Übungsblatt Nr. 9: in der Übungsstunde

1) Initialisieren Sie den `std::string` name mit Ihrem Namen.
   a) Geben Sie den String zwischen Gänsefüßchen aus und drucken Sie die Anzahl der Zeichen darin. Drucken Sie mit einem `range-for` alle Zeichen in `name` mit einzelnen Apostrophen. Verwenden Sie in a) zum Vergleich nicht das `using namespace std;` Statement.
   b) Schreiben Sie ein Lambda, das einen einzelnen `char` ausgibt und speichern Sie es unter `print_char`.
   c) Verwenden Sie statt des `range_for` den Algorithmus `std::for_each()`, definiert in der Headerdatei `algorithm`.
   d) Geben Sie die Zeichen von hinten nach vorn aus.
   e) Wie d) aber vom vorletzten bis zum 3. von vorn.

2) a) Anstatt `name` nur auszugeben, wollen wir den String modifizieren. Schreiben Sie ein Lambda `increment(auto c)`, das den Parameter inkrementiert. Wenden Sie das Lambda mit `for_each` auf den String an. Funktioniert es?
   b) Überlegen Sie, warum es nicht funktioniert und machen Sie die nötige Änderung (1 Zeichen fehlt!). Schreiben Sie für den Testausdruck und die Anwendung des Lambdas ein weiteres Lambda `apply(auto f, string& s)`.
   c) Verbessern Sie das Lambda `increment`: Es soll um eine beliebige Zahl `inc` inkrementieren. Versuchen Sie zuerst die Variante mit 2 Parametern `(auto& i, auto inc)`. Das wird nicht funktionieren, da `for_each` nur ein Argument übergeben will. Capturen Sie daher `inc`. In einer Eingabeschleife lesen Sie Werte für `inc` ein und wenden das Lambda an. Was müssen Sie tun, damit es funktioniert?
   d) Aufgabe c) ist natürlich extrem hässlich gelöst. Programmieren Sie statt des Lambdas `increment` das Lambda `mk_increment`, das das Lambda `increment` erzeugt und zurückgibt.

Übungsblatt Nr. 9 Hausübung: Die folgenden Aufgaben sind Pflicht und zählen 1 Punkt!

1) a) Berechnen Sie mit dem Datentyp `Bruch` die Summe $1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{9}$ und geben Sie aus.
   b) Berechnen Sie die Summe mit einer `for`-Schleife.
   c) Schreiben Sie die Funktion `double to_double(Bruch b)` , die den Quotienten als `double` zurückgibt. Geben Sie damit das Rechenergebnis auch als Dezimalzahl aus.

2) a) Programmieren Sie für `Vec2` die Multiplikation Skalar-Vektor, Vektor-Skalar und Vektor-Vektor (Skalarprodukt), d.h.
   `Vec2 operator*(double t, Vec2 a)`
   `Vec2 operator*(Vec2 a, double t)`
   `double operator*(Vec2 a, Vec2 b)`
   b) Programmieren Sie Funktionen `double len(Vec2 a)` (Länge des Vektors) und `double operator<(Vec2 a, Vec2 b)` (Winkel zwischen Vektoren):

$$\|a\| = \sqrt{\langle a, a\rangle}, \qquad \phi = \arccos \frac{\langle a, b\rangle}{\sqrt{\langle a, a\rangle \cdot \langle b, b\rangle}}$$

   Testen Sie ihre neuen Funktionen an `a{3, 4}, b{-8,6}` , indem Sie $2 * a, a * 2, \langle a, b\rangle$, `len(a)` und den Winkel zwischen `a` und `b` ausgeben.

3) a) Starten Sie mit dem String `ich` , der Ihren Namen uns ihre Matrikelnummer enthalten soll. Schreiben Sie das Lambda
   `capitalize(auto& i)` , das einen Kleinbuchstaben in einen Großbuchstaben umwandelt. Wenn Sie `cctype` inkludieren, können Sie mit `islower(i)` nachfragen, ob `i` ein Kleinbuchstabe ist und `toupper(i)` liefert den passenden Großbuchstaben zurück (ändert `i` nicht!). Nehmen Sie das Lambda `apply` der Vorbereitung, aber übergeben Sie diesmal den String per Wert (ohne Ampersand!). Testen Sie das Ganze: `apply(capitalize, name)` .
   b) Schreiben Sie auch ein Lambda `nodigit(auto& i)` , das jede Dezimalziffer durch ein Fragezeichen ersetzt. Wenden Sie auch dieses Lambda auf `ich` an.

4) Starten Sie mit dem Programm `u3.cpp`. In dieser Aufgabe wollen wir das Lambda `nodigit` analog zu `v3e.cpp` verbessern. Es soll alle Zeichen zwischen `int from, int to` durch Fragezeichen ersetzen. Diesmal soll das Lambda zur Abwechslung durch eine Funktion (kein Lambda!) `auto hide_chars(int from , int to)` erzeugt werden. Probieren Sie es aus, indem Sie
die Ziffern anonymisieren.
die Großbuchstaben anonymisieren.
die Kleinbuchstaben anonymisieren.
nichts anonymisieren.
alles anonymisieren.

Extra-Aufgabe(n): Diese Aufgaben sind freiwillig und zählen 2 Punkte!

5) Starten Sie mit `Vec2` aus Aufgabe 2 und dem Rechteck der letzten Übung: Speichern Sie die Eckpunkte `A, B, C, D` in einem `std::vector<Vec2>`. Das Programm wird übersichtlicher, wenn Sie diesem Typ einen Namen geben, etwa `Polygon`. Schreiben Sie also im globalen Raum

```
using Polygon = std::vector<Vec2>;
```

Damit können Sie ihr Polygon in `main()` so erzeugen:

```
Polygon p{A, B, C, D}; oder gleich
Polygon{{0., 0.}, {3., 0.}, {3., 2.}, {0., 2.}};
```

Berechnen Sie vor `main()` den Polygonumfang mit der Funktion

```
double umfang(const Polygon& p) {...}
```

Schreiben Sie nach der Umfangsfunktion eine Streamausgabe für das Polygon:

```
ostream& operator<<(ostream& os, const Polygon& p)...
    -> Polygon { ... } U: ...
```

und wenden Sie das ganze auf das Rechteck an!