Bus

Implementieren Sie die Klassen **`Bus`** und **`Fleet`**.

Vordefiniert ist eine Enumeration **`Station`** mit Werten, die Stationen repräsentieren, die von Bussen angefahren werden können.

Die Beschreibung für einen **`Bus`** umfasst die Liniennummer (nicht leerer String), die nicht leere Liste der Stationen (in der Reihenfolge in der sie angefahren werden) und die Kosten pro Streckenabschnitt (also für die Fahrt von einer Station zur nächsten; in Cent; ganze Zahl >0 und <=20). Folgende Methoden und Operatoren sind für die Klasse **`Bus`** zu implementieren:

- Konstruktor(en) mit zwei und drei Parametern. Liniennummer, Stationsliste und Kosten pro Streckenabschnitt in dieser Reihenfolge. Die Angabe der Kosten ist optional. Der Defaultwert beträgt 7 Cent**.** Entspricht einer der Parameter nicht den Vorgaben (z.B. Liniennummer oder Stationsliste leer bzw. Kosten nicht im erlaubten Bereich) so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`bool stops_at(Station) const`**: Retourniert **`true`**, wenn der Bus die Station, die als Parameter übergeben wurde, anfährt, **`false`** sonst.
- **`int cost(Station from, Station to) const`**: Liefert die Kosten in Cent für eine Fahrt von **`from`** nach **`to`**. Sollten die Stationen nicht (oder nicht in dieser Reihenfolge) angefahren werden, so ist 0 zu retournieren. (Anmerkung: Die Stationen werden in der Reihenfolge angefahren, in der sie in der Stationenliste des Busses auftreten. Eine implizite Rückfahrt ist nicht vorgesehen. Die Kosten sind vom ersten Anfahren von **`from`** bis zum ersten darauffolgenden Anfahren von **`to`** zu berechnen.)
- **`operator<<`**: Die Ausgabe eines Objekts des Typs **`Bus`** muss in folgender Form erfolgen: [*Liniennummer Kosten pro Streckenabschnitt* cent/station; {*Liste der Stationen}*], z.B.: [42Y 7 cent/station {Main, Church}]. Der vordefinierte Vektor **`station_names`** kann für die Ausgabe der Enumerationswerte verwendet werden.

Für eine Busflotte (**`Fleet`**) werden der Name des Besitzers (nicht leerer String) und die Liste der Busse erfasst. Folgende Methoden und Operatoren sind für die Klasse **`Fleet`** zu implementieren:

- Konstruktor(en) mit einem und zwei Parametern. Name des Besitzers und Liste der Busse in dieser Reihenfolge. Die Busliste ist optional mit dem Defaultwert einer leeren Liste. Entspricht einer der Parameter nicht den Vorgaben (z.B. Besitzername leer), so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`void add_line(const Bus&)`**: Fügt den als Parameter erhaltenen Bus am Ende der Busliste der Flotte hinzu.
- **`vector<Bus> stops_at(Station) const`**: Retourniert eine Liste aller Busse in der Flotte, die die als Parameter erhaltene Station anfahren. Die relative Reihenfolge der Busse im Ergebnis soll der in der ursprünglichen Busliste der Flotte entsprechen.
- **`operator<<`**: Die Ausgabe eines Objekts des Typs **`Fleet`** muss in folgender Form erfolgen: [*Besitzername* {*Liste der Busse}*], z.B.: [Dr. Ritchart {[42Y 7 cent/station {Main, Church}], [15A 20 cent/station{}]}].
- Zusatz für 10 Punkte: **`vector<Bus> has_loop() const`**: Retourniert eine Liste aller Busse der Flotte, die dieselbe Station mehrmals anfahren. Die relative Reihenfolge der Busse im Ergebnis soll der in der ursprünglichen Busliste der Flotte entsprechen.
- Zusatz für 15 Punkte: **`vector<Bus> cheapest_connection(Station from, Station to) const`**: Liefert eine Liste aller Busse, für die die Fahrt von **`from`** nach **`to`** den geringsten Preis (innerhalb der Flotte) hat. Die relative Reihenfolge der Busse im Ergebnis soll der in der ursprünglichen Busliste der Flotte entsprechen. Gibt es keinen Bus, der die beiden Stationen in der vorgegebenen Reihenfolge verbindet, so ist eine leere Liste zu retournieren.

Implementieren Sie die Klassen **`Bus`** 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.