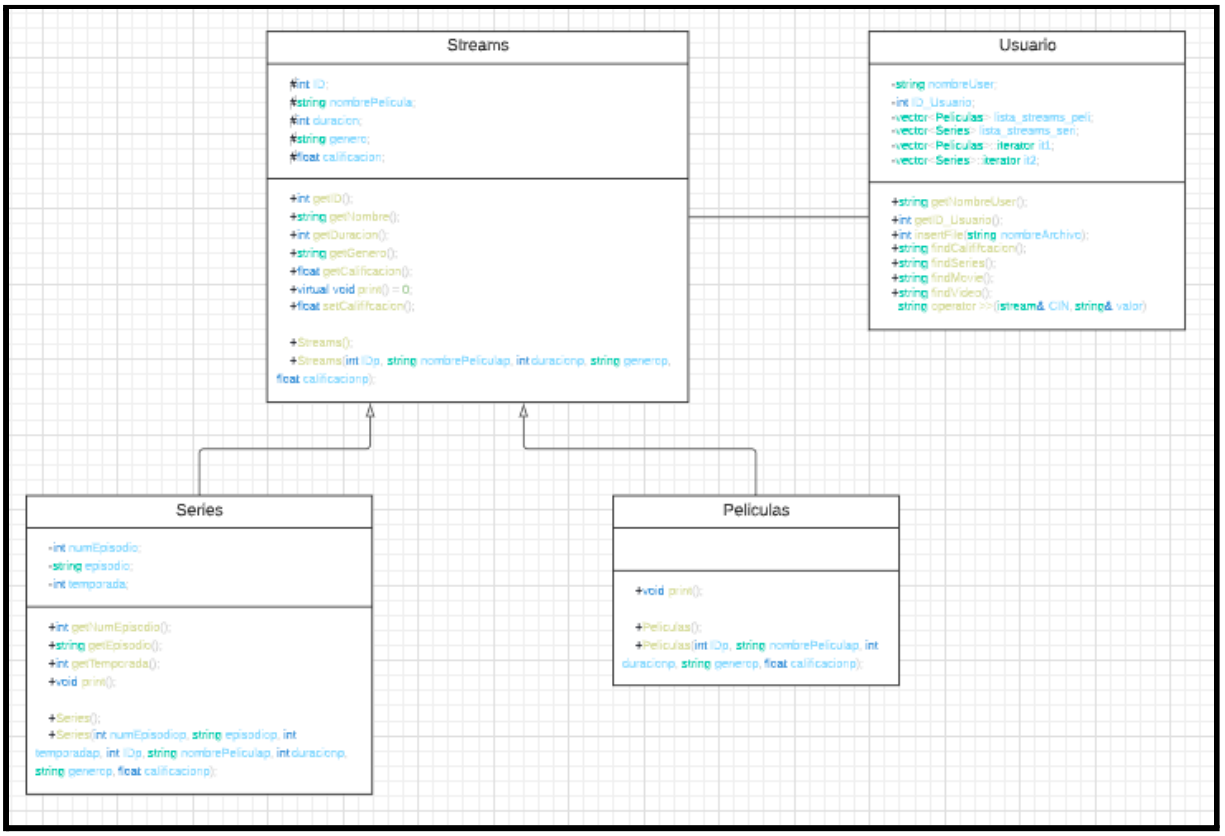
Índice

Introducción

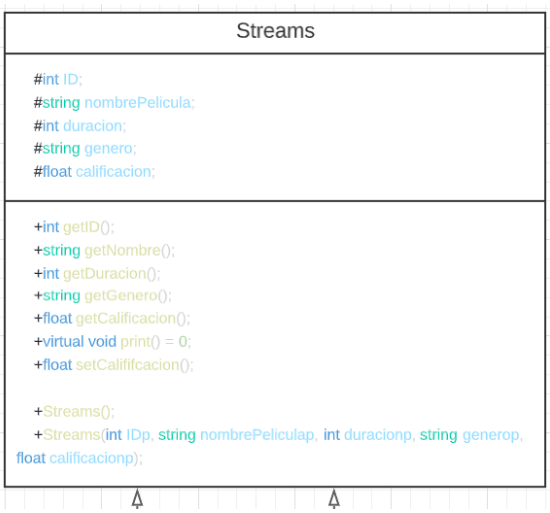
La situación problema que se buscó resolver fue crear un sistema de streaming en el cual los usuarios puedan ingresar series y películas, y poder acceder a ellas. El objetivo del código era proporcionar una manera en la que las personas pueden buscar y encontrar de una manera eficiente cualquier película o serie, por su calificación o género. Además, la solución plantea una opción para poder cambiar la calificación de alguno de los videos, para que así el usuario tenga la potestad de ordenar su catálogo con respecto a sus gustos. Este sistema no solo ayudaría mucho a los usuarios a organizar sus películas y a calificarlas, si no que no tendría la limitante de estar sujeta a una única marca. A diferencia de Netflix, HBO MAX y Disney, nuestro sistema de streaming permitirá el ingreso de cualquier tipo de película, independientemente de su origen o quien posea sus derechos.

