NPRG041 – 2022/23 ZS – Cvičení MS – **Malý úkol C09**

Databáze filmů III

V rámci posledního úkolu v tematickém celku databáze filmů vyjdeme ze stávající aplikace a rozšíříme ji přidáním několika pomocných indexů a hlavně dotazů. Smyslem těchto indexů jakožto pomocných datových struktur založených na nejrůznějších kontejnerech bude simulovat tradiční databázové indexy, a tedy umožnit efektivnější vyhodnocování našich databázových dotazů.

Zmíněné indexy konkrétně vytvoříme tři: nazvěme je index titulů, roků a herců. V uvedeném pořadí je realizujeme pomocí standardních kontejnerů `std::map`, `std::multimap` a `std::unordered_multimap`. Ve všech případech předpokládáme, že prázdná instance daného indexu bude předem připravena ve funkci `main`, naším úkolem bude implementovat následující globální funkce, pomocí kterých daný index naplníme potřebnými údaji na základě aktuálního stavu databáze. Aneb patřičným způsobem zaindexujeme každý titul, který v kontejneru databáze v daný okamžik existuje.

- `void db_index_titles(`
  `const std::vector<std::shared_ptr<Title>>& db,`
  `std::map<std::string, std::shared_ptr<Title>>& index`
  `)`: index umožňující vyhledávat tituly na základě jejich (unikátních) názvů
- `void db_index_years(`
  `const std::vector<std::shared_ptr<Title>>& db,`
  `std::multimap<unsigned short, std::shared_ptr<Title>>& index`
  `)`: index pro vyhledávání titulů na základě roků jejich natočení; předpokládáme, že při vytvoření kontejneru tohoto indexu bude jako porovnávací funktor použit `std::less<unsigned short>`; ten již existuje, a tak jej implementovat samozřejmě nebudeme
- `void db_index_actors(`
  `const std::vector<std::shared_ptr<Title>>& db,`
  `std::unordered_multimap<Actor, std::shared_ptr<Title>>& index`
  `)`: index umožňující vyhledávat tituly podle herců, kteří v nich hráli; při vytvoření kontejneru tohoto indexu bude jako hašovací funktor použit `std::hash<Actor>` a analogicky `std::equal_to<Actor>` jako porovnávací funktor; první uvedený neexistuje, a tedy jej budeme potřebovat implementovat sami (v hlavičkovém souboru) jako specializaci šablony hašovacího funktoru, tedy v podobě struktury `template<> struct std::hash<Actor> { … }`, v rámci které naprogramujeme jedinou metodu, a to operátor kulatých závorek `()` v podobě členské funkce `std::size_t operator()(const Actor& actor) const noexcept`, v rámci které jen vrátíme zahašovanou hodnotu založenou na příjmení herce, k čemuž využijeme existující funktor `std::hash<std::string>`; pokud jde o porovnávací funktor `std::equal_to<Actor>`, stačí jen implementovat operátor testu rovnosti `==` v podobě globální funkce, tedy v podobě `bool operator==(const Actor& actor1, const Actor& actor2)`

Všechny čtyři stávající databázové dotazy zachováme, ve stejném stylu přidáme i následující nové dotazy. Z hlediska rozhraní jim však už nebudeme předávat referenci na celou databázi, ale jen příslušný index diskutovaný výše. Nadále platí, že pro každý nalezený titul vypíšeme výsledek v požadovaném formátu na standardní výstup, případně vypíšeme hlášku, že se žádný vyhovující záznam najít nepodařilo. Vypsání každého záznamu opět ukončíme koncem řádku.

- `void db_query_5(`
  `const std::map<std::string, std::shared_ptr<Title>>& index,`
  `const std::string& name`
  `)`: na základě indexu titulů najdeme konkrétní titul, který má název `name`; pokud jej najdeme, vypíšeme jej ve tvaru `"name" -> titul`, kde `name` nahradíme hodnotou požadovaného jména a `titul` nahradíme kompletním JSON objektem daného titulu; pokud hledaný titul neexistuje, vypíšeme jen `"name" -> Not found!`, kde `name` opět bude nahrazeno za hledaný název
- `void db_query_6(`
  `const std::multimap<unsigned short, std::shared_ptr<Title>>& index,`
  `unsigned short year`

