Geschichtete Smoothies

Implementieren Sie die Klassen **`Zutat`** und **`Smoothie`** zur Unterstützung der Verwaltung einer Bar.

Eine Zutat hat einen Namen (String mit Länge>0) und einen physiologischen Brennwert (in kJ, ganze Zahl zwischen 10 und 75 jeweils inklusive). Folgende Methoden und Operatoren sind in der Klasse **`Zutat`** zu implementieren:

- Konstruktor(en) mit zwei Parametern und einem Parameter. Name und Brennwert in dieser Reihenfolge. Der Name muss immer spezifiziert werden, bei fehlender Angabe des Brennwerts ist der Defaultwert 32 zu verwenden. Sollte einer der übergebenen Werte nicht den Anforderungen entsprechen (z.B. Name mit Länge 0, bzw. Brennwert nicht im erlaubten Bereich), so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`int brennwert() const`**: Liefert den Brennwert der Zutat zurück.
- **`operator==:`** Vergleicht zwei Zutaten und liefert **`true`**, wenn sie in Name und Brennwert übereinstimmen, **`false`** sonst.
- **`operator<<`**: Die Ausgabe eines Objekts des Typs **`Zutat`** soll in der Form [Name Brennwert kJ] erfolgen, z.B.: [Apfel 27 kJ].

Ein Smoothie besteht aus einer Aufeinanderfolge von Schichten einzelner Zutaten. Die gleiche Zutat kann mehrmals zu einem Smoothie hinzugefügt werden, um dickere Schichten oder gewünschte Muster zu erzeugen. Ein Smoothie hat eine Bezeichnung (String mit Länge > 0) sowie eine Liste von Zutaten in der Reihenfolge, wie sie (von unten) aufgeschichtet werden. Folgende Methoden und Operatoren sind in der Klasse **`Smoothie`** zu implementieren:

- Konstruktor mit einem Parameter, der die Bezeichnung des Smoothies festlegt. Es gibt keinen Defaultwert. Wird eine Bezeichnung mit Länge 0 übergeben, so ist eine Exception vom Typ **`runtime_error`** zu werfen.
- **`void hinzu(const Zutat& z)`**: Fügt eine Schicht mit einer weiteren Zutat zum Smoothie oben hinzu.
- **`int brennwert() const:`** Berechnet den gesamten Brennwert des Smoothies als Summe der Brennwerte der verwendeten Zutaten.
- **`operator<<:`** Die Ausgabe eines Objekts des Typs **`Smoothie`** soll in der Form [{Liste der Zutaten}, Bezeichnung] erfolgen. Die Liste der Zutaten ist in der Reihenfolge auszugeben, wie sie hinzugefügt wurden, z.B.: [{[Apfel 27 kJ], [Birne 31 kJ], [Apfel 27 kJ]}, Turm von Hanoi]
- Zusatz für 10 Punkte: **`void unterheben(const Zutat& z)`**: Eine geschickte Bartenderin schiebt unter jede schon vorhande Schicht eine weitere Schicht der Zutat z. Sollte der Smoothie zu Beginn noch gar keine Schicht enthalten, so ist eine Exception vom Typ **`runtime_error`** zu werfen. Z.B.: Der Smoothie enthalte die Schichten a, b und a. Ein Aufruf von **`unterheben`** mit der Zutat c ändert den Smoothie so, dass er nun aus den Schichten c, a, c, b, c und a besteht.
- Zusatz für 15 Punkte: **`void liste(ostream& o) const`** soll eine Einkaufsliste der benötigten Zutaten auf den Stream **`o`** ausgeben. Die Zutaten sind in der Reihenfolge auszugeben, wie sie aufgeschichtet sind, allerdings jede Zutat nur einmal mit einer Angabe, wie oft sie benötigt wird. Ob in zwei Schichten dieselbe Zutat verwendet wird, kann mit **`Zutat::operator==`** geprüft werden. Für den Smoothie „Turm von Hanoi“ (siehe **`operator<<`**) würde die ausgegebene Liste so aussehen: [Apfel 27 kJ]*2, [Birne 31 kJ]*1.

Implementieren Sie die Klassen **`Zutat`** und **`Smoothie`** 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.