Para construir este sistema, se decidió utilizar el lenguaje de programación C + +, con el objetivo de diseñar un software gratuito para este problema. Se decidió utilizar este lenguaje gracias a su gran eficiencia, y su uso adecuado de los objetos. Dentro de C + + las clases y estructuras permiten hacer una gran variedad de funciones que ayudan a optimizar y simplificar procesos que de otra manera serían más complejos. Dentro del programa fue necesario la utilización de diferentes conceptos como herencia, clases abstractas, polimorfismo y sobrecarga de operadores. Muchos de estos elementos serán mostrados y ejemplificados a lo largo de este reporte, con el fin de que el usuario no solo entienda como usarlo, si no que también tenga un idea general del funcionamiento interno del mismo.

Diagramas UML



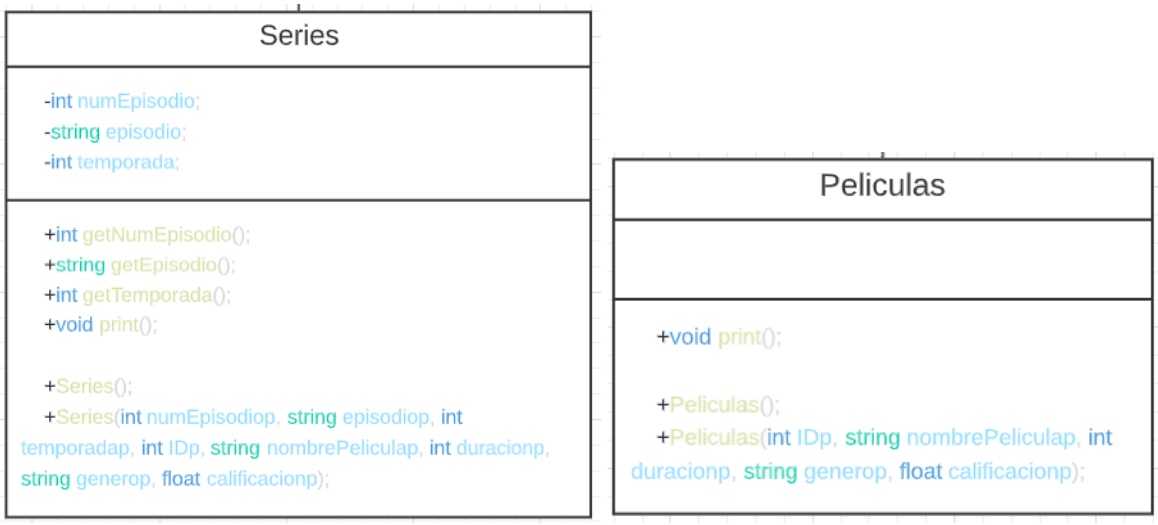
Para la solución de la situación problema se utilizan 4 clases dentro del código, cada una con diferentes funciones. Dentro de estas 4 clases podemos encontrar una clase de tipo abstracto que funciona como clase padre de algunas de las otras clases. Esta clase lleva el nombre de Streams, y es la encargada de construir la mayoría de los atributos y métodos de nuestro código.



