NPRG041 – 2024/25 ZS – Cvičení MS – **Malý úkol C12**

Databáze filmů IV

Cílem posledního úkolu v tématu databáze filmů bude další rozšíření stávajícího projektu o několik nových dotazů, v rámci kterých budeme používat nejrůznější standardní algoritmy (knihovna `<algorithm>`), funktory, lambda výrazy, falešné iterátory (`<iterator>`) a také rozsahy a pohledy (`<ranges>`).

Jelikož už máme z předchozích úkolů implementovaný náš vlastní kontejner gumového pole, byla by škoda jej nepoužít. Jinými slovy současné pojetí databáze v podobě `std::vector<std::shared_ptr<Title>>` nahradíme za `lib::Array<std::shared_ptr<Title>>`. Tedy alespoň v případě, že gumové pole opravdu máte a máte jej dostatečně odladěné. V každém případě do projektu přidáme nový hlavičkový soubor `Storage.h`, do kterého kromě `#include` direktiv pro všechny potřebné knihovny umístíme jedinou věc, a to definici typového aliasu `database_t`, prostřednictvím kterého si explicitně vybrané úložiště pro naši databázi sami zvolíte. Veškerý stávající kód (funkce pro existující dotazy, vytváření indexů, import databáze atp.) pak upravíte tak, aby pracoval právě s tímto aliasem.

Následně do projektu přidáme konkrétně následujících pět nových dotazů. Aby se opravdu naplnil cíl celého úkolu, je potřeba všechny předepsané konstrukty i postupy věrně dodržet.

- `std::vector<std::shared_ptr<Title>> db_query_12(`
  `const database_t& database,`
  `const Actor& actor`
  `)`: cílem je najít tituly, ve kterých hrál uvedený herec `actor`; toho dosáhneme tak, že nejprve úplně všechny tituly z databáze (myšleno sdílené chytré ukazatele na ně) překopírujeme do námi připraveného výstupního kontejneru `std::vector`, 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 `Predicate_Q12_Actor`; nakonec všechny aktuálně zbývající tituly, tedy ty hledané, seřadíme primárně sestupně podle roku natočení a sekundárně vzestupně podle jejich jména, a to pomocí funkce `std::sort` a funktoru `Comparator_Q12_Years` simulujícího chování operátoru `<`

- `void db_query_13(`
  `const database_t& database,`
  `unsigned short year,`
  `std::ostream& stream = std::cout`
  `)`: najdeme všechny filmy (tituly mající typ `Type::MOVIE`) natočené v roce `year`, a to překopírováním tentokráte přímo jen vyhovujících titulů do námi vytvořeného prázdného pomocného kontejneru typu `database_t`; použijeme k tomu funkci `std::copy_if`, predikát implementovaný pomocí lambda výrazu a instanci falešného iterátoru pro výstup získanou voláním `std::back_inserter`; následně tituly seřadíme, a to opět pomocí lambda výrazu simulujícího chování operátoru `<`, přičemž tituly chceme řadit vzestupně podle jejich názvů; nakonec všechny tituly vypíšeme v transformované podobě do předaného výstupního streamu, a to pomocí funkce `std::transform`; vlastní pak transformaci provedeme pomocí lambda výrazu, který pro daný titul vrátí řetězec `{ name: "`*title*`", year: `*year*`, ... }`, kde *title* a *year* nahradíme jeho názvem resp. rokem natočení; pro výstup pak použijeme falešný iterátor `std::ostream_iterator<std::string>`, přičemž každý záznam ještě ukončíme vypsáním konce řádku

- `int db_query_14(`
  `const database_t& database,`
  `Type type,`
  `const std::string& genre`
  `)`: spočítá celočíselné průměrné hodnocení všech titulů majících náš enumerační typ `type` (s hodnotami `MOVIE` a `SERIES`) a žánr `genre`; pro iteraci nad jednotlivými tituly použijeme funkci `std::for_each`, přičemž zpracování konkrétního titulu provedeme pomocí funktoru s názvem `Visitor_Q14_Rating`; ten naprogramujeme tak, aby veškerá logika počítání tohoto průměru byla schována uvnitř něho; pro nulový počet titulů vrátíme průměr `0`; dodejme dále, že funkce `std::for_each` si vytvoří a následně používá vlastní kopii předaného funktoru, tu pak vrátí pomocí návratové hodnoty

- `size_t db_query_15(`
  `const database_t& database,`
  `const std::string& genre`
  `)`: zjistíme celkový počet titulů majících žánr `genre`; opět využijeme funkci `std::for_each`, tentokráte ale zpracování konkrétních titulů budeme řešit pomocí lambda výrazu

- `void db_query_16(`
  `const database_t& database,`
  `unsigned short begin,`
  `unsigned short end,`
  `std::ostream& stream = std::cout`
  `)`: najdeme všechny tituly natočené během let [`begin`, `end`); k jejich nalezení použijeme pohledy `std::views::filter` a `std::views::transform`, které zřetězíme pomocí pipe operátoru `|`, přičemž filtrovací predikát i transformační akci naprogramujeme v podobě lambda výrazů; cílem zmíněné transformace je pro každý titul získat trojici v podobě `std::tuple<std::string, unsigned short, size_t>`, kde postupně uložíme název titulu, rok jeho natočení a počet herců v něm hrajících; získané záznamy následně uložíme do kontejneru `std::vector` pomocí jeho rozsahového konstruktoru; všechny záznamy dále uspořádáme pomocí funkce `std::ranges::sort`, a to opět pomocí lambda výrazu řadícího tituly podle roků natočení a následně jejich názvů; nakonec záznamy transformujeme pomocí `std::views::transform`, tentokráte do řetězců v podobě `{ name: "`*title*`", year: `*year*`, actors: `*actors*` }`, kde *title*, *year* a *actors* nahradíme názvem titulu, rokem natočení a počtem herců; takto získané řetězce opět vypíšeme do uvedeného výstupního streamu pomocí funkce `std::ranges::copy` a falešného iterátoru `std::ostream_iterator`

Odevzdejte opět jen modul dotazů, tedy soubory `Queries.h` a `Queries.cpp`, a nově také hlavičkový soubor `Storage.h`. Pokud jste se rozhodli používat gumové pole, odevzdejte i jeho implementaci. Zbytek projektu databáze tedy opět odevzdávat nebudete. Nadále pak předpokládáme, že všechno potřebné týkající se titulů, filmů, seriálů a herců je dostupné skrze hlavičkový soubor `Movie.h`.