Assignment 8: Music database and queries

Course 'Imperative Programming' (IPC031)

Make sure to start working on the assignment **before the lab** session, otherwise you will be too late for the deadline

1 Background

In this assignment you work with vectors of structured data. On Brightspace, you find a .zip file. The input data, a database consisting of tracks from Peter's CD collection, is given in the text file "`Tracks.txt`". Your task is to implement a number of functions for querying the database. The "`main.cpp`" in the .zip file contains definitions that must be used in this assignment. Queries may contain text arguments. These can be used for string matching with (a part of) a database entry.
Currently, "`Tracks.txt`" contains 6,868 entries of CD tracks. In an entry information about the following data items of a track are stored:

- the name of the artist
- the title of the CD
- the year of recording, or `"0"` if year is not known
- the track number
- the track title
- the tags, may contain spaces; tags are separated by a `","`
- the track length, in "mm:ss" format
- the countries of artist; may contain space; countries are separated by a `","`

In an entry, each data item is given by a string presented on a separate text line. Each entry, as a whole, is ended with a newline.
The database is sorted. The ordering of data items, which are strings, is based on the ordering of ASCII codes of characters. This has consequences on the ordering of tracks. For example, the CD track with number 10 precedes the CD track with number 2, because the string `"10"` is lexicographically smaller than the string `"2"`. As, in ASCII coding, upper-case characters have a lower value than lower-case characters have, artist `"dEUS"` follows `"Yes"` (the ASCII code of `'Y'` is 89, the ASCII code of `'d'` is 100).
You may assume that each CD entry is complete (all tracks are present) and no artist has two or more different albums with the same album name.

2 Learning objectives

After doing this assignment you are able to:

- Work with structured data;
- Work with vectors of structured data and realize a number of search-operations on them.

3 Assignment

Part 1: Operator overloading on time lengths

In "`main.cpp`" you find the struct `Length` which has to be used for track / CD lengths. Design and implement the overloaded operators `<<`, `>>`, `+` on `Length` values. The format used by the `<<` operator must be "mm:ss" (see "`Tracks.txt`" for examples) where $0 \leq$ "mm" and $0 \leq$ "ss" ≤ 59.
Test your implementation of `<<` and `>>` for at least two different `Length` values. (Desktop) test your algorithm and implementation of `+` for all combinations of the following `Length` values:

```
Length l1 = {42,55}, l2 = {3,4}, l3 = {24,6};
```

Part 2: Customized printing of tracks

In "`main.cpp`" tracks are stored as `Track` values. The queries to be implemented in Part 3 produce different output for a track. Instead of introducing different functions for each query, define a single function:

```
void show_track (Track track, TrackDisplay td, ostream& os)}
```

The `TrackDisplay` parameter, defined in "`main.cpp`", specifies which data items of a `Track` value must be displayed. The `ostream&` parameter specifies where the track should be printed to. This could be to the console by providing `cout`, but also to a file or string stream. Design and implement `show_track`. Use this function for displaying `Track` values in Part 3.

Part 3: Database queries

Design and implement functions for queries as explained below. In "`main.cpp`" you find the function `bool match (string sub, string source)` which returns true only if sub is a substring of source. Use this function for matching strings. Matching is case-sensitive. Note that an empty string matches any string. Test your implementations by using the `vector<Track> testDB` in "`main_test.cpp`".

- `int match_tracks (const vector<Track>& tracks, string track, bool display)`:
  This function finds every track of which the track title matches `track`. If the value of `display` is `true`, then, in each match, the artist, CD title, year of recording, track number, track title, track length, tags, and country are shown on the screen by making use of the function `show_track`. If the value of `display` is `false`, then no output is sent to the screen. The function returns the number of tracks found, irrespective of the value of `display`.
  Example: In `testDB` there are **4** tracks that contain the string `"el"`: Melt my heart to stone, Make you feel my love, Mellowing, Instrumental 2 (welcome to the caveman future).
- `int match_artists (const vector<Track>& tracks, string artist, bool display)`:
  This function finds every artist whose name matches `artist`. If the value of `display` is `true`, the function displays the matching artist name by making use of the function `show_track`. Each matching artist name is displayed and counted at most once (thus not for every track). If the value of `display` is `false`, then no output is sent to the screen. The function returns the number of artists found, irrespective of the value of `display`.
  Example: In `testDB` there are **2** artists that contain the string `"el"`: Adele, Colin Stetson and Sarah Neufeld.
- `int match_cds (const vector<Track>& tracks, string artist, bool display)`:
  This function finds every CD title of which the artist's name matches `artist`. If the value of `display` is `true`, then display the artist name, CD title, and year of recording by making use of the function `show_track`. Each CD title is displayed and counted at most once (thus not for every track). If the value of `display` is `false`, then no output is sent to the screen. The function returns the number of CD titles found, irrespective of the value of `display`.
  Example: In `testDB` there are **3** albums by artists with name containing the string `"el"`: 19, 21, Never were the way she was.
- `int number_of_cds (const vector<Track>& tracks)`: This function returns the total number of CDs. Each CD is counted exactly once.
  Example: In `testDB` there are **5** CDs.

Part 4: Reading the database file

Design and implement the function

```
int read_tracks (string filename, vector<Track>& tracks, bool show_content)
```

It attempts to open the file with file name `filename`, and reads and writes all of the found `Track` elements into the vector `tracks`. If the Boolean condition `show_content` is true, then the content of the read file should be displayed (useful for yourself to see if everything is working as expected). The result of the function is the number of read `Track` elements. If the file could not be opened, or it is empty or ill-formatted, `tracks` is empty and the result of the function is zero.
As part of this task, design and implement the overloaded operator `>>` for `Track` values. To this end, make use of the overloaded operator `>>` for `Length` values, in Part 1.

Afterwards, make use of `read_tracks` and `number_of_cds` in your `main` function in order to show the amount of tracks and cds in “`Tracks.txt`”.
Tip: the files folder contains, besides the target database “`Tracks.txt`”, a number of smaller files to test your implementation with:

- “`TracksZeroTrackDB.txt`”: contains no entries at all, so `tracks` should be empty after the function;
- “`TracksOneTrackDB.txt`”: contains one entry, so `tracks` should contain exactly one element;
- “`TracksTestDB.txt`”: this is the file representation of the `TEST_DB` vector in “`main_test.cpp`”.

4 Products

As product-to-deliver you upload to Brightspace:

- “`main.cpp`” that you have created with solutions for each part of the assignment.
- “`main_test.cpp`” that has been extended with non-trivial unit tests for each new function that you have developed in “`main.cpp`”.

Deadline

Lab assignment: Friday, November 10, 2023, 23:59h