Neuerungen durch C++

In diesem Praktikum trainieren Sie Objekte in einen Container (`std::map` und `std::list`) zu füllen und den gesamten Container-Inhalt durzugehen. Sie arbeiten mit Ein- und Ausgabe-Streams weiter.

Einstieg

1. **Eine Zähler-Klasse**
   (a) Erzeugen Sie mit CMake eine Projektmappe für die von Ihnen verwendete Entwicklungsumgebung. Sie soll zunächst lediglich die Datei `main.cpp` enthalten.
   (b) Legen Sie in einem separaten Paar aus Header- (`.hpp`) und Quelltextdatei (`.cpp`) eine neue Klasse `Counter` an. Diese enthält eine private Membervariable `name_` vom Typ `std::string`. Diese wird initialisiert mit dem Wert des Parameters, der dem Konstruktor übergeben wird. Weiterhin enthält diese Klasse eine private Membervariable `count_` vom Typ `unsigned int`, die mit `0` initialisiert wird.
   (c) Implementieren Sie in der Klasse `Counter` eine öffentliche Methode `void Touch(unsigned int increment)`, die die Membervariable `count_` um den als Parameter `increment` angegebenen Wert erhöht.
   (d) Implementieren Sie in der Klasse `Counter` eine öffentliche Methode `void Print() const`, die die Inhalte der Membervariablen in folgendem Format ausgibt:

```
[name_]:[count_]
```

   (e) Erstellen Sie in der Datei `main.cpp` eine `main`-Funktion. Erzeugen Sie dort eine Instanz der Klasse `Counter`. Rufen Sie mehrfach deren Methode `Touch` mit verschiedenen Ganzzahlen als Parameter auf. Rufen Sie anschließend deren Methode `Print` auf.

2. **Ein Zähler-Katalog**

   Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.
   (a) Legen Sie in einem separaten Paar aus Header- (`.hpp`) und Quelltextdatei (`.cpp`) eine neue Klasse `CounterCatalogue` an. Diese enthält eine private Membervariable `counters_` vom Typ `std::map<std::string, Counter>`, also ein Wörterbuch, das einer Zeichenkette einen Zähler zuordnet.
   (b) Implementieren Sie in der Klasse `CounterCatalogue` eine öffentliche Methode `void Add(const std::string&)`, mit der Zähler in `counters_` hinzugefügt werden. Der übergebene Parameter dient sowohl als Key in `counters_` als auch als Name des jeweiligen Zählers.
   Verwenden Sie zum Hinzufügen zu `counters_` die Methode `insert` der `std::map`[a] und nicht die eckigen Klammern.
   erläuterndes Beispiel:

```
std::map<char, int> m;
m.insert({'a', 1});
m.insert({'b', 2});
```

   [a] `https://en.cppreference.com/w/cpp/container/map/insert`

   (c) Implementieren Sie in der Klasse `CounterCatalogue` eine öffentliche Methode `void Touch(const std::string& name, unsigned int increment)`, die den enthaltenen Zähler mit Namen `name` um `increment` erhöht.
   Verwenden Sie zum Elementzugriff- die Methode `at` der `std::map`[a] und nicht die eckigen Klammern.
   erläuterndes Beispiel (Fortsetzug von oben):

```
std::cout << m.at('b') << std::endl;
```

   [a] `https://en.cppreference.com/w/cpp/container/map/at`

   (d) Implementieren Sie in der Klasse `CounterCatalogue` eine öffentliche Methode `void Print() const`, die die `Print`-Methode jedes enthaltenen Zählers aufruft. Verwenden Sie eine range-based for-Schleife und den Datentyp `auto`.

(e) Legen Sie in der `main`-Funktion eine `CounterCatalogue`-Instanz an und fügen dieser mehrere Zähler mit verschiedenen Namen hinzu. Lassen Sie den Benutzer mit der Tastatur einen Zählernamen und ein Inkrement im Format

```
[Name] [Inkrement]
```

angeben, bis der Nutzer durch Drücken von Strg+D signalisiert, dass keine weiteren Zahler und Inkremente mehr angegeben werden sollen. In diesem Fall wird die `Print`-Methode der `CounterCatalogue`-Instanz aufgerufen.

3. Sortierte Ausgabe

Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

(a) Ändern Sie die Methode `Print` der Klasse `CounterCatalogue` so, dass die enthaltenen Zähler zunächst in eine lokale Liste vom Typ `std::list<Counter>` übertragen werden. Erst in einer zweiten Schleife werden die Counter dieser Liste ausgegeben.

(b) Implementieren Sie in der Klasse `Counter` den öffentlichen Operator `bool operator<(const Counter& other) const`, mit dem die Zähler `*this` und `other` verglichen werden können.
Implementieren Sie diesen Operator so, dass er `true` zurückgibt, falls

```
this->count_ > other.count_
```

Sind die beiden `count_`-Attribute gleich, gibt der Operator `true` zurück, falls `this->name_` lexikografisch kleiner ist als `other.name_`. Verwenden Sie für den lexikografischen Vergleich den Operator `operator<`[a] der Klasse `std::string`.
Testen Sie diesen Operator. Stellen Sie sicher, dass er korrekt funktioniert.

