NPRG041 – 2022/23 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. Množinu jako takovou 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 standardní 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 musí být naprogramovány 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 naší třídy `Movie` bude `void print_json(std::ostream& os)`. 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 `std::vector`, do kterého postupně budeme jednotlivé vytvořené instance filmů vkládat. K tomu můžeme využít např. metody `push_back` nebo `emplace_back`. Uživateli však nabídneme i možnost importu filmů ze vstupních souborů resp. obecného vstupního streamu. Za tímto účelem implementujeme třídu `Database` a v rámci ní následující dvě statické veřejné metody:

- `static bool import(const std::string& filename, std::vector<Movie>& db)`: otevře uvedený vstupní soubor, provede import filmů a jejich instance vloží do připraveného vektoru
- `static bool import(std::istream& is, std::vector<Movie>& db)`: provede import z obecného 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. Připomeňme, že počet herců může být i nulový, jak je ostatně vidět i na následujícím příkladu.

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

Můžeme předpokládat, že vstup je syntakticky korektní, nikde uvnitř hodnot našich položek se používané oddělovače nevyskytují. Jednotlivé hodnoty jsou neprázdné. Návratové hodnoty obou funkcí tradičně použijeme pro signalizaci úspěchu (`true`) nebo naopak neúspěchu (`false`). Pokud dojde k problémům při parsování čísel, instanci daného filmu nevytvoříme, ve zpracování případných dalších filmů na vstupu ale budeme pokračovat dál. Prázdné řádky ve vstupním streamu za chybu nepovažujeme.

Pro samotné parsování vstupu je potřeba použít globální funkci `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 do kontejneru vkládejte co nejefektivněji, tedy za použití již zmíněné metody `emplace_back` v kombinaci s konstrukcí `std::move`.

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 na standardní výstup, přičemž výpis každého takového filmu ještě ukončíme koncem řádku.

- `void db_query_1(const std::vector<Movie>& db)`: najdeme všechny filmy (tedy žádné filtrování neprovádíme); každý z nich vypíšeme jako kompletní JSON objekt (pomocí naší připravané funkce)
- `void db_query_2(const std::vector<Movie>& db)`: 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 vektoru pro filmy. Následně je možné 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 hlavičkovém souboru `Database.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 deklarování pojmenovaných konstant pro námi používané oddělovače středníků a čárek ve vstupních CSV souborech, stejně jako na používání konvence postfixování názvů privátních datových položek v našich třídách pomocí symbolu `_`. Nezapomeňte ani na ošetřování chybových situací při parsování číselných hodnot a otevírání souborů.

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.

Na závěr se ještě podívejme na parsování jednotlivých údajů pro konkrétní film při importu ze vstupních CSV dat ve spojitosti s používáním funkce `std::getline`. Pro tento záměr není vhodné použít cyklus (ať už `while` nebo `for`), protože každý údaj potřebujeme zpracovat jinak. Jinými slovy se cykly obecně hodí 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`) a řešit jej samostatně. To by vedlo k velmi neefektivnímu kódu, takže se takového řešení vyvarujeme.