Como se puede apreciar esta clase cuenta con los atributos principales que comparten todos los tipos de video: ID, nombre, duración, género, y calificación. Cada uno de estos atributos

almacenan diferentes tipos de datos, pero aun así todos comparten el mismo modificador de acceso **protegido**. Estos atributos fueron almacenados de esta manera para que sus clases sean capaces de acceder y utilizar los valores de estos atributos dentro de sus propios métodos. Además de estos atributos la clase también cuenta con métodos *getters* para cada uno de sus atributos, al igual que un método *setter* para poder modificar el valor de la calificación. En el diagrama también podemos apreciar los constructores, tanto por defecto como por atributos, y un método de tipo **virtual** que nos permite conocer que esta clase es abstracta. El método *print()* es el encargado de plasmar los atributos de una forma agradable para el usuario, y es redefinido en cada una de las clases hijos, ya que al no tener exactamente los mismos atributos el método es editado.

Las clases que heredan Usuario son: Series, Películas.

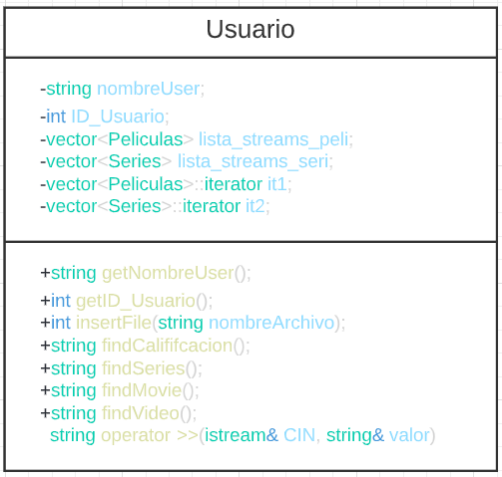


Estas dos clases heredan los atributos y métodos de la clase Streams, y redefinen sus métodos **virtuales** dentro de ellas. Como se puede apreciar del diagrama UML Películas no agrega ningún atributo adicional, y únicamente se encarga de definir el método *print()*. Por el otro lado, Clases si agrega los atributos para el número del episodio, el nombre del episodio, y la temporada. Estos atributos son agregados ya que las series que se ingresan al sistema tiene ciertas características que son exclusivas para este tipo de video. Siguiendo la misma estructura anterior se construyen métodos de tipo *getter* para cada una de los atributos nuevos de Series.

```
void Series::print(){
    cout << "--------------------" << endl;
    cout << "Serie ID: " << getID() << endl;
    cout << "Serie: " << getNombre() << endl;
    cout << "Duracion: " << getDuracion() << "min" << endl;
    cout << "Genero: " << getGenero() << endl;
    cout << "Calificacion: " << getCalificacion() << endl;
    cout << "Temporada: " << getTemporada() << endl;
    cout << "Episodio: " << getNumEpisodio() << endl;
    cout << "Titulo: " << getEpisodio() << endl;
    cout << "--------------------" << endl;
    cout << endl;
}
```

Por último, cabe destacar que a la hora de ingresar los archivos y construir los objetos de tipo Series y Películas, las mismas a su vez crean objetos dentro de Streams con el fin de poder accederlas desde cualquiera de los dos lugares.

La cuarta clase que se ve presente en nuestro código es la clase Usuario, en la que corren todos las funciones que el usuario puede hacer. Es decir, todas las opciones que el usuario efectúa desde el menu, cómo cambiar una calificación o buscar una serie, se ejecutan como métodos de Usuario



Dentro de esta clase podemos ver un gran número de elementos distintos. Primero de todo podemos ver que el usuario define dos atributos principales: nombre e ID, al igual que sus respectivos *getters*. Además de esto, podemos ver que la clase define otros 4 atributos como privados, una lista de objetos *Películas*, una lista de objetos *Series*, un iterador para la lista de películas, y por último otro iterador para la lista de series.

