UTN – FRMDP Mar del Plata ***TUP - Laboratorio 2*** ***Trabajo Práctico Final***	***Integrantes del grupo***	***Nota***

Introducción

Con el propósito principal de integrar contenidos aprendidos en la materia laboratorio 2, sumando lo trabajado en laboratorio 1 hemos planteado la siguiente problemática:

- Codificar un sistema de Logueo que gestione la estructura **usuario**.
- Administrar un archivo de Canciones (Alta, Baja, Modificación, Consulta y Listados)
- Administrar una playlist por cada usuario en un archivo independiente.

Fundamentación

El valor pedagógico de la propuesta se apoya en el aprendizaje colaborativo **(se formarán grupos de 2 o 3 alumnos)** y la integración de contenidos de otras asignaturas a partir del desarrollo de un proyecto de software. Para que este tipo de proyectos sea más exitoso, deben llevarse a cabo desde un enfoque que facilite alcanzar los Objetivos de Aprendizaje propuestos.
Una de las ideas centrales es desarrollar competencias profesionales y preparar al futuro programador para el mundo laboral y el trabajo en equipo.
En un ambiente de aprendizaje colaborativo, los estudiantes:

- Construyen conocimiento y en lugar de recibirlos en forma pasiva;
- Se involucran y se comprometen directamente con el descubrimiento de nuevo conocimiento;
- Se exponen a puntos de vista alternativos e ideas contrapuestas, de forma tal que pueden sacar sus propias conclusiones y así transformar conocimientos y experiencias previas y de esta manera comprender con mayor profundidad;
- Transfieren conocimientos y habilidades a nuevas situaciones o circunstancias;
- Se responsabilizan y apropian tanto de su aprendizaje continuo de contenidos curriculares, como del desarrollo propio de competencias;
- Los estudiantes colaboran para el aprendizaje del grupo y el grupo colabora en el aprendizaje individual de estos.

Objetivos

De aprendizaje:

- Gestión de archivos binarios.
- Recursividad.
- Listas enlazadas, simples o dobles
- Arboles binarios.
- Estructura de datos compuestos. (arreglo de listas, listas de listas, listas de árboles, etc.)
- Trabajar en forma colaborativa.

Metodológicos:

- Ser capaces de trabajar en un proyecto complejo, aplicando técnicas de desarrollo de software.
- Lograr integrar contenidos de otras asignaturas.
- El grupo deberá ir mostrando el avance sobre el trabajo en clase.

Modo de Evaluación del Trabajo Práctico

- Se establece el desarrollo de un trabajo práctico final, brindando una fecha límite de entrega del mismo: **Según planificación de cada comisión**
- Si el sistema está codificado en su totalidad y funciona correctamente, se considerará aprobado con una nota mínima de 6.
- Si el sistema no está codificado en su totalidad (como mínimo un 60 % en cada inciso), se considerará desaprobado y el grupo presentará la versión final en la fecha de recuperatorio.
- En la fecha de recuperatorio deberá cumplir las pautas mínimas establecidas precedentemente para la aprobación de la instancia recuperatoria. Vale aclarar que no puede aprobar de manera directa.
- Es obligatorio la presentación de este trabajo para aprobar la materia.

Pautas Generales

El sistema deberá permitir gestión de Usuarios, Canciones y Canciones escuchadas (*similar a Spotify*) persistiendo la información en archivos binarios.

Como metodología de trabajo, se requiere crear un documento de texto en Google Drive que será compartido a todos los miembros del grupo (y también al equipo docente, publicando el link vía campus virtual en el foro correspondiente), con el fin de plasmar los avances del proyecto de forma de construir la siguiente documentación a entregar:

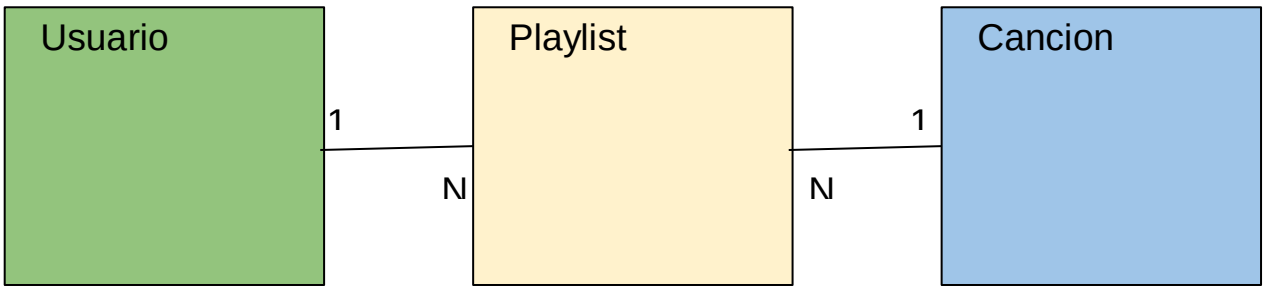
Informe final - Documentación a entregar: [10 puntos]

- Diario de trabajo: Día a día qué actividades se desarrollaron y el responsable de cada una.
- Matriz de soluciones: Que problema tuvieron y cómo lo resolvieron.
- Manual de usuario: Breve explicación de cómo funciona el sistema, pueden usar imágenes, videos, presentaciones, etc.
- Diagrama de estructuras: Esquema de las estructuras utilizadas y sus relaciones.

