NPRG041 – 2024/25 ZS – Cvičení MS – **Malý úkol C09**

Databáze filmů III

V rámci třetího 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 standardních kontejnerech, bude simulovat tradiční databázové indexy, a tedy umožnit efektivnější vyhodnocování našich dotazů.

Zmíněných indexů konkrétně vytvoříme pět: nazvěme je index titulů podle názvů, herců podle roků, titulů podle roků, titulů podle herců a hereckého obsazení podle žánrů. Pro jejich realizaci využijeme standardních kontejnerů `std::map`, `std::multimap` a `std::unordered_multimap` (z knihoven `<map>` resp. `<unordered_map>`). 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 očekávanými záznamy na základě aktuálního obsahu databáze.

- `void db_index_titles_by_names(`
  `const std::vector<std::shared_ptr<Title>>& db,`
  `std::map<std::string, std::shared_ptr<Title>>& index`
  `)`: index umožní vyhledávat tituly na základě jejich (unikátních) názvů
- `void db_index_actors_by_years(`
  `const std::vector<std::shared_ptr<Title>>& db,`
  `std::map<unsigned short, std::set<Actor>>& index`
  `)`: index pro vyhledávání herců podle roků jejich narození; při vkládání záznamů do tohoto indexu se očekává použití operátoru `[]` na vnější mapě
- `void db_index_titles_by_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 nebudeme
- `void db_index_titles_by_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 `size_t operator()(const Actor& actor) const noexcept`, v rámci které jen vrátíme zahašovanou hodnotu založenou na příjmení herce, k čemuž využijeme instanci existujícího funktoru `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& actor_1, const Actor& actor_2)`
- `void db_index_cast_by_genres(`
  `const std::vector<std::shared_ptr<Title>>& db,`
  `std::multimap<`
  `std::tuple<std::string, unsigned short>,`
  `std::tuple<std::string, std::string, std::shared_ptr<Title>>`
  `>& index`
  `)`: index umožní ukládat herecké obsazení v titulech na základě jejich žánrů a roků, konkrétně v podobě trojic (křestní jméno herce, příjmení herce, ukazatel na daný titul) na základě dvojic (žánr titulu, rok natočení titulu); pro reprezentování těchto dvojic resp. trojic využijeme strukturu `std::tuple` z knihovny `<tuple>`

Všechny čtyři stávající databázové dotazy zachováme beze změny, 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 do uvedeného výstupního streamu. 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,`
  `std::ostream& stream = std::cout`
  `)`: na základě indexu titulů podle názvů najdeme konkrétní titul, který má název `name`; pokud jej najdeme, vypíšeme jej ve tvaru "*name*" `->` *title*, kde *name* nahradíme hodnotou požadovaného jména a *title* 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::map<unsigned short, std::set<Actor>>& index,`
  `unsigned short begin, unsigned short end,`
  `std::ostream& stream = std::cout`
  `)`: na základě indexu herců podle roků spočítáme celkový počet herců narozených během roků patřících do zprava otevřeného intervalu [`begin`, `end`); výsledek vypíšeme ve tvaru `[`*begin*`,` *end*`) ->` *count* `actors` nebo `actor` podle zjištěného počtu *count*

- `void db_query_7(`
  `const std::multimap<unsigned short, std::shared_ptr<Title>>& index,`
  `unsigned short year,`
  `std::ostream& stream = std::cout`
  `)`: pomocí indexu titulů podle 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_8(`
  `const std::multimap<unsigned short, std::shared_ptr<Title>>& index,`
  `unsigned short begin, unsigned short end,`
  `std::ostream& stream = std::cout`
  `)`: 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!`

Dále implementujeme dotaz, který umožní nalezení titulů na základě obecné vyhledávací podmínky implementované v podobě samostatné funkce.

