UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA

Programación II – Curso 23/24

Práctica 1: Enunciado

1. El problema

El problema a resolver en esta Práctica 1 consiste en implementar una serie de funcionalidades para MUSFIC, una plataforma de música bajo demanda. Será necesario diseñar una estructura de datos que permita almacenar conjuntamente toda la información asociada a usuarios y reproducciones. En esta primera práctica se abordarán las tareas de gestión de usuarios, incluyendo altas, bajas y reproducciones.

Como el objetivo de este trabajo es practicar la independencia de la implementación en los Tipos Abstractos de Datos (TADs), se pide crear dos implementaciones de una LISTA NO ORDENADA, las cuales deberán funcionar de manera intercambiable: una implementación ESTÁTICA y otra DINÁMICA. De este modo, el programa principal no deberá realizar ninguna suposición sobre la forma en que está implementado el TAD.

Para facilitar la elaboración de esta primera práctica, se recomienda organizar el trabajo siguiendo las fases que se detallan a continuación.

2. Fase 1

Esta primera fase se centrará en el TAD. Para ello: (1) implementaremos una librería donde se incluyen los tipos de datos necesarios para el problema a resolver; y (2) implementaremos el TAD Lista en sus dos versiones, estática y dinámica.

2.1 Librería Types

Algunos tipos de datos se definirán en esta librería (`types.h`) ya que son necesarios para el problema a resolver y los usará tanto el TAD como el programa principal.

`NAME_LENGTH_LIMIT`	Longitud máxima de un nombre de usuario y de un título de canción (constante).
`tUserName`	Nombre de un usuario (`string`).
`tUserCategory`	Categoría de usuario (tipo enumerado: `{basic, pro}`)
`tNumPlay`	Número de reproducciones realizadas (`int`).
`tItemL`	Datos de un elemento de la lista (un usuario). Compuesto por: • `userName`: de tipo `tUserName` • `numPlay`: de tipo `tNumPlay` • `userCategory`: de tipo `tUserCategory`

`tSongTitle`	Título de una canción (`string`).
`tSong`	Datos de una canción. Contendrá el campo: • `songTitle` de tipo `tSongTitle`

2.2. TAD Lista

Para mantener la lista de usuarios y su información asociada, el sistema utilizará un TAD Lista. Se realizarán dos implementaciones:

1. **ESTÁTICA** con arrays (`static_list.c`) con tamaño máximo 25.
2. **DINÁMICA**, simplemente enlazada, con punteros (`dynamic_list.c`).

2.2.1. Tipos de datos incluidos en el TAD Lista

`tList`	Representa una lista de usuarios.
`tPosL`	Posición de un elemento de la lista.
`LNULL`	Constante usada para indicar posiciones nulas.

2.2.2. Operaciones incluidas en el TAD Lista

Una precondición común para todas estas operaciones (salvo `createEmptyList`) es que la lista debe estar previamente inicializada:

- `createEmptyList (tList) → tList`
  Crea una lista vacía.
  PostCD: La lista queda inicializada y no contiene elementos.

- `isEmptyList (tList) → bool`
  Determina si la lista está vacía.

- `first (tList) → tPosL`
  Devuelve la posición del primer elemento de la lista.
  PreCD: La lista no está vacía.

- `last (tList) → tPosL`
  Devuelve la posición del último elemento de la lista.
  PreCD: La lista no está vacía.

- `next (tPosL, tList) → tPosL`
  Devuelve la posición en la lista del elemento siguiente al de la posición indicada (o `LNULL` si la posición no tiene siguiente).
  PreCD: La posición indicada es una posición válida en la lista.

- `previous (tPosL, tList) → tPosL`
  Devuelve la posición en la lista del elemento anterior al de la posición indicada (o `LNULL` si la posición no tiene anterior).
  PreCD: La posición indicada es una posición válida en la lista.

- `insertItem (tItemL, tPosL, tList) → tList, bool`
  Inserta un elemento en la lista antes de la posición indicada. Si la posición es `LNULL`, entonces se añade al final. **Devuelve un valor `true` si el elemento fue insertado; `false` en caso contrario.**
  PreCD: La posición indicada es una posición válida en la lista o bien nula (`LNULL`).
  PostCD: Las posiciones de los elementos de la lista posteriores a la del elemento insertado pueden haber variado.

