EJERCICIOS PREPARACIÓN EVALUACIÓN 3 - LP

EJERCICIO 1

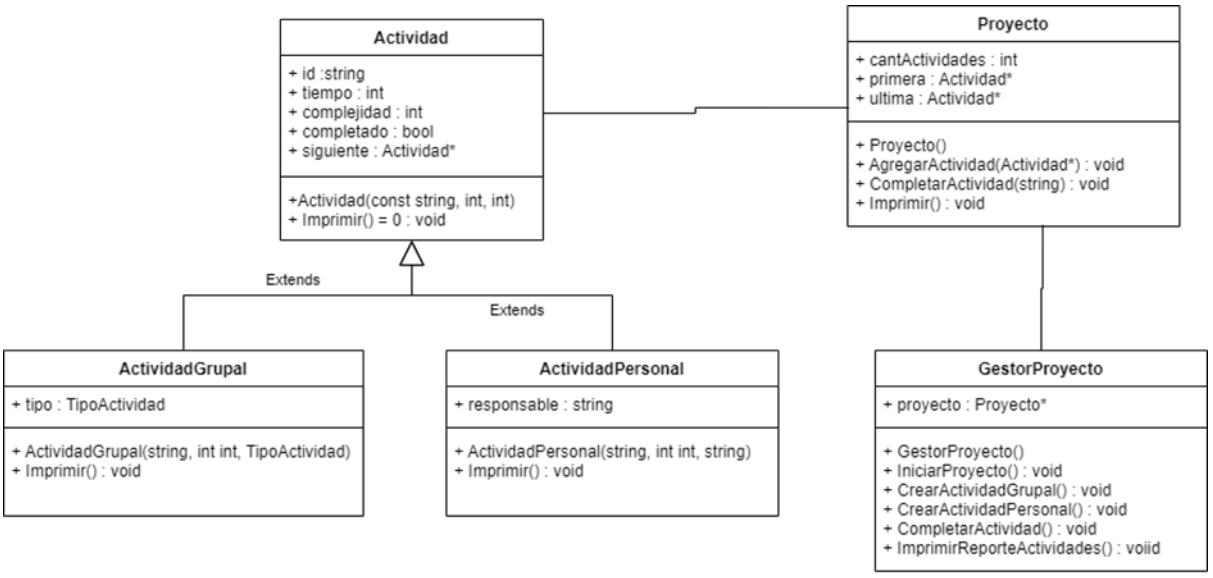
Se le pide implementar un proyecto que permita gestionar actividades de un proyecto de desarrollo de software. Estas actividades pueden ser de dos tipos: Grupal o Persona.

Las actividades en general deberán tener los siguientes campos: id (string), tiempo (int), complejidad (int) y completado (bool). Las actividades personales además tendrán el campo responsable (string). Las actividades grupales además de los campos generales tendrán el campo TipoActividad.

Este campo TipoActividad debe ser un enum class que tendrá los siguientes valores: Programacion, Planificacion, Configuracion, e Investigacion.

Se le pide realizar la implementación de la clase Proyecto que representará a una lista enlazada de actividades. Tomar en cuenta que una ActividadPersonal o una ActividadGrupal heredan de una clase Actividad. Además, cada una de las clases hijas deberá sobreescribir el método abstracto Imprimir y tener su propia implementación.

El diagrama UML de lo que se le pide implementar es el siguiente:



Debe completer la implementación del código (archivo proyectos.cpp) que se le envía:

1. Completar la implementación del método IniciarProyecto de la clase GestorProyecto. Este método debe instanciar una referencia a Proyecto y almacenarla.

2. Completar la implementación del método CrearActividadGrupal. Este método debe crear una actividad grupal y agregarla en el proyecto (ListaEnlazada).
3. Completar la implementación del método CrearActividadIndividual. Este método debe crear una actividad personal y agregarla en el proyecto (ListaEnlazada).
4. Completar la implementación del método CompletarActividad de la clase GestorProyecto. Esta clase debe pedir el código de una actividad para buscarla en la lista de actividades del proyecto y marcarla como completado.
5. El método Imprimir de la clase Actividad está implementado en la clase Padre. Se requiere convertir este método a abstracto y luego implementar sobreescrituras en sus hijos (ActividadPersonal y Actividad Grupal).
    - Por ejemplo: si ha registrado 2 actividades (una grupal y una personal), el reporte debería mostrar lo siguiente:

```
============================

REPORTE DE AVANCE DE PROYECTO

============================

(1)   G001     5       3     Investigacion   -
(2)     P001   2       1       Nacho   X
```

Como se ve, por actividad se imprime un número consecutivo, el id de la actividad, el tiempo en días, la complejidad y en el caso que sea una actividad grupal, se imprimirá su TipoActividad, mientras que, si era una actividad personal, se imprimirá el responsable. La X significa que la actividad ha sido marcada como completada. El “-” significa que no está completada.

