Dr. Ole Klein, Dr. Steffen Müthing

IWR, Universität Heidelberg

Abgabe 1. Dezember 2017

Übung 1 *Statistik*

Bevor Sie diese Übung auf einem der Pool-Computer bearbeiten: Aktualisieren Sie Ihren Compiler wie auf Blatt 3 beschrieben, falls Sie dies noch nicht getan haben!

In dieser Aufgabe berechnen wir ein paar grundlegende Statistiken zu einer Menge von Zahlen.

Um Vektoren zu erstellen, mit denen Sie die Statistikfunktionen testen können, laden Sie folgende Dateien herunter:

`https://conan.iwr.uni-heidelberg.de/data/teaching/ipk_ws2017/io.hh`
`https://conan.iwr.uni-heidelberg.de/data/teaching/ipk_ws2017/io.cc`

Diese Dateien enthalten Routinen, um einen Vektor entweder von der Standardeingabe einzulesen oder zufällig zu erzeugen (siehe Kommentare in `io.hh`).

Aufgaben:

(a) Erstellen Sie drei Programme `readvector`, `uniform` und `normal`, die jeweils eine der drei unterschiedlichen Generatorfunktionen in `io.hh` verwenden und den Vektor dann auf die Standardausgabe schreiben.

Alle Funktionen in den folgenden Aufgaben sollen in wiederverwendbarer Weise geschrieben werden: Erstellen Sie einen Header `statistics.hh` und eine Implementierungsdatei `statistics.cc` für die Funktionen. Den Header includen Sie dann in jedem der drei Programme aus der ersten Aufgabe und wenden die Funktion(en) in jedem der drei Programme jeweils auf den eingelesenen Vektor an. Hierfür entfernen Sie am besten wieder den Code, der alle Vektoreinträge schreibt. Die Programme sollen immer alle schon implementierten Statistiken ausgeben.

(b) Erstellen Sie eine Funktion `double mean(const std::vector<double>& v)`, die den Mittelwert $\mathbb{E}[v]$ aller Einträge in dem Vektor v zurückliefert:

$$\mathbb{E}[v] = \mu = \frac{1}{N}\sum_{i=1}^{N} v_i,$$

wobei N die Anzahl der Einträge im Vektor angibt.

(a) Erstellen Sie eine Funktion `double median(const std::vector<double>& v)`, die den Median $M(v)$ aller Einträge in dem Vektor v zurückliefert. Um den Median zu berechnen, erstellen Sie eine sortierte Kopie $\tilde{v}$ von v und bestimmen $M(v)$ als

$$M(v) = \begin{cases} \tilde{v}_{\frac{N+1}{2}} & N \text{ ungerade} \\ \frac{1}{2}\left(\tilde{v}_{\frac{N}{2}} + \tilde{v}_{\frac{N}{2}+1}\right) & N \text{ gerade} \end{cases}$$

wobei N die Anzahl der Einträge im Vektor angibt. Beachten Sie die unterschiedlichen Indizierungsstrategien (1-basiert in der Formel oben, 0-basiert in C++) sowie den Spezialfall eines leeren Vektors!

(b) Erstellen Sie eine Funktion `double moment(const std::vector<double>& v, int k)`, die das k-te statistische Moment m_k aller Einträge in dem Vektor zurückliefert:

$$m_k = \mathbb{E}[v^k] = \frac{1}{N}\sum_{i=1}^{N} v_i^k,$$

wobei N die Anzahl der Einträge im Vektor angibt.

(c) Erstellen Sie eine Funktion `double standard_deviation(const std::vector<double>& v)`, die die Standardabweichung des Vektors berechnet:

$$s = \left(\mathbb{E}[(v-\mu)^2]\right)^{\frac{1}{2}}.$$

Überprüfen Sie, ob folgende Relation gilt:

$$\mathbb{E}[(v-\mu)^2] = \mathbb{E}[v^2] - \mathbb{E}[v]^2.$$

(Grundlagen)

Übung 2 *CMake*

In dieser Aufgabe erweitern Sie die vorherige Aufgabe um ein CMake-basiertes Buildsystem. Verwenden Sie hierfür die Informationen aus der Vorlesung.

