NPRG041 – 2023/24 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, ty stávající pak přizpůsobíme nové situaci.

Nejprve se zaměřme na změny v reprezentaci herců. Zatímco doteď jsme herce reprezentovali jen pomocí jejich atomického jména (a tedy technicky pomocí řetězce `std::string`), nově budeme chtít o hercích uchovávat více informací, navíc strukturovaně. 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 navrhneme analogicky jako u třídy pro filmy, konkrétně tedy budeme očekávat následující dvojici konstruktorů a rovněž metody `name`, `surname` a `year` pro přístup k jednotlivým položkám.

- `Actor(const std::string& name, const std::string& surname, unsigned short year)`
- `Actor(std::string&& name, std::string&& surname, unsigned short year)`

Abychom s instancemi herců vůbec mohli pracovat v kontejneru množiny `std::set`, musíme nad nimi nejprve definovat vzájemné uspořádání. Toho lze dosáhnout implementací globální funkce `bool operator<(const Actor& actor1, const Actor& actor2)`, čímž definujeme chování operátoru `<` pro porovnávání dvojic herců. Z logického pohledu bude toto porovnávání definováno trojicí příjmení, křestní jméno a rok narození (v tomto pořadí). Funkce vrátí `true` právě tehdy, když první herec bude předcházet druhého.

Vypsání herce do určeného (nebo standardního) výstupního streamu vyřešíme pomocí metody `void print_json(std::ostream& stream = std::cout) const` a nově rovněž i pomocí implementace vlastního operátoru `<<`, tedy pomocí globální funkce `std::ostream& operator<<(std::ostream& stream, 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.

Pro usnadnění importu herců ze vstupního streamu přidáme vlastní operátor `>>`, tedy implementujeme funkci `std::istream& operator>>(std::istream& stream, Actor& actor)`. Očekáváme, že jednotlivé údaje o daném herci budou uvedeny ve správném pořadí a že budou odděleny mezerami, 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 doplníme dle potřeby, nově přidané položky dáme vždy na konec, dodržíme jejich pořadí a také názvy přístupových metod `length`, `seasons` resp. `episodes`.

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 bude 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 tituly 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`. Pokud první položka neobsahuje platný selektor, vrátíme naši strukturovanou výjimku s kódem `2` a textem `Invalid type selector in field <type> on line <`*line*`>` s příslušným číslem řádku. Platné rozsahy nově přidaných číselných hodnot jsou následující: délka `0` až `300`, sezóny `0` až `100` a epizody `0` až `10000`.

Při parsování herců zcela prázdné herce přeskakujeme, detekci chybových situací založíme na chování konverze streamu na logickou hodnotu, a tedy u atributů `name` a `surname` budeme vracet výjimku `Missing attribute <`*attribute*`> in actor <`*actor*`> on line <`*line*`>` a u atributů roku narození `year` pouze jedinou sloučenou výjimku `Missing, invalid or overflow value in attribute <year> in actor <`*actor*`> on line <`*line*`>` nebo následně výjimku na hodnotu mimo povolený rozsah `1850` až `2100` v podobě `Integer out of range <`*min*`, `*max*`> in attribute <year> in actor <`*actor*`> on line <`*line*`>`. Ve všech případech půjde o naše strukturované výjimky s kódem `2`, kde *actor* nahradíme za celý vstupní řetězec daného herce. Napovězme rovněž, že textový obsah uvedených výjimek nemusíme vytvořit najednou aneb část vygenerujeme přímo ve funkci operátoru `>>` a zbytek doplníme později.

Pro jistotu se pojďme podívat na následující tři ilustrativní příklady:

- `MOVIE;Pres prsty;2019;comedy;56;Petra Hrebickova 1979,Jiri Langmajer 1966;101`
  je v pořádku,
- `MOVIE;Pres prsty;2019;comedy;56;Petra Hrebickova 1979,Jiri;101`
  vyvolá výjimku s textem `Missing attribute <surname> in actor <Jiri> on line <1>` a
- `MOVIE;Pres prsty;2019;comedy;56;Petra Hrebickova 1979,Jiri Langmajer NaN;101`
  vyvolá výjimku s textem `Missing, invalid or overflow value in attribute <year> in actor <Jiri Langmajer NaN> on line <1>`.

Konečně třetí změnou bude částečně i vynucená úprava reprezentace naší simulované databáze jako takové, tedy kontejneru filmů. Nově totiž 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 (včetně vypsání konce řádku za každým nalezeným titulem):

- ```
  void db_query_3(
    const std::vector<std::shared_ptr<Title>>& db,
    unsigned short seasons, unsigned short episodes,
    std::ostream& stream = std::cout
  ```
  ): 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,
    std::ostream& stream = std::cout
  ```
  ): 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 konstrukci `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 budeme opět předpokládat direktivy `#include` pro hlavičkové soubory `Database.h` a `Queries.h`. Cílem úkolu je ověřit schopnost implementace vybraných vlastních operátorů a také práce se sdílenými chytrými ukazateli používanými ve spojitosti s hierarchií tříd a polymorfním kontejnerem.