- `deleteAtPosition (tPosL, tList) → tList`
  Elimina de la lista el elemento que ocupa la posición indicada.
  PreCD: La posición indicada es una posición válida en la lista.
  PostCD: Las posiciones de los elementos de la lista posteriores a la de la posición eliminada pueden haber variado.

- `getItem (tPosL, tList) → tItemL`
  Devuelve el contenido del elemento que ocupa la posición indicada.
  PreCD: La posición indicada es una posición válida en la lista.

- `updateItem (tItemL, tPosL, tList) → tList`
  Modifica el contenido del elemento situado en la posición indicada.
  PreCD: La posición indicada es una posición válida en la lista.
  PostCD: El orden de los elementos de la lista no se ve modificado.

- `findItem (tUserName, tList) → tPosL`
  Devuelve la posición **del primer elemento de la lista** cuyo nombre de usuario se corresponda con el indicado (o `LNULL` si no existe tal elemento).

2.2.3. Testeo de la implementación del TAD

Una vez terminada la implementación del TAD Lista, es necesario comprobar el correcto funcionamiento de ésta mediante el fichero de prueba facilitado (`test_list.c`).

3. Fase 2

Una vez implementado el TAD, nos centraremos en el programa principal. La tarea consiste ahora en implementar un único programa (`main.c`) que procese las peticiones recibidas por la aplicación MUSFIC, que tienen el siguiente formato:

`N userName userCategory`	***[N]ew:*** Alta de un usuario de categoría `basic` o `pro`.
`D userName`	***[D]elete:*** Baja de un usuario.
`U userName`	***[U]pgrade:*** Actualización de un usuario de categoría `basic` a `pro`.
`P userName songTitle`	***[P]lay:*** Reproducción de una canción por un usuario.
`S`	***[S]tats:*** Listado de los usuarios actuales de MUSFIC y sus datos.

En el programa principal se implementará un bucle que procese una a una las peticiones de los usuarios. Para simplificar tanto el desarrollo como las pruebas, el programa no necesitará introducir ningún dato por teclado, sino que leerá y procesará las peticiones de usuarios contenidas en un fichero (ver documento `EjecucionScript.pdf`) En cada iteración del bucle, el programa leerá del fichero una nueva petición y la procesará. Para facilitar la corrección de la práctica todas las peticiones del fichero van numeradas correlativamente.

Para cada línea del fichero de entrada, el programa hace lo siguiente:

1. **Muestra una cabecera con la operación a realizar.** Esta cabecera está formada por una primera línea con 20 asteriscos y una segunda línea que indica la operación tal y como se muestra a continuación:

```
********************
CC␣T:␣user␣XX␣category/song␣YY
```

donde `CC` es el número de petición, `T` es el tipo de operación (`N`, `D`, `P` o `S`), `XX` es el nombre del usuario (`userName`) (cuando corresponda), `YY` es la categoría del usuario (`userCategory`) o el título de la canción (`songTitle`) (cuando y según corresponda), y ␣ indica un espacio en blanco. Sólo se imprimirán los parámetros necesarios; es decir, para una petición `[S]tats` se mostrará únicamente "`01 S`", mientras que para una petición `[P]lay` se mostrará "`02 P: user User1 song Song1`".

2. **Procesa la petición correspondiente:**

- Si la operación es **`[N]ew`**, se incorporará el usuario al final de la lista de usuarios, con la categoría indicada y con su número de reproducciones inicializado a 0. Además, se imprimirá el mensaje:

```
*␣New:␣user␣XX␣category␣YY
```

  donde, nuevamente, `XX` es el `userName`, `YY` es el `userCategory` y ␣ representa un espacio en blanco. El resto de mensajes sigue el mismo formato.

  Si ya existiese un usuario con ese userName o no se ha podido realizar la inserción, se imprimirá el siguiente mensaje:

```
+␣Error:␣New␣not␣possible
```

- Si la operación es **`[D]elete`**, se buscará al usuario en la lista, se eliminará de la misma y se imprimirá el siguiente mensaje:

```
*␣Delete:␣user␣XX␣category␣YY␣numplays␣ZZ
```

  Si no existiese ningún usuario con ese nombre o la lista estuviese vacía, se imprimirá el siguiente mensaje:

```
+␣Error:␣Delete␣not␣possible
```

- Si la operación es **[U]pgrade**, se buscará al usuario en la lista, se actualizará su categoría a `pro` y se mostrará el siguiente mensaje:

```
*␣Upgrade:␣user␣XX␣category␣YY
```