Wichtig: Falls Sie die Aufgabe auf einem der Pool-Rechner bearbeiten, lassen Sie die Zeile

```
set(CMAKE_CXX_STANDARD 14)
```

weg, die CMake-Version auf diesen Rechnern ist zu alt, um den Befehl zu verstehen. Dafür müssen Sie auf diesen Rechnern CMake mitteilen, dass Sie einen speziellen Compiler verwenden wollen:

```
cmake -DCMAKE_CXX_COMPILER=ipkc++ <weitere Optionen> <Pfad zu CMakeLists.txt>
```

Aufgaben:

(a) Erstellen Sie die Datei `CMakeLists.txt` wie in der Vorlesung beschrieben und fügen die drei Programme hinzu. Führen Sie `cmake` und `make` in einem Unterverzeichnis aus, um die Programme zu bauen.

(b) Verändern Sie verschiedene Header und Implementierungsdateien und führen Sie `make` erneut aus, um zu sehen, welche Dateien neu kompiliert und gelinkt werden.

(c) Wie Sie sehen, werden `io.cc` und `statistics.cc` für jedes Programm einzeln übersetzt. Dies können wir vermeiden, wenn wir eine `Bibliothek` anlegen. Dies funktioniert folgendermassen:

```
# Erzeuge Bibliothek statistics
add_library(statistics statistics.cc io.cc)

# Ausführbares Programm ohne Quellen für Bibliothek
add_executable(uniform uniform.cc)
# Programm gegen die Statistik-Bibliotek linken
target_link_libraries(uniform statistics)
```

Passen Sie Ihr Buildsystem entsprechend an.

(d) Erstellen Sie ein Unterverzeichnis `release-build` und rufen dort ebenfalls CMake auf. Setzen Sie beim CMake-Aufruf den Build Type auf `Release` (siehe Vorlesung).

Vergleichen Sie die Laufzeit Ihrer Programme in den beiden Verzeichnissen für große, zufällig erzeuge Vektoren ($\approx 10^6 - 10^8$ Einträge).

Übung 3 *Statistik mit Templates*

In der grundlegenden Statistik-Aufgabe haben wir ein paar einfache Funktionen geschrieben, die leider nur mit `std::vector<double>` funktionieren. Im folgenden wollen wir diese Funktionen allgemeiner machen, indem wir das C++-Feature *Templates*[1] verwenden.

Aufgaben:

(a) Verändern Sie die statistischen Funktionen so, dass sie statt `std::vector<double>` einen Template-Parameter akzeptieren. Damit sollte es auch möglich sein, die Funktion für andere Typen, z.B. ein `std::array<float>`, aufzurufen. Testen Sie das!

Tipp: Wenn Ihr Container den Template-Typ `T` hat, können Sie eine Variable vom enthaltenen Element-Typ mit `typename T::value_type x = ...;` anlegen.

Warum ist es schwieriger, die IO-Funktionen auch fit für arrays zu machen?

(b) Auf `cppreference.com` gibt es eine lange Liste mit weiteren Zufallsverteilungen[2]. Es wäre sehr praktisch, eine Funktion zu haben, der man einfach die gewünschte Verteilung mitgeben könnte, um den Vektor damit zu füllen. Außerdem soll die Funktion einen Container der passenden Größe als Parameter bekommen und einfach alle Einträge mit Zufallszahlen überschreiben, so dass folgendes funktioniert:

```
1 std::uniform_distribution<double> dist(0.0,1.0);
2 std::array<double,10> v;
3 int seed = 42;
4 fill_random(42,dist,v);
```

Schreiben Sie auf Basis der Funktionen in `io.cc` die Funktion `fill_random` als Template-Funktion, so dass sie mit beliebigen Distributionen und mindestens mit `std::array` und `std::vector` funktioniert und testen Sie das.

(c) **Bonus**
Überarbeiten Sie die Funktionen so, dass sie (ähnlich wie die Standard Library) zwei Iteratoren als Parameter nehmen statt eines Containers.

[1] https://en.wikipedia.org/wiki/Template_(C%2B%2B)
[2] http://en.cppreference.com/w/cpp/numeric/random