NPRG041 – 2023/24 ZS – Cvičení MS – **Malý úkol C05**

Databáze filmů I

Cílem tohoto úkolu je naprogramovat první část jednoduché aplikace, pomocí které budeme schopni simulovat práci s databází filmů, čímž myslíme schopnost jednotlivé filmy ukládat (vytvářet jejich instance a uchovávat je ve standardním vektoru) a následně se nad nimi dotazovat (tedy hledat jednotlivé filmy odpovídající vyhledávacím kritériím a ty pak následně vypisovat na standardní výstup).

U každého filmu potřebujeme evidovat následující datové položky: název filmu (typu `std::string`), rok natočení (`unsigned short`), žánr (`std::string`), hodnocení (`unsigned short`) a množinu jmen herců (`std::string`), kteří v daném filmu hráli. Samotnou množinu realizujeme pomocí dalšího typu kontejneru, konkrétně `std::set`, který najdeme v knihovně `<set>`. Všechny údaje jsou povinné, počet herců ale může být zcela libovolný, tedy klidně i nulový.

Pro reprezentaci popsaných filmů vytvoříme běžnou třídu s názvem `Movie`, každý konkrétní film bude reprezentován jednou instancí této třídy. Všechny výše uvedené údaje budou realizovány jako privátní datové položky, přístup k nim zvenku bude možný jen přes veřejné bezparametrické metody `name`, `year`, `genre`, `rating` resp. `actors` vracející nemodifikovatelné reference na příslušné položky (v případě názvu, žánru a herců) resp. příslušné hodnoty kopií (u roku a hodnocení). Tyto metody navíc musí být deklarované jako konstantní (abychom je mohli volat i na nemodifikovatelných instancích filmů) a z důvodu efektivity je naprogramujeme jako inline metody.

Třída `Movie` nabídne i následující dva konstruktory. V případě prvního z nich si u sebe uložíme kopie předaných položek, v případě druhého si položky předané rvalue referencemi přivlastníme. Pro uložení jednotlivých položek využijeme výhradně mechanismu inicializačních seznamů.

- `Movie(const std::string& name, unsigned short year, const std::string& genre, unsigned short rating, const std::set<std::string>& actors)`
- `Movie(std::string&& name, unsigned short year, std::string&& genre, unsigned short rating, std::set<std::string>&& actors)`

Poslední metodou třídy `Movie` bude `void print_json(std::ostream& stream)`. Jejím prostřednictvím budeme schopni daný film vypsat do uvedeného výstupního streamu. Volání této funkce je potřeba umožnit i bez uvedení jediného parametru, v takovém případě budeme předpokládat standardní výstup `std::cout`. Film jako takový vypíšeme v podobě JSON objektu, jeho struktura musí odpovídat následující šabloně:

```
{ name: "Bobule", year: 2008, genre: "comedy", rating: 65, actors: [ "Krystof Hadek", "Te
reza Voriskova" ] }
```

Kromě mezer a celkové struktury musíme dodržet i pořadí a názvy jednotlivých položek, textové hodnoty budeme zapouzdřovat do uvozovek. Jména herců budeme vypisovat přesně v tom pořadí, v jakém nám je vrátí příslušný iterátor. Pokud film nemá ani jednoho herce, pak položku `actors` vůbec nevypíšeme. Pokud herci přítomní jsou, za posledním z nich už oddělovací čárku samozřejmě vypisovat nebudeme. Součástí výpisu nejsou žádné konce řádků, a to ani na konci za uzavírací závorkou celého JSON objektu.

Samotnou databázi realizujeme pomocí jediné instance standardního kontejneru vektoru `std::vector`, do kterého postupně budeme jednotlivé instance filmů vkládat. K tomu můžeme využít např. metody `push_back` nebo lépe `emplace_back`. Uživateli však nabídneme i možnost importu filmů ze vstupních souborů resp. obecného vstupního streamu. Za tímto účelem navrhneme třídu `Database`, její deklaraci umístíme do hlavičkového souboru `Database.h`. Cílem této třídy není uchovávat obsah databáze, ale jen zapouzdřit funkcionalitu importu, a to prostřednictvím následujících dvou veřejných statických metod:

