Postamt

Implementieren Sie die Klassen `Letter` und `Mailbox`:

Ein `Letter`-Objekt hat die Instanzvariablen `recipient` (`string`, nicht leer), `sender` (`string`, nicht leer), einen Wert für die Briefmarke `postage` (`int`, mindestens 80) und ein Format. Das Format ist ein Wert aus der vordefinierten Enumeration `Format(Format::Standard, Format::Compact, Format::Big, Format::Maxi)`. Für die Klasse `Letter` sind folgende Methoden und Funktionen zu implementieren:

- Konstruktor(en) mit 3 bzw. 4 Parametern: Empfänger, Absender, Briefmarkenwert und Format. Das Format is optional und per Default `Format::Standard`. Sollte einer der Parameter leer sein oder nicht den Vorgaben entsprechen (z. B. Briefmarkenwert zu gering), so ist eine Exception vom Typ `runtime_error` zu werfen.
- `Format get_format() const`: Retourniert das Format des Briefes.
- `int get_postage() const`: Retourniert das Porto (Wert der Briefmarke) des Briefes.
- `int capacity_consumption() const`: Retourniert, wie viel Platz ein Brief verbraucht. Der Platzverbrauch eines Briefes berechnet sich hierbei aus dem Format, indem zum Doppelten der `int`-Präsentation des Enumwertes (0 für `Format::Standard`, 2 für `Format::Compact`, 4 für `Format::Big` etc.) 10 addiert wird.
- `operator<<`: Ein `Letter`-Objekt muss in der Form [*recipient*, *sender*, *postage*, *format*] ausgegeben werden, z.B.: [Anna, Peter, 120, Maxi]. Der vordefinierte Vektor `format_names` kann für die Ausgabe der Formate verwendet werden.

Ein `Mailbox`-Objekt hat die Instanzvariablen maximale Kapazität `capacity` (`int`, mindestens 50), nicht leere Liste von erlaubten Formaten `allowed_formats` (`vector<Format>`) und Liste von enthaltenen Briefen `contained_letters` (`vector<Letter>`, kann leer sein). Für die Klasse `Mailbox` sind folgende Methoden und Funktionen zu implementieren:

- Konstruktor(en) mit 1, 2 bzw. 3 Parametern: Kapazität, Liste von erlaubten Formaten und Liste von Briefen (`Letter`-Objekten) in dieser Reihenfolge. Die Liste von Briefen ist optional und per Default leer. Ebenso ist die Liste von erlaubten Formaten optional und enthält per Default nur den einen Eintrag `Format::Standard`. Sollte einer der Parameter nicht den Vorgaben entsprechen (z. B., leere Liste von erlaubten Formaten oder Kapazität zu gering), ist eine Exception vom Typ `runtime_error` zu werfen. Falls in der Liste der erlaubten Formate mindestens ein Eintrag mehrfach auftritt, der gesamte Platzverbrauch der Briefe (`capacity_consumption()`) die Kapazität übersteigt oder mindestens einer der Briefe ein nicht erlaubtes Format hat, ist ebenfalls eine Exception vom Typ `runtime_error` zu werfen.
- `int total_value() const`: Retourniert den Gesamtwert der Briefmarken aller im Briefkasten vorhandenen Briefe.
- `void empty()`: Leert den Briefkasten, d.h., entfernt alle Briefe aus der Liste der momentan vorhandenen Briefe.
- `operator<<`: Die Ausgabe eines Objekts vom Typ `Mailbox` muss in der Form [{*Liste erlaubter Formate*}, Kapazitaet: *capacity*, {*Liste von Briefen*}] erfolgen, z.B.: [{Standard, Maxi}, Kapazität: 250, {[Anna, Peter, 120, Maxi], [Reinhold, Julia, 90, Standard]}].
- Zusatz für 10 Punkte: Erweitern Sie die Klasse `Mailbox` um die Methode `vector<Letter> post_letters(const vector<Letter>& letters)`: Diese versucht alle Briefe aus `letters` unter Beibehaltung der Reihenfolge am Ende der Liste der bereits vorhandenen Briefe hinzuzufügen. Ein Brief darf nur hinzugefügt werden, sofern die Kapazität des Briefkastens nicht überschritten wird und der Brief ein passendes Format hat. Zu retournieren ist die Liste der Briefe, die nicht hinzugefügt werden konnten, in derselben relativen Reihenfolge wie im Fuktionsparameter `letters`.
- Zusatz für 15 Punkte: Erweitern Sie die Klasse `Mailbox` um die Methode `void sort(const vector<Format>& order)`: Diese soll die Liste der im Briefkasten enthaltenen Briefe so umordnen, dass zuerst alle mit dem Format kommen, das im Vektor `order` zuerst kommt, dann jene mit dem Format, das im Vektor als zweites kommt usw. Briefe, mit einem Format, das im Vektor `order` nicht vorkommt, werden anschließend nach Format in der Reihenfolge der Enumerationskonstanten geordnet. Briefe, die dasselbe Format haben, sollen ihre relative Ordnung beibehalten. Sollten für ein bestimmtes Format keine Briefe vorhanden sein, tut das der Sortierreihenfolge keinen Abbruch. Der Vektor `order` kann auch leer sein, dann wird nur nach der Reihenfolge der Enumerationskonstanten geordnet. Im Falle, dass im Vektor `order` ein Eintrag mehrfach auftritt, ist eine Exception vom Typ `runtime_error` zu werfen. Beispiel: `order({Format::Maxi, Format::Compact})` führt dazu, dass die Briefe mit Format `Format::Maxi` zuerst kommen, dann die mit `Format::Compact`, die mit `Format::Standard` und schließlich die mit `Format::Big`.

Implementieren Sie die Klassen `Letter` und `Mailbox` 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.