): pomocí indexu roků najdeme všechny tituly, které byly natočeny v roce `year`; každý takový titul vypíšeme ve formátu `year -> "name"`, kde `year` nahradíme za hodnotu hledaného roku a `name` za název titulu; pokud žádný vyhovující neexistuje, vypíšeme po nahrazení analogicky `year -> Not found!`

- ```
  void db_query_7(
    const std::multimap<unsigned short, std::shared_ptr<Title>>& index,
    unsigned short begin, unsigned short end
  ```
  ): pomocí stejného indexu tentokráte najdeme všechny tituly, které byly natočeny v roce patřícím do zprava otevřeného intervalu [`begin`, `end`); nalezené tituly vypíšeme ve stejném formátu jako u předchozího dotazu; pokud žádný nenajdeme, vypíšeme po nahrazení `[begin, end) -> Not found!`
- ```
  void db_query_8(
    const std::unordered_multimap<Actor, std::shared_ptr<Title>>& index,
    const Actor& actor
  ```
  ): na základě indexu herců najdeme všechny tituly, ve kterých hrál herec `actor`; nalezené záznamy vypíšeme ve tvaru `actor -> "name"`, kde `actor` nahradíme JSON objektem herce a `name` názvem nalezeného titulu; pokud nic nenajdeme, vypíšeme po nahrazení `actor -> Not found!`

Nakonec přidáme ještě následující nové dotazy. V jejich případě však již na standardní výstup nic vypisovat nebudeme, nalezený nebo spočítaný výsledek totiž vždy předáme pomocí příslušného výstupního parametru volajícímu. Používané standardní algoritmy najdeme v knihovně `<algorithm>`.

- ```
  void db_query_9(
    const std::vector<std::shared_ptr<Title>>& db,
    std::vector<std::shared_ptr<Title>>& result,
    const Actor& actor
  ```
  ): cílem je opět najít tituly, ve kterých hrál herec `actor`; toho tentokráte dosáhneme tak, že nejprve úplně všechny tituly z databáze (myšleno sdílené chytré ukazatele na ně) překopírujeme do předem připraveného prázdného výstupního kontejneru vektoru, a to pomocí funkce `std::copy`; následně pomocí funkce `std::remove_if` odebereme ty tituly, které nám nevyhovují, a to pomocí predikátu v podobě funktoru implementujícího metodu `bool operator()(const std::shared_ptr<Title>& title_ptr)`; nakonec všechny ponechané tituly, tedy ty hledané, uspořádáme primárně sestupně podle roku natočení a sekundárně vzestupně podle jména, a to pomocí funktoru implementujícího metodu simulující chování operátoru `<`, konkrétně v podobě metody `bool operator()(const std::shared_ptr<Title>& title_ptr_1, const std::shared_ptr<Title>& title_ptr_2)`
- ```
  void db_query_10(
    const std::vector<std::shared_ptr<Title>>& db,
    std::vector<std::shared_ptr<Title>>& result,
    const std::string& genre
  ```
  ): najdeme všechny tituly mající žánr `genre`, a to překopírováním vyhovujících titulů do předem připraveného prázdného výstupního kontejneru vektoru pomocí funkce `std::copy_if` a predikátu implementovaného pomocí lambda výrazu; nakonec filmy opět seřadíme, a to opět pomocí lambda výrazu simulujícího chování operátoru `<`, přičemž tituly chceme řadit vzestupně podle jejich názvů
- ```
  void db_query_11(
    const std::vector<std::shared_ptr<Title>>& db,
    Type type, const std::string& genre,
    int& result
  ```
  ): cílem je spočítat celočíselné průměrné hodnocení titulů majících typ `type` a žánr `genre`; takto získanou hodnotu uložíme do připraveného výstupního parametru `result`; pro zpracování jednotlivých titulů použijeme funkci `std::for_each`, přičemž zpracování konkrétního filmu provedeme pomocí vlastního funktoru implementujícího metodu `void operator()(const std::shared_ptr<Title>& title_ptr)`
- ```
  void db_query_12(
    const std::vector<std::shared_ptr<Title>>& db,
    const std::string& genre,
    int& result
  ```
  ): zjistíme celkový počet titulů majících žánr `genre`; výsledek uložíme do připraveného výstupního

parametru `result`; opět využijeme funkci `std::for_each`, tentokráte ale zpracování konkrétního titulu budeme řešit pomocí lambda výrazu

Odevzdejte všechny vytvořené zdrojové soubory (`*.cpp` a `*.h`) kromě hlavního souboru `Main.cpp`. U něho opět předpokládejte direktivu `#include "Database.h"`. Samozřejmě se očekává dodržení obvyklých požadavků pro naše úkoly. Pokud v rámci jednotlivých dotazů bylo předepsáno použití určitých konstruktů nebo funkcí, je nutné takový záměr dodržet.

Cílem tohoto úkolu je ověřit schopnost práce se standardními kontejnery `std::map`, `std::multimap` a `std::unordered_multimap` nad vlastními třídami, strukturou `std::pair`, předdefinovanými funktory `std::less`, `std::hash` a `std::equal_to`, standardními algoritmy nad kontejnery, konkrétně pak funkcemi `std::copy`, `std::copy_if`, `std::remove_if`, `std::sort` a `std::for_each`, stejně jako funktory a lambda výrazy v obecné rovině.