- `static void import(const std::string& filename, std::vector<Movie>& db)`: otevře zadaný vstupní soubor, provede import filmů a jejich instance vloží do připraveného vektoru
- `static void import(std::istream& strem, std::vector<Movie>& db)`: uskuteční import filmů z uvedeného vstupního streamu a instance filmů opět vloží do připraveného vektoru

V obou případech předpokládáme vstupní data ve formátu CSV. To znamená, že na každém jednotlivém řádku budou údaje o jednom filmu, případné zcela prázdné řádky budeme přeskakovat. Pro daný film postupně očekáváme jeho název, rok natočení, žánr, hodnocení a množinu jmen herců. Jednotlivé údaje budou odděleny pomocí středníku, jména jednotlivých herců pak pomocí čárky.

Položky názvu filmu a jeho žánr musí být neprázdné řetězce. Pokud jde o povolené hodnoty číselných položek, rok natočení se očekává z uzavřeného intervalu `1900` až `2100`, hodnocení filmu `0` až `100`. Můžeme předpokládat, že nikde uvnitř hodnot našich položek se používané oddělovače vyskytovat nebudou. Jelikož v navazujících úkolech přidáme několik dalších položek, vyplatí se kód vhodně dekomponovat. Připomeňme také, že počet herců může být i nulový, jak je ostatně vidět i na následujícím příkladu u druhého filmu:

```
Dira u Hanusovic;2014;comedy;49;Tatiana Vilhelmova,Ivan Trojan,Klara Meliskova
Vlastnici;2019;comedy;74;
```

Pro samotné parsování vstupu se očekává použití globální funkce `std::getline`, tentokráte i ve variantě s volitelným třetím parametrem, pomocí kterého můžeme určit i jiný separátor než výchozí. V kombinaci s tím se nám bude hodit možnost vytvořit stream nad řetězcem, a to pomocí `std::istringstream` z knihovny `<sstream>`. Instanci filmu budeme do kontejneru databáze vkládat co nejefektivněji, tedy za použití již zmíněné metody `emplace_back` v kombinaci s konstrukcí `std::move`.

Veškeré chybové situace budeme řešit pomocí strukturovaných výjimek, a to v podobě struktury `struct Exception { int code; std::string text; }`, kde první položka bude obsahovat číselný kód typu chyby a druhá textový řetězec blíže vysvětlující příčinu chyby. Konkrétně očekáváme následující chování:

- Kód `1` (chyby vstupů)
  - `Unable to open input file <filename>`: nepodaří se otevřít vyžádaný vstupní soubor
- Kód `2` (parsovací chyby)
  - `Missing field <name> on line <line>`: chybí příslušná položka aneb ve vstupním řetězci už žádná další položka není
  - `Empty string in field <name> on line <line>`: prázdný řetězec u textových položek, které prázdné hodnoty nepřipouští
  - `Invalid integer in field <name> on line <line>`: nevalidní číselná hodnota u číselných položek jakožto reakce na výjimku `std::invalid_argument` z funkce `std::stoi`
  - `Overflow integer in field <name> on line <line>`: analogicky přetečená číselná hodnota jakožto reakce na výjimku `std::out_of_range`
  - `Malformed integer in field <name> on line <line>`: analogicky špatně formovaná číselná hodnota jakožto reakce na nesplněný poziční test
  - `Integer out of range <min, max> in field <name> on line <line>`: číselná hodnota nepatřící do intervalu povolených hodnot

Parametry v lomených závorkách se samozřejmě nahradí konkrétními hodnotami, závorky samotné se ponechají: `filename` je jméno požadovaného souboru, `name` jméno problematické položky (tedy `name`, `year`, `genre`, `rating` nebo `actors`), `line` číslo řádku ve vstupním souboru (počítáno od `1`), a `min` a `max` je interval povolených hodnot pro číselné položky.

Pokud dojde k problémům při parsování záznamu filmu, instanci daného filmu nevytvoříme, vyhodíme příslušnou výjimku, a ve zpracování případných dalších filmů na vstupu tedy už nepokračujeme. Všechny filmy, které se nám už podařilo do databáze (vektoru filmů) vložit, v něm už zůstanou.

