FACULTY OF COMPUTER SCIENCE
Programming II - Course 23/24

Practical 1: Instructions

1. The Problem

The problem to be solved in Practical 1 consists of implementing the main features of MUSFIC, a music-on-demand platform. To do this, it will be necessary to design a data structure capable of storing all the information associated with the platform users and the number of times they played each song ("stream counts"). In this first practical, the students will develop the user management system, including operations for adding and removing users and counting their "streams".

The aim of this work is to practice the concept of independence of implementation in the context of Abstract Data Types (ADTs). The student is asked to create two different implementations of an UNORDERED LIST: a STATIC implementation and a DYNAMIC implementation, which must both work in a fully interchangeable way. So, the main program must not make any assumptions about the way an ADT is implemented.

To facilitate the development of this first practical, we encourage the students to organize their work according to the phases detailed below.

2. Phase #1

This first phase will focus on the ADT. We will do the following: (1) implement a *header file* with the data types necessary to solve the problem; and (2) implement the two versions of the ADT List, that is, its static and dynamic implementations.

2.1. Header File Types

The following data types will be defined in the header file `types.h`, since they are necessary to solve the problem and they are used by both the ADT and the main program.

`NAME_LENGTH_LIMIT`	Maximum length of usernames and song titles (constant)
`tUserName`	Name of a user (`string`)
`tUserCategory`	Category of a user (enumerated type `{basic, pro}`)
`tNumPlay`	Number of streams, the total per user (all songs) (`int`)
`tItemL`	Data for an element of the list (a user). It contains: • `username`: type `tUserName` • `numPlay`: type `tNumPlay` • `userCategory`: type `tUserCategory`

`tSongTitle`	Title of a song (`string`)
`tSong`	Data for a song. It contains the field: • `songTitle`: type `tSongTitle`

2.2. ADT List

The system will make use of an ADT List to hold the list of users and their associated data. Two different implementations of this ADT will be created:

1. A **STATIC** one using arrays (`static_list.c`) with a maximum size of 25 elements.
2. A singly-linked **DYNAMIC** one using pointers (`dynamic_list.c`).

2.2.1. Data types included in the ADT List

`tList`	Represents a list of users
`tPosL`	Position of an element in the list
`LNULL`	Constant used to represent null positions

2.2.2. Operations included in the ADT List

A common precondition for all these operations (except `createEmptyList`) is that the list must be previously initialised:

- `createEmptyList (tList) → tList`
  Creates an empty list.
  PostCD: The list is initialised and has no elements.

- `isEmptyList (tList) → bool`
  Determines whether the list is empty or not.

- `first (tList) → tPosL`
  Returns the position of the first element of the list.
  PreCD: The list is not empty.

- `last (tList) → tPosL`
  Returns the position of the last element of the list.
  PreCD: The list is not empty.

- `next (tPosL, tList) → tPosL`
  Returns the position following the one we indicate (or `LNULL` if the specified position has no next element).
  PreCD: The indicated position is a valid position in the list.

- `previous (tPosL, tList) → tPosL`
  Returns the position preceding the one we indicate (or `LNULL` if the specified position has no previous element).
  PreCD: The indicated position is a valid position in the list.

- `insertItem (tItemL, tPosL, tList) → tList, bool`
  Inserts an element containing the provided data item in the list. If the specified position is `LNULL`, then the element is added at the end of the list; otherwise, it will be placed right before the element currently holding that position. **It the element could be inserted, the value `true` is returned, `false` otherwise.**
  PreCD: The specified position is a valid position in the list or a `LNULL` position.
  PostCD: The positions of the elements in the list following that of the inserted one may have changed.

- `deleteAtPosition (tPosL, tList) → tList`
  Deletes the element at the given position from the list.
  PreCD: The indicated position is a valid position in the list.
  PostCD: The positions of the elements in the list following that of the deleted one may have changed.

- `getItem (tPosL, tList) → tItemL`
  Retrieves the content of the element at the position we indicate.
  PreCD: The indicated position is a valid position in the list.

- `updateItem (tItemL, tPosL, tList) → tList`
  Modifies the content of the element at the position we indicate.
  PreCD: The indicated position is a valid position in the list.
  PostCD: The order of the elements in the list has not been modified.

- `findItem (tUserName, tList) → tPosL`
  Returns the position **of the first element in the list** whose username matches the one indicated (or `LNULL` if there is no such element).

2.2.3. Testing the ADT implementation

Once the ADT List has been implemented, it is necessary to check its correct functioning using the provided test file (`test_list.c`).

3. Phase #2

Once the ADT is implemented, we will focus on the main program. The task now consists of implementing a single program (`main.c`) that processes the requests received by the MUSFIC system. The requests have the following format:

`N username userCategory`	***[N]ew:*** A new user of category `basic` or `pro` is added.
`D username`	***[D]elete:*** The user is removed.
`U userName`	***[U]pgrade:*** Upgrade of a user from `basic` to `pro` category.
`P username songTitle`	***[P]lay:*** (Re)play of a song by a user, "stream".
`S`	***[S]tats:*** List of current MUSFIC users and their data.

