Házi feladat

Programozás alapjai 2.

Dokumentáció

Ónodi-Kiss Viktor

2025. május 17.

Tartalomjegyzék

1. Feladat

Készítsen filmeket nyilvántartó rendszert. Minden filmnek tároljuk a címét, lejátszási idejét és kiadási évét. A családi filmek esetében korhatár is van, a dokumentumfilmek esetében egy szöveges leírást is tárolunk. Tervezzen könnyen bővíthető objektummodellt a feladathoz! Demonstrálja a működést külön modulként fordított tesztprogrammal! A megoldáshoz **ne** használjon STL tárolót!

2. Feladatspecifikáció

Készíteni kell egy olyan rendszert, amely képes filmeket nyilvántartani, az alábbi jellemzőkkel:

- Minden filmhez tároljuk:
    - **Cím**
    - **Lejátszási idő**
    - **Kiadási év**
- A **családi filmek** esetén a rendszer tárolja a **korhatárt**.
- A **dokumentumfilmek** esetén a rendszer tárolja a filmekhez tartozó **szöveges leírást**.

A rendszer felépítésekor figyelembe vesszük az OO szemléletet.

A rendszernek képesnek kell lennie a következőkre: új film hozzáadása, film törlése, film módosítása, film keresése.

A kiadott feladatot kibővítem felhasználói bejelentkezési rendszerrel. A felhasználó csak keresni tud, az admin viszont törölni, szerkeszteni és hozzáadni is.

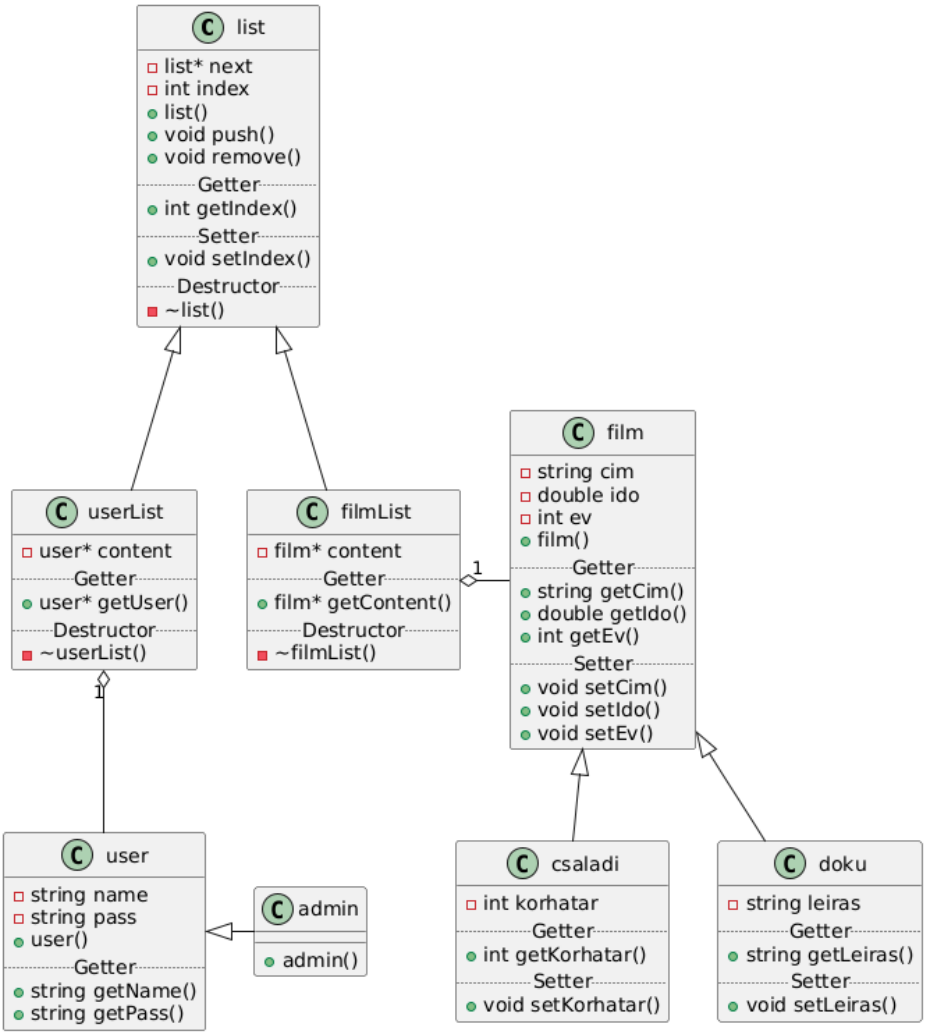
A filmek és a felhasználók listája egy-egy fájlban kerül eltárolásra. A program alapállapotában (első futtatás) ezek a fájlok még valószínű hogy nem léteznek, ezért a programnak létre kell tudnia hozni azokat.

