Andreas Naumann
IWR, Universität Heidelberg

Übung 5

Aufgabe 1: Statistiken

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 die Dateien `io.h` und `io.cc` von der Vorlesungs-Homepage[1] herunter. Diese Dateien enthalten Routinen, um einen Vektor entweder von der Standardeingabe einzulesen oder zufällig zu erzeugen (siehe Kommentare in `io.h`).

Aufgaben:

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

Alle Funktionen in den folgenden Teilaufgaben sollen in wiederverwendbarer Weise geschrieben werden: Erstellen Sie einen Header `statistics.h` und eine Implementierungsdatei `statistics.cc` für die Funktionen. Den Header includen Sie dann in jedem der drei Programme aus der Teilaufgabe a) 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.

(c) 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!

(d) 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.

[1] https://scoop.iwr.uni-heidelberg.de/teaching/2023ss/grundkurscpp/

(e) 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:

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

Aufgabe 2: CMake

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

Hinweis: Sie müssen nur die Teilaufgabe a) der vorherigen Aufgabe lösen. Falls Sie die anderen Teilaufgaben nicht lösen konnten, schreiben Sie einfach Dummy-Implementierungen für die diversen Statistik-Funktionen, die immer 0 zurückgeben.

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. Um eine Datei als verändert zu markieren, können Sie einfach Leerzeilen hinzufügen / entfernen oder `touch <dateiname>` aufrufen.

(c) Wie Sie sehen, werden `io.cc` und `statistics.cc` für jedes Programm einzeln übersetzt. Legen Sie eine Bibliothek mit diesen beiden Dateien an, um die Mehrfachkompilierung zu vermeiden.

(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).

(e) Erstellen Sie Tests für die Funktionen `mean()` und `median()`, die jeweils folgende Inputs testen:

- Einen leeren Vektor
- Einen im Test vorgegebenen Vektor mit 4 Einträgen
- Einen im Test vorgegebenen Vektor mit 5 Einträgen

Die Tests sollen jeweils überprüfen, ob die Funktion das korrekte Ergebnis zurückgibt und das wie in der Vorlesung beschrieben an CMake signalisieren. Wenn Sie möchten, können Sie dafür wie in der Vorlesung gezeigt die Funktion `assert()`[2] verwenden. Sie können entweder separate Programme für jeden Test erstellen oder nur jeweils ein Programm für jede Funktion. Binden Sie die Tests wie in der Vorlesung gezeigt ins Buildsystem ein und führen Sie sie mit `ctest` aus.

Hinweis: Falls Ihr Test wider Erwarten fehlschlägt, kann es sein, dass sich das Ergebnis aufgrund von Rundungsfehlern minimal unterscheidet. Vergleichen Sie zwei `double`-Werte am besten so:

```
#include <cmath>
double test_val = ...;
double reference_val = ...;
if (std::abs(test_val - reference_val) < 1e-10) {
  // test passed
}
```

[2] https://en.cppreference.com/w/cpp/error/assert