The main program will contain a loop to process, one by one, the requests of the users. In order to simplify the development and testing of the system, the program will not prompt the user to input the data of each request. Instead, the program will take as input a file containing the sequence of requests to be executed (**see document `RunScript.pdf`**). For each loop iteration, the program will read a new request from the file and then process it. In order to make correction easier, all requests in the input file have been numbered consecutively.

For each line of the input file, the program will do the following:

1. Show a header with the operation to be performed. This header consists of a first line with 20 asterisks and a second line indicating the operation as shown below:

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

where `CC` is the number of the request, `T` is the type of operation (`N`, `D`, `P` or `S`), `XX` is the name of the user (`userName`) (when applicable), `YY` is the category of the user (`userCategory`) or the title of the song (`songTitle`) (when and as appropriate), and ␣ represents a blank. Only the necessary parameters are printed; that is, for a `[S]tats` request we will only show "`01 S`", while for a `[P]lay` request we will show "`02 P: user User1 song Song1`".

2. Process the corresponding request:

- If the operation is **`[N]ew`**, that user must be added **at the end** of the user list, with the specified user category and the stream count initialized to `0`. In addition, a message like this will be displayed:

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

  where, again, `XX` is the `userName`, `YY` is the `userCategory` y, ␣ represents a blank. The rest of messages follow the same format.

  If a user with that `userName` already exists or the insertion could not be performed, the following message will be printed:

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

- If the operation is **`[D]elete`**, the system will locate and remove that user from the list. In addition, a message like this will be displayed:

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

  In the event that there is no user with the given username or the list is empty, the following message must be printed:

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

- If the operation is **[U]pgrade**, the user is located, his category is upgraded to `pro`, and the following message is displayed:

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

  In the event that there is no user with the given username, the user has already category `pro`, or the list is empty, the following message must be printed:

  ```
  + Error: Upgrade not possible
  ```

- If the operation is **[P]lay**, the user is located and their total stream count is incremented by 1. In addition, a message like this will be displayed:

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

  If there is no user with the given username or the list is empty, the following message must be printed:

  ```
  + Error: Play not possible
  ```

- If the operation is **[S]tats**, the whole list of current users will be displayed as follows:

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

  Below this list we will also print a table showing, for each user category, the number of users with that category, their number of songs played, and their average stream count (to two decimal places). The table must follow the following format:

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

  In the event that the user list is empty, the following message must be printed:

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

4. Running the Program

To facilitate the development of this practical, we provide the following materials: (1) a folder `CLion` that includes a template project (`P1.zip`) along with a file that explains how to use it (`Howto_use_IDE.pdf`); and (2) a folder `script` which contains a file (`script.sh`) that allows you to test the system with all the test files supplied at once. A document explaining how to run it is also provided (`RunScript.pdf`). Finally, to avoid problems when running the script, it is highly recommended **NOT to directly copy-paste the text of this document into the source code**, since the PDF format may include

invisible characters that may result in (apparently) valid outputs to be considered incorrect.

5. Commenting the source code

The source code must be appropriately **commented**, including the variables used. Comments must be concise but informative and useful (do not include unnecessary comments which will only clutter the code). Additionally, below the header of each function, the following **specification** must be included:

- *Goal* of the function (subroutine).
- *Inputs* (identifier and brief description, one per line).
- *Outputs* (identifier and brief description, one per line).
- *Preconditions* (those conditions to be met by the input entries for the proper functioning of the subroutine).
- *Postconditions* (other consequences of the execution of the subroutine that are not reflected in the descriptions of its goal nor the outputs).

6. Important information

The document `DeliveryGuidelines_AssessmentCriteria.pdf`, available on the course website, clearly outlines the delivery guidelines to be followed. For an adequate **evaluation of this practical**, two **mandatory partial deliveries** must be made before the deadlines and with the contents indicated below:

1. **Checkpoint #1:** Tuesday **March 5th**, at 22:00. Implementation and testing of the static version of the ADT List: submission of files `types.h`, `static_list.c` y `static_list.h` (only these files).

2. **Checkpoint #2:** Tuesday **March 12th**, at 22:00. Implementation and testing of the dynamic version of the ADT List: submission of files `types.h`, `dynamic_list.c` y `dynamic_list.h` (only these files).

To check the correct functioning of the ADT implementations, the test file `test_list.c` is provided. The implementations delivered by the students will be assessed automatically. For this purpose, the provided script will be used to check whether the corresponding checkpoint is passed or not (see doc `DeliveryGuidelines_AssessmentCriteria.pdf`).

Final submission deadline: Tuesday **March 19th**, at 22:00.