Laborator 11 – Standard Template Library

Tema: Cinema City – Sistem de gestionare a rezervărilor 🎞

Autor: ing. Eduard Donea

Poveste

Cinema City procesează zilnic mii de rezervări pentru săli, filme, clienți și promoții. Fiecare sală poate avea locuri ocupate, locuri libere, rezervări noi care intră în coadă, clienți fideli care vin frecvent, și un istoric complet al rezervărilor anterioare.

Pentru a eficientiza gestionarea acestui flux intens de date, Cinema City are nevoie de un sistem modern implementat în C++ folosind puterea **STL** (vector, map, set, queue, list, unordered_map) și algoritmi standard (std::sort, std::count, std::copy_if, lambdas, iterators).

În acest laborator, studenții vor implementa un mini–sistem de rezervări în două fișiere: `CinemaCity.h` și `CinemaCity.cpp`, plus un `main.cpp` pentru testare.

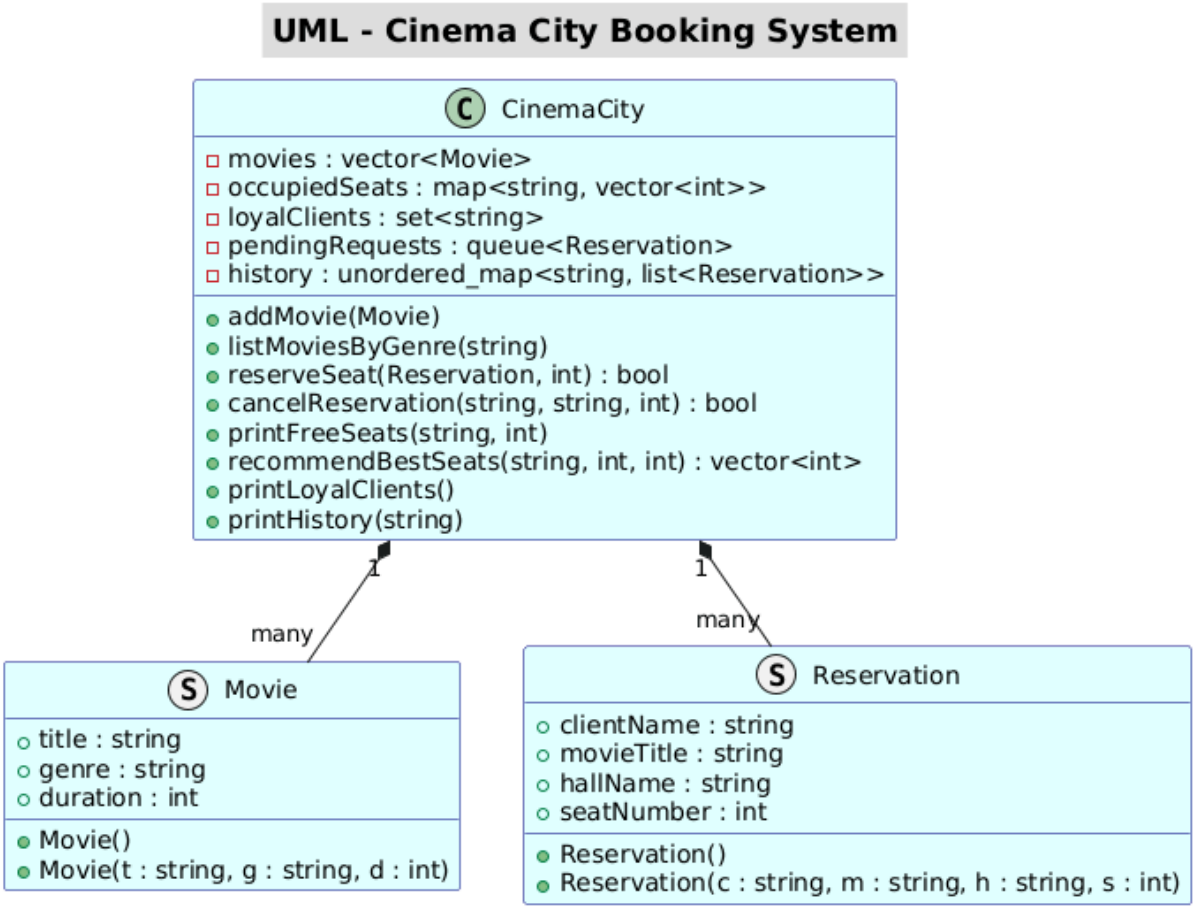
Scopul laboratorului

- utilizarea containerelor STL: `vector`, `list`, `set`, `map`, `queue`, `unordered_map`;
- folosirea algoritmilor STL: `sort`, `binary_search`, `lower_bound`, `copy_if`;
- utilizarea funcțiilor **lambda** pentru sortări și criterii personalizate;
- gestionarea eficientă a rezervărilor și locurilor ocupate;
- utilizarea unui **struct** pentru modelarea datelor simple (film și rezervare);
- manipularea colecțiilor mari folosind iteratoare directe și reverse.

Structura proiectului

- **CinemaCity.h – Movie** – struct pentru filme (titlu, gen, durată);
- **CinemaCity.h – Reservation** – struct pentru rezervări (client, film, sală, loc);
- **CinemaCity.h – CinemaCity** – clasa principală care gestionează toate containerele STL;
- **main.cpp** – fișierul de testare unde se realizează toate operațiile cerute.

