2023. november 13.

C++ PARCIÁLIS – INFORMATIKA D. CSOPORT

Adott egy Game osztály header állománya:

```
class Game {
public:
    // Constructors
    Game();
    Game(const std::string &title, double price);
    Game(const Game &other);
    Game(Game &&other);

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

    // Getter methods
    std::string getTitle() const;
    double getPrice() const;

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

    // Inserter operator
    friend std::ostream& operator<<(std::ostream &os, const Game &game);
    // Extractor operator
    friend std::istream& operator>>(std::istream &is, Game &game);

private:
    std::string title;
    double price;
};
```

Implementálja a Game osztály függvényeit (`game.cpp`), majd tesztelje ezeket a `main.cpp` Állományban.

Magyarázat:

- `bool operator==(const Game &other);`
    - Akkor egyenlő két játék, ha megegyezik a címük.

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_PRICE
};


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
    std::string getUsername() const;

    // Wishlist management
    void addToWishlist(const Game &game);
    void removeFromWishlist(const std::string &gameTitle);

    // Sort wishlist by name or price
    void sortWishlistBy(SortOption option);

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

private:
    std::string username;
    std::vector<Game> wishlist;
};
```

Magyarázat:

- `void addToWishlist(const Game &game);`
  - Hozzáad egy új játékot az adott felhasználó kívánság listájához. Amennyiben a játék már hozzá volt adva, váltson ki egy `std::invalid_argument` típusú kivételt.
- `void removeFromWishlist(const std::string &gameTitle);`
  - Megkeresi az adott címmel rendelkező játékot, és amennyiben létezik, letörli a felhasználó kívánság listájáról. Ha nem létezik ilyen címmel rendelkező játék, nem történik semmi.
- `void sortWishlistBy(SortOption option);`
  - Rendezi a felhasználó kívánság listáját egy megadott kritérium szerint. Ez a kritérium a `SortOption` enumban van kódolva, rendezni lehet cím (alfabetikus sorrend) és ár szerint – mindegyik növekvő sorrenben.
  - A rendezést meg lehet oldani `SortOption` használata nélkül is, ez esetben a maximális pontszám felét éri a rendezéses feladat/függvény. Helyette lehet két külön függvényt bevezetni.
- `friend std::ostream& operator<<(std::ostream &os, const User &user);`
  - Nincs megkötött sorrendje a kiíratásnak, minden releváns információt irassunk ki, köztük a felhasználó játékait is. Használjuk a Game osztály << operátorát is.
- `friend std::istream& operator>>(std::istream &is, User &user);`
  - Az adatok sorrendje a következő a bemeneti állományban (`user.txt`):
    - Felhasználó neve
    - Játékok száma
    - Játék címe
    - Játék ára
  - A játék címe és ára annyiszor ismétlődik ahány játék van. Az egyszerűség kedvéért mindenképp egy szóból állnak a felhasználók nevei és a játékok címei is. Használjuk a Game 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;

    // Add new game
    Game game("Dota 2", 0.00);
```

```
    user.addToWishlist(game);

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

    // Remove game
    user.removeFromWishlist("Doesn't Exist");
    user.removeFromWishlist("Valorant");

    std::cout << "After removing Valorant:" << std::endl;
    std::cout << user << std::endl;

    // Test sorting
    user.sortWishlistBy(SortOption::BY_PRICE);
    std::cout << "After sorting by price:" << std::endl;
    std::cout << user << std::endl;

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

    // Test copy
    std::cout << "Game title: " << game.getTitle() << std::endl;

    Game game2 = game;
    Game game3(game);
    std::cout << "Game 2 title: " << game2.getTitle() << std::endl;
    std::cout << "Game 3 title: " << game3.getTitle() << std::endl;

    // Test move
    Game game4 = std::move(game);
    Game game5(std::move(game2));
    std::cout << "Game 4 title: " << game4.getTitle() << std::endl;
    std::cout << "Game 5 title: " << game5.getTitle() << std::endl;

    return 0;
}
```