Laborator 11

Tema: Standard Template Library (STL)

Poveste

O aplicație populară de streaming muzical dorește să implementeze un modul inteligent pentru gestionarea playlist-urilor utilizatorilor. Un playlist este format dintr-o serie de melodii (**Song**), fiecare având un titlu, un artist, o durată și o listă de etichete (tags - ex: "Rock", "90s", "Chill"). Pentru a oferi recomandări mai bune, aplicația trebuie să poată analiza rapid playlist-ul curent:

- Să sorteze melodiile după durată (pentru cei care vor sesiuni scurte/lungi).
- Să găsească rapid o melodie după titlu.
- Să filtreze melodiile care aparțin unui anumit gen (tag).
- Să genereze un raport ("Music DNA") care arată care sunt genurile predominante în playlist.

Vom utiliza puterea **STL (Standard Template Library)** pentru a gestiona aceste date eficient, fără a ne preocupa de alocarea manuală a memoriei.

Scopul laboratorului

- Utilizarea containerelor secvențiale (`std::vector`) pentru stocarea obiectelor;
- Utilizarea containerelor asociative (`std::map`) pentru frecvență și statistici;
- Manipularea șirurilor de caractere cu `std::string`;
- Scrierea de funcții **Lambda** pentru a personaliza algoritmii STL;
- Utilizarea algoritmilor standard: `sort`, `find_if`.

Cerințe de implementare (9p + 1p oficiu)

În acest laborator **vom folosi exclusiv** `std::string` în loc de `char*`.

1. **Clasa Song (2p)**
   - Creați o clasă `Song` care să conțină:
     - Titlu (`std::string`);
     - Artist (`std::string`);
     - Durata în secunde (`int`);
     - O listă de tag-uri/genuri (folosiți `std::vector<std::string>`).
   - Implementați constructorul și metodele de acces (getters) necesare.
2. **Gestionarea Playlist-ului (3p)**
   - În `main`, populați un `std::vector<Song>` cu cel puțin 5 melodii diferite.
   - Folosind `std::sort` și o **funcție lambda**, sortați playlist-ul crescător după durată și afișați-l.
   - Sortați din nou playlist-ul alfabetic după numele artistului.
3. **Căutare și Filtrare (2p)**
   - Folosind `std::find_if`, căutați o melodie după titlu (ex: "Bohemian Rhapsody").
   - Implementați o filtrare care afișează toate melodiile care conțin un anumit tag (ex: "Rock"). Puteți itera prin vector și verifica fiecare melodie.
4. **Statistici cu Map (2p)**
   - Folosiți un `std::map<std::string, int>` pentru a număra frecvența fiecărui tag în playlist.
   - Cheia este tag-ul (ex: "Pop"), iar valoarea este numărul de melodii care au acel tag.
   - Afișați "Music DNA"-ul utilizatorului (genurile și numărul de melodii aferente).

Ghid pentru `main.cpp`

Acesta este scheletul de cod. Trebuie să creați fișierele `Song.h` și `Song.cpp` și să completați `main`-ul.

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm> // sort, find_if
#include <map>       // map
#include "Song.h"    // Creati voi acest fisier
int main() {
    // 1. Initializare playlist
    std::vector<Song> playlist;

    // TODO: Adaugati 5 melodii.
    // Ex: playlist.push_back("Numb", "Linkin Park", 187,
    ↪  std::vector<std::string>{"Rock", "2000s"});

    // 2. Afisare initiala
    std::cout << "--- Playlist Initial ---\n";
    for (const auto& s : playlist) {
        // TODO: Afisati detaliile melodiei (operator << sau metoda print)
    }

    // 3. Sortare dupa durata (Lambda)
    // TODO: std::sort cu lambda: [](const Song& a, const Song& b) { return
    ↪  a.getDuration() < b.getDuration(); }

    std::cout << "\n--- Playlist Sortat dupa Durata ---\n";
    // TODO: Afisare playlist sortat

    // 4. Cautare melodie dupa titlu
    std::string titluCautat = "Numb";
    // TODO: std::find_if cu lambda care verifica daca s.getTitle() == titluCautat.
    // auto it = std::find_if(...);
    // if (it != playlist.end()) { cout << "Gasit: " << ... }

    // 5. Statistici Tag-uri (Music DNA)
    std::map<std::string, int> tagStats;
    // TODO: Iterati prin playlist.
    // TODO: Pentru fiecare melodie, iterati prin vectorul ei de tag-uri.
    // TODO: tagStats[tag]++;

    std::cout << "\n--- Music DNA (Top Genuri) ---\n";
    for (const auto& pair : tagStats) {
        std::cout << "Gen: " << pair.first << " -> " << pair.second << " melodii\n";
    }
    return 0;
}
```

Compilare și verificare

```
# Compilare
g++ -o lab11 main.cpp Song.cpp
# Rulare
./lab11
# Verificare (optional)
valgrind ./lab11
```

Bonus GitHub Classroom

Studenții pot obține până la **3p bonus** dacă încarcă toate laboratoarele până la finalul semestrului.

Pași pentru încărcarea laboratorului:

(a) Accesați linkul **GitHub Classroom** pentru Laboratorul 11:

`https://classroom.github.com/a/RpAZgwml`

(b) Clonați repository-ul, creați fișierele `Song.h/cpp` și completați `main.cpp`.

(c) Încărcați soluția pe branch-ul main:

```
git add .
git commit -m "Lab 11 - Music Playlist STL"
git push origin main
```