PR1 VU Programmierung 1	Abschlussklausur	29.06.2021

Star Wars Prequel

Implementieren Sie die Klassen `Spaceship` und `Fleet`:

Ein `Spaceship`-Objekt hat die Instanzvariablen `name`(`string`, nicht leer), `faction` und `type`. Die Fraktion und der Typ sind Werte aus den Enumerationen `Faction(Faction::Empire, Faction::Rebels)` bzw. `Type(Type::Squadron=10, Type::Small=50, Type::Medium=100, Type::Big=200)`. Für die Klasse `Spaceship` sind folgende Methoden und Funktionen zu implementieren:

- Konstruktor(en) mit 2 bzw. 3 Parametern: Name, Fraktion und Typ in dieser Reihenfolge. Angabe des Typs ist optional und der Default-Typ ist `Type::Squadron`. Bei leerem Namen ist eine Exception vom Typ `runtime_error` zu werfen.
- `int get_value() const`: Retourniert den Wert eines `Spaceship`-Objekts. Dieser Wert berechnet sich aus dem Wert der sich aus dem Typ des Objekts ergibt (10, 50, 100 oder 200) multipliziert mit 2 falls das Objekt der Fraktion `Faction::Rebels` angehört bzw. multipliziert mit 3 falls es der Fraktion `Faction::Empire` angehört.
- `operator<`: Vergleicht zwei Raumschiffe. Retourniert `true`, falls der Wert des linken Operanden kleiner ist als der Wert des rechten Operanden, `false` sonst.
- `static bool same_faction(const vector<Spaceship>& ships, Faction f)`: Retourniert `false`, falls mindestens ein Raumschiff im Vektor `ships` nicht der Fraktion `f` angehört, `true` sonst.
- `operator<<`: Ein `Spaceship`-Objekt muss in der Form [*name*, *faction*, *type*, *value*] ausgegeben werden, z. B.: [X-Wing, Rebels, Squadron, 20]. Der vordefinierte Vektor `faction_names` kann für die Ausgabe der Fraktion verwendet werden. Die vordefinierte globale Funktion `string to_string(Type)` liefert die für die Ausgabe passenden Werte zur Enumeration `Type`.

Ein `Fleet`-Objekt hat die Instanzvariablen `name`(`string`, nicht leer), `faction` (`Faction`) und `ships` (`vector<Spaceship>`). Für die Klasse `Fleet` sind folgende Methoden und Funktionen zu implementieren:

- Konstruktor mit 3 Parametern: Name, Fraktion und Liste von `Spaceship`-Objekten in dieser Reihenfolge. Sollten die Raumschiffe in der Liste nicht alle der Fraktion des neu zu erstellenden `Fleet`-Objekts angehören oder der Name leer sein, ist eine Exception vom Typ `runtime_error` zu werfen.
- `bool add(const vector<Spaceship>& ships)`: Ist der Vektor `ships` leer, ist eine Exception vom Typ `runtime_error` zu werfen. Sollten nicht alle Schiffe in `ships` derselben Fraktion angehören wie das aktuelle `Fleet`-Objekt (`this`-Objekt), so ist `false` zu retournieren. Andernfalls werden alle Raumschiffe der Liste am **Beginn** der Raumschiffliste des aktuellen `Fleet`-Objekts unter Beibehaltung der relativen Reihenfolge eingefügt und `true` wird retourniert.
- `operator<<`: Die Ausgabe eines Objekts vom Typ `Fleet` muss in der Form [*name*, *faction*, {*list of ships*}] erfolgen, z. B.: [Rogue Squadron, Rebels, {[X-Wing, Rebels, Squadron, 20], [MC-80, Rebels, Big, 400]}].
- Zusatz für 10 Punkte: Erweitern Sie die Klasse `Fleet` um die Methode `vector<int> extremes() const`: Ermitteln Sie den minimalen und den maximalen Wert aller Raumschiffe der aktuellen Flotte (`this`-Objekt). Der zu retournierende Vektor soll 2 Elemente beinhalten. Das erste Element ist das gefundene Minimum, das zweite Element das Maximum. Es ist möglich, dass beide Werte gleich sind. Sollte die Raumschiffliste der aktuellen Flotte leer sein, ist eine Exception vom Typ `runtime_error` zu werfen.
- Zusatz für 15 Punkte: Erweitern Sie die Klasse `Fleet` um die Methode `vector<Spaceship> elite(Fleet& f)`: Sollte Flotte `f` nicht derselben Fraktion angehören wie die aktuelle Flotte (`this`-Objekt) oder die beiden Flotten eine unterschiedliche Anzahl an Raumschiffen umfassen, so ist eine Exception vom Typ `runtime_error` zu werfen. Andernfalls werden die Raumschiffe beider Flotten Position für Position verglichen und gegebenenfalls so vertauscht, dass das Raumschiff mit dem höheren Wert in der aktuellen Flotte (`this`-Objekt) ist. Zu retournieren ist ein Vektor, der alle Raumschiffe enthält, die in die aktuelle Flotte (`this`-Objekt) übernommen wurden, allerdings in der umgekehrten Reihenfolge in Bezug auf die Raumschiffliste im `this`-Objekt. Beispiel: `*this` habe die Raumschiffliste {R1,R2, R3} und `f` habe {R4, R5, R6}, dann werden die Raumschiffe R1 und R4, R2 und R5 sowie R3 und R6 verglichen und gegebenenfalls getauscht. Unter der Annahme, dass alle Tauschoperationen durchgeführt werden müssen, enthielte der zu retournierende Vektor die Raumschiffe {R6, R5, R4} in dieser Reihenfolge.

Implementieren Sie die Klassen `Spaceship` und `Fleet` mit den notwendigen Konstruktoren, Methoden und Operatoren, sodass jedenfalls das Rahmenprogramm kompiliert und ausgeführt werden kann und die gewünschten Ergebnisse liefert. Achten Sie in Ihren Konstruktoren darauf, dass nur gültige Objekte erstellt werden können. Werfen Sie gegebenenfalls eine Exception vom Typ `runtime_error`.

Für Ihr Programm dürfen Sie **nur** die im vorgegebenen Rahmenprogramm angeführten include-Dateien verwenden!

Instanzvariablen sind `private` zu definieren und die Verwendung globaler Variablen ist (abgesehen von im Rahmenprogramm eventuell bereits definierten) nicht erlaubt! Die Datenkapselung darf nicht durchbrochen werden. Es ist daher unter anderem nicht erlaubt, Referenzen oder Pointer auf private Instanzvariablen einer Klasse nach außen zu vermitteln, `friend`-Deklarationen (mit Ausnahme bei Operatorfunktionen) zu verwenden, oder setter-Methoden zu implementieren, die die Integrität der Daten nicht gewährleisten. Interpretationsspielraum in der Angabe können Sie zu Ihren Gunsten nutzen.

Die Teilaufgaben, bei denen keine Punkteanzahl angegeben ist, gelten als Basisfunktionalität. Für eine positive Beurteilung ist zumindest die Basisfunktionalität zu implementieren. Diese wird mit 30 Punkten bewertet. Die übrigen Teilaufgaben müssen nicht unbedingt implementiert werden, führen aber im Falle einer korrekten Implementierung zu einer entsprechenden Erhöhung der Punkteanzahl.