Todos estos atributos fueron definidos ya que en los métodos de esta clase se están utilizando constantemente for loops. Estos bucles son utilizados en los métodos como *findSeries()*, *findMovie()*, *findVideo()*, and *find Calificación()*, en los que se buscan ciertos atributos específicos dentro de los objetos de Películas y Series. Estos no son los únicos métodos, ya que también se construyó un método para ingresar los archivos *.csv*. Por último, cabe destacar que se hizo una sobrecarga de operador a ">>" el cual funciona para que el usuario ingrese una función a una variable. Dentro de C + +, este operador no permite que se ingrese una oración con espacios, pero alguno de los títulos los llevan. Por esta razón sobrecargamos el operador para que el usuario pueda ingresar oraciones completas sin preocuparse de los espacios.

Ejecución

```
Gabriels-MacBook-Air-4:Reto2doSemestre gabrielrodriguezdelosreyes$ ./a.out
Bienvenido a la plataforma de streaming
Ingrese su usuario:
```

El sistema primero te dará la bienvenida y posteriormente te solicitará que ingreses tu nombre el cual será registrado como uno de los atributos del objeto Usuario.

```
Ingrese su usuario: Gabo_Rodriguez

================================================================================
1. Cargar archivo de datos
2. Mostrar las peliculas en general con una cierta calificación o de un cierto género
3. Mostrar los episodios de una determinada serie con una calificacion determinada
4. Mostrar las películas con cierta calificacion
5. Calificar un video

0. Salir
================================================================================
```

Posteriormente aparece el menú de todas las posibles opciones que tiene el usuario. Al ingresar 1 el código le solicitará al usuario el nombre del archivo *.csv* que quiere abrir. En este escenario nosotros ya habíamos introducido el nombre del archivo para agilizar el proceso.

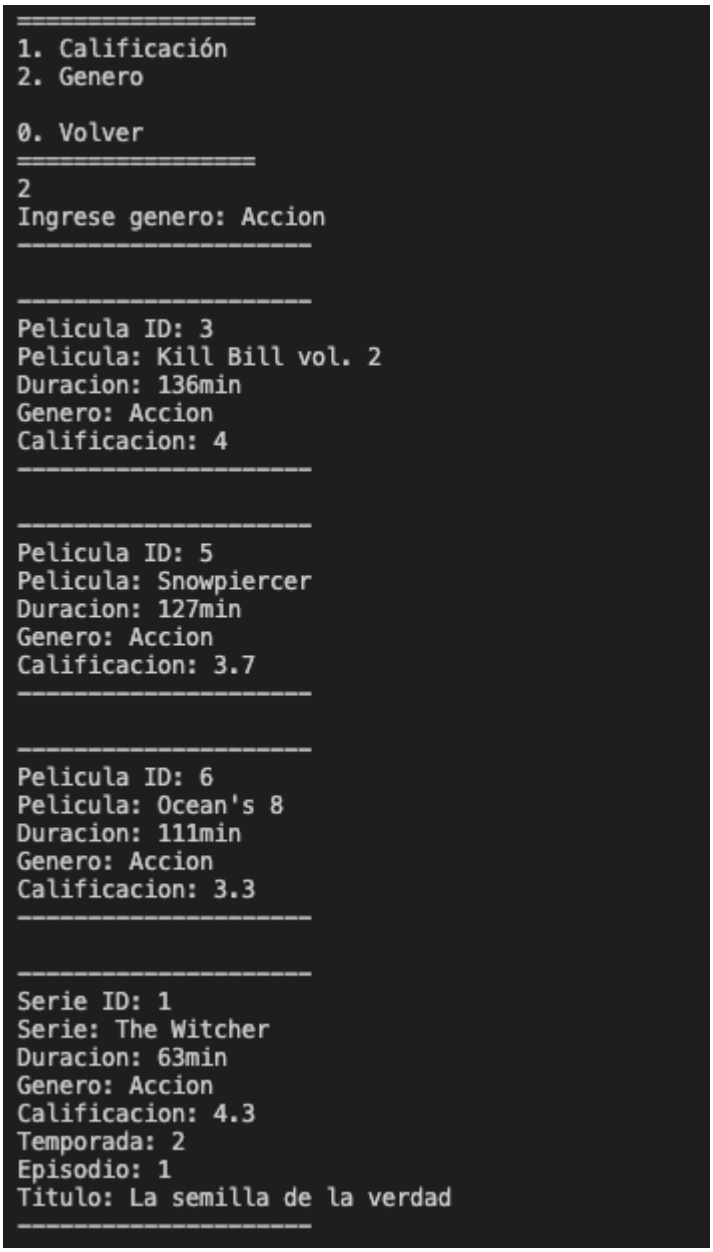
```
case 1:
    // cout << "Ingrese nombre del archivo" << endl;
    // cin >> nombreArchivo;
    nombreArchivo1 = "series.csv";
    nombreArchivo2 = "peliculas.csv";
    usuario1.insertFile(nombreArchivo1);
    usuario1.insertFile(nombreArchivo2);
    break;
```

