NPRG041 – 2025/26 ZS – Cvičení MS – **Malý úkol C07**

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 třech 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 uložené konkrétně tyto údaje: křestní jméno (`std::string`), příjmení (`std::string`) a rok narození (`int`). Konstruktory a další metody navrhneme analogicky jako u třídy pro filmy, konkrétně tedy budeme očekávat následující dvojici konstruktorů a rovněž getter metody `name`, `surname` a `year` pro přístup k jednotlivým privátním datovým položkám.

- `Actor(const std::string& name, const std::string& surname, int year)`
- `Actor(std::string&& name, std::string&& surname, int 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 konstrukci `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 nepovinnou délku v minutách (`int`). Pro její reprezentaci a vytváření hodnot použijeme typ `std::optional` resp. funkci `std::make_optional` nebo konstantu `std::nullopt` z knihovny `<optional>`. V případě seriálů naopak přidáme počet sezón a celkový počet epizod (obojí `int`). 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` resp. `seasons` a `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 }`. Pokud délka chybí, položku vůbec nevypíšeme. V případě seriálů jde 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.

Upravíme samozřejmě 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 úplném konci budou uvedeny specifické údaje, tedy konkrétně u filmů nepovinná délka, např. `…;112`, u seriálů počet sezón a epizod, např. `…;8;73`. Platné rozsahy nově přidaných číselných hodnot jsou následující: délka `0` až `300`, sezóny `0` až `100` a epizody `0` až `10000`. Pokud první položka neobsahuje platný selektor, vyhodíme následující strukturovanou výjimku:

- Kód `2` (parsovací chyby)
  - `Invalid type selector <`*selector*`> in field <type> on line <`*line*`>`:
    neplatná hodnota typového selektoru *selector*

Obecně očekáváme, že texty výjimek nově konstruujeme pomocí formátovaných řetězců `std::format` z knihovny `<format>` (přičemž refaktorujeme i existující výjimky z předchozího úkolu). Při parsování herců (tedy přímo uvnitř extrakčního operátoru `>>` pro herce) založíme detekci chybových situací na kontextuální konverzi streamu na logickou hodnotu. Konkrétně předpokládáme následující strukturované výjimky:

- Kód `2` (parsovací chyby)
  - `Missing attribute <`*attribute*`>`:
    chybí hodnota atributu křestního jména (`name`) nebo příjmení (`surname`)
  - `Missing, invalid, or overflow value in attribute <`*attribute*`>`:
    chybí hodnota atributu roku narození (`year`), nebo sice nechybí, ale nejde o validní číslo (bez rozlišení konkrétní příčiny)
  - `Integer <`*value*`> out of range <`*min*`, `*max*`> in attribute <`*attribute*`>`:
    hodnota atributu roku narození je mimo povolený rozsah `1850` až `2100`

V rámci procesu importu filmů pak textové zprávy těchto výjimek (se zachováním jejich kódu) doplníme o další informace popisující kontext vstupního filmu. Konkrétně přidáme `... in actor <`*actor*`> on line <`*line*`>`, kde *actor* nahradíme za celý vstupní řetězec daného herce a *line* je opět číslo řádku. Zcela prázdné herce budeme opět přeskakovat. Pro jistotu se ještě pojďme podívat na následující 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>`
- `MOVIE;Pres prsty;2019;comedy;56;Petra Hrebickova 1979,Jiri Langmajer NaN;101`
  vyvolá výjimku `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 aneb hledat 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 následující tři 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 database_t& database,
    Type type, int begin, int end,
    std::ostream& stream = std::cout
  ```
  ```

