Aufgabe 1. Das folgende Programm soll eine Klasse enthalten, mit der sich Punkte im $\mathbb{R}^3$ abspeichern lassen. Allerdings enthält es diverse Fehler. Finden Sie sie alle.

```
#include <iostream>

enum Dimension {x_dim, y_dim, z_dim};

class Punkt_3D {
public
   Punkt_3D(double x_coor,
            double y_coor,
            double z_coor)
   {
      _coors[1] = x_coor;
      _coors[2] = y_coor;
      _coors[3] = z_coor;
   }

   double get_coor(Dimension dim)
   {
      return _coors[dim];
   }

   void set_coor(Dimension dim, double coor)
   {
      _coors[dim] = coor;
   }
private
   double _coors[3];
};

void punkt_ausgeben(const Punkt_3D & punkt)
{
   std::cout << "x: " << punkt.get_coor(x_dim)
             << " y: " << punkt.get_coor(y_dim)
             << " z: " << punkt.get_coor(z_dim) << std::endl;
}

int main()
{
   Punkt_3D p1(37, 42.5, 5.2);
   Punkt_3D p2(21, 753, 2.3);

   p2.set_coor(x_dim, 12.3);
   p2.set_coor(y_dim, 17);
   p2.set_coor(z_dim, 3.1);

   punkt_ausgeben(p1);
   punkt_ausgeben(p2);
}
```

Aufgabe 2. Betrachten Sie die folgende Klasse, mit der man Brüche abspeichern kann:

```
#include <iostream>

class Bruch {
public:
   Bruch(int zaehler,
         int nenner)
   {
      _zaehler = zaehler;
      _nenner = nenner;
   }

   void print()
   {
      std::cout << _zaehler << " / " << _nenner << std::endl;
```

```
15    }
16
17    Bruch multiplizieren (const Bruch & bruch) const
18    {
19       return Bruch(_zaehler * bruch.get_zaehler(),
20                    _nenner * bruch.get_nenner());
21    }
22
23    int get_zaehler() const
24    {
25       return _zaehler;
26    }
27
28    int get_nenner() const
29    {
30       return _nenner;
31    }
32
33 private:
34    int _zaehler;
35    int _nenner;
36 };
```

Die Klasse kann auch von der Veranstaltungshomepage heruntergeladen werden.
Erweitern Sie die Klasse um Methoden, mit denen sich die Klasseneinträge nachträglich ändern lassen. Schreiben Sie auch eine Methode mit der Schnittstelle
`Bruch addieren (const Bruch & bruch) const;`
die es erlaubt zwei Brüche zu addieren. Implementieren Sie ebenso eine Methode, um Brüche zu kürzen (siehe dazu Algorithmus 3 vom 1. Zettel).
Bemerkung: Für diese Aufgabe können Sie annehmen, dass die vorkommenden Nenner stets von 0 verschieben sind. Wie man solche Fälle abfangen kann, werden wir noch sehen.

Aufgabe 3. Schreiben Sie eine Klasse `QuadPol`, mit der sich quadratische Polynome $p : \mathbb{R} \to \mathbb{R}$ speichern lassen. Dazu sollten Sie ein Polynom $p(x) = ax^2 + bx + c$ durch die drei Werte a, b und c abspeichern. Ihre Klasse soll insbesondere einen Konstruktor haben, dem die Werte a, b und c übergeben werden. Die Einträge selbst sollten `private` sein und nur durch Klassenmethoden manipulierbar.
Außerdem soll es eine Methode geben, welche die Addition von zwei quadratischen Polynomen erlaubt. Implementieren Sie auch eine Methode, die per `bool` ausgibt, ob die Funktion für steigendes x gegen unendlich läuft.
Schreiben Sie darüber hinaus eine Methode, der zwei Zahlen x_d und y_d übergeben werden und die den Graph des Polynoms um x_d in x-Richtung und y_d in y-Richtung verschiebt.
Überlegen Sie sich als Ergänzung auch selbst nützliche Erweiterungen und implementieren Sie diese (beispielsweise ein Konstruktor, bei dem das Polynom in der Form $p(x) = d(x - e)^2 + f$ gegeben ist, oder eine Methode zur Nullstellenbestimmung).

Aufgabe 4. Modifizieren Sie die Klasse `QuadPol` aus der vorigen Aufgabe so, dass eine Klasse `Polynom` entsteht, mit der sich Polynome von beliebigem Grad speichern lassen. Speichern Sie dazu in der Darstellung $p(x) = \sum_{i=0}^{n} a_i x^i$ eines Polynoms $p : \mathbb{R} \to \mathbb{R}$ vom Grad n die Koeffizienten a_i in einem Vektor ab. Überlegen Sie sich selbst einen geeigneten Konstruktor für die Klasse und passen Sie die Aufgaben für die Klasse `QuadPol` geeignet für die Klasse `Polynom` an.
Implementieren Sie neben der Addition von Polynomen auch eine Multiplikation.