NPRG041 – 2022/23 ZS – Cvičení MS – **Malý úkol C08**

Databáze filmů II

V tomto domácím úkolu se opět vrátíme k tématu databáze filmů. Stávající kód nejprve celý převezmeme a následně jej refaktorujeme, abychom vyhověli upraveným a nově přidaným požadavkům. To konkrétně znamená, že upravíme reprezentaci našich herců, nově začneme kromě filmů pracovat i se seriály a upravíme i kontejner, pomocí kterého simulujeme naši databázi jako takovou. Nakonec přidáme i vyhodnocení dvou dalších dotazů.

Nejprve se podívejme na změny v reprezentaci herců. Připomeňme, že ke každému filmu umíme přiřadit libovolně velkou množinu herců, kteří v něm hráli. To zachováme beze změny, ovšem zatímco doteď jsme herce reprezentovali jen pomocí jejich atomického jména (a tedy technicky pomocí řetězce `std::string`), nově budeme chtít o hercích uchovat více informací.

Za tímto účelem vytvoříme třídu `Actor`, o každém herci pak budeme mít konkrétně tyto údaje: křestní jméno (`std::string`), příjmení (`std::string`) a rok narození (`unsigned short`). Konstruktory a další metody ve třídě herců navrhněte dle potřeby, samozřejmě bychom ale neměli vybočovat z přístupu, který jsme už aplikovali nad třídou pro filmy.

Abychom s instancemi našich herců vůbec mohli pracovat v kontejneru množiny `std::set`, musíme nad nimi definovat vzájemné uspořádání. Toho lze dosáhnout např. tím, že implementujeme globální funkci `bool operator<(const Actor& actor1, const Actor& actor2)`, čímž definujeme chování operátoru `<` pro porovnávání dvojic herců. Z logického pohledu toto porovnávání bude definováno trojicí příjmení, křestní jméno a rok narození (v tomto pořadí).

Vypsání herce do určeného výstupního streamu vyřešíme pomocí implementace vlastního operátoru `<<`, tedy pomocí globální funkce `std::ostream& operator<<(std::ostream& os, const Actor& actor)`. Cílem je vypsat daného herce v podobě JSON objektu, např. `{ name: "Ivan", surname: "Trojan", year: 1964 }`. Opět věrně zachováme pořadí položek i oddělovače v podobě čárek a mezer. Na konci žádný konec řádku nevypisujeme.

Analogicky pro usnadnění importu herců ze vstupního streamu přidáme vlastní operátor `>>`, tedy implementujeme funkci `std::istream& operator>>(std::istream& is, Actor& actor)`. Očekáváme, že jednotlivé údaje o daném herci budou uvedeny opět ve stejném pořadí a že budou odděleny právě jednou mezerou, např. `Ivan Trojan 1964`. Abychom takovýto operátor čtení ze streamu mohli implementovat efektivně a jednotlivé datové položky herce mohly stále zůstat privátní, využijeme konstrukce `friend`.

Druhá hlavní změna ve stávajícím programu spočívá v tom, že nově budeme chtít kromě filmů pracovat i se seriály. Tuto změnu technicky provedeme tak, že nejprve současnou třídu `Movie` přejmenujeme na `Title` a následně od ní, jakožto společného abstraktního předka, odvodíme pomocí dědičnosti dvě specifické třídy, konkrétně `Movie` pro filmy a `Series` pro seriály.

V případě filmů chceme kromě společných údajů pro všechny tituly uchovávat také délku v minutách (`unsigned short`), v případě seriálů naopak počet sezón a celkový počet epizod (obojí `unsigned short`). Pro rozlišení obou typů titulů přidáme virtuální funkci `Type type() const` vracející hodnotu z enumerační třídy `enum class Type { MOVIE, SERIES }`. Ostatní metody a vhodné konstruktory doplňte dle potřeby.