Asimismo, deberán crear un tablero en www.trello.com para distribuir las tareas entre los integrantes del grupo y trabajar de forma organizada. A medida que avancen con el desarrollo del trabajo, realizarán capturas de pantalla y las adjuntarán al Diario de trabajo. Deberán compartir el tablero con el equipo docente, publicando el link vía campus virtual en el foro correspondiente.

Se pide únicamente la impresión del presente enunciado para entregar al equipo docente, el resto del material puede entregarse de forma digital.

Para la persistencia de datos en los archivos (“usuarios.dat”, “canciones.dat” y “playlist.dat”) utilizaremos las siguientes estructuras de datos:



Los registros del archivo binario “***playlist”***, se construyen a partir de los datos más representativos del Usuario (idUsuario), de la Canción (idCancion) y de un idPlaylist (campo autoincremental) para identificar el número de registro. Se establece una relación de 1 a N entre los archivos, donde un Usuario puede haber escuchado muchas canciones y una canción puede haber sido escuchada muchas veces.

Detalle de estructuras y funcionalidad básicas:

Integración y/o adaptación de funciones del TP Final de Laboratorio 1: [10 puntos]

Estructura de Canción

```
typedef struct {
   int idCancion;
   char titulo[30];
   char artista[20];
   int duracion;
   char album[20];
   int anio;
   char genero[20];
   char comentario[100];
   int eliminado;  // indica 1 o 0 si la canción fue eliminada
} stCancion;
```

Importar y adaptar las funciones de Alta, Baja, Modificación, Consulta y Listados de Canciones que ya fueron desarrollados en el TP Final de Laboratorio 1.

Estructura de Usuario

```
typedef struct
{
    int idUsuario;
    char nombreUsuario[30];
    char pass[20];
    int anioNacimiento;
    char genero;
    char pais[20];
    int eliminado;   // indica 1 o 0 si el cliente fue eliminado
} stUsuario;
```

Importar y adaptar las funciones de Alta, Baja, Modificación, Consulta y Listados de Usuarios que ya fueron desarrollados en el TP Final de Laboratorio 1.

Lista de Canciones (lista simple) [15 puntos]

```
typedef struct
{
    stCancion c;
    struct nodoListaCancion * sig;
} nodoListaCancion;
```

Deberán codificar todas las funciones necesarias para administrar el TDA Lista Simple, a saber (como mínimo):

inicLista()
crearNodoLista()
agregarAlPrincipio()
agregarAlFinal()
agregarEnOrdenPorNombreDeCancion()
mostrarLista() // modularizar
borrarNodoPorIdCancion()

Árbol de Canciones [20 puntos]

```
typedef struct
{
    stCancion c;
    struct nodoArbolCancion * izq;
    struct nodoArbolCancion * der;
} nodoArbolCancion;
```

Deberán codificar todas las funciones necesarias para administrar el TDA Árbol, a saber (como mínimo):

inicArbol ()
crearNodoArbol ()
insertarNodoArbol (ordenado por idCancion)
mostrarArbol (son tres funciones, recorriendo inOrder, postOrder, preOrder) // modularizar
borrarUnNodoArbol (buscarlo por idCancion)
buscarCancion (por idCancion)

cargarArbolDesdeArchivo(): Al inicio del sistema, deberán cargar todas las canciones del archivo, sobre un árbol binario ordenado por idCancion, de forma tal que las búsquedas se realicen de forma más eficiente.

Tenga en cuenta que, seguramente, su archivo de canciones está ordenado de forma creciente por idCancion y que si realiza un recorrido secuencial del archivo, la inserción en el árbol no se realizará de una forma óptima. Desarrolle una función (o varias) que logren realizar la inserción en el árbol, logrando que este quede lo más balanceado posible.

Estructura de Arreglo de Usuarios (Arreglo de listas) [20 puntos]

```
typedef struct
{
    stUsuario usr;
    nodoListaCancion * listaCanciones;
} stCelda;
```

En el programa principal (main) se definirá un Arreglo de Listas de dimensión justa, utilizando la función malloc(), para lo cual deberán determinar la cantidad de usuarios activos en tiempo de ejecución.

Este arreglo se cargará de forma automática al iniciar el sistema, con todos los usuarios activos y sus canciones escuchadas. El trabajo en el sistema se realizará sobre esta estructura y, por ejemplo, al momento de escuchar una canción, se realizarán las acciones necesarias para actualizar la información sobre el ADL. Luego, una vez que el usuario quiera desloguearse o antes de cerrar el programa, se persistirán estos los datos en los archivos.

