

Aufgabenblatt 8

Allgemeine Hinweise:

- Für die Aufgaben auf diesem Übungsblatt müssen Sie am 17.12. votieren.
 - Lesen Sie sich die Aufgaben jeweils vollständig durch, bevor Sie anfangen, sie zu bearbeiten!
-

Aufgabe 1

Templates 101

1 Punkt

In einer früheren Aufgabe haben Sie die Template-Funktion `std::swap()` verwendet, die Referenzen auf zwei Variablen gleichen Typs bekommt und diese vertauscht (dafür wird eine temporäre Variable angelegt). Schreiben Sie Ihre eigene Version der Funktion (ebenfalls als Template), mit der folgende `main()`-Funktion korrekt funktioniert:

```
1 int main(int argc, char** argv)
2 {
3     int a = 1;
4     int b = 2;
5     std::cout << "a = " << a << " b = " << b << std::endl;
6     swap(a,b);
7     std::cout << "a = " << a << " b = " << b << std::endl;
8     double c = 3;
9     double d = 4;
10    std::cout << "c = " << c << " d = " << d << std::endl;
11    swap(c,d);
12    std::cout << "c = " << c << " d = " << d << std::endl;
13    return 0;
14 }
```

Stellen Sie sicher, dass der Code nicht mehr kompiliert, wenn Sie folgende Zeile hinzufügen:

```
1 swap(a,c);
```

Aufgabe 2

Statistik mit Templates

(a): 1 Punkt, (b): 1 Punkt, (c): 1 Punkt

In der Statistik-Aufgabe auf Blatt 6 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 Blatt 6 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`¹ 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.
 - Den Vektor mit `resize()` auf die richtige Größe bringen und die Werte mit einem Aufruf von `|std::copy()|`² kopieren.

Passen Sie Ihre Implementierung entsprechend an. Zum Testen aktivieren Sie den auskommentierten Block im heruntergeladenen Testprogramm.

Aufgabe 3

Punkte im \mathbb{R}^d

(a): 1 Punkt, (b): 1 Punkt, (c): 1 Punkt (Bonus)

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 {

```

¹https://conan.iwr.uni-heidelberg.de/data/teaching/ipk_ws2018/statisticsdriver.cc

²<https://en.cppreference.com/w/cpp/algorithm/copy>

```

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 Default-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.
- Es soll eine Methode `Coord& x(int i)` geben, die die i -te Koordinate des Punktes zurückgibt. Da diese Methode eine Referenz zurückgibt, kann man mit

```

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

```

auch die im Punkt gespeicherten Koordinaten ändern.

Schreiben Sie ein Testprogramm, das Punkte in 1D, 2D und 3D anlegt und ihre Koordinaten ausgibt.

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

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

berechnet und zurückgibt.

- (c) 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.

Testen Sie, ob das ganze funktioniert!