Nakonec ještě implementujeme dvě globální funkce, pomocí kterých budeme simulovat vyhodnocování dotazů nad naší databází. V obou případech budeme jednoduše iterovat přes všechny filmy v předaném kontejneru a ty hledané vypíšeme v požadované podobě do uvedeného výstupního streamu, přičemž výpis každého takového filmu ještě ukončíme koncem řádku. Obě funkce deklarujeme v hlavičkovém souboru `Queries.h`.

- `void db_query_1(const std::vector<Movie>& db, std::ostream& stream = std::cout)`: najdeme všechny filmy (tedy žádné filtrování neprovádíme); každý z nich vypíšeme jako kompletní JSON objekt (pomocí naší připravené funkce)

- `void db_query_2(const std::vector<Movie>& db, std::ostream& stream = std::cout)`: najdeme všechny komedie (*comedy*) natočené před rokem *2010*, ve kterých hrál *Ivan Trojan* nebo *Tereza Voriskova*; pro každou z nich vypíšeme jen příslušný název filmu

Podobně jako v minulém úkolu, i tentokráte bude průběh celého testu řízen přímo z funkce `main`, kterou odevzdávat nebudete, protože se použije její předpřipravená varianta. V rámci ní nejprve dojde k vytvoření instance naší databáze aneb vektoru pro filmy. Následně bude opakovaně a v libovolném pořadí manuálně vytvářet a vkládat nové instance filmů, importovat je ze vstupních souborů nebo volat diskutované dotazovací funkce.

Veškerý kód samozřejmě rozdělte do jednotlivých modulů s hlavičkovými soubory. Třídu `Database` a funkce `db_query_*` je nutné deklarovat v hlavičkových souborech `Database.h` resp. `Queries.h`. Odevzdejte všechny vytvořené zdrojové soubory (`*.cpp` a `*.h`) kromě hlavního souboru `Main.cpp`.

Konkrétním cílem úkolu je ověřit schopnost práce s následujícími konstrukty: návrh a použití obyčejných datových tříd, návrh jejich parametrických konstruktorů v kombinaci s inicializačními seznamy, návrh inline metod, použití konstruktu `std::move` a práce s rvalue referencemi, seznámení se s kontejnerem `std::set` a pokročilejšími způsoby vkládaní prvků do kontejneru `std::vector` pomocí obecného mechanismu `emplace` a konečně také práce s funkcí `std::getline` s jiným než výchozím separátorem a v kombinaci se streamy `std::istringstream`. Jiný slovy je nutné všechny tyto konstrukty v řešení úkolu použít.

Předpokládáme dodržení všech postupů, které jsme se už naučili, stejně jako požadavků, které na úkoly máme. Specificky nezapomeňte na pojmenované konstanty pro námi používané oddělovače středníků a čárek ve vstupních CSV souborech (aby je v případě potřeby bylo možné snadno změnit), stejně jako na používání konvence postfixování názvů privátních datových položek v našich třídách pomocí symbolu `_`.

Každou lokální proměnnou deklarujeme v nejvíce specifickém bloku, tedy jen v takovém bloku (který je vždy definován párem složených závorek `{}`), kde ji opravdu potřebujeme používat. Případné další pomocné metody pro parsování filmů je opět vhodné přidávat jako privátní statické členské funkce třídy `Database`, nikoli jako samostatné globální funkce. Inline funkce musí být naprogramovány tak, aby v místě jejich použití byly známé nejenom jejich deklarace, ale také plné definice. To v našem případě znamená, že jejich těla musíme implementovat přímo v daném hlavičkovém souboru.

Během parsování jednotlivých položek konkrétního filmu ze vstupu není vhodné použít cyklus (ať už `while` nebo `for`), protože každý údaj potřebujeme zpracovat jinak. Jinými slovy se cykly obecně hodí jen v takových situacích, kdy každá iterace probíhá řekněme podobně nebo má alespoň nějaký dostatečně významný společný základ. V našem případě bychom ale tělo cyklu museli zase zpětně rozvětvit na každý jednotlivý případ (třeba pomocí konstrukce `switch` nebo jinak) a řešit jej samostatně. Bez ohledu na zjevnou neefektivitu by takové řešení především značilo špatný návrh, a proto se jej vyvarujeme.