C++ parciális (I)

1. Adott a következő osztály header állománya. Implementáljuk a metódusokat (**flight.cpp**), majd teszteljük le őket a **main.cpp** állományban.

```cpp
#include <string>
#include <iostream>

class Flight {
public:
   // Konstruktorok
   Flight();
   Flight(const std::string &departure, const std::string &destination, double price,
double popularity = 0.0);
   Flight(const Flight &other);
   Flight(Flight &&other);

   // Értékadás operátorok
   Flight& operator=(const Flight &other);
   Flight& operator=(Flight &&other);

   // Getterek
   std::string getDeparture() const;
   std::string getDestination() const;
   double getPrice() const;
   double getPopularity() const;

   // Egyenlőség operátor
   bool operator==(const Flight &other) const;

   // << operátor
   // Irassuk ki szép formátumban a repülés adatait
   friend std::ostream& operator<<(std::ostream &os, const Flight &flight);

   // >> operátor
   // Olvassuk be a repülés adatait
   // A departure, destination, price, popularity sorban szerepelnek
   // A departure és destination nem tartalmaz szóközt
   friend std::istream& operator>>(std::istream &is, Flight &flight);

private:
   std::string departure;
   std::string destination;
   double price;
   double popularity;
};
```

2. Adott egy másik osztály, a User osztály header állománya. Implementáljuk a metódusokat (**user.cpp**), majd teszteljük le őket a **main.cpp** állományban.

```
#include <vector>
#include "flight.h"

class User {
public:
   // Konstruktorok
   User();
   User(const std::string &name);
   User(const User &other);
   User(User &&other);

   // Értékadás operátorok
   User& operator=(const User &other);
   User& operator=(User &&other);

   // Getter
   std::string getName() const;

   // Repülések kezelése

   // Egy új járat hozzáadása a tervhez
   // Dobjon std::invalid_argument kivételt, ha a járat már szerepel a tervben
   void addToItinerary(const Flight &flight);

   // Egy járat eltávolítása a tervből
   // Dobjon std::invalid_argument kivételt, ha a járat nem szerepel a tervben
   void removeFromItinerary(const std::string &departure, const std::string
&destination);

   // A tervben szereplő járatok rendezése népszerűség szerint, csökkenő sorrendben
   void sortItineraryByPopularity();

   // A minPopularity-nál kisebb népszerűségű járatok törlése a tervből
   void removeUnpopularFlights(double minPopularity);

   // << operátor
   // A felhasználó nevét és a tervben szereplő járatokat írja ki
   // Használd a Flight osztály operátorát a járatok kiírásához
   friend std::ostream& operator<<(std::ostream &os, const User &user);

   // >> operátor
   // A felhasználó nevét és a repülések számát (n), majd a tervben szereplő
járatokat olvassa be
   // Használd a Flight osztály operátorát a járatok beolvasásához
   friend std::istream& operator>>(std::istream &is, User &user);

private:
   std::string name;
   std::vector<Flight> itinerary;
};
```