El código marcará cuantas películas y series encuentra dentro de los documentos y separará las películas y las series en listas por separado.

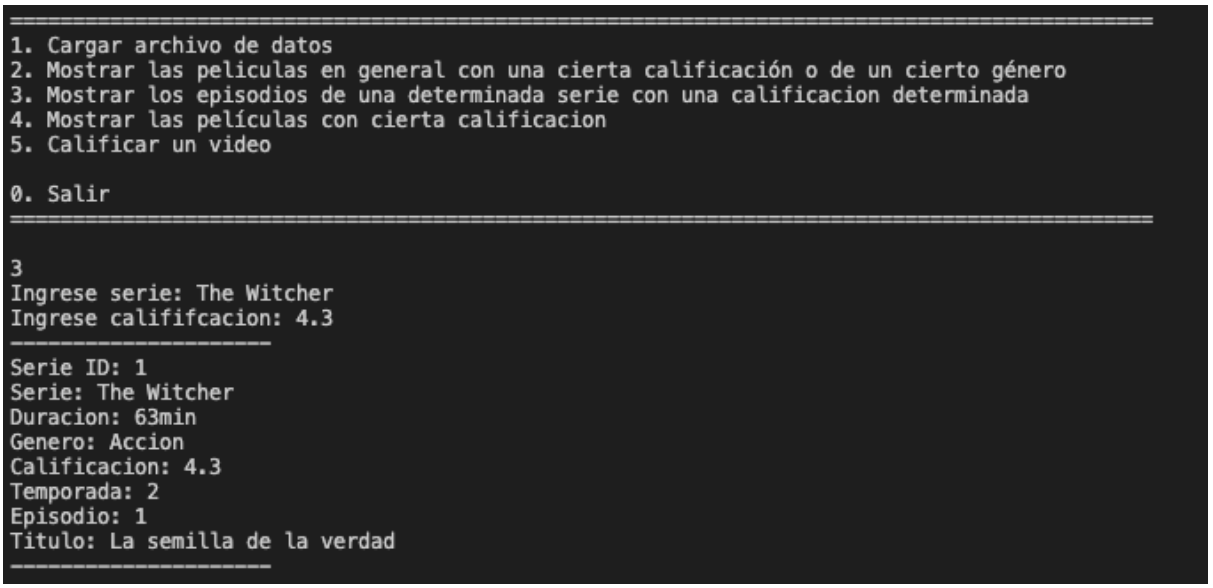
```
SERIES===>10
PELICULAS===>10

================================================================================
1. Cargar archivo de datos
2. Mostrar las peliculas en general con una cierta calificación o de un cierto género
3. Mostrar los episodios de una determinada serie con una calificacion determinada
4. Mostrar las películas con cierta calificacion
5. Calificar un video

0. Salir
================================================================================
```

En el caso de marcar la opción 2 aparecerá otro menú el cual te preguntará si deseas buscar por género o por calificación.

Dependiendo la opción que elijas te solicitará que ingreses lo que estás buscando y te mostrará todas las series y películas que contengan ese parámetro. Luego te devolveré al menú principal nuevamente.



En el caso de marcar la opción 3, el código te pedirá que ingreses el nombre de la serie y la calificación que está buscando, y detectará si dentro de esa serie hay algún episodio con la calificación dada.



Continuando con la simulación la opción 4 te solicitara una calificación y devolverá todas las películas que tengas la misma calificación.