Stávající funkci na vypisování titulů upravíme tak, aby první položkou ve vytvářeném JSON objektu byla položka určující typ titulu, konkrétně `{ type: "MOVIE", … }` v případě filmů a naopak `{ type: "SERIES", … }` v případě seriálů. Pole herců vypisujeme stejným způsobem jako minule, akorát každý herec je nově reprezentován vlastním JSON objektem, jak už jsme ostatně popsali. Specifické položky titulů umístíme až na konec. V případě filmů jde o jejich délku, např. `{ …, length: 112 }`, v případě seriálů o počet sezón a dílů, např. `{ …, seasons: 8, episodes: 73 }`. Pro zvýšení komfortu uživatelů nabídneme i vlastní operátor zápisu do streamu `<<` pro filmy jako takové.

Analogicky upravíme i stávající funkci na import filmů ze vstupních CSV dat. Nově předpokládáme, že u každého titulu bude prvním údajem selektor určující jeho typ, konkrétně `MOVIE;…` u filmů a `SERIES;…` u seriálů. Opět na konci budou uvedeny specifické údaje, tedy u filmů délka, např. `…;112`, u seriálů počet sezón a epizod, např. `…;8;73`.

Konečně třetí změnou bude částečně i vynucená úprava reprezentace naší simulované databáze jako takové, tedy kontejneru filmů. Nově totiž samozřejmě budeme potřebovat polymorfní kontejner umožňující pracovat s jakýmikoli tituly, tedy filmy i seriály. Tentokráte však už nevyužijeme klasické céčkové ukazatele,

ale chytré ukazatele, konkrétně ve variantně sdílených chytrých ukazatelů. To nám usnadní situaci z hlediska dynamické alokace a především dealokace jednotlivých instancí titulů. Nově tedy předpokládáme databázi titulů v podobě `std::vector<std::shared_ptr<Title>>`, jednotlivé instance titulů pak budeme vytvářet pomocí konstrukce `std::make_shared<Movie>(…)` pro filmy a analogicky pro seriály.

Oba existující databázové dotazy zachováme, a to beze změny jejich významu. To však bude obnášet následující dvě technické úpravy: v případě obou z nich budeme nově uvažovat všechny tituly (ne jen původní filmy) a konkrétně u dotazu `db_query_2` budeme v návaznosti na nové strukturované herce hledat dvojici `Ivan Trojan 1964` a `Tereza Voriskova 1989`.

Nakonec ještě implementujeme dva nové dotazy, technicky opět pomocí globálních funkcí ve stejném stylu jako minule:

- `void db_query_3(`
  `const std::vector<std::shared_ptr<Title>>& db,`
  `unsigned short seasons, unsigned short episodes`
  `)`: najdeme všechny seriály, které mají alespoň uvedený počet sezón `seasons` nebo alespoň uvedený počet epizod `episodes`; vypisovat budeme celé JSON objekty nalezených titulů
- `void db_query_4(`
  `const std::vector<std::shared_ptr<Title>>& db,`
  `const std::type_info& type, unsigned short begin, unsigned short end`
  `)`: najdeme všechny tituly uvedeného typu (film nebo seriál), které byly natočeny v uvedeném intervalu roků [`begin`, `end`), přičemž tento interval chápeme jako zprava otevřený; pro určení typu titulu `type` použijeme informační typové třídy `std::type_info` z knihovny `<typeinfo>` a konstrukce `typeid(…)`, kterou lze volat nad třídami i jejich instancemi; pro každý vyhovující titul vypíšeme jen jeho název; dodejme, že předpokládáme korektní interval roků, v opačném případě bude chování nedefinováno; pokud jsou hodnoty `begin` a `end` stejné, je daný interval prázdný

Jako již tradičně odevzdejte všechny vytvořené zdrojové soubory (`*.cpp` a `*.h`) kromě hlavního souboru `Main.cpp`. U něho opět předpokládejte direktivu `#include "Database.h"`. Cílem úkolu je ověřit schopnost implementace vlastních vybraných operátorů a také práce se sdílenými chytrými ukazateli používanými ve spojitosti s hierarchií tříd a polymorfním kontejnerem.