Programación 4 - 2021
Laboratorio 0

Consideraciones generales:

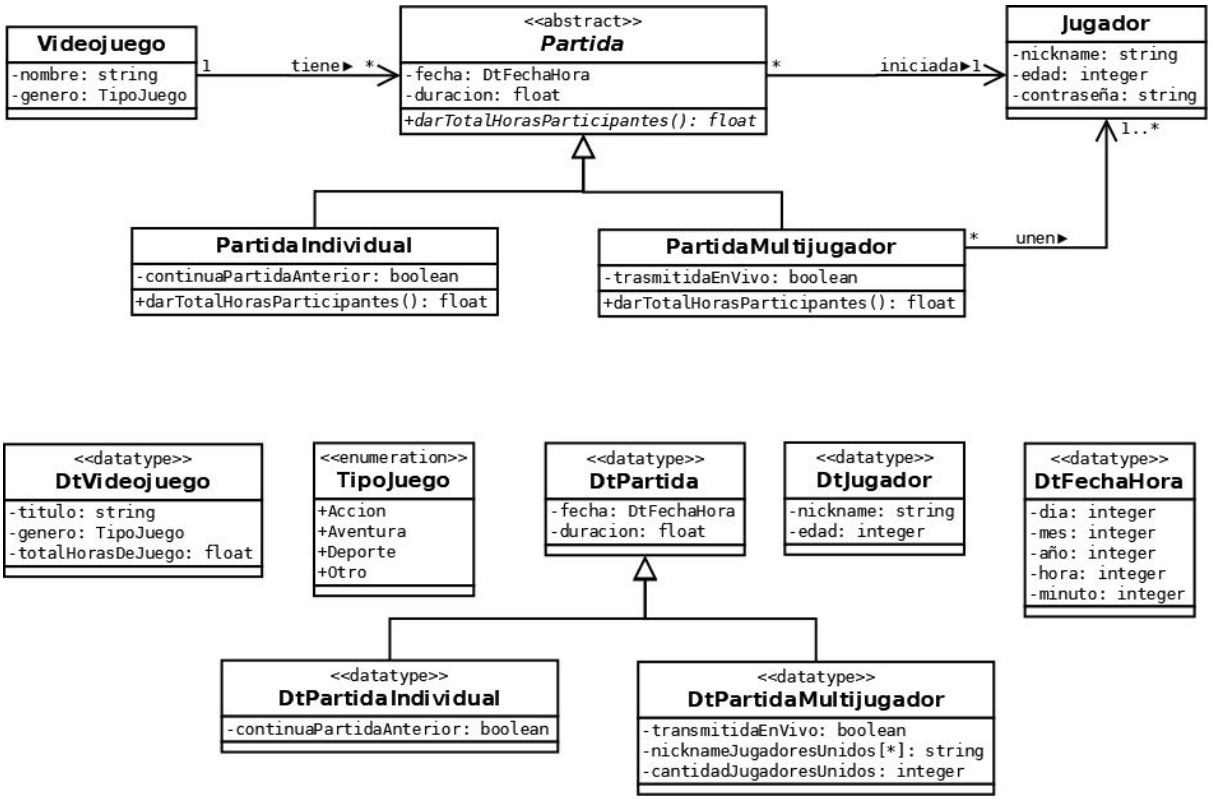
- La entrega podrá realizarse hasta el **lunes 29 de marzo de 2021 a las 15hrs**.
- El código fuente y el archivo Makefile deberán ser entregados mediante el EVA del curso dentro de un archivo con nombre `<número de grupo>_lab0.zip (o tar.gz)`. Dentro del mismo archivo comprimido, se deberán entregar 2 archivos de nombres: `resp_lab0.txt` e `integrantes.txt`. En el primero, se incluirán las respuestas a las preguntas planteadas y las aclaraciones que se consideren necesarias. En el segundo se incluirán los nombres y correos electrónicos institucionales (`@fing.edu.uy`) de cada uno de los miembros del grupo.
- El archivo `Makefile` entregado debe ser independiente de cualquier entorno de desarrollo integrado (IDE, por sus siglas en inglés).
- Las entregas que no cumplan estos requerimientos no serán consideradas. El hecho de no realizar una entrega implica la insuficiencia del laboratorio completo.

