2023. november 16.

C++ PARCIÁLIS – INFORMATIKA C. CSOPORT

Adott egy Song osztály header állománya:

```
class Song {
public:
    // Constructors
    Song();
    Song(const std::string &title, const std::string &artist, int duration);

    Song(const Song &other);
    Song(Song &&other);

    // Assignment operators
    Song& operator=(const Song &other);
    Song& operator=(Song &&other);

    // Getter methods
    std::string getTitle() const;
    std::string getArtist() const;
    int getDuration() const;
    int getMinutes() const;
    int getSeconds() const;

    // Equality operator overloading
    bool operator==(const Song &other);

    // Inserter operator
    friend std::ostream& operator<<(std::ostream &os, const Song &song);

    // Extractor operator
    friend std::istream& operator>>(std::istream &is, Song &song);

private:
    // Helper methods
    std::string padNumber(int number) const;
    std::string title;
    std::string artist;
    int duration;
};
```

Implementálja a Song osztály függvényeit (`song.cpp`), majd tesztelje ezeket a `main.cpp` állományban.

Magyarázat:

- A könnyebbség kedvéért az `artist` és a `title` garantáltan egyszavas.
- `int getMinutes() const;`
  - Visszatéríti a zeneszám perceinek számát. Például, ha egy zeneszám 87 másodperces (azaz 01:27), akkor az 1 percnek számít.
- `int getSeconds() const;`
  - Visszatéríti a zeneszám perceinek számát. Például, ha egy zeneszám 87 másodperces (azaz 01:27), akkor a visszatérített érték 27.
- `bool operator==(const Song &other);`
  - Két zeneszám egyenlő, ha a `title` és az `artist` is megegyezik.
- `std::string padNumber(int number) const;`
  - Zéróval kiegészíti a paraméterként kapott számot, amennyiben az egyjegyű szám. Segédfüggvény a zeneszámok hosszának kiírásához.
- `friend std::ostream& operator<<(std::ostream &os, const Song &song);`
  - A kiíratás a következő formában kell történjen:

    ```
    - <song_name> by <artist> (<song_duration>)
    - ...
    ```

  - Például:

    - AttackOfTheKillerQueen by LenaRaine (02:00)
    - BeneathTheMask by ShojiMeguro (04:37)
    - Cat by C418 (03:06)

  - A `<song_duration>` esetén az egyjegyű perceket és másodperceket egészítsük ki 0-kal, hogy szép legyen a kimenet: például 1:3 → 01:03. Használjuk a `padNumber` függvényt.

Adott egy User osztály deklaráció egy **user.h** header állományban. Implementálja a metódusokat egy **user.cpp** állományban.

```cpp
// Enum for sorting options
enum SortOption {
    BY_TITLE,
    BY_ARTIST,
    BY_DURATION
};

class User {
public:
    // Constructors
    User();
    User(const std::string &username);
    User(const User &other);
    User(User &&other);

    // Assignment operators
    User& operator=(const User &other);
    User& operator=(User &&other);

    // Getter methods
    int getId() const;
    std::string getUsername() const;
    const std::vector<Song> &getPlaylist() const;

    // Playlist management
    void addToPlaylist(const Song &song);
    int removeArtistFromPlaylist(const std::string &artist);
    int removeShortSongsFromPlaylist(int minDuration);

    // Sort playlist by name or price
    void sortPlaylistBy(SortOption option);

    friend std::ostream& operator<<(std::ostream &os, const User &user);
    friend std::istream& operator>>(std::istream &is, User &user);

private:
    int id;
    std::string username;
    std::vector<Song> playlist;
    // First user's ID must be 1
    static int nextId;
};
```

Magyarázat:

- `void addToPlaylist(const Song &song);`
  - Hozzáad egy új dalt az adott felhasználó lejátszási listájához. Amennyiben a dal már hozzá volt adva, váltson ki egy `std::invalid_argument` típusú kivételt.
- `int removeArtistFromPlaylist(const std::string &artist);`
  - Megkeresi az összes dalt egy adott nevű zeneszerzőtől és kitörli azokat a felhasználó lejátszási listájáról.
  - Visszatéríti, hogy hány dalt törölt ki.
- `int removeShortSongsFromPlaylist(int minDuration);`
  - Kitörli a megadott hossznál rövidebb dalokat.
  - Visszatéríti, hogy hány dalt törölt ki.
- `void sortPlaylistBy(SortOption option);`
  - Rendezi a felhasználó lejátszási listáját egy megadott kritérium szerint. Ez a kritérium a `SortOption` enumban van kódolva, rendezni lehet cím, szerző (alfabetikus sorrend) és hossz szerint – mindegyik növekvő sorrenben.
  - A rendezést meg lehet oldani `SortOption` használata nélkül is, ez esetben a maximális pontszám felét éri a rendezéses feladat/függvény.
- `friend std::ostream& operator<<(std::ostream &os, const User &user);`
  - A kiíratás a következő formában kell történjen:

    ```
    <username> has <playlist_length> songs saved:
    - <song_name> by <artist> (<song_duration>)
    - ...
    ```

  - Például:

    emanresU has 3 songs:
    - AttackOfTheKillerQueen by LenaRaine (02:00)
    - BeneathTheMask by ShojiMeguro (04:37)
    - Cat by C418 (03:06)

  - Használjuk a Song osztály << operátorát is.
- `friend std::istream& operator>>(std::istream &is, User &user);`
  - Az adatok sorrendje a következő a bemeneti állományban (`user.txt`):
    - Felhasználó neve
    - Dalok száma
    - Dal címe
    - Dalszerző neve
    - Dal hossza másodpercben
  - A dalok címe, szerzője és hossza annyiszor ismétlődik ahány dal van. Az egyszerűség kedvéért mindenképp egy szóból állnak a felhasználók nevei, a dalok címei és a szerzők nevei. Használjuk a Song osztály >> operátorát is.

Adott a `main.cpp` állomány tartalma:

```
int main() {
    // Read user data from user.txt
    User user;
    std::ifstream input("user.txt");

    input >> user;
    std::cout << user << std::endl;

    if (user.getId() != 1) {
        throw std::runtime_error("User ID is not 1 as expected!");
    }

    // Add new song
    Song song("Korobeiniki", "HirokazuTanaka", 84);
    user.addToPlaylist(song);

    try {
        user.addToPlaylist(song);
        throw std::runtime_error("Failed to raise exception during add.");
    } catch (std::invalid_argument& e) {
        std::cout << "Successfully prevented adding duplicate song." <<
std::endl;
    }

    std::cout << "After adding hey:" << std::endl;
    std::cout << user << std::endl;

    // Remove short songs
    user.removeShortSongsFromPlaylist(50);

    std::cout << "After removing short songs:" << std::endl;
    std::cout << user << std::endl;

    for (auto &song : user.getPlaylist()) {
        if (song.getDuration() < 50) {
            throw std::runtime_error("Failed to remove short songs.");
        }
    }

    // Remove artist
    user.removeArtistFromPlaylist("TobyFox");

    std::cout << "After removing TobyFox:" << std::endl;
```

```
    std::cout << user << std::endl;

    for (auto &song : user.getPlaylist()) {
        if (song.getTitle() == "TobyFox") {
            throw std::runtime_error("Failed to remove TobyFox.");
        }
    }

    // Test sorting
    user.sortPlaylistBy(SortOption::BY_DURATION);
    std::cout << "After sorting by duration:" << std::endl;
    std::cout << user << std::endl;

    user.sortPlaylistBy(SortOption::BY_ARTIST);
    std::cout << "After sorting by artist:" << std::endl;
    std::cout << user << std::endl;

    user.sortPlaylistBy(SortOption::BY_TITLE);
    std::cout << "After sorting by title:" << std::endl;
    std::cout << user << std::endl;

    // Test copy
    std::cout << "Song 1 title: " << song.getTitle() << std::endl;

    Song song2 = song;
    Song song3(song);
    std::cout << "Song 2 title: " << song2.getTitle() << std::endl;
    std::cout << "Song 3 title: " << song3.getTitle() << std::endl;

    // Test move
    Song song4 = std::move(song);
    Song song5(std::move(song2));
    std::cout << "Song 4 title: " << song4.getTitle() << std::endl;
    std::cout << "Song 5 title: " << song5.getTitle() << std::endl;

    return 0;
}
```