PR1 VU Programmierung 1	Abschlussklausur	25.06.2019

Flughafen Check In

Implementieren Sie die Klassen `Passenger` und `Flight`: Ein `Passenger`-Objekt hat die Instanzvariablen `name` (`string`, nicht leer) und `category` (ein Wert aus der vordefinierten Enumeration `Category` `Category::Economy`, `Category::Business` und `Category::First`). Für die Klasse `Passenger` sind folgende Methoden und Funktionen zu implementieren:

- Konstruktor(en) mit 1 oder 2 Parametern: Name und Kategorie in dieser Reihenfolge. Kategorie ist optional mit dem Defaultwert `Category::Economy`. Sollte einer der Parameter nicht die Voraussetzungen erfüllen (z. B. Name ist leer), ist eine Exception vom Typ `runtime_error` zu werfen.
- `bool in_category(Category c) const`: Retourniert `true`, wenn das `this`-Objekt der Kategorie `c` angehört, `false` sonst.
- `operator==`: Der Einfachheit halber wird angenommen, dass Passagiernamen eindeutig sind. Zwei `Passenger`-Objekte sind also gleich, wenn die Namen gleich sind.
- `operator<<`: Passagiere müssen in der Form [*name*: *category*] ausgegeben werden, z. B. **[Renate Musterfrau: Business]**. Der vordefinierte Vektor `category_names` kann für die Ausgabe der Enumerationswerte verwendet werden.

Ein `Flight`-Objekt hat die Instanzvariablen `no` (`string`, nicht leer), `seats` (`int` Wert, nicht negativ) sowie `checked_in` (Liste der eingecheckten Passagiere) und `boarded` (Liste der Passagiere im Flugzeug). Für die Klasse `Flight` sind folgende Methoden und Funktionen zu implementieren:

- Konstruktor mit zwei Parametern Flugnummer und Anzahl Sitzplätze in dieser Reihenfolge. Die Listen der eingecheckten und geboardeten Passagiere sind für ein neues `Flight`-Objekt leer zu setzen. Sollte einer der Parameter die Voraussetzungen nicht erfüllen (z.B. Flugnummer ist leer oder Anzahl Sitzplätze ist negativ), ist eine Exception vom Typ `runtime_error` zu werfen.
- `bool check_in(const Passenger& p)`: Retourniert `false`, falls das `Passenger`-Objekt schon eingecheckt ist oder der Platz im Flugzeug nicht mehr ausreicht (also bereits so viele Passagiere eingecheckt haben, wie Plätze zur Verfügung stehen). Andernfalls wird der Passagier am Ende der Liste der eingecheckten Passagiere hinzugefügt und `true` retourniert.
- `bool board(const Passenger& p)`: Retourniert `false`, falls der Passagier nicht eingecheckt ist, oder bereits zuvor geboardet wurde. Andernfalls wird der Passagier am Ende der Liste der geboardeten Passagiere eingetragen und `true` retourniert.
- `operator<<`: Die Ausgabe eines Objekts vom Typ `Flight` muss in der Form [*no*, {*Liste der eingecheckten Passagiere*}] erfolgen. In der Liste der Passagiere werden jene, die auch schon geboardet wurden, durch einen nachgestellten Stern ('*') gekennzeichnet, z. B.: **[A4711, {[Renate Musterfrau: Business], [Maria Musterfrau: Economy]*}]**.
- Zusatz für 10 Punkte: Erweitern Sie die Klasse `Flight` um die Methode
  `bool ready(const vector<int>& seats_per_category) const`:
  Wenn `seats_per_category` nicht genau 3 Einträge enthält, die alle nicht negativ sind, ist eine Exception vom Typ `runtime_error` zu werfen. Andernfalls sind die Einträge als Maximalanzahl der verfügbaren Plätze in jeder Kategorie zu interpretieren und zwar der Eintrag mit Index 0 für die Klasse `Category::Economy`, der Eintrag mit Index 1 für die Klasse `Category::Business` usw. Es ist `true` zu retournieren, wenn alle eingecheckten Passagiere auch geboardet sind und in keiner Kategorie die Anzahl der Passagiere den durch `seats_per_category` festgelegten Wert überschreitet, `false` sonst.
- Zusatz für 15 Punkte: Erweitern Sie die Klasse `Flight` um die Methode `void pretty_print() const`:
  Diese gibt das Objekt auf cout in folgendem Format aus:
  no[, boarded: {*Liste der geboardeten Passagiere*}][, missing: {*Liste der nur eingecheckten Passagiere*}]
  Die Teile in eckigen Klammern sind optional und werden nur ausgegeben, wenn die jeweilige Liste nicht leer ist. Die Passagierlisten sind in der Reihenfolge des Check-Ins bzw. des Boardings auszugeben, z. B.:
  A4711, boarded: {[Renate: Business], [Maria: Economy]}, missing: {[Irene: First]} oder
  A4711, missing: {[Irene: First], [Renate: Business]}, falls noch niemand geboardet wurde oder **A4711** falls noch niemand eingecheckt hat.

Implementieren Sie die Klassen `Passenger` und `Flight` 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.