Por último, si ingresamos la quinta opción el sistema nos solicitará el nombre del video, y nos permitirá ingresar el nuevo valor de su calificación.

```
================================================================================
1. Cargar archivo de datos
2. Mostrar las peliculas en general con una cierta calificación o de un cierto género
3. Mostrar los episodios de una determinada serie con una calificacion determinada
4. Mostrar las películas con cierta calificacion
5. Calificar un video

0. Salir
================================================================================

5
Ingrese nombre de video: Seven
Ingrese la calificacion: 1.2
```

Como se puede ver a continuación si buscamos nuevamente esta calificación saldrá la película, aunque en un principio no contará con ese valor.

```
================================================================================
1. Cargar archivo de datos
2. Mostrar las peliculas en general con una cierta calificación o de un cierto género
3. Mostrar los episodios de una determinada serie con una calificacion determinada
4. Mostrar las películas con cierta calificacion
5. Calificar un video

0. Salir
================================================================================

4
Ingrese califcacion: 1.2
----------------------
Pelicula ID: 4
Pelicula: Seven
Duracion: 127min
Genero: Misterio
Calificacion: 1.2
----------------------
```

Cabe destacar que si en algún momento el usuario ingresa algún valor, tanto de calificación, género, nombre u otro, que no está asociado a ningún video, la página devolverá alguno de los siguientes mensajes.

```
3
Ingrese serie: Las locuras del emperador
Ingrese califcacion: 3.5
No se encontro la serie
```

```
==================
2
Ingrese genero: Miedo
----------------------

No se encontraron videos
```

```
4
Ingrese califcacion: 2.1
No se encontro la pelicula
```

Argumentación

A lo largo de este código se tomaron muchas decisiones con respecto a cómo construir el código, principalmente enfocándose en que podría simplificar y optimizar el código en la mejor medida. La decisión de utilizar 4 clases parte de la necesidad de tener una clase padre que hiere sus atributos y sus métodos a otras clases, esto nos permite simplificar mucho el código teniendo una base para nuestras clases. Esto se debe a que al tener unos atributos en común y unos métodos *virtuales*, las clases hijo podían simplemente redefinir la función para sus necesidades específicas implementando el mismo método independientemente del tipo de dato(Series o Películas).

Por otro lado, decidimos que debíamos crear una clase que interactúa con las clases Películas y Series. Pese a poder interactuar directamente desde el *main()*, preferimos crear una clase, ya que de esta manera varias personas podrían usar el código a la vez sin afectar mutuamente sus listas. Pese a que no esté actualmente implementado, hacer que varios usuarios entren y salgan del código sería muy rápido y accesible.

Con respecto a los iteradores como atributos, decidimos que por el uso constante de los mismo definirlos dentro de la clase nos permitirá siempre usar el mismo iterador, en vez de redefinir en cada uno de los métodos. Por otro lado, en un principio planteamos tener una única lista de Streams que se accedera a través de un iterador de apuntadores, pero nos dimos cuenta que esto simplemente nos obligaría a constante construir un comando que detectase si era una película o una serie. Por este motivo separamos las listas y las guardamos dentro por separado, ya que de esta manera solo tendríamos que revisar si son series o películas una única vez.

Otro elemento clave del código fue la sobrecarga del operador. Pese a que simplemente podrían usar continuamente el comando *getline* pareció más lógico dentro del código redefinir la manera en la que se utilizaba el comando “>>”, permitiéndonos utilizarlo como de costumbre. Pese a ser un cambio muy sutil, a la hora de estar escribiendo el código, este cambio simplifica y facilita mucho el entendimiento y la implementación del mismo.

Pese a obtener grandes resultados con respecto a la funcionalidad e implementación de nuestro código, el mismo no es perfecto. Una de sus limitaciones principales parte de la función encargada de insertar los archivos al código. Esta función fue encontrada en internet, página que se encuentra en la bibliografía, y la misma separa los elementos con el uso de “,” ya que así está en los documentos *.csv* que nos proporcionaron. Por este motivo, si algún título de una serie o película contiene el uso de comas, el código lo detectará como otro elemento y dará un error. En el caso que venía en la actividad, pese a no ser la mejor solución, decidimos eliminar todas las comas que había en los nombres para asegurar el funcionamiento adecuado de nuestro código. Este error se podría arreglar solicitando que los archivos *.csv* tengan otro separador como “-” que son menos recurrentes en los títulos de las series y películas.