Figura 1 – Diagrama UML a sistemului Cinema City



Fișierul `CinemaCity.cpp`

```cpp
#include "CinemaCity.h"

// =========================
// STRUCTS IMPLEMENTATION
// =========================

// TODO: Implementati constructorii din struct-uri.

// =========================
// CINEMACITY IMPLEMENTATION
// =========================

// TODO: Implementati constructorul din clasa.

// =========================
// AUXILIAR: LOYAL STATUS
// =========================

// ==============================
//      PRIVATE: Loyal Status
// ==============================
void CinemaCity::updateLoyalStatus(const std::string& clientName) {
    // TODO:
    // - Daca history[clientName].size() >= x -> adaugati in loyalClients.
    // - Daca < x -> eliminati-l din loyalClients daca exista.
    // - x e ales de voi.
}

// ==============================
//      GESTIONARE FILME
// ==============================
void CinemaCity::addMovie(const Movie& m) {
    // TODO: Adaugati filmul in vectorul movies.
}

void CinemaCity::listMoviesByGenre(const std::string& g) const {
    // TODO:
    // - Parcurgeti movies
    // - Afisati doar filmele cu acelasi gen
}

void CinemaCity::listAllMoviesSorted() const {
    // TODO:
    // - Creati o copie locala
    // - Sortati cu lambda dupa titlu
    // - Afisati rezultatul
}
```

```cpp
// ==============================
//      REZERVARI
// ==============================
bool CinemaCity::reserveSeat(const Reservation& r, int capacity) {
    // TODO:
    // - Verificati seatNumber 1..capacity
    // - Obtineti vectorul occupiedSeats[r.hallName]
    // - Sortati-l
    // - binary_search pentru loc ocupat
    // - Daca liber -> inserati cu lower_bound
    // - Adaugati in pendingRequests
    // - Adaugati in history[clientName]
    // - updateLoyalStatus(clientName)
}

bool CinemaCity::cancelReservation(const std::string& client,
                                   const std::string& movie,
                                   int seat) {
    // TODO:
    // - Cautati rezervarea in history[client]
    // - Salvati hallName
    // - Stergeti-o din history
    // - Scoateti seat din occupiedSeats[hallName]
    // - updateLoyalStatus(client)
}

void CinemaCity::printFreeSeats(const std::string& hall, int cap) const {
    // TODO:
    // - Generati vector freeSeats
    // - Parcurgeti 1..cap
    // - Folositi binary_search pentru occupiedSeats[hall]
}

std::vector<int> CinemaCity::recommendBestSeats(const std::string& hall,
                                                int cap,
                                                int howMany) const {
    // TODO:
    // - Generati vector freeSeats
    // - Sortati-l cu lambda dupa distanta fata de centru
    // - Returnati primele howMany locuri
}

void CinemaCity::printLoyalClients() const {
    // TODO: Afisati set-ul loyalClients
}

void CinemaCity::printHistory(const std::string& client) const {
    // TODO: Parcurgeti lista history[client]
}
```

Structurile `Movie` și `Reservation`

Structurile `Movie` și `Reservation` sunt cele mai simple forme de organizare a datelor.
Movie – titlu, gen, durată **Reservation** – client, film, sală, loc
Aceste două structuri sunt trecute în UML ca două clase simple de date. Rolul lor este de a transporta informația între containerele STL fără logică suplimentară.

Clasa `CinemaCity`

Clasa `CinemaCity` gestionează tot fluxul logic:

- `vector<Movie>| - lista filmelor;`
- `map<string, vector<int>>| - locurile ocupate;`
- `queue<Reservation>| - rezervări noi;`
- $\texttt{unordered}_map < string, list < Reservation >> | - istoriculperclient$;
- `set<string>| - clienți fideli (>= x rezervări), cu x la alegere;`

De asemenea, conține funcții esențiale: sortări cu lambda, filtrare, recomandări automate de locuri, iteratoare, căutări, binary_search și lower_bound.

Fișierul `main.cpp`

```cpp
#include "CinemaCity.h"
#include <iostream>

int main() {

    CinemaCity cc;

    // ============================
    // TODO A: Adaugati filme
    // ============================
    // Exemplu: cc.addMovie({"Dune 2", "SF", 165});

    // TODO B: Afisati filmele sortate

    // TODO C: Afisati filme dupa gen

    const std::string hall = "NumeSala";
    const int capacity;

    // ============================
    // TODO D: Faceti rezervari
    // ============================
    // Exemplu: cc.reserveSeat({"Alice", "Dune 2", hall, 5}, capacity);

    // TODO E: Afisati locurile libere

    // TODO F: Recomandare locuri (lambda)

    // TODO G: Afisati clientii fideli

    // TODO H: Afisati istoricul unui client

    // TODO I: Anulare rezervare

    return 0;

}
```

Cerințe (10p)

1. (2p) Implementați toate funcțiile din `CinemaCity.cpp`.
2. (2p) Folosiți sortări cu lambdas pentru filme și pentru locuri.
3. (2p) Implementați recomandarea automată de locuri (lambda + sort).
4. (2p) Implementați gestionarea rezervărilor: rezervare, anulare, istoric.
5. (2p) Implementați testele în `main.cpp` conform TODO-urilor.

Compilare și rulare

```
g++ -std=c++17 -g *.cpp -o lab11
valgrind ./lab11
```

Dacă programul nu compilează, se scad 4p.

Bonus GitHub Classroom

Studentii pot obține până la **3p bonus** dacă încarcă toate laboratoarele până la finalul semestrului sau **1p** dacă trimit cel puțin jumătate.

1. Accesați cursul de pe OCW – Programare Orientată pe Obiecte.
2. Mergeți la secțiunea **Bonus GitHub laborator**.
3. Acceptați tema pentru Laborator 11.

Atenție: Repository-urile sunt monitorizate automat. Codul copiat între studenți anulează bonusul pentru toți participanții.