`)`: najdeme všechny tituly uvedeného typu `type` (film nebo seriál podle naší enumerace), které byly natočeny v uvedeném intervalu roků [`begin`, `end`), přičemž tento interval chápeme jako zprava otevřený; předpokládáme jen korektní intervaly roků, v opačném případě bude chování nedefinováno; pokud jsou hodnoty `begin` a `end` stejné, je daný interval prázdný; pro každý vyhovující titul vypíšeme textový řetězec v podobě JSON objektu `{ type: "`*selector*`", name: "`*name*`", year:` *year* `}`, kde *selector* nahradíme typem titulu (textovou hodnotou `MOVIE` nebo `SERIES`), *name* názvem titulu a *year* rokem jeho natočení

- ```
  void db_query_4(
    const database_t& database,
    int seasons, int episodes,
    std::ostream& stream = std::cout
  ```
  `)`: najdeme všechny seriály, které mají alespoň uvedený počet sezón `seasons` nebo alespoň uvedený počet epizod `episodes`; pro přetypování obecných titulů na seriály (přesněji řečeno ukazatelů na ně) je nutné použít statické přetypování pomocí konstruktů `static_cast<...>(...)` nebo případně také `std::static_pointer_cast<...>(...)`; pro každý vyhovující titul vypíšeme textový řetězec v podobě JSON objektu `{ name: "`*name*`", seasons:` *seasons*`, episodes:` *episodes* `}`, kde *name* nahradíme názvem titulu, *seasons* počtem sezón a *episodes* počtem epizod

- ```
  void db_query_5(
    const database_t& database,
    int length,
    std::ostream& stream = std::cout
  ```
  `)`: najdeme všechny filmy, jejichž délka je alespoň `length` minut; filmy, které délku uvedenou nemají, budeme ignorovat; pro přetypování je tentokrát naopak nutné použít dynamické přetypování pomocí konstruktů `dynamic_cast<...>(...)` nebo případně také `std::dynamic_pointer_cast<...>(...)` a korektně úspěšnost takové konverze otestovat; pro každý vyhovující titul opět vypíšeme textový řetězec v podobě JSON objektu `{ name: "`*name*`", length:` *length* `}`, kde *name* nahradíme názvem titulu a *length* jeho délkou

Jako obvykle 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`, skrze které musí být přímo či nepřímo dostupná veškerá očekávaná funkcionalita. Nadále také předpokládáme hlavičkový soubor `Storage.h` s definicí typového aliasu pro naši databázi.

Cílem úkolu je ověřit schopnost práce s vybranými vlastními operátory (zejména operátory zápisu do streamu a čtení ze streamu), sdílenými chytrými ukazateli používanými ve spojitosti s hierarchií tříd a polymorfním kontejnerem, třídou `std::optional`, formátovanými řetězci `std::format` a mechanismy statického i dynamického přetypování.

Z pohledu kvality kódu nezapomeňte pracovat s pojmenovanými konstantami. Jejich definice umisťujte do modulů, do kterých logicky patří, stejně tak je pojmenovávejte systematicky (např. pomocí vhodných prefixů). Aneb máme jich už velké množství, takže by už na první pohled mělo být zřejmé, čeho konkrétně se týkají (tituly, filmy, seriály, herci, …).

V rámci importu titulů je nutné pro zpracování jednotlivých herců opravdu používat jejich operátor extrakce ze streamu `>>`. Právě proto jsme jej ostatně navrhli. Uvnitř tohoto operátoru pak položky křestního jména, příjmení a roku narození opět zpracováváme pomocí jejich operátorů extrakce a získané hodnoty rovnou ukládáme do připravené výstupní instance herce. Dodejme nakonec, že i import filmů jako takových bychom analogicky mohli řešit přes jejich vlastní operátor extrakce, bohužel by to ale kvůli formátu CSV (a tedy jiným oddělovačům než bílým znakům) bylo komplikovanější a bez výrazných přínosů pro nás. Proto takovou změnu nemá smysl realizovat.

Při importu a vypisování konkrétních variant titulů, tedy filmů a seriálů, se vyvarujte opakování stejného nebo podobného kódu. Jinými slovy společné položky (patřící do každého titulu) řešíme jen jednou, ne opakovaně. V případě vypisování titulů do JSON objektů pak využijte výhod, které přináší mechanismus virtuálních metod. A přestože obecné tituly jako takové jsou abstraktní, i tak by jejich hypotetické vypsání mělo vést k dobře formovanému a smysluplnému JSON výstupu.

Jelikož jsme už nyní seznámeni s pohledy na řetězce `std::string_view`, zkuste zvážit, jestli by nebylo rozumné je na vhodných místech použít (kromě závazně daného rozhraní, tedy jen pokud jde o vaše interní metody, ať už nově nebo refaktorováním stávajícího kódu). Na úplný konec ještě připomeňme náš požadavek

týkající se databázového dotazu `db_query_2`, kdy ověřování přítomnosti herců je i nadále nutné zvládnout v logaritmickém čase.