Conclusión Personal

El objetivo de esta situación problema era desarrollar un programa que funcionase como una plataforma de streaming, que permitiese al usuario ejecutar funciones características de este tipo de páginas. Este código se debió llevar a cabo con el uso de diferentes conceptos relacionados con la programación orientada a objetos. Conceptos que pese a parecer complejos en un principio, posteriormente se vuelven intuitivos y ayudan al programador a optimizar y organizar su código. Lo que en un principio yo pensé que era simplemente una manera de complicarse la vida, resultó siendo todo lo contrario, hoy en día cuando pienso en cualquier tipo de programa automáticamente empiezo a planear que tipo de clases y métodos podría construir para ejecutarlo.

El desarrollo de este reto fue un proceso el cual disfruté mucho, pero esto no quiere decir que no tuve ciertos obstáculos que tuve que superar. En estos 2 semestres estudiando ITC me he dado cuenta que la mejor manera de aprender a programar es programando. En mi opinión personal yo siento que haciendo estos retos es cuando más aprendo y verdaderamente entiendo todos aquellos conceptos que nos enseñan en las clases de forma teórica. En este caso particular, el uso de la sobrecarga de operadores, al igual que la implementación de iteradores fueron conceptos que yo pensé que entendía, pero haciendo el código me di cuenta que no. Este reto me obligó a adentrarme más a estos conceptos y alcanzar un alto grado de entendimiento de los mismos, el cual ahora sí poseo.

Con respecto a las otras limitaciones que tuve al hacer este código, una de las principales fue tener que corregir todos aquellos errores que no son tan claros dentro del código. Un ejemplo de esto fue cuando mi compilador marcaba un error llamado “Illegal instruction 4”, el cual no daba ningún tipo de especificación de porge surgía. Tras investigar acerca de este error parecía que el mismo surge por un error que tenían las MacBooks en ciertos tipos de código. Al principio me frustre, pero decidí que este error no me prohibiría acabar mi código, por lo que trae aún más investigación y reconstruir gran parte del código logré arreglar el error. Al final resultó que un iterador *it*, estaba accediendo a espacio de la RAM que no estaba previamente reservado, por lo que mi compilador mostraba un error para proteger mis datos.

La conclusión general que yo me quedo es que dentro de la programación siempre vamos a aprender información y técnicas nuevas que nos harán sentir que no sabemos nada, y queremos volver a nuestra zona de confort. Pero en realidad debemos adentrarnos más dentro de estos nuevos conceptos hasta alcanzar un entendimiento total de ellos. De esta manera, nosotros logramos expandir nuestra zona de confort dentro de la programación y logramos tener un entendimiento más completo de nuestra carrera. Dentro de la avenida ICT yo siento que lo más importante es ser capaz de no frustrarnos pese a todos los obstáculos o barreras que encontremos en el camino.

Referencia

Guntheroth, K. (2020). What is an 'illegal instruction' error in C++? - quora. Quora. Retrieved May 2, 2022, from https://www.quora.com/What-is-an-illegal-instruction-error-in-C

Raj, A. (2022, February 17). How to read CSV files in C++? Java2Blog. Retrieved May 2, 2022, from https://java2blog.com/read-csv-file-in-cpp/#:~:text=To%20read%20a%20CSV%20file%2C,variable%20as%20its%20 second%20argument.

Seymour, M. (1960, October 1). Illegal instruction 4 on Mac using C++. Stack Overflow. Retrieved May 2, 2022, from https://stackoverflow.com/questions/14261820/illegal-instruction-4-on-mac-using-c

Whitney, T., Robertson, C., Jones, M., & Hogenson, G. (2021, June 12). Overloading the >> operator for your own classes. Microsoft Docs. Retrieved May 2, 2022, from https://docs.microsoft.com/en-us/cpp/standard-library/overloading-the-input-operator-for-your-own-classes?view=msvc-170