[a] `https://en.cppreference.com/w/cpp/string/basic_string/operator_cmp`

(c) Ändern Sie die Methode `Print` der Klasse `CounterCatalogue` so, dass die Zähler sortiert ausgegeben werden. Verwenden Sie dazu die Methode `sort`[a] der Klasse `std::list`.

[a] `https://en.cppreference.com/w/cpp/container/list/sort`

Aufgaben für die Abgabe

Die folgenden Aufgaben setzen die Projektmappe für die Abgabe des vorherigen Praktikums fort.

4. Eine Liga

Arbeiten Sie mit der Projektmappe weiter, die für die Abgabe zum vorherigen Praktikum entstanden ist.

(a) Legen Sie in einem separaten Paar aus Header- (`.hpp`) und Quelltextdatei (`.cpp`) eine neue Klasse für eine gesamte Liga an. Diese enthält eine `std::map` als private Membervariable, die einem Team-Kürzel eine `TeamStatistics`-Instanz zurodnet (siehe vorheriges Praktikum).

(b) Implementieren Sie in der Liga-Klasse eine öffentliche Methode, mit der Team-Statistiken hinzugefügt werden. Als Parameter soll das Team-Kürzel angegeben werden.

(c) Implementieren Sie in der Liga-Klasse eine öffentliche Methode, die Spielpaarungen samt Ergebnis verarbeitet: Als Parameter werden die Kürzel der Heim- und Auswärtsmannschaft sowie die durch die jeweilige Mannschaft erzielten Tore übernommen. Die Methode aktualisiert die enthaltenen Team-Statistiken entsprechend.

(d) Implementieren Sie in der Liga-Klasse eine öffentliche Methode, die sämtliche Team-Statistiken am Bildschirm ausgibt.

(e) Legen Sie in der `main`-Funktion eine Instanz der Liga-Klasse an.
Mit der Tastatur soll zunächst die Anzahl der aufzunehmenden Teams angegeben werden. Anschließend werden zeilenweise Team-Kürzel angegeben. Danach folgen Spielpaarungen in folgendem Format:

```
[Heim-Kürzel] [Heim-Tore] [Auswärts-Tore] [Auswärts-Kürzel]
```

Es können so lange Spielpaarungen angeben werden, bis der Nutzer durch Drücken von Strg+D signalisiert, dass keine weiteren Eingaben gemacht werden sollen. In diesem Fall werden sämtliche Team-Statistiken ausgegeben.
Beispielsweise soll folgende Eingabe

```
2
AAA
BBB
AAA 1 0 BBB
BBB 2 0 AAA
```

folgende Ausgabe erzeugen:

```
AAA 2 3 1:2 -1
BBB 2 3 2:1 +1
```

5. Die Meisterschafts-Tabelle

Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

(a) Implementieren Sie in der Klasse `TeamStatistics` den öffentlichen Operator `bool operator<(const TeamStatistics& other) const`. Der Vergleich ergibt `true`, wenn

- `*this` mehr Punkte erzielt hat als `other`,
- bei Punktgleichheit `*this` eine höhere Tordifferenz hat als `other`,
- zusätzlich bei gleicher Tordifferenz `*this` mehr Tore erzielt hat als `other`,
- zusätzlich bei Tor-Gleichheit das Kürzel von `*this` lexikografisch kleiner ist als das Kürzel von `other`.

(b) Ändern Sie in der Liga-Klasse die Methode, die sämtliche Team-Statistiken am Bildschirm ausgibt, sodass eine nach obigen Regeln sortierte Tabelle angezeigt wird.
Beispielsweise soll folgende Eingabe

```
2
AAA
BBB
AAA 1 0 BBB
BBB 2 0 AAA
```

folgende Ausgabe erzeugen:

```
BBB 2 3 2:1 +1
AAA 2 3 1:2 -1
```

VERTIEFUNG

6. Überladene Funktionen

Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

Implementieren Sie eine freistehende Funktion `Print`, mit jeweils einer Überladung für die folgenden Parameter:

- eine Team-Statistik

- eine Liga
- ein Spielergebnis (aus der Vertiefung des vorherigen Aufgabenblatts).

7. **Stream-Operator**

   Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

   Implementieren Sie für die angelegten Klassen den Ausgabe-Stream-Operator `operator<<`. Verändern Sie alle Methoden, die direkt auf `std::cout` arbeiten. Verwenden Sie stattdessen den implementierten Stream-Operator.

   Auf diese Weise werden Ihre Klassen unabhängig von einem konkreten Stream und können flexibler eingesetzt werden.

   Wichtig: Gestatten Sie nach wie vor keinen Zugriff von außen auf private Member – weder durch Getter noch durch `friend`. Überlegen Sie sich eine andere Methode.