

Aufgabenblatt 7

Allgemeine Hinweise:

- Für die Aufgaben auf diesem Übungsblatt müssen Sie am 15.01. votieren.
- Lesen Sie sich die Aufgaben jeweils vollständig durch, bevor Sie anfangen, sie zu bearbeiten!
- Die Beiden Aufgaben bauen auf Lösungen früherer Übungen auf. Entsprechende Musterlösungen finden Sie auf der Vorlesungshomepage¹. Bauen Sie bitte Ihre Lösungen für diesen Zettel auf den bereitgestellten Musterlösungen auf.

Aufgabe 1: Punkte im \mathbb{R}^d

(1 Punkt)

Auf dem letzten Übungsblatt haben Sie eine Klasse geschrieben, die einem Punkt im \mathbb{R}^2 repräsentiert. In dieser Aufgabe erweitern wir die Klasse so, dass sie Punkte im \mathbb{R}^d darstellen kann.

- (a) Schreiben Sie die Klasse `Point` so um, dass Sie eine Template wird, die zwei Template-Parameter hat, einen für den Typ der Koordinate und einen für die Dimension:

```
1     template<typename Coord, int dim>
2     class Point {
3         ...
4     };
```

- Starten Sie mit Ihrer Implementierung des 2D-Punkts und verlagern Sie die komplette Implementierung in den Header.
- Verwenden Sie ein `std::array`, um die Koordinaten zu speichern.
- Die Klasse soll den Typ der Koordinaten mit einem Type Alias mit dem Namen `Coordinate` exportieren.
- Die Klasse soll die Dimension als Konstante exportieren. Dies funktioniert so:

```
1     template<typename Coord, int dim>
2     class Point {
3     public:
4         static const int dimension = dim;
5     };
```

Auf diese Konstante kann man später mit der gleichen Syntax zugreifen wie auf geschachtelte Type Aliases.

- Es soll einen argumentlosen Konstruktor geben, der den Ursprung erzeugt.
 - Es soll einen Konstruktor geben, der ein `std::array` passender Länge mit den Koordinaten eines Punkts akzeptiert und in dem neuen Punkt speichert.
- (b) Fügen Sie eine Methode `Coord& x(int i)` hinzu, die die i -te Koordinate des Punktes zurückgibt (wobei i von 0 bis $dim - 1$ geht). Da diese Methode eine Referenz zurückgibt, kann man mit

```
1     Point<double,3> p;
2     p.x(2) = 3.0;
```

¹https://conan.iwr.uni-heidelberg.de/teaching/ipk_ws2020/

auch die im Punkt gespeicherten Koordinaten ändern.

- (c) Schreiben Sie eine Methode `norm()` `const`, die die Norm des Punktes als

$$\|x\| = \sqrt{\sum_{i=1}^d x_i^2}$$

berechnet und zurückgibt.

- (d) Um die Lesbarkeit beim Zugriff auf die Koordinaten zu verbessern, wäre es schön, wenn man statt `p.x(i)` einfach `p[i]` schreiben könnte. Um dies zu ermöglichen, fügen Sie der Klasse die zwei Methoden

```
1     Coord& operator[] (int i);
2     const Coord& operator[] (int i) const;
```

hinzu. Im Body dieser Funktionen geben Sie jeweils einfach den i -ten Eintrag des arrays zurück.

- (e) Sorgen Sie dafür, dass Sie ihr Programm mit CMake bauen können und schreiben Sie einen Test, der Ihre Implementierung exemplarisch in 1D und 3D prüft.

Aufgabe 2: Statistik mit Templates

(1 Punkt)

In der Statistik-Aufgabe auf Blatt 5 haben wir ein paar einfache Funktionen zum Berechnen von Statistiken geschrieben, die aber leider nur mit Zahlenmengen funktionieren, die in einem `std::vector<double>` gespeichert sind. Im folgenden verallgemeinern wir diese Funktionen, indem wir sie zu Templates umwandeln.

Aufgaben:

- (a) Nehmen Sie Ihre Lösung von Übungblatt 5 oder die Musterlösung von der Vorlesungshomepage und verwandeln Sie die Statistik-Funktionen in Templates.

Hinweise:

- Templates müssen vollständig in der Header-Datei definiert sein, die `.cc`-Datei brauchen Sie nicht mehr.
- Die Funktionen sollen statt einem `std::vector<double>` einen beliebigen Container von Zahlen akzeptieren, dessen Typ nun ein Template-Parameter ist.
- Wenn Ihr Container den Template-Typ `T` hat, können Sie eine Variable vom enthaltenen Element-Typ mit `typename T::value_type x = ...;` anlegen oder den Trick mit `auto` aus der Vorlesung verwenden.

Stellen Sie sicher, dass die Programme aus der Original-Aufgabe weiterhin ohne Änderungen funktionieren (bis auf das Entfernen von `statistics.cc` aus dem Buildsystem).

- (b) Laden Sie die Datei `statisticsdriver.cc`² herunter. Diese Datei testet Ihre Implementierung für verschiedene Containertypen. Binden Sie die Datei in Ihr Buildsystem ein und stellen Sie sicher, dass das Programm kompiliert und durchläuft.
- (c) Um den Median zu berechnen, benötigen Sie eine sortierte Kopie des Containers. Nicht alle Container in der C++-Standardbibliothek lassen sich sortieren, daher muss man für eine robuste Implementierung von `median()` nicht den Container kopieren, sondern einen leeren `std::vector` mit dem korrekten Elementtyp (aus dem Original-Container extrahiert) anlegen und den Inhalt des Original-Containers von Hand in den neuen Container kopieren. Hierfür gibt es zwei Möglichkeiten:
- Mit einem vereinfachten `for`-Loop über den Container iterieren und die Werte mit `push_back()` an den `std::vector` anfügen.

²https://conan.iwr.uni-heidelberg.de/data/teaching/ipk_ws2020/statisticsdriver.cc

- Den Vektor mit `resize()` auf die richtige Grösse bringen und die Werte mit einem Aufruf von `std::copy()`³ kopieren.

Implementieren Sie eine neue Funktion `robust_median`, welche das oben beschriebene Vorgehen umsetzt.

³<https://en.cppreference.com/w/cpp/algorithm/copy>