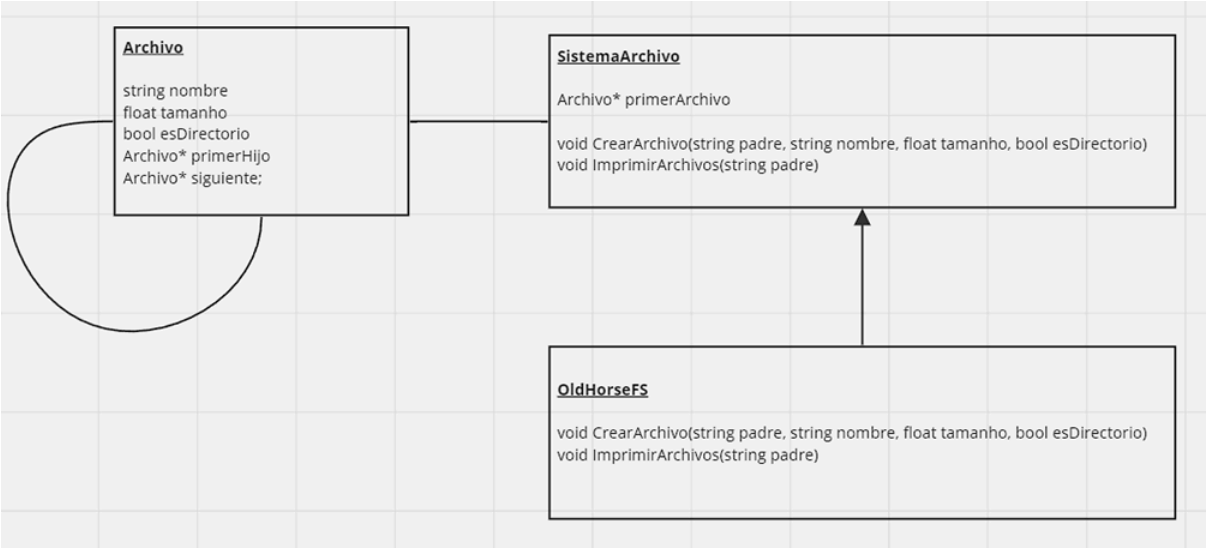
6. Crear correctamente el archivo Makefile de compilación.

EJERCICIO 2

Los sistemas de archivos (o *filesystem* en inglés) son un módulo importante que los sistemas operativos implementan y cuya función es gestionar todas las funcionalidades realizadas con los archivos y directorios.

Un directorio es un archivo especial, dado que puede tener archivos “hijos”.

Para el presente laboratorio, deberá de implementar un sistema de archivos llamados OldHorseFS, tomando en cuenta el siguiente diagrama de clase.



Como puede ver, la clase archivo es un elemento de una lista enlazada, así como también apunta a una nueva lista enlazada de archivos (sus archivos hijos).

Con el fin de que nuestro software pueda funcionar con distintos sistemas de archivos, debe de implementar una clase padre llamada SistemaArchivos que define 2 métodos abstractos (polimórficos): CrearArchivo e ImprimirArchivo. Estos métodos abstractos deben ser implementados en la subclase llamada OldHorseFS, donde estará la implementación concreta de este sistema de archivos.

PREGUNTA 0

Crear las clases y los métodos según lo que se le pide. Estos deberán ser creados en una unidad de compilación llamada filesystems. Nota: No olvidarse de utilizar también el namespace llamado filesystem.

