PR1 VU Programmierung 1	Abschlussklausur	29.01.2019

Star Wars

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

Eine `Person` hat einen Namen (string), einen Erfahrungswert (int, Wert zwischen 0 und 100, jeweils inklusive) und einen Heimatplaneten. Der Heimatplanet ist ein Wert aus der vordefinierten Enumeration `Planet(Planet::Kashyyyk, Planet::Coruscant, Planet::Corellia, Planet::Vulcan)`. Für die Klasse `Person` sind folgende Methoden und Operatoren zu implementieren:

- Konstruktor(en) mit 2 bzw. 3 Parametern. Name, Erfahrungswert und Planet in dieser Reihenfolge. Der Planet ist optional und per Default `Planet::Kashyyyk`. Bei leerem Namen oder Erfahrungswert nicht im erlaubten Bereich ist eine Exception vom Typ `runtime_error` zu werfen.
- `bool make_expedition(Planet p)`: Der Erfahrungswert der Person steigt um 2, wenn `p` der Heimatplanet der Person ist, um 5 sonst. Sollte dadurch der maximale Erfahrungswert 100 überschritten werden, so ist der Erfahrungswert unverändert auf dem ursprünglichen Wert zu lassen und `false` zu retournieren. Konnte der Erfahrungswert erfolgreich erhöht werden, so ist `true` zu retournieren.
- `bool is_experienced() const`: Liefert `true`, wenn der Erfahrungswert der Person größer als 75 ist, `false` sonst.
- `bool operator==`: Für den Vergleich zweier `Person`-Objekte. Liefert `true` falls Name, Erfahrungswert und Heimatplanet gleich sind, sonst `false`.
- `operator<<`: Die Ausgabe eines Objekts des Typs `Person` muss in der Form [*Name: Erfahrungswert*, *Planet*] erfolgen. Der vordefinierte Vektor `planet_names` kann für die Ausgabe der Enumerationswerte verwendet werden, z.B.: [Han Solo: 100, Corellia].

Ein `Spaceship`-Objekt hat einen Namen (string), eine Crewliste (`vector<Person>`) und einen Planeten auf dem es sich aktuell befindet. Für die Klasse `Spaceship` sind folgende Methoden und Operatoren zu implementieren:

- Konstruktor(en) mit 2 bzw. 3 Parametern. Name, Vektor von `Person`-Elementen und ein Planet in dieser Reihenfolge. Der Planet ist optional und per Default `Planet::Corellia`. Die Elemente aus dem Parametervektor müssen unter Beibehaltung der Reihenfolge in die Crewliste übernommen werden. Ist eine `Person` mehr als einmal in der Crewliste enthalten, ist eine Exception vom Typ `runtime_error` zu werfen. Bei leerem Namen oder einer leeren Liste ist ebenfalls eine Exception vom Typ `runtime_error` zu werfen.
- `bool hire(const Person&)`: Fügt die als Parameter übergebene Person am Ende der Crewliste ein, sofern sich diese Person noch nicht in der Crewliste befindet. Konnte die Person erfolgreich hinzugefügt werden, ist `true` zu retournieren, ansonsten `false`.
- `operator<<`: Die Ausgabe eines Objekts des Typs `Spaceship` muss in der Form [*Name*, Crew: {*Crewliste*}, *Planet*] erfolgen, z.B.: [Millennium Falcon, Crew: {[Han Solo: 100, Corellia], [Chewbacca: 99, Kashyyyk]}, Corellia].
- Zusatz für 10 Punkte: Erweitern Sie die Klasse `Spaceship` um folgende Methode:
  `vector<Person> expedition()`: Alle Crewmitglieder werden auf dem aktuellen Planeten des Raumschiffs auf Expedition geschickt (`Person::make_expedition`). Zu retournieren ist eine Liste aller Crewmitglieder, die nach der Expedition erfahren (Erfahrungswert größer als 75) sind, es aber vor Antritt der Expedition nicht waren.
- Zusatz für 15 Punkte: Erweitern Sie die Klasse `Spaceship` um folgende Klassenmethode:
  `static vector<Person> check_fleet(const vector<Spaceship>&)`: Retourniert eine Liste aller Personen, die Crewmitglied in mehr als einem der als Parameter empfangenen Raumschiffe sind. Die Reihenfolge der Personen im Ergebnis muss dem jeweils ersten Auffinden der betroffenen Personen entsprechen, wenn man für jedes Raumschiff in der Reihenfolge, in der sie im Parameter übergeben wurden, die jeweilige Crewliste in der gegebenen Reihenfolge prüft.

Implementieren Sie die Klassen `Person` und `Spaceship` 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.