UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA

Programación II – Curso 2020/21

Práctica 1: Enunciado

1. El problema

En esta práctica se implementarán una serie de funcionalidades para VIMFIC, una plataforma de vídeo 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.

2. 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 nick y de un título de vídeo (constante)
`tNickname`	Nick de un usuario (`string`)
`tUserCategory`	Categoría de usuario (tipo enumerado: `{standard, premium}`)
`tNumPlay`	Número de reproducciones realizadas (`int`)
`tItemL`	Datos de un elemento de la lista (un usuario). Compuesto por: • `nickname`: de tipo `tNickname` • `numPlay`: de tipo `tNumPlay` • `userCategory`: de tipo `tUserCategory`
`tTitleVideo`	Título de un vídeo (`string`)
`tVideo`	Datos de un vídeo. Contendrá el campo: • `titleVideo` de tipo `tTitleVideo`

3. 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`).

3.1. 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

3.2. 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 de la lista 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 (tNickname, tList) → tPosL`

Devuelve la posición **del primer elemento de la lista** cuyo nick de usuario se corresponda con el indicado (o `LNULL` si no existe tal elemento).

4. Descripción de la tarea

La tarea consiste en implementar un único programa principal (`main.c`) que procese las peticiones de los usuarios de VIMFIC con el siguiente formato:

`N nickname userCategory`	***[N]ew:*** Alta de un usuario de categoría standard o premium
`D nickname`	***[D]elete***: Baja de un usuario
`P nickname titleVideo`	***[P]lay***: Reproducción de un vídeo por parte de un usuario
`S`	***[S]tats:*** Listado de los usuarios actuales de VIMFIC 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:

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:␣nick␣XX␣category/video␣YY
```

donde `CC` es el número de petición, `T` es el tipo de operación (`N`, `D`, `P` o `S`), `XX` es el valor del *nick* del usuario (cuando corresponda) e YY es la categoría del usuario o el título del vídeo (cuando y según corresponda), ␣ 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: nick Nickname1 video Video1`".

2. **Procesa la petición** correspondiente:

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

```
*␣New:␣nick␣XX␣category␣YY
```

donde, de nuevo, ␣ representa un espacio en blanco. El resto de mensajes siguen el mismo formato.

Si ya existiese un usuario con ese *nick*, se imprimirá el siguiente mensaje:

```
+ Error: New not possible
```

• Si la operación es **`[D]elete`**, se buscará al usuario en la lista, se borrará y se imprimirá el siguiente mensaje:

```
* Delete: nick XX category YY numplays ZZ
```

Si no existiese ningún usuario con ese *nick*, se imprimirá el siguiente mensaje:

```
+ Error: Delete not possible
```

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

```
* Play: nick XX plays video YY numplays ZZ
```

Si no existiese ningún usuario con ese *nick* 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:

```
Nick XX1 category standard numplays ZZ1
Nick XX2 category premium numplays ZZ2
...
Nick XXn category standard numplays ZZn
```

  A continuación, se mostrarán, para cada categoría de usuario, el número de usuarios, el número de vídeos reproducidos, y la media de reproducciones, con el formato siguiente:

```
Category__Users__Plays__Average
Standard____TTn_____PPn_____AAn
Premium_____TTp_____PPp_____AAp
```

  Si la lista estuviese vacía se imprimirá el mensaje:

```
+ Error: Stats not possible
```

5. Lectura de ficheros

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 del mismo (`Presentacion_uso_IDE.pdf`) y, (2) Un directorio `script` donde se proporciona 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 el `script` no dé problemas se recomienda **NO copiar directamente el código de este documento**, ya que el formato PDF puede incluir caracteres invisibles que darían por incorrectas salidas válidas.

6. Información importante

El documento `NormasEntrega.pdf`, disponible en la página web de la asignatura detalla claramente las normas de entrega.
Fecha límite de entrega: **Viernes 9 de Abril de 2021 a las 22:00 horas**