Mivel az STL tárolók használata tilos, így egy dinamikus tömb kezelésen alapuló osztályt (listát) is implementálni kell.

3. Terv

A feladat elkészítéséhez szükség van 8 objektum és egy tesztprogram megtervezésére.

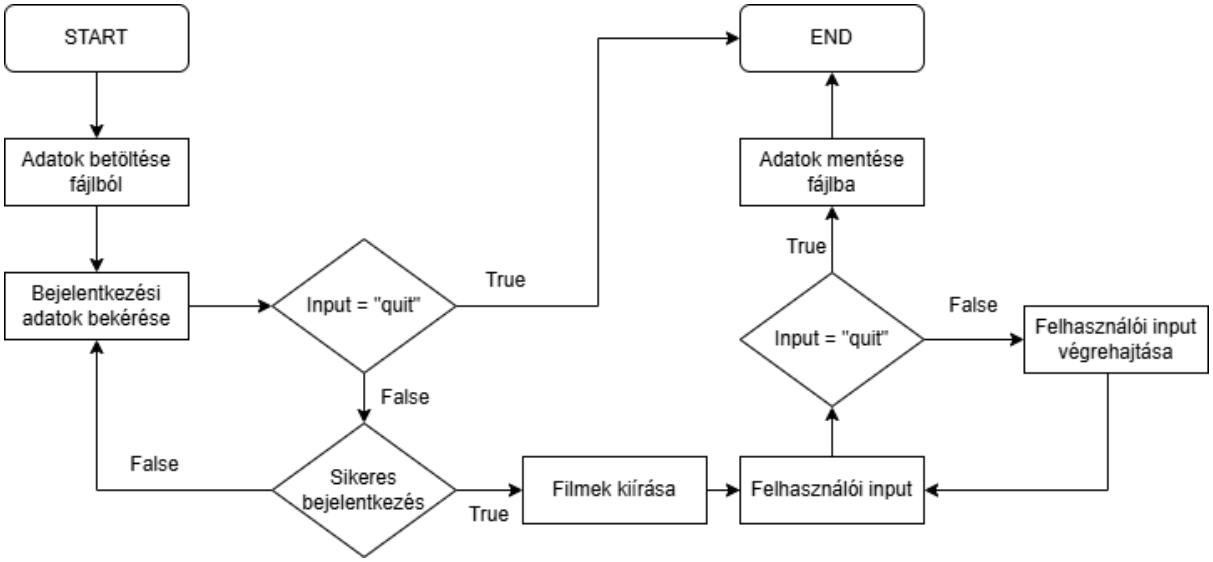
3.1. Objektum terv



1. ábra - A programhoz tartozó osztály diagram

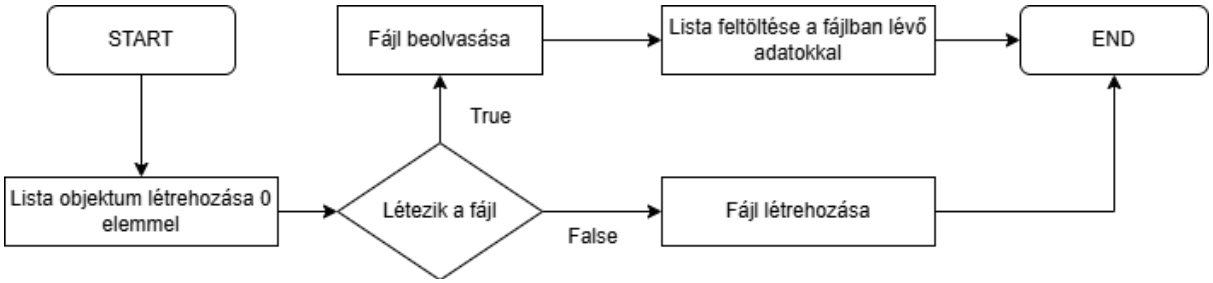
3.2. Algoritmusok

3.2.1. A fő ciklus



3. ábra - A program fő ciklusa

3.2.2. Fájl beolvasás



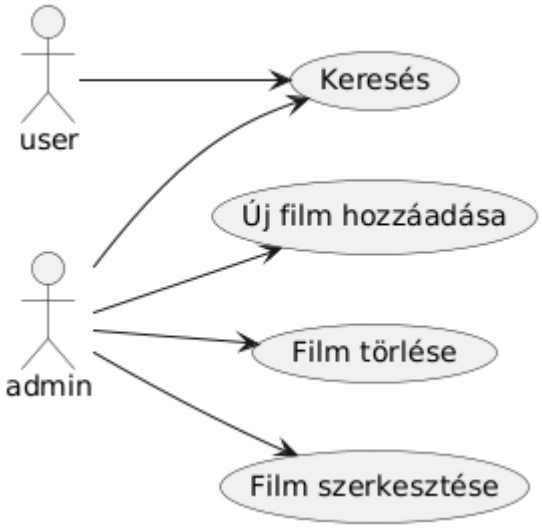
2. ábra - A fájl beolvasás és adatok betöltése

