C/C++ Programmierung
Übungsblatt 7

Florian Oetke

1 Vektor Statistiken mit C++-Algorithmen

Die Aufgabenstellung in dieser Aufgabe ist identisch zu Aufgabe 3 auf dem letzten Übungsblatt. Hier sollen Sie allerdings versuchen, die beschriebene Funktionalität ohne eine eigene Schleife, sondern unter Verwendung passender Algorithmen aus der C++ Standardbibliothek zu implementieren.

Schreiben Sie eine Funktion `statistics`, die einen `std::vector<float>` als Parameter erhält und für diesen einige Statistiken berechnet und zurückgibt.

Der Rückgabewert soll ein `struct Statistics` sein, das Sie ebenfalls noch schreiben müssen und das folgende Member-Variablen enthält:

- `min`: Der kleinste Wert im Vector
- `max`: Der größte Wert im Vector
- `avg`: Der Durchschnitt aller Werte im Vector
- `range`: Eine `enum class Number_range`, die Sie ebenfalls noch schreiben müssen und die angibt, ob alle Werte ≥ 0 (`positive`) bzw. ≤ 0 (`negative`) sind oder ob sowohl positive als auch negative Werte im Vector vorkommen (`mixed`)

Als Basis für die Implementierung können Sie entweder Ihre Lösung vom letzten Übungsblatt oder die Vorlage aus der Handout ZIP-Datei verwenden.

```cpp
#include <iostream>
#include <vector>

// TODO

int main() {
    const auto data = std::vector<float>{-20.f, 14.1f, 9.f, 9.f, 200.f};
    const auto stats = statistics(data);

    std::cout << "min   : " << stats.min << "\n"
              << "max   : " << stats.max << "\n"
              << "avg   : " << stats.avg << "\n"
              << "range : ";

    switch(stats.range) {
        case Number_range::positive:
            std::cout << "positive\n";
            break;
        case Number_range::negative:
            std::cout << "negative\n";
            break;
        case Number_range::mixed:
            std::cout << "mixed\n";
            break;
    }
}
```

2 Häufigsten Wörter aus Wort-Histogramm

In dieser Aufgabe geht es darum, die Wort-Histogramm-Aufgabe vom letzten Übungsblatt weiter zu verbessern. Falls Sie die Aufgabe nicht bearbeitet oder eine andere Datenstruktur gewählt haben, können Sie diese Aufgabe aber trotzdem bearbeiten.

Als Basis für die Implementierung können Sie folgenden Programmcode verwenden:

```cpp
#include <algorithm>
#include <iostream>
#include <string>
#include <unordered_map>
#include <vector>


std::unordered_map<std::string, int> count_words() {
    auto words = std::unordered_map<std::string, int>{};

    // TODO: mit tatsaechlichem Code ersetzen
    words = std::unordered_map<std::string, int>{
        {"verdeutlich", 1}, {"der", 1}, {"aus", 1}, {"den", 1},
        {"datei", 1}, {"mit", 1}, {"beispiel", 3}, {"zum", 1},
        {"dies", 1}, {"uns", 1}, {"wir", 1}, {"koenn", 1}
    };

    return words;
}

int main() {
    auto words = count_words();

    // TODO
}
```

Wenn Sie die Aufgabe bearbeitet und mit einer `std::unordered_map<std::string, int>` gelöst haben, können Sie Ihre Lösung in der `count_words` Funktion einfügen. Sie können aber auch den Platzhalter in der Funktion belassen und nur den Rest der Aufgabe unten bearbeiten.

Das Ergebnis der `count_words` Funktion ist eine `std::unordered_map<std::string, int>`, die jedem Wort aus der Eingabe seine Häufigkeit zuordnet. Ergänzen Sie die `main` Funktion so, dass die 5 häufigsten Wörter, sortiert nach ihrer Häufigkeit ausgegeben werden. Verwenden Sie hierzu einen geeigneten Algorithmus aus der C++ Standardbibliothek.

Bei der Eingabe `Wenn wir uns zum Beispiel mal dieses Beispiel anschauen A A A STOP` sollte die Ausgabe z.B. so aussehen (welche Wörter genau ausgegeben werden und in welcher Reihenfolge kann abhängig von Ihrem Vorgehen abweichen):

```
1. A (3)
2. Beispiel (2)
3. dieses (1)
4. mal (1)
5. zum (1)
```

3 Verarbeitung von Log-Einträgen

In dieser Aufgabe sollen Sie sich etwas mit den, in der Vorlesung vorgestellten, Algorithmen aus der C++ Standardbibliothek vertraut machen. Implementieren Sie dazu, die folgenden Funktionen zum Verarbeiten eines `std::vector<Log_entry>`, ohne selbst eigene Schleifen zu schreiben:

- `remove_old_entries`: Bekommt neben dem Vector einen Zeitstempel vom Typ `std::int64_t` und entfernt alle Einträge aus dem Vector deren Zeitstempel kleiner ist.
- `count_system_entries`: Gibt die Anzahl der Einträge zurück, deren `source=="SYSTEM"` ist.
- `contains`: Bekommt neben dem Vector einen String und gibt `true` zurück, falls es mindestens einen Eintrag gibt, dessen `message` den übergebenen String **enthält**.