PREGUNTA 1

Crear una unidad de compilación llamada pregunta1 que deberá contener una función llamada Pregunta1 que se encargue de leer los datos del sistema de archivo del archivo de texto llamado input.txt, según el siguiente ejemplo:

```
10

archivo1 archivo2 archivo3 archivo4 archivo5 archivo6 archivo7 archivo8 archivo9 archivo10

a d a a a a d a a a

2.3 10.2 1.1 3.0 4.5 6.2 7.7 3.1 5.3 1.0
```

En la primera línea se tiene la cantidad de archivos que tendrá el primer nivel del sistema de archivos. Luego se leerán los nombres de los archivos, seguidamente se leerán el tipo de archivo (a: archivo, d:directorio), para posteriormente finalizar leyendo el tamaño de cada archivo.

Para implementar esto, debe utilizar la función CrearArchivo de una subclase de SistemaArchivos.

Nota:

- El árbol de archivos solo podrá ser de dos niveles.

- Tomar en cuenta que la función CrearArchivo puede recibir como argumento de entrada el nombre del archivo padre donde se desea crear el archivo. Caso que se reciba un string vacío (“”), deberá crear el archivo en la raíz.

PREGUNTA 2

Crear una unidad de compilación llamada pregunta2 que deberá contener una función llamada Pregunta2 que se encargue de Imprimir los archivos del sistema de archivos creado en la pregunta 1, según el siguiente formato:

archivo1

-hijo1

-hijo2

archivo2

archivo3

-hijo1

Nota:

- El árbol de archivos solo podrá ser de dos niveles.

- Tomar en cuenta que la función ImprimirArchivo puede recibir como argumento de entrada el nombre del archivo padre desde donde se desea listar los hijos. Caso que se reciba un string vacío (“”), deberá listar todos los archivos del sistema de archivos.

PREGUNTA 3

Crear una unidad de compilación llamada pregunta3 que deberá contener una función llamada Pregunta3 que se encargue de eliminar un archivo del sistema de archivos. Esta función debe recibir como argumento de entrada un puntero a un sistema de archivos, el nombre del archivo padre del que quiera eliminarse el archivo, así como el del archivo a eliminar En caso de que el nombre del padre llegue como una cadena vacía, debe eliminar directamente el archivo con el nombre enviado.

Ejemplo:

Pregunta3(“archivo1”, “hijo3”, sistema) -> Eliminará el hijo llamado hijo3 del padre llamado padre1.

Pregunta3(””, “archivo2”, sistema) -> Eliminará el archivo llamado archivo2 del primer nivel.

Tomar en cuenta:

- En caso de que quiera eliminar hijos de un archivo que no es un directorio, deberá de imprimir un mensaje de error: “El archivo que quiere eliminar no se encuentra en un directorio valido”.

- En caso de que quiera eliminar un archivo que no existe, deberá de imprimir un mensaje de error: “El archivo que quiere eliminar no existe”.

- Luego de eliminar un archivo, debe de liberarse la memoria ocupada por este.

PREGUNTA 4

Crear una unidad de compilación llamada pregunta4 que deberá contener una función llamada Pregunta4 que se encargue de limpiar todo el sistema de archivos, esto es, eliminar todos los archivos del sistema de archivos. Esta función debe recibir como argumento de entrada un puntero a un sistema de archivos.

Tomar en cuenta:

- Luego de eliminar un archivo, debe de liberarse la memoria ocupada por este.

- Eliminar TODOS los archivos (de nivel 1 y 2).

PREGUNTA 5

Se le pide implementar un nuevo sistema de archivo llamado NewHorseFS que herede de la clase SistemaArchivos. Debe de crear una nueva unidad de compilación llamada pregunta5 que implemente una función llamada Pregunta5 que encargue de agregar un archivo en este nuevo sistema de archivos. Para esto, esta función recibirá como argumentos de entrada el nombre del padre, el nombre del archivo a crear y un puntero a un sistema de archivos. La diferencia de la implementación del otro sistema de archivos es que ahora los archivos deben de guardarse siguiendo un esquema de LIFO (cola), esto es que el primer elemento de la lista de archivos sea el último que entró.

Tomar en cuenta:

- La función Pregunta5 solo debe de trabajar con el sistema de archivos NewHorseFS.

PREGUNTA 6

También debe implementar el método ImprimirArchivos del nuevo sistema de archivos NewHorseFS. Para esto, debe de crear una nueva unidad de compilación llamada pregunta6 que implemente una función llamada Pregunta6 que utilice esta función llamada ImprimirArchivos de la nueva clase NewHorseFS.

Esta función ImprimirArchivos de NewHorseFS se diferencia de la otra clase en que ahora no solo debe pintar el nombre del archivo, si no también el tamaño de este según la siguiente lógica:

- Si se está imprimiendo un archivo simple, debe imprimir su tamaño que se encuentra almacenado.