3.2.3. Tesztprogram algoritmusai

A tesztprogram a standard inputról file végéig olvas. Az első beolvasott adat egy teszteset sorszámot jelent. Ezt követően egy megjegyzés lehet az adott sorban. A beolvasott szám dönti el, hogy melyik teszteset fut a megjegyzés pedig az adott tesztesetre vonatkozhat.

A tesztprogram teszteli a hibás bemenetet is.

3.3. Usecase



4. ábra - A programhoz tartozó usecase diagram

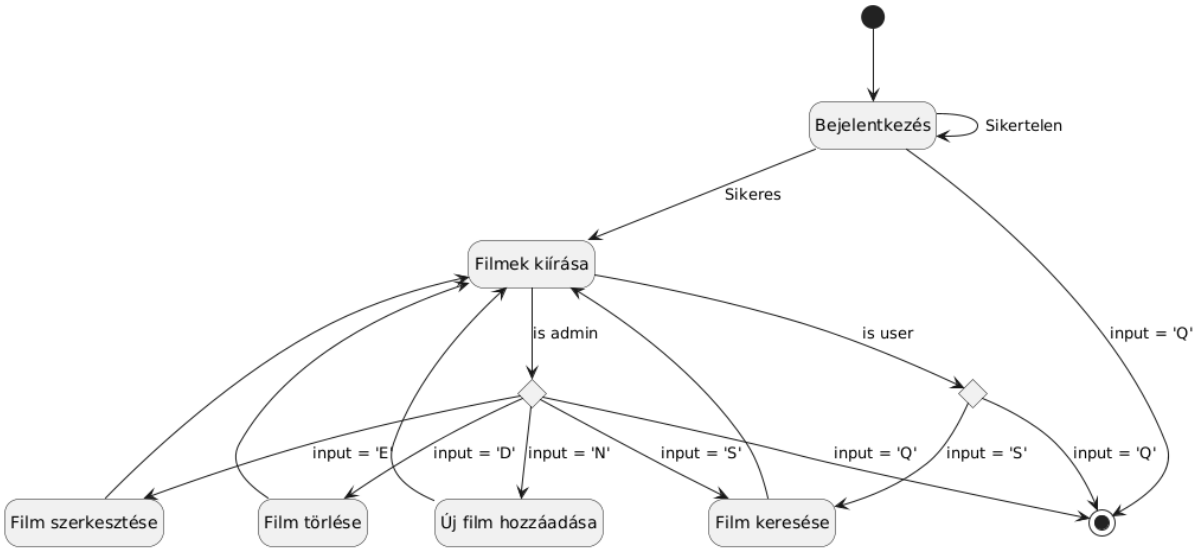
A program kettő felhasználó típust különböztet meg.

Az **admin** tud filmeket hozzáadni, törölni és a hozzájuk tartozó adaatokat szerkeszteni. Ezen túl tud keresni is.

A **user** csak keresni tud.

A filmek olvasása mind a kettő típusnak lehetséges.

3.4. Állapot diagram



5. ábra - A programhoz tartozó állapot diagram

4. Megvalósítás

A feladat megoldása 10 osztály elkészítését igényelte, a tesztelés pedig még ehhez 3 hozzáadását. Az osztályok megvalósítása többnyire követi a tervben leírtat, de több helyen módosításokat tartalmaz az Objektum-oriantált programozás megvalósításának érdekében. Az userlist és a filmlist osztályokat megszűntettem, helyettük a list osztály felhasználásával létrehoztam egy generikus list osztályt. Ezen kívül a Model-View-Persistence OOP tervezési séma megvalósítására létrehoztam a Model, View és a Persistence osztályokat.

4.1. Osztályok leírása

4.1.1. View

A Model-View-Persistence OOP séma szerint a View tartalmazza a felhasználói felület (CLI ebben az esetben) kezelésére szolgáló metódusokat.

Metódusok:

- **void writeOutFilms(list<film>*);**

  Kiírja az összes filmet a megadott listából

- **void writeOutFilm(film*, int);**

  Kiír egy filmet (az indexével)

- **String askForUsername();**

  Bekér egy felhasználónevet

- **String askForPass();**

  Bekér egy jelszót

- **String getInput(String);**

  Bekér adatot

- **void invalidInput(String);**

  Hibás bemenet esetén a paraméterként kapott String-et kiírja az outputra.

