NPRG041 – 2025/26 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.

Hned na úvod poznamenejme, že pro úspěšné vypracování této úlohy ve skutečnosti není potřeba mít kompletně implementovaný předcházející úkol v této tematické sérii. Implementace samotné databáze se totiž odevzdávat nebude, jen nově přidané indexy a dotazy. Postačí se tedy seznámit s veřejným rozhraním tříd `Title`, `Movie`, `Series` a `Actor`. Tím se myslí rozhraní jejich konstruktorů, getter metod pro datové položky, možnosti jejich vypisování pomocí operátorů `<<` a rozlišení typů titulů pomocí enumerace `Type`.

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>`) a pro přehlednější práci s nimi rovněž budeme předpokládat následující typové aliasy.

- `using db_index_titles_by_names_t = std::map<`
  `std::string, std::shared_ptr<Title>`
  `>`: index umožní vyhledávat tituly na základě jejich (unikátních) názvů
- `using db_index_actors_by_years_t = std::map<`
  `int, std::set<Actor>`
  `>`: index pro vyhledávání herců podle roků jejich narození; při vkládání záznamů do tohoto indexu se očekává použití operátoru `[]` na vnější mapě
- `using db_index_titles_by_years_t = std::multimap<`
  `int, std::shared_ptr<Title>`
  `>`: index pro vyhledávání titulů na základě roků jejich natočení; předpokládáme, že při vytvoření kontejneru tohoto indexu bude jako porovnávací funktor použit `std::less<int>`; ten již existuje, a tak jej implementovat nebudeme
- `using db_index_titles_by_actors_t = std::unordered_multimap<`
  `Actor, std::shared_ptr<Title>`
  `>`: 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 `bool operator==(const Actor& actor_1, const Actor& actor_2)`; vlastní porovnání realizujeme pomocí již implementovaného operátoru `==` nad n-ticemi `std::tuple` z knihovny `<tuple>`, které pro tento účel uměle vytvoříme pomocí funkce `std::tie`
- `using db_index_cast_by_genres_t = std::multimap<`
  `std::tuple<std::string, int>,`
  `std::tuple<std::string, std::string, std::shared_ptr<Title>>`
  `>`: 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)

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_1(const database_t& database, db_index_titles_by_names_t& index)`
- `void db_index_2(const database_t& database, db_index_actors_by_years_t& index)`
- `void db_index_3(const database_t& database, db_index_titles_by_years_t& index)`
- `void db_index_4(const database_t& database, db_index_titles_by_actors_t& index)`
- `void db_index_5(const database_t& database, db_index_cast_by_genres_t& index)`

Všech pět stávajících databázových dotazů 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_6(`
  `const index_titles_by_names_t& index,`
  `std::string_view name,`
  `std::ostream& stream = std::cout`
  `)`: na základě indexu titulů podle názvů najdeme konkrétní titul, který má název `name`; pokud jej najdeme, vypíšeme jeho kompletní JSON objekt; jinak vypíšeme jen hlášku `Not found!`

- `void db_query_7(`
  `const index_actors_by_years_t& index,`
  `int begin, int 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 *count* `actors` nebo `actor` podle zjištěného počtu *count* (např. `7 actors` nebo `1 actor`)

- `void db_query_8(`
  `const index_titles_by_years_t& index,`
  `int year,`
  `std::ostream& stream = std::cout`
  `)`: pomocí indexu titulů podle roků najdeme všechny tituly, které byly natočeny v roce `year`; každý takový titul vypíšeme jako řetězec obsahující JSON objekt `{ name: "`*name*`", year: `*year*` }`, kde *name* nahradíme za název titulu a *year* za rok jeho natočení; pokud žádný vyhovující titul neexistuje, vypíšeme jen hlášku `Not found!`

- `void db_query_9(`
  `const index_titles_by_years_t& index,`
  `int begin, int 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, opět jen vypíšeme `Not found!`

V dalších dvou dotazech se obejdeme bez indexů aneb budeme v nich opět pracovat přímo s databází. V prvním z nich si vyzkoušíme práci s kontejnerem, který bude využívat jiný než výchozí funktor na řazení. Druhý pak umožní nalezení titulů na základě obecné vyhledávací podmínky implementované v podobě samostatné funkce nebo funktoru.