  Si no existiese ningún usuario con ese nombre, si el usuario ya fuese `pro` o si la lista estuviese vacía, se imprimirá el mensaje:

```
+␣Error:␣Upgrade␣not␣possible
```

- Si la operación es **[P]lay**, se buscará al usuario en la lista, se incrementará su contador de reproducciones en 1 y se mostrará el siguiente mensaje:

```
*␣Play:␣user␣XX␣plays␣song␣YY␣numplays␣ZZ
```

  Si no existiese ningún usuario con ese nombre o si la lista está vacía se imprimirá el mensaje:

```
+␣Error:␣Play␣not␣possible
```

- Si la operación es **[S]tats**, se mostrará la lista completa de usuarios actuales de la siguiente forma:

```
User␣XX1␣category␣basic ␣numplays␣ZZ1
User␣XX2␣category␣pro␣numplays␣ZZ2
...
User␣XXn␣category␣basic␣numplays␣ZZn
```

  A continuación, se mostrará una tabla donde, para cada categoría de usuario, se indicará el número de usuarios con esa categoría, su número de canciones reproducidas, y su media de reproducciones (con dos decimales). El formato de dicha tabla es el siguiente:

```
Category␣␣Users␣␣Plays␣␣Average
Basic␣␣␣␣␣%5d␣%6d␣%8.2f
Pro␣␣␣␣␣␣␣%5d␣%6d␣%8.2f
```

  Si la lista estuviese vacía se imprimirá el mensaje:

```
+␣Error:␣Stats␣not␣possible
```

4. Ejecución de la práctica

Para facilitar el desarrollo de la práctica se proporciona el siguiente material de especial interés: (1) un directorio `CLion` que incluye un proyecto plantilla (`P1.zip`) junto con un fichero que explica cómo hacer uso de éste (`Presentacion_uso_IDE.pdf`); y (2) un directorio `script` que contiene un fichero (`script.sh`) que permite probar de manera conjunta los distintos archivos proporcionados. Además, se facilita un documento de ayuda para su ejecución (`Ejecucion_Script.pdf`). Nótese que, para que dicho *script* de pruebas

no dé problemas, se recomienda encarecidamente **NO copiar-pegar directamente texto desde este documento al fichero de código**, ya que el formato PDF puede incluir caracteres invisibles que darían por incorrectas salidas (aparentemente) válidas.

5. Documentación del código

El código deberá estar convenientemente **comentado**, incluyendo las variables empleadas. Los comentarios han de ser concisos pero explicativos. Asimismo, después de la cabecera de cada procedimiento o función del programa y/o librería, se incluirá la siguiente información correspondiente a su **especificación**, tal y como se explicó en el TGR correspondiente:

- *Objetivo* del procedimiento/función.
- *Entradas* (identificador y breve descripción, una por línea).
- *Salidas* (identificador y breve descripción, una por línea).
- *Precondiciones* (condiciones que han de cumplir las entradas para el correcto funcionamiento de la subrutina).
- *Postcondiciones* (otras consecuencias de la ejecución de la subrutina que no quedan reflejadas en la descripción del objetivo o de las salidas).

6. Información importante

El documento `NormasEntrega_CriteriosEvaluacion.pdf`, disponible en la página web de la asignatura detalla claramente las normas de entrega. Para un adecuado **seguimiento de la práctica** se realizarán **entregas obligatorias parciales** antes de las fechas y con los contenidos que se indican a continuación:

1. **Entrega parcial #1**: martes **5 de marzo** a las 22:00 horas. Implementación estática y prueba del TAD Lista: entrega de los ficheros `types.h`, `static_list.c` y `static_list.h` (solamente dichos ficheros).

2. **Entrega parcial #2:** martes **12 de marzo** a las 22:00 horas. Implementación dinámica y prueba del TAD Lista: entrega de los ficheros `types.h`, `dynamic_list.c` y `dynamic_list.h` (solamente dichos ficheros).

Para comprobar el correcto funcionamiento de los TAD se facilita el fichero de prueba `test_list.c`. Se realizará una corrección automática usando el *script* proporcionado para ver si se supera o no el control de seguimiento (véase el documento `NormasEntrega_CriteriosEvaluacion.pdf`).

Fecha límite de entrega de la práctica: martes **19 de marzo** a las 22:00 horas.