4.1.2. Model

Az Model-View-Persistence objektum-orientált séma szerint a Model tárolja a programhoz tartozó logikát

Változók:

- View view;
- Persistence per;
- list<user> *users;
- list<film> *films;

Metódusok:

- **void mainLoop();**

  A program folyamatos futásáért felelő "loop"

  Felhasználói inputre leáll, egyébként végtelenciklus

- **Answer inputMgmt(String);**

  Lekezeli a felhasználó által megadott input-ot

- **bool getPriv(list<user>*);**

  Bekéri, majd leellenőrzi a megadott felhasználónév és jelszó helyességét

- **void resetAll(list<film>*);**

  A keresés műveletnél eltűntetett lista elemeket megjeleníti

- **list<film>* addLogic(list<film>*);**

  Hozzáad új elemet a listához a felhasználó inputja alapján

- **list<film>* updateLogic(list<film>*);**

  Frissíti a listában lévő filmeket a felhasználó inputja alapján

- **list<film>* deleteLogic(list<film>*);**

  Kitörli a lista egy elemét a felhasználó inputja alapján

- **void searchLogic(list<film>*);**

  Eltűnteti az összes filmet, amiben nem szerepel a felhasználó által megadott szöveg

4.1.3. Persistence

Metódusok:

- **list<user>* getUsers(String fileN = "users.txt");**

  Beolvassa a felhasználókat és visszadja listában őket

- **list<film>* getFilms(String fileN = "films.txt");**

  Beolvassa a filmeket és visszaadja listában őket

- **void writeUsers(list<user>*);**

  Fájlba írja a paraméterként megadott user lista tartalmát

- **void writeFilms(list<film>*);**

  Fájlba írja a paraméterként megadott film lista tartalmát

4.1.4. list<T>

Változók:

- list* next;
- int index;
- T* content;
- bool show;

Metódusok:

- **void push(list<T>*);**

  Belehelyezi a paraméterként megadott objektumot a következő node-ba

- **void removenext();**

  Megsemmisíti a következő node-ot és a helyére rakja az azt követőt.

- **int getIndex();**
- **list* getNext();**
- **T* getContent();**
- **bool getShow();**
- **void setIndex(int idx);**
- **void setNext(list*);**
- **void setContent(T*);**
- **void setShow(bool);**

4.1.5. film

változók:

- String cim;
- double ido;
- int ev;
- int type;

Metódusok:

- **String getCim() const;**
- **double getIdo() const;**
- **int getEv();**

- **int getType();**
- **void setCim(String);**
- **void setIdo(double);**
- **void setEv(int);**

4.1.6. user

Változók:

- String name;
- String pass;

Metódusok:

- **String getName();**
- **String getPass();**
- **bool checkPass(String name, String pass);**

  Leellenőrzi, hogy a paraméterként megadott name és pass egyeznek-e az osztálypéldányban eltároltakkal.

4.2. Tesztprogram bemutatása

A tesztprogram a megfelelő bemenetre „TEST” fut le. Összesen 29 tesztet tartalmaz, a lehető legtöbb irányból lefedi a program működését.

A Tesztprogram 3 osztály tartalmaz:

- **CoutRedirect**

A View osztály tesztelésére való, CLI-be std::cout-on kimenő adatokat irányítja át.

- **CinRedirect**

A View osztály tesztelésére való, a CLI-be std::cin-on bejövő adatokat irányítja át.

- **TestClass**

A Tesztmetódusokat gyűjti össze.

5. Tesztelés

A program tesztelése során a memóriát ellenőriztem, hogy ne tartalmazzon memóriszívárgást, az interfész megfelelő működését, illetve a program funkcionalitását. A program manuális teszteléséhez felhasználhatóak az „admin” felhasználónév és az „admin123” jelszó.

5.1. Memóriakezelés tesztje

A memóriakezelés ellőnrzését a laborgyakorlatokon is használt MEMTRACE modullal végeztem. Ehhez minden modul fejlécében include-oltam a „memtrace.h” állományt. Nem tapasztaltam memóriakezlési hibát.

5.2. Interfész teszt

A 24-29.-ik tesztek szolgálnak az interface ki- és bemenetének tesztelésére a CoutRedirect és a CinRedirect osztályok felhasználásával.

A megfelelő inputra a megfelelő feladatot végezte el a program és a megfelelő programra az elvárt kimenetet írta ki. Az interfész megfelelően működik.

5.3. Funkcionalitás teszt

TESZTEK SORSZÁMA	TESZTELT OSZTÁLY
1-8	String
9-10	list
11-13	film
14	user
15-16	Persistence
17-23	Model
24-29	View

A tesztek során a program pontosan azt a működést demonstrálta, amit a megíráskor elvártam, a specifikációban leírt feladatot teljesíti.