Deberán codificar todas las funciones necesarias para administrar el TDA Arreglo de Listas, a saber (como mínimo):

agregarUsuario() // crea un nuevo usuario en el arreglo
buscarUsuario() // busca un usuario por idUsuario y retorna la posición que ocupa en el arreglo
mostrarUsuarios() // muestra todo el arreglo de listas, cada usuario con sus canciones escuchadas
agregarCancionToUsuario() // agrega una Cancion al Usuario correspondiente
limpiarArregloDeListas() // esta función "vacía" todo el arreglo de listas, dejando la estructura preparada para volver a trabajar
persistirCancionesEscuchadas() // esta función realizará la persistencia de todas las canciones escuchadas en el archivo correspondiente

Estructura de canciones escuchadas por cada Usuario [15 puntos]

```
typedef struct
{
    int idPlaylist;
    int idUsuario;
    int idCancion;
} stPlaylist;
```

Esta estructura da forma al archivo de canciones escuchadas por cada usuario, en cada registro se almacena el id del usuario, el id de la canción y un id autoincremental para contabilizar los registros.

A partir de esta información, se carga el arreglo de listas, buscando los datos del usuario en el archivo y los datos de la canción en el árbol de canciones. Para hacer esto, deberá desarrollar una serie de funciones que sean invocadas por la función **pasarDeArchivoPlaylistToADL().**

Asimismo, deberá desarrollar las funciones necesarias para hacer el trabajo inverso. A partir del arreglo de listas que se va cargando y actualizando en memoria, realizar la persistencia de los datos en el archivo de canciones escuchadas recorriendo el ADL y tomando los datos allí almacenados.
actualizarCancionesEscuchadas().

Codificar las funciones necesarias para persistir esta estructura en un archivo binario y las que necesite para la interacción con el sistema.

La función principal - Main() y menús: [10 puntos]

El sistema deberá contar con una presentación amigable con el usuario, construir menús de acceso a las diferentes estructuras y funcionalidades del sistema, y **de manera directa o indirecta, permitir probar todas las funciones desarrolladas**.

El desarrollo del sistema deberá ser ordenado, **identificando con comentarios cada una de las funciones realizadas, explicando brevemente lo que realizan**. **Se tendrá en cuenta, al momento de evaluar, la prolijidad del código y la organización de los módulos. Se recomienda agrupar los mismos por funcionalidad.**

El sistema tendrá que proporcionar el acceso a las diferentes funcionalidades mínimas, aunque se deja a libre desarrollo del grupo la forma de construir los menús:

Menú principal

1. Ingreso con User y Pass para administradores
2. Ingreso con User y Pass
3. Registrarse

1- Ingreso con User y Pass Solo administradores:

Sub-Menú Administración Usuarios:

- **Alta**: Una vez completado el formulario de alta se valida que no exista el usuario. Si no existe, se guardan los datos en los archivos correspondientes.
- **Baja**: Modificar el campo eliminado en la estructura y guardarlo.
- **Modificación**: En modificación se ingresa el nro de usuario. Si no existe, se muestra mensaje de error; si existe lo muestra y se ve una forma de poder modificar los campos. Ej: mostrar los campos con un número de orden y pedir el ingreso del nro de camp a modificar.
- **Consulta**: Para la consulta la metodología sería similar a listados.
- **Listados**: Para los listados se abre el archivo y se carga en un array y se ordena por nombre de usuario antes de mostrarlo.

Sub-Menu Administración de Canciones:

- **Alta**: Una vez completado el formulario de alta se persiste la canción en el archivo.
- **Baja**: Modificar el campo eliminado en la estructura y guardarlo.
- **Modificación**: En modificación se ingresa el nro de canción. Si no existe, se muestra mensaje de error; si existe se ve una forma de poder modificar los campos. Por ejemplo: mostrar los campos con un número de orden y pedir el ingreso del nro de camp a modificar.
- **Consulta**: Para la consulta la metodología sería similar a listados.
- **Listados**: Para los listados se abre el archivo y se carga en un array. Mostrarlos según los siguientes criterios:
    1 Por título (ordenado por selección)
    2 Por género (ordenado por inserción)

2- Ingreso Con User y Pass: Esta pantalla pide que se ingrese Usuario y Contraseña, si el usuario existe comprueba que la contraseña sea correcta y muestra los datos del cliente. Si el usuario no existe << muestra mensaje >> y si el usuario existe pero la contraseña no es correcta << muestra mensaje >>

Sub-Menú de login exitoso

- **Ver perfil:** Muestra la información completa del usuario logueado.
- **Mostrar playlist:** Muestra el catálogo con dos opciones, ordenado por nombre o género.
- **Escuchar cancion:** Se ingresa el id de la canción y se agrega en la playlist del usuario.

- **Canciones recomendadas:** Muestra un listado de las canciones recomendadas para el usuario en base a su playlist. (Queda a criterio del equipo el algoritmo de recomendación).

3. **Registrarse:**
   Redirige al alta de usuario.

Tabla de puntuación:

Obtenido	10	20	30	40	50	60	70	80	90	100
Nota	1	2	2	4	5	6	7	8	9	10
Observación	*Desaprobado*					*Aprobado*				