Aufgabe 1. Im folgenden Programm finden sich 5 Fehler. Welche?

```
#include <iostream>
#include <vector>

/* Liefert das Maximum einer Menge von nichtnegativen Zahlen.
   Wenn keine Zahlen gegeben werden, ist die  Ausgabe 0. */
unsigned maximum(const std::vector<unsigned> & zahlen)
{
   unsigned result = 0;

   for(unsigned i = 0; i < zahlen.size; ++i)
   {
      if(zahlen[i] > result)
      {
         result = zahlen[i];
      }
   }
   return result;
}

int main()
{
   std::vector<int> eintraege;

   std::cout << "Geben Sie nicht-negative Zahlen ein. " << std::endl;
   std::cout << "Mit einer negativen Zahl koennen Sie die Eingabe beenden"  << std::endl;

   do
   {
      int eingabe;
      std::cout << "Naechster Eintrag bitte. ";
      std::cin  >> eingabe;
      if(eingabe >= 0)
      {
         eintraege.push(eingabe);
      }
   } while(eingabe >= 0)

   if(eintraege.size() > 0)
   {
      unsigned groesste_zahl = maximum(eintraege);
      std::cout << "Das Maximum ist: " << groesste_zahl << std::endl;
   }
   return 0;
}
```

Aufgabe 2. Schreiben Sie eine Funktion, der (per `const`-Referenz) zwei `int`-Vektoren `a` und `b` übergeben werden, die beide in nicht-absteigender Reihenfolge sortiert sind. Die Funktion soll einen `int`-Vektor zurückgeben, der alle Einträge aus `a` und `b` in nicht-absteigender Reihenfolge enthält. In allen Vektoren können Zahlen mehrfach vorkommen. Testen Sie Ihre Funktion, indem Sie die sortierten Vektoren `a` und `b` über die Konsole einlesen lassen.
Bemerkung: Rekursiv auf Teil-Instanzen aufgerufen, kann man mit diesem Ansatz auch Vektoren sortieren. Als Zusatzaufgabe können Sie sich die Details dazu überlegen und ein entsprechendes Sortierverfahren implementieren.

Aufgabe 3. Betrachten Sie die folgende Klasse, mit der man Brüche abspeichern kann:

```
#include <iostream>

class Bruch {
public:
   Bruch(int zaehler,
```

```
6           int nenner)
7    {
8       _zaehler = zaehler;
9       _nenner = nenner;
10   }
11
12   void print()
13   {
14      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 PreCampus-Seite (dort, wo die Übungszettel stehen) 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 (überlegen Sie sich dazu einen Algorithmus).
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.