A **main.cpp** fájl tartalma legyen a következő, használjátok tesztelésre:

```
#include <iostream>
#include "flight.h"
#include "user.h"

void testFlightClass() {
   std::cout << std::endl;
   std::cout << "==================== Flight Teszt ====================";
   std::cout << std::endl;

   // Alapértelmezett konstruktor
   Flight flight1;
   std::cout << "Alapértelmezett konstruktor:\n" << flight1 << std::endl;

   // Paraméteres konstruktor
   Flight flight2("New York", "Paris", 500.0, 10.0);
   std::cout << "Paraméteres konstruktor:\n" << flight2 << std::endl;

   // Copy konstruktor
   Flight flight3(flight2);
   std::cout << "Copy konstruktor: " << flight3 << std::endl;

   // Move konstruktor
   Flight flight4(std::move(flight2));
   std::cout << "Move konstruktor:\n" << flight4 << std::endl;

   // Értékadó operátor
   flight1 = flight3;
   std::cout << "Értékadó operátor:\n" << flight1 << std::endl;

   // Move értékadó operátor
   Flight flight5;
   flight5 = std::move(flight3);
   std::cout << "Move értékadó operátor:\n" << flight5 << std::endl;

   // Lekérdezők
   std::cout << "Indulás: " << flight4.getDeparture() << std::endl;
   std::cout << "Érkezés: " << flight4.getDestination() << std::endl;
   std::cout << "Ár: " << flight4.getPrice() << std::endl;
   std::cout << "Népszerűség: " << flight4.getPopularity() << std::endl;

   // Egyenlőség operátor
   if (flight4 == flight5) {
       std::cout << "A repülőjáratok egyenlőek." << std::endl;
   } else {
       std::cout << "A repülőjáratok nem egyenlőek." << std::endl;
   }
}

void testUserClass() {
   std::cout << std::endl;
   std::cout << "==================== User Teszt ====================";
```

```cpp
    std::cout << std::endl;

    // Alapértelmezett konstruktor
    User user1;
    std::cout << "Alapértelmezett konstruktor:\n" << user1 << std::endl;

    // Paraméteres konstruktor
    User user2("Alice");
    std::cout << "Paraméteres konstruktor:\n" << user2 << std::endl;

    // Járatok hozzáadása az útitervhez
    Flight flight1("London", "Tokyo", 700.0, 5.0);
    Flight flight2("Tokyo", "Sydney", 400.0, 3.0);

    user2.addToItinerary(flight1);
    user2.addToItinerary(flight2);
    std::cout << "Felhasználó járatokkal:\n" << user2 << std::endl;

    // Duplikált járat hozzáadásának hibakezelése
    try {
        user2.addToItinerary(flight1);
    } catch (const std::invalid_argument &e) {
        std::cout << "Várt kivétel elkapva: " << e.what() << std::endl;
    }

    // Járat eltávolítása az útitervből
    user2.removeFromItinerary("London", "Tokyo");
    std::cout << "Járat eltávolítása után:\n" << user2 << std::endl;

    // Nem létező járat eltávolításának hibakezelése
    try {
        user2.removeFromItinerary("NonExistent", "Destination");
    } catch (const std::invalid_argument &e) {
        std::cout << "Várt kivétel elkapva: " << e.what() << std::endl;
    }

    // Rendezés népszerűség alapján
    user2.sortItineraryByPopularity();
    std::cout << "Rendezett útiterv:\n" << user2 << std::endl;

    // Népszerűtlen járatok eltávolítása
    user2.removeUnpopularFlights(4.0);
    std::cout << "Népszerűtlen járatok eltávolítása után:\n" << user2 << std::endl;

    // Copy konstruktor
    User user3(user2);
    std::cout << "Copy konstruktor:\n" << user3 << std::endl;

    // Move konstruktor
    User user4(std::move(user2));
    std::cout << "Move konstruktor:\n" << user4 << std::endl;

    // Értékadó operátor
    user1 = user3;
    std::cout << "Értékadó operátor:\n" << user1 << std::endl;

    // Move értékadó operátor
```

```
    User user5;
    user5 = std::move(user3);
    std::cout << "Move értékadó operátor:\n" << user5 << std::endl;
}

void testInputUser() {
    std::cout << std::endl;
    std::cout << "=================== Felhasználó Bemenet Teszt ===================";
    std::cout << std::endl;

    User user;
    std::cin >> user;
    std::cout << "Felhasználó bemenettel:\n" << user << std::endl;
}

void testInputFlight() {
    std::cout << std::endl;
    std::cout << "=================== Repülőjárat Bemenet Teszt ===================";
    std::cout << std::endl;

    Flight flight;
    std::cin >> flight;
    std::cout << "Repülőjárat bemenettel:\n" << flight << std::endl;
}

int main() {
    testFlightClass();
    testUserClass();
    testInputUser();
    return 0;
}
```