Objetivo

En el laboratorio 0 se espera que el estudiante se introduzca a la implementación en el lenguaje C++ (que se usará en el laboratorio de Programación 4) de operaciones básicas y otras construcciones del lenguaje así como el uso/repaso básico del entorno de programación en Linux. También se espera que el estudiante consulte el material disponible en el EVA del curso (ver referencias al final de este documento) y complemente las consultas recurriendo a Internet con espíritu crítico y corroborando, en la medida de lo posible, que las fuentes consultadas sean confiables

Problema

Se desea implementar un prototipo de sistema para aficionados a los videojuegos, el cual debe contar con con un catálogo de videojuegos de diversos géneros, disponibles exclusivamente para los jugadores registrados en el sistema. De cada videojuego se registra su nombre, el cual lo identifica, y su género. Los jugadores se registran ingresando un nickname (seudónimo que lo identifica), edad y contraseña, y pueden iniciar tanto partidas individuales como multijugador. En estas últimas, siempre participa más de un jugador y pueden ser transmitidas en vivo. El sistema almacena las partidas jugadas registrando la fecha en la que se realizan y su duración (en horas). Para cada partida individual interesa saber además si la misma fue una partida nueva o una continuación de una anterior. Por último, es de interés contar con estadísticas que permitan conocer cuáles son los videojuegos preferidos por los usuarios. Para esto, el sistema permite conocer la cantidad total de horas dedicadas por los jugadores a determinado videojuego. En base a la descripción anterior se ha diseñado la estructura de clases que muestra la siguiente figura.



Ejercicio 1

Se pide implementar en C++:

1. Todas las `clases` (incluyendo sus atributos, pseudo-atributos, constructores y destructores, y los getters y setters necesarios para las operaciones que se indican más adelante), `enumerados` y `datatypes` que aparecen en el diagrama. Para las fechas en caso de recibir dd > 31 o dd < 1 o mm > 12 o mm < 1 o aaaa < 1900, se debe lanzar la excepción `std::invalid_argument`. No se deben hacer más que estos controles en la fecha (ej. la fecha 31/2/2000 es válida para esta realidad).

2. Las siguientes operaciones:

a) **`void agregarJugador(string nickname, int edad, string contraseña)`**
Registra un nuevo jugador en el sistema. Si ya existe un jugador registrado con el mismo nickname, se lanza una excepción de tipo `std::invalid_argument`.

b) **`void agregarVideojuego(string nombre, TipoJuego genero)`**
Registra un nuevo videojuego en el sistema. Si ya existe un videojuego registrado con el mismo nombre, se lanza una excepción de tipo `std::invalid_argument`.

c) **`DtJugador** obtenerJugadores(int& cantJugadores)`**

Devuelve un arreglo con información sobre los jugadores registrados en el sistema. El parámetro `cantJugadores` es un parámetro de salida donde se devuelve la cantidad de jugadores devueltos por la operación (corresponde a la cantidad de instancias de `DtJugador` retornadas).

d) **`DtVideojuego** obtenerVideojuegos(int& cantVideojuegos)`**

Devuelve un arreglo con información sobre los videojuegos registrados en el sistema. El parámetro `cantVideojuegos` es un parámetro de salida donde se devuelve la cantidad de jugadores devueltos por la operación (corresponde a la cantidad de instancias de `DtVideojuego` retornadas). El atributo `totalHorasDeJuego` corresponde a la suma de horas jugadas por todos los jugadores del videojuego. Esto debe calcularse sumando las horas devueltas por las invocaciones a *`darTotalHorasParticipantes()`* sobre cada partida del juego. Si la partida es individual, la cantidad de horas devuelta por dicha operación es igual a su duración, mientras que si es multijugador devuelve su duración multiplicada por la cantidad de participantes de la partida.

e) **`DtPartida** obtenerPartidas(string videojuego, int& cantPartidas)`**

Devuelve un arreglo con información de las partidas del videojuego identificado por `videojuego`. El parámetro `cantPartidas` es un parámetro de salida donde se devuelve la cantidad de partidas devueltas por la operación (corresponde a la cantidad de instancias de `DtPartida` retornadas). Entre los datos devueltos para ambos tipos de partida se encuentra la duración. Además, entre los específicos de cada partida individual se encuentra si la misma es o no continuación de una partida anterior, mientras que para cada partida multijugador se indica si la misma es transmitida en vivo y la cantidad total de jugadores que se unen a la misma junto con sus nicknames. Si no existe un videojuego registrado con el nombre enviado, se lanza una excepción de tipo `std::invalid_argument`.

f) **`void iniciarPartida(string nickname, string videojuego, DtPartida* datos)`**