- `void db_query_10(`
  `const database_t& database,`
  `int begin, int end,`
  `std::ostream& stream = std::cout`
  `)`: najdeme každé jednotlivé obsazení herců narozených v letech patřících do intervalu [`begin`, `end`); jinými slovy projdeme všechny tituly a v rámci nich najdeme všechny vyhovující herce; zajímají nás jen herci jako takoví, všechny duplicity zachováme; jelikož navíc chceme tyto herce vypsat v určitém pořadí (jiném než výchozím, tedy v tom námi definovaném v předchozím úkolu), budeme je nejprve

interně ukládat do pomocného kontejneru multimnožiny `std::multiset`, ve které využijeme náš vlastní funktor pro řazení prvků; konkrétně půjde o třídu `Comparator_Q10_Actors` s veřejnou metodou `void operator()(const Actor& actor_1, const Actor& actor_2) const`, která bude simulovat chování operátoru `<` pro řazení vzestupně podle roků narození, křestních jmen a příjmení; pro její elegantní implementaci opět využijeme uměle vytvořené n-tice `std::tuple`, tentokráte získané pomocí funkce `std::make_tuple`; pro každého nalezeného herce vypíšeme jeho úplný JSON objekt; pokud žádný neexistuje, vypíšeme hlášku `Not found!`

- `void db_query_11(`
  `const database_t& database,`
  `const std::function<bool(const Title*)>& predicate,`
  `std::ostream& stream = std::cout`
  `)`: najdeme všechny tituly, které splňují obecnou 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 jako jeho úplný JSON objekt; pokud žádný nenajdeme, vypíšeme `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. Ani v jednom případě pak pro uvedené konkrétní hodnoty pojmenované konstanty nevytvářejte, bylo by to proti smyslu těchto funkcí.

- Globální funkce `bool predicate_Q11_movies(const Title* title)`: najdeme všechny filmy (nikoli seriály aneb chceme jen tituly s typem `Type::MOVIE`), ve kterých hráli alespoň tři herci a nejsou to komedie (`comedy`)
- Funktor `Predicate_Q11_Titles` implementující metodu `bool operator()(const Title* title) const`: najdeme všechny tituly s hodnocením alespoň 80, ve kterých hrála `Tatiana Vilhelmova 1978`

Nakonec přidáme ještě následující dva dotazy. Vrátíme se zpět k práci s indexy, navíc již nebudeme nic vypisovat do výstupního streamu, protože nalezené nebo spočítané výsledky volajícímu vždy předáme formou návratové hodnoty.

- `std::vector<Title*> db_query_12(`
  `const index_titles_by_actors_t& index,`
  `std::string_view surname`
  `)`: na základě indexu titulů podle herců najdeme všechny tituly, ve kterých hrál herec s uvedeným 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_13(`
  `const index_cast_by_genres_t& index,`
  `std::string_view genre, int year`
  `)`: pomocí posledního definovaného indexu hereckého obsazení podle žánrů najdeme jména herců a názvy titulů v titulech s žánrem `genre` natočených v roce `year`; každý nalezený záznam vrátíme jako řetězec `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

Jak již bylo vysvětleno v úvodu, během práce na úkolu budete používat aktuální verzi svého projektu databáze filmů (tedy pokud jej máte). Odevzdávat však budete jen modul pro databázové dotazy jako takové. Přesněji řečeno odevzdáte pouze dotazy Q6 až Q13, všechny indexy a související kód. Odevzdáte-li také starší dotazy Q1 až Q5, nic se nestane, testovány ale nebudou. 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, kterou jste si sami vytvořili.

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 `Storage.h`, prostřednictvím kterého získáte přístup ke všemu potřebnému, tedy zejména všemu týkajícímu se titulů, filmů, seriálů a herců.

V rámci tohoto úkolu máme mimo jiné naprogramovat i specializaci hašovacího funktoru `std::hash` a operátor testu rovnosti `==` pro naše herce. Z povahy věci by pochopitelně bylo nejrozumnější umístit je do modulu pro herce. Ten však odevzdáváte nebudete, a tedy je z důvodu možnosti otestování umístěte do modulu dotazů.

Očekává se dodržení obvyklých požadavků a postupů pro naše úkoly. Pokud v rámci jednotlivých indexů nebo dotazů bylo předepsáno použití určitých konstruktů nebo funkcí, je nutné takový záměr dodržet. Cílem tohoto úkolu je ověřit schopnost práce s dalšími standardními kontejnery `std::map`, `std::multimap`, `std::unordered_multimap` a `std::multiset` 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.