- Si se está imprimiendo un archivo que es un directorio, su tamaño será el tamaño que tiene almacenado, sumado con el tamaño de todos sus hijo.

Ejemplo:

archivo1 (10)

-hijo1 (5)

-hijo2 (2)

archivo2 (7)

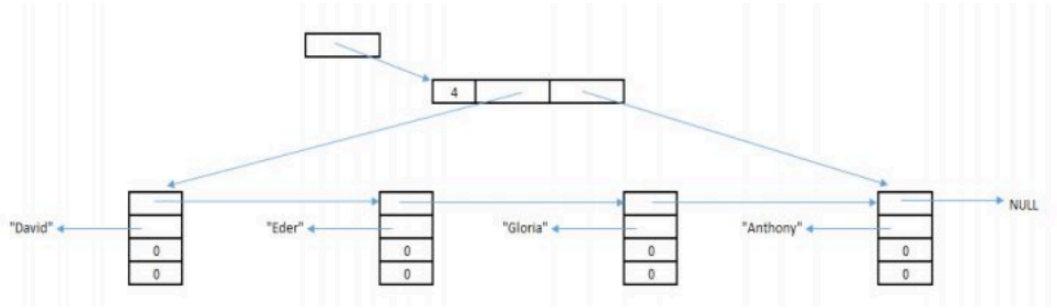
archivo3 (5)

-hijo1 (3)

Como ve, el archivo “archivo1” tiene un tamaño de 3, y como sus hijos tienen un tamaño de 5 y 2 respectivamente, se imprime como nuevo tamaño 10.

EJERCICIO 3

El juego del 21 (BlackJack) es un juego de cartas muy conocido que incluso tiene su propia película. Nosotros realizaremos una adaptación de este juego y lo llamaremos El juego del 21
de LP (BlackJack LP).
Este juego se realiza en una mesa con un número determinado de jugadores, esta mesa de jugadores la representaremos mediante una lista simplemente enlazada de la siguiente manera:



Cada elemento de la Lista es un Jugador y este jugador tiene la siguiente información: nombre, cantidad de partidas ganadas, suma de cartas y un enlace al siguiente jugador. Siguiendo con el juego, además de los jugadores, también participa un Crupier y es este Crupier quien juega contra todos los jugadores de la mesa. El Crupier NO SE representará en la lista.

El Crupier tiene un grupo de cartas al cual llamaremos “Deck” y será de este Deck de donde obtenga las cartas para jugar las partidas. Para generar el Deck se debe considerar que se utilizarán 6 barajas de cartas, es decir 78 cartas.

Una partida del juego inicia con el Crupier repartiendo UNA CARTA del Deck a todos los jugadores, empezando por el jugador que está al inicio de la lista hasta el último. Obviamente el crupier al repartir una carta a un jugador, la siguiente carta que sacará será la que continúa en el Deck y utilizará esta lógica siempre que saque una carta del deck.

Luego de repartirles una carta a todos los jugadores, el crupier también reparte una carta para él.

A continuación empieza a jugar la partida el Crupier con cada jugador. Primero inicia con el jugador que está al inicio de la Lista y le reparte cartas hasta que la suma de todas las cartas que tiene el jugador (ojo incluye la primera que ya le repartió) sea mayor o igual que 17.

Cuando se cumpla esa condición, el Crupier deja de repartirle cartas al jugador y va con el siguiente jugador y repite la misma acción. Esto se repite por cada jugador que está en la mesa (Lista).

Una vez que ha terminado con todos los jugadores, el crupier reparte cartas para él bajo la misma lógica de los jugadores, es decir, hasta que la suma de cartas que tiene el Crupier sea mayor o igual que 17.

Luego que el Crupier termino de repartir cartas, comienza a verificar que jugadores le ganaron a él o que jugadores perdieron o empataron contra él. Las reglas para determinar quién gana o pierde son las siguientes y se dan en este orden:
● Si la suma de cartas del Crupier es mayor que 21, entonces todos los jugadores ganaron la partida.
● Si la suma de cartas de un jugador es mayor que 21, ese jugador pierde la partida.
● Si la suma de cartas del Crupier es mayor que 17 y menor o igual que 21 y es mayor o igual que la suma de cartas del jugador, entonces el jugador pierde la partida.
● Si la suma de cartas del Crupier es mayor que 17 y menor o igual que 21 y es menor que la suma de cartas del jugador, entonces el jugador gana la partida y aumenta su cantidad de partidas ganadas.

