PR1 VU Programmierung 1	Klausurvorbereitung	

Auktion

Implementieren Sie die Klassen **`Lot`** und **`Gebot`** zur Verwaltung von (Online)Auktionen.

Ein **`Gebot`** umfasst den Namen der Bieterin (String mit Länge >0) und den gebotenen Betrag (ganze Zahl >0 und <=10.000.000). Der Einfachheit halber dürfen Sie davon ausgehen, dass Namen von Bieterinnen eindeutig sind. Folgende Methoden und Operatoren sind für die Klasse **`Gebot`** zu implementieren:

- Konstruktor(en) mit einem und zwei Parametern. Name und Betrag in dieser Reihenfolge. Der Betrag ist optional mit Defaultwert **`100.`** Entspricht einer der Parameter nicht den Vorgaben (z.B. leerer Name oder Betrag nicht im erlaubten Bereich) so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`bool selbe_bieterin(const Gebot&) const`**: Liefert **`true`**, wenn beide Gebote (this-Objekt und Parameter) von derselben Bieterin abgegeben wurden, **`false`** sonst.
- **`bool operator==(const Gebot&) const`**: Liefert **`true`**, wenn beide Gebote (this-Objekt und Parameter) gleich hoch sind (Betrag stimmt überein), **`false`** sonst.
- **`bool operator<(const Gebot&) const`**: Liefert **`true`**, wenn das this-Objekt das niedrigere Gebot ist (Betrag ist kleiner als im Parameterobjekt), **`false`** sonst.
- **`bool operator>=(int) const`**: Liefert **`true`**, wenn der Betrag im this-Objekt zumindest so hoch ist wie der als Parameter erhaltene Betrag, **`false`** sonst.
- **`operator<<`**: Die Ausgabe eines Objekts des Typs **`Gebot`** muss in der Form [*Name*: *Betrag* Euro] erfolgen, z.B.: [Susi: 263 Euro].

Ein **`Lot`** (Gut, das versteigert wird) hat eine Bezeichnung (String mit Länge >0) und einen gewünschten Mindesterlös (ganze Zahl >0 und <=10.000.000). Es gehört einer Warenkategorie (ein Wert aus der vordefinierten Enumeration **`Kategorie`**: **`Kategorie::Schmuck`**, **`Kategorie::Moebel`**, **`Kategorie::Kunst`**, **`Kategorie::Sonstiges`**) an und zusätzlich ist für jedes **`Lot`** eine Liste der zugehörigen Gebote zu verwalten. Folgende Methoden und Operatoren sind für die Klasse **`Lot`** zu implementieren:

- Konstruktor(en) mit zwei und drei Parametern. Bezeichnung, gewünschter Mindesterlös und Warenkategorie in dieser Reihenfolge. Die Warenkategorie ist optional mit Defaultwert **`Kategorie::Sonstiges`**. Entspricht einer der Parameter nicht den Vorgaben (z.B. leere Bezeichnung oder Mindesterlös nicht im erlauben Bereich), so ist eine Exception vom Typ **`runtime_error`** zu werfen. Die Liste der Gebote für neu erstellte **`Lot`** Objekte ist immer leer.
- **`bool ist_offen() const`**: Retourniert **`true`**, falls das **`Lot`** einer der Kategorien **`Kategorie::Schmuck`** oder **`Kategorie::Kunst`** angehört (Waren aus diesen Kategorien werden offen versteigert, andere verdeckt), **`false`** sonst.
- **`bool bieten(const Gebot&)`** : Retourniert **`false`**, falls es sich um eine offene Versteigerung handelt und der gebotene Betrag kleiner ist als der gewünschte Mindesterlös. Sonst ist eine Exception vom Typ **`runtime_error`** zu werfen, wenn dieselbe Bieterin schon einen höheren Betrag geboten hat. Andernfalls wird das neue Gebot am Ende der Liste der Gebote für dieses **`Lot`** eingefügt und **`true`** retourniert.
- **`operator<<:`** Die Ausgabe eines Objekts des Typs **`Lot`** soll in der Form [*Bezeichnung*: *Mindesterlös* Euro, *Kategorie* {*Liste der Gebote*}] erfolgen, z.B.: [Kohinoor: 42 Euro, Schmuck {[Susi: 263 Euro], [Erwin: 43 Euro]}]. Der vordefinierte Vektor **`kategorie_namen`** kann für die Ausgabe der Enumerationswerte verwendet werden.
- Zusatz für 10 Punkte: **`Gebot zuschlag() const`**: Ein Zuschlag kann nicht erfolgen, wenn gar kein Gebot vorliegt oder im Fall einer verdeckten (nicht offenen) Auktion das höchste Gebot nicht zumindest so hoch ist, wie der gewünschte Mindesterlös. Kann kein Zuschlag erfolgen, dann ist eine Exception vom Typ **`runtime_error`** zu werfen. Andernfalls erfolgt der Zuschlag an das höchste Gebot. Sollten mehrere gleich hohe Maximalgebote existieren, so erfolgt der Zuschlag an das zuerst abgegebene (das erste Maximalgebot, wenn man die Liste der Gebote nach aufsteigenden Positionsindizes durchläuft). Das Gebot, das den Zuschlag erhält, ist zu retournieren.
- Zusatz für 15 Punkte: **`static vector<Lot> probleme(vector<Lot>&)`** : Lots, für die keine Gebote abgegeben wurden, bzw. die verdeckt versteigert werden und für die kein **`Gebot`** zumindest so hoch ist wie der gewünschte Mindesterlös, gelten als Problemfälle. Die Methode **`probleme`** muss alle Problemfälle aus der als Parameter erhaltenen Liste von Lots entfernen (unter Beibehaltung der relativen Reihenfolge). Eine Liste aller entfernten Lots (in derselben Reihenfolge, in der sie in der ursprünglichen Lotliste auftraten) ist zu retournieren.

Implementieren Sie die Klassen **`Gebot`** und **`Lot`** 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.