Registra una partida individual o multijugador del videojuego identificado por `videojuego`, iniciada por el jugador identificado por `nickname`. El parámetro de entrada `datos` contiene la información completa de la partida. Entre los datos comunes a ambos tipos de partida se encuentra la duración. Además, si `datos` es una instancia de `DtPartidaIndividual` contiene si la partida es o no una continuación de una partida anterior, mientras que si es un instancia de `DtPartidaMultijugador`, indica si la partida es transmitida en vivo y la cantidad total de jugadores que se unen a la misma junto con sus nicknames. La partida se da de alta con la fecha del

sistema al momento del registro. Si no existe un jugador o videojuego registrado con el nickname y nombre enviados, se lanza una excepción de tipo `std::invalid_argument`.

Notas:

- A los efectos de este laboratorio, considere que el sistema maneja un conjunto acotado de jugadores, videojuegos, y partidas por videojuego, donde la cota se define por las constantes MAX_JUGADORES, MAX_VIDEOJUEGOS y MAX_PARTIDAS, respectivamente.
- Puede implementar operaciones auxiliares en las clases dadas en el diagrama si considera que le facilitan para la resolución de las operaciones pedidas.
- Se puede utilizar el tipo std::string para implementar los atributos de tipo string.
- En este laboratorio no se pueden utilizar estructuras de datos de la biblioteca STL, tales como vector, set, map, etc.
- Se sugiere crear una clase auxiliar llamada `Sistema` que implemente las operaciones pedidas y almacene el conjunto de jugadores y videojuegos.

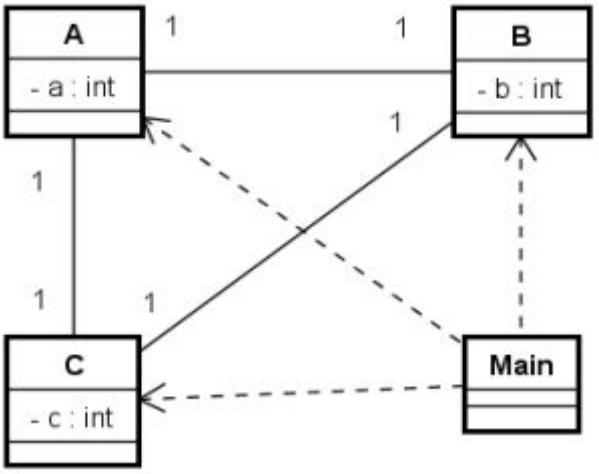
3. Sobrecargar el operador de inserción de flujo (ej. <<) en un objeto de tipo `std::ostream`. Este operador debe "imprimir" todos los datos de las distintos datatypes de `DtPartida` (`DtPartidaIndividual`, `DtPartidaMultijugador`), siguiendo un formato similar al ejemplo:

Tipo Partida: *Individual/Multijugador*
Fecha partida: *dd/mm/aaaa*
Duración partida : *hh hs*
/* Solo para partidas individuales */
Continuación de una partida anterior: *Si/No*
/* Solo para partidas multijugador*/
Transmitida en vivo: *Si/No*
Cantidad jugadores unidos a la partida: *N*
Jugadores unidos a la partida: *nickname1, nickname2, ...*

4. Implementar un menú sencillo e interactivo con el usuario para probar las funcionalidades requeridas en los puntos anteriores. Al ejecutar el programa debe primero pedir el ingreso de un número especificando la acción a realizar, y luego pedir los datos necesarios para cada operación.

Ejercicio 2

En este ejercicio se busca que el estudiante se familiarice con la problemática de dependencias circulares en C++ [3]. Para ello considere la estructura dada por el siguiente diagrama.



Se pide

1. Implementar en C++ y compilar la estructura dada en el diagrama. Incluya en cada una de las clases A, B y C una operación que despliegue el texto de un mensaje. Implemente un main que defina un objeto de cada una de las clases e invoque a la operación de cada uno de ellos.

2. Responder las siguientes preguntas:
    a. ¿Cuáles son las dependencias circulares que fueron necesarias solucionar para que el programa compile?
    b. ¿Qué es una *forward declaration*?

Referencias

[1] Eva Programación 4

[2] Eva Programación 4, Material > Bibliografía y material complementario

[3] Eva Programación 4, Material > Material adicional

[4] Eva Programación 4, Unidad de Recursos Informáticos de la Facultad (Configuraciones y Salas estudiantes Linux)

- https://www.fing.edu.uy/sysadmin/configuracioneshttps://www.fing.edu.uy/sysadmin/gestion-estudiantil