- `void db_query_9(`
  `const std::vector<std::shared_ptr<Title>>& db,`
  `const std::function<bool(const Title*)>& predicate,`
  `std::ostream& stream = std::cout`
  `)`: najde všechny tituly, které splňují selekční podmínku, kterou vyhodnotí předaná funkce `predicate`; tato funkce očekává jediný parametr v podobě nemodifikujícího céčkového observer ukazatele na titul, pro který se podmínka má vyhodnotit, přičemž tato funkce vrátí hodnotu `true` právě tehdy, když daný titul ve výsledku dotazu zahrnout chceme; abychom mohli pro tyto podmínky používat nejenom obyčejné funkce, ale také funktory nebo později i lambda výrazy, předáme tento parametr pomocí struktury `std::function` z knihovny `<functional>`; každý nalezený titul vypíšeme v podobě "*name*", kde *name* nahradíme názvem daného titulu; pokud žádný titul nenajdeme, vypíšeme jen `Not found!`

Pro předchozí dotaz pak ještě připravíme následující dvě konkrétní funkce s podmínkami. První z nich realizujeme v podobě obyčejné globální funkce, druhý v podobě funktoru.

- Globální funkce `bool predicate_Q9_movies(const Title* title)`: najde všechny filmy, ve kterých hráli alespoň 3 herci
- Funktor `Predicate_Q9_Titles` implementující operátor `()` v podobě členské funkce `bool operator() (const Title* title) const`: najde všechny tituly s hodnocením alespoň 80, ve kterých hrála `Tatiana Vilhelmova 1978`

Nakonec přidáme ještě následující nové dotazy. V jejich případě však již na výstup nic vypisovat nebudeme, nalezený nebo spočítaný výsledek totiž vždy volajícímu předáme formou návratové hodnoty.

- ```
  std::vector<Title*> db_query_10(
    const std::unordered_multimap<Actor, std::shared_ptr<Title>>& index,
    const std::string& surname
  ):
  ```
  na základě indexu titulů podle herců najdeme všechny tituly, ve kterých hrál herec s příjmením `surname`; výsledek vrátíme v podobě vektoru céčkových observer ukazatelů na odpovídající tituly; v rámci iterace přes jednotlivé záznamy indexu se očekává použití strukturované vazby `auto&& [key, value]`
- ```
  std::vector<std::string> db_query_11(
    const std::multimap<
      std::tuple<std::string, unsigned short>,
      std::tuple<std::string, std::string, std::shared_ptr<Title>>
    >& index,
    const std::string& genre, unsigned short year
  ):
  ```
  pomocí posledního definovaného indexu hereckého obsazení podle žánrů najde jména herců a názvy titulů v titulech s žánrem `genre` natočených v roce `year`; každý nalezený záznam vrátíme v podobě řetězce `std::string`, který bude obsahovat JSON objekt v následujícím formátu `{ name: "`*name*`", surname: "`*surname*`", title: "`*title*`" }`, kde *name* je křestní jméno herce, *surname* jeho příjmení a konečně *title* název daného titulu

Během práce na úkolu budete používat aktuální verzi celého vašeho projektu databáze filmů. Odevzdávat však budete jen modul pro databázové dotazy jako takové. Konkrétně tedy nejspíše odevzdáte jen hlavičkový soubor `Queries.h` a tomu odpovídající céčkový soubor `Queries.cpp`. Žádné další soubory původního projektu už odevzdávat nebudete. Jinými slovy se vaše dotazy budou vyhodnocovat vůči implementaci kompletní databáze obsažené v připraveném testu, nikoli té vaší.

Test samotný bude obsahovat direktivu `#include` pouze a jenom pro hlavičkový soubor `Queries.h`. V rámci něho můžete (kromě potřebných standardních knihoven) mít `#include` jen na soubor `Movie.h`, prostřednictví kterého získáte přístup ke všemu potřebnému, tedy zejména všemu týkajícímu se titulů, filmů, seriálů a herců. V jejich případě to znamená i implementaci operátoru porovnávání `==` (ten tedy také odevzdávat nebudete).

Očekává se 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, strukturami `std::pair`, `std::tuple` a `std::function`, předdefinovanými funktory `std::less`, `std::hash` a `std::equal_to` a funktory jako takovými v obecné rovině.