Die Ausgabe des fertigen Programms sollte so aussehen:

```
System?   : 3
Requests? : 1
Requests? : 0

3 [UI] Button pressed
3 [SYSTEM] Please do not press this button again
6 [SYSTEM] Terminated
```

Als Basis für die Implementierung können Sie folgenden Programmcode verwenden:

```
 1 #include <algorithm>
 2 #include <iostream>
 3 #include <string>
 4 #include <vector>
 5
 6 struct Log_entry {
 7     std::int64_t timestamp;
 8     std::string  source;
 9     std::string  message;
10 };
11 // TODO
12
13 int main() {
14     auto logs = std::vector<Log_entry> {
15         Log_entry{1, "SYSTEM", "System startet"},
16         Log_entry{2, "APP", "Received request"},
17         Log_entry{3, "UI", "Button pressed"},
18         Log_entry{3, "SYSTEM", "Please do not press this button again"},
19         Log_entry{6, "SYSTEM", "Terminated"}
20     };
21
22     std::cout << "System?   : " << count_system_entries(logs) << "\n";
23     std::cout << "Requests? : " << contains(logs, "request") << "\n";
24
25     remove_old_entries(logs, 3);
26     std::cout << "Requests? : " << contains(logs, "request") << "\n\n";
27
28     for(auto& entry : logs) {
29         std::cout << entry.timestamp << " [" << entry.source << "] "
30                   << entry.message << "\n";
31     }
32 }
```

4 Musikbibliothek

In dieser Aufgabe geht es darum, den Umgang mit mehreren Containern zu vertiefen und sich weiter mit Pointern und Iteratoren auseinanderzusetzen.

Ergänzen Sie den im Handout vorgegebenen Code um eine Klasse `Music_library`, die eine Menge von `Song` Objekten verwaltet und die folgenden Methoden implementiert:

- `insert`: Bekommt einen `Song` und speichert ihn in der Klasse. Sie können davon ausgehen, dass ein Artist zwar mehrere Songs haben kann, aber die Titel der einzelnen Songs und Namen der Artists eindeutig sind.
- `find_by_title`: Bekommt einen `std::string` als Parameter und gibt entweder einen Pointer auf den `Song` mit dem Titel oder `nullptr` zurück, falls es keinen entsprechenden Eintrag gibt.

  `find_by_artist`: Bekommt einen `std::string` als Parameter und gibt entweder einen Pointer auf einen `std::vector<Song*>` mit allen Songs eines Artists oder `nullptr` zurück, falls es keinen entsprechenden Eintrag gibt.

Überlegen Sie sich, welche Member-Variablen und Container Sie benötigen, um diese Funktionen effizient zu implementieren. Unter Umständen ist es hier angebracht, wie in Vorlesung 6 gezeigt, eine Map zu verwenden, um Pointer auf Objekte zu verwalten, die in einem anderen Container gespeichert werden. Bedenken Sie hierbei auch die Regeln für Pointer-Invalidation, die in der Vorlesung angesprochen wurden.

Die Ausgabe des fertigen Programms sollte so aussehen:

```
Song gefunden: Blind Guardian - The Bard's Song
Artist gefunden:
    The Bard's Song
    Valhalla
    Sacred Worlds
    Silence
```

5 Brüche

Fließkommazahlen wie `float` und `double` sind in C++ nur Approximationen, mit der nicht alle rationalen Zahlen exakt abgebildet werden können. In dieser Aufgabe sollen Sie nun im Gegensatz dazu die Klasse `Fraction` aus dem Handout ergänzen, mit der Brüche exakt repräsentiert werden können.

Ihre Klasse sollte den Zähler (`numerator_`) und den Nenner (`denominator_`) als Member-Variablen vom Typ `int` speichern und die folgenden Funktionen und Operatoren implementieren:

- Einen `operator<<` um den Bruch auf einen IO-Stream auszugeben (bereits im Handout vorgegeben)
- Eine Member-Funktion `reciprocal`, die den Kehrwert des Bruchs zurückgibt
- Einen unären `operator-()`
- Eine Member-Funktion `simplified`, die den gekürzten Bruch zurückgibt und die Sie auch in ihrem `operator<<` verwenden sollten
- Die Operatoren `+`, `-`, `*` und `/`, die auf zwei Brüchen als Operanden arbeiten. Überlegen Sie, ob Sie die Operatoren besser als Member-Funktionen oder als freie Funktionen implementieren sollten.

Beachten Sie, dass keine der oben genannten Funktionen das Objekt selbst ändert, sondern ggf. immer nur ein neuer Bruch mit dem entsprechenden Wert zurückgegeben wird.

Darüber hinaus sollte Ihre Klasse einen Konstruktor mit Zähler und Nenner haben und eine implizite Konvertierung von `int` zu `Fraction` erlauben (notwendig für `(x / 2)` in Z.17).

Bei der Implementierung der Funktionen und Operatoren müssen Sie u.a. das kleinste gemeinsame Vielfache und den größten gemeinsamen Teiler zweier Zahlen berechnen. Hierfür haben wir am Ende von Vorlesung 7 entsprechende Hilfsfunktionen kennengelernt, die Sie ggf. benutzen wollen.

Als Basis für die Implementierung können Sie den im Handout bereitgestellten Programmcode verwenden, der bei einer korrekten Lösung die folgende Ausgabe erzeugen sollte:

```
       x         = 1/2
      -x         = -1/2
 x.reciprocal() = 2/1
     21/28       = 3/4

x + y = 9/10
x - y = 1/10
x * y = 1/5
x / y = 5/4
x / 2 = 1/4
```