Una vez que se terminó de verificar el resultado de todos los jugadores, la suma de cartas de los jugadores y del crupier vuelve a 0 y se inicia otra partida, siguiendo los mismos pasos explicados anteriormente.

Se debe repetir todas las partidas posibles que se puedan realizar hasta que la cantidad de cartas que queden en el deck sea menor o igual a 20. Ojo, si ya se inició una partida, esta debe terminar y recién después verificar la cantidad de cartas restantes para ver si se continúa con otra partida o ya se termina de jugar.

Un ejemplo de la acción jugar sería el siguiente:

Partida 1

Jugador David tiene 19
Crupier tiene 20
El resultado es: Crupier gana o quedaron empates.

Jugador Anita tiene 22
Crupier tiene 20
El resultado es: Crupier gana o quedaron empates.

Partida 2

Jugador David tiene 19
Crupier tiene 24
El resultado es: Jugador David gana.

Jugador Anita tiene 23
Crupier tiene 24
El resultado es: Crupier gana o quedaron empates.

Fin de la mesa

Se le pide implementar el juego en el lenguaje de programación C++.

a) Definir en C++ las clases a utilizar de acuerdo a lo especificado en el diagrama.
b) Un metodo GenerarDeck de una clase Deck que permita dentro de esta función, generar de manera aleatoria las cartas del deck que utilizará el crupier para repartir las cartas. OJO, debe tener en consideración que solo se manejarán cartas del 1 a 13 y se utilizarán 6 barajas, es decir 78 cartas. Con esta condición, solo se permitirá como máximo que los números del 1 al 13 se repitan 6 veces en el deck.
c) Una función AgregarJugador de una clase Juego que permita agregar un jugador al final de la mesa (al final de la Lista).
d) Una función MostrarJugadorMayor de una clase Juego que imprima la información asociada al jugador de la mesa (Lista) que tenga más partidas ganadas, en caso más de un jugador sea el que tenga más partidas ganadas se deberá mostrar la información de todos ellos.
e) Una función EliminarJugador de una clase Juego que permita eliminar un jugador de la mesa (Lista). Para ello dentro de esta función debe solicitar al usuario ingrese el nombre del jugador a eliminar. En caso el jugador no exista dentro de la mesa, se deberá imprimir el mensaje “El jugador no existe”.
f) Una función MostraMenu de la clase Juego que permita mostrar el menú de opciones que existe en el juego. Esta función solo muestra el menú, no debe realizar ninguna lectura de opción dentro de esta función.
g) Una función Jugar de la clase Juego que permita implementar toda la lógica de jugar partidas para una mesa de acuerdo a lo descrito en el enunciado.
h) Una función main que permita mostrar un menú de opciones que llame a las funciones anteriores. Tenga en cuenta que debe tener una opción para salir del programa y que el programa se debe ejecutar hasta que se ingrese esta opción de salida. Tenga en cuenta que debe validar que para iniciar el juego (opción Jugar), el deck debe estar generado y debe existir al menos un jugador en la mesa (Lista).

EJERCICIO 4

Crear una clase PartidaAjedrez que representa una partida de ajedrez y contenga los siguientes métodos:

- verTurno: muestra en la consola que color de piezas se deben mover en este turno.
- mostrarPartida: muestra en la consola el tablero y las fichas en su respectiva ubicación. Imprimir también los ejes del tablero.
- mostrarPosiblesMovimientos: muestra en consola las casillas a las que se podría mover la pieza de la casilla indicada. En caso no haya una pieza en dicha casilla indicarlo en la consola.
- moverFicha: recibe de parámetros las coordenadas de la ficha a mover y las coordenadas del destino. Mueve la ficha a la casilla indicada en caso sea posible, en caso contrario muestra en la consola que la jugada es incorrecta.
- reiniciarPartida: devuelve todos los valores a su estado inicial.

Sugerencia: crear una clase abstracta que representa a las piezas y luego crear clases hija a partir de esta para cada una de las piezas en particular (peón, caballo, etc)

Luego, modificar la clase PartidaAjedrez para que muestre en consola el resultado de la partida cuando esta acabe. Puede modificar las clases, métodos y atributos que considere necesarios.