technische universität dortmund

fakultät für informatik

Lutz Oettershagen
Jurij Kuzmic

Dortmund, den 20. Dezember 2018

# Übung zur Vorlesung EidP (WS 2018/19)

# Blatt 9

Block grün

**Es können 8 Punkte erreicht werden.**

**Abgabedatum:** 10. Januar 2019, 23:59 Uhr

## Hinweise

- Bitte beachten Sie die aktuellen Hinweise unter

  `https://ls11-www.cs.tu-dortmund.de/teaching/ep1819uebung/`

- Stellen Sie sicher, dass alle von Ihnen abgegebene Dateien reine Textdateien im UTF-8-Format sind.
- Für die Abgabe sind die Dateien `Aufgabe_09_1.txt`, `Aufgabe_09_2.h`, `Aufgabe_09_2.cpp`, `Aufgabe_09_3.h`, `Aufgabe_09_3.cpp`, `Fraction.h`, `Fraction.cpp` und `FractionTest.cpp` notwendig.
- Für die Kompilierung muss ebenfalls sichergestellt werden, dass sich Ihre Programme mit den Parametern `-pedantic` und `-Werror` kompilieren lassen.
- Für die Programmieraufgaben kopieren Sie immer die Ergebnisse als Block-Kommentar an das Ende der Datei, welche das jeweilige Hauptprogramm enthält.
- Dies ist das erste Blatt im dritten, grünen Block.

## Aufgaben

### Aufgabe 1: Grundlagen (1 Punkt)

Legen Sie für Ihre Antworten eine Text-Datei `Aufgabe_09_1.txt` an.

**a)** Beschreiben Sie anhand eines Beispiels, in welchen Fällen eine Implementierung des Kopierkonstruktors durchgeführt werden muss. (0.3 Punkte)

**b)** Was ist der Unterschied zwischen ASCII- und Binärdateien? (0.2 Punkte)

**c)** Wie kann man feststellen, ob eine geöffnete Datei unbekannter Länge zu Ende gelesen wurde? (0.1 Punkte)

**d)** Was ist beim Öffnen einer zu schreibenden Datei zu beachten? (0.1 Punkte)

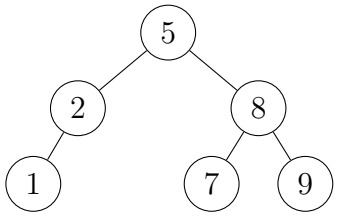
**e)** Was passiert, wenn eine schreibgeschützte Datei zum Schreiben geöffnet wird? (0.1 Punkte)

**f)** Was sind die möglichen Durchlaufstrategien bei der Tiefensuche in einem binären Suchbaum? (0.2 Punkte)

## Aufgabe 2: ADT Binärer Suchbaum (2 Punkte)

In der Vorlesung und in der vorherigen Übung haben Sie den ADT *binärer Suchbaum* für den effizienten Zugriff auf sortierbare Elemente kennengelernt. Die Implementierung des ADT *binärer Suchbaum* sowie die Testeingabe sind auf der Website der Übung zu finden. Laden Sie für Ihre Abgabe die Dateien `Aufgabe_09_2.h` und `Aufgabe_09_2.cpp` von der Website der Übung herunter. Erweitern Sie den ADT um die folgende Methode:
Die Methode `void preorder()` soll für den Suchbaum, für den sie aufgerufen wird, die Werte der Knoten des Baums in Preorder-Reihenfolge auf der Konsole ausgeben.

**Beispiel:**




Der Aufruf `preorder()` gibt auf der Konsole die Werte `5 2 1 8 7 9` aus.

Kompilieren Sie Ihr Programm, überprüfen Sie Ihre Ergebnisse und kopieren Sie diese als Block-Kommentar an das Ende der cpp-Datei `Aufgabe_09_2.cpp`.

## Aufgabe 3: Kopierkonstruktor und Zuweisungsoperator (2 Punkte)

In der Vorlesung und in der vorherigen Übung haben Sie den ADT *binärer Suchbaum* für den effizienten Zugriff auf sortierbare Elemente kennengelernt. In der Vorlesung wurden jedoch der Kopierkonstruktor und der Zuweisungsoperator nicht implementiert, was bei einer Klasse mit dynamischen Speicher ja eigentlich notwendig wäre. Erweitern Sie den ADT binärer Suchbaum um Kopierkonstruktor und Zuweisungsoperator so, wie es für andere ADTs in der Vorlesung vorgemacht wurde.

**Hinweis:** Die Implementierung des ADT *binärer Suchbaum* sowie die Testeingabe sind auf der Website der Übung zu finden. Laden Sie für Ihre Abgabe die Dateien `Aufgabe_09_3.h` und `Aufgabe_09_3.cpp` von der Website der Übung herunter. Speichern Sie Ihre Methoden in der h-Datei `Aufgabe_09_3.h`.

## Aufgabe 4: Rechnen mit Brüchen (3 Punkte)

Laden Sie für Ihre Abgabe die Dateien `Fraction.h`, `Fraction.cpp` und `FractionTest.cpp` von der Website der Übung herunter.

**a)** Erweitern Sie die vorgegebene Klasse `Fraction` zum Rechnen mit Brüchen. Die vorgegebene Klasse enthält zwei private Attribute `numerator` (deutsch Zähler) und `denominator` (deutsch Nenner) vom Typ `int`. Achten Sie bei der Erzeugung des Bruches darauf, dass das Vorzeichen im Attribut `numerator` geführt wird. Die Erzeugung der Brüche mit dem Nenner 0 erzeugt standardmäßig den Bruch $\frac{1}{1}$. Handeln Sie ebenfalls nach dem „`information hiding`“ Prinzip. Überladen Sie die folgenden Operatoren zum Rechnen mit Brüchen:

- `Fraction operator+ (Fraction const &f)` (Addition von Brüchen)
- `Fraction operator- (Fraction const &f)` (Subtraktion von Brüchen)
- `Fraction operator* (Fraction const &f)` (Multiplikation von Brüchen)
- `Fraction operator/ (Fraction const &f)` (Division von Brüchen)

(1 Punkt)

**b)** Schreiben Sie für die Klasse `Fraction` eine öffentliche Methode `void print()`, die für die Ausgabe eines Bruches zuständig ist. Überladen Sie ebenfalls den Ausgabeoperator `ostream& operator<< (ostream &os, Fraction const &f)` zur Ausgabe des Bruches mit `cout << bruch` auf der Konsole. Für den Zugriff auf die privaten Attribute der Klasse werden Getter-Funktionen benötigt. Der Ausgabeoperator gehört nicht zu der Klasse und greift somit über die öffentlichen **Getter-Funktionen** auf die privaten Attribute `numerator` und `denominator` zu.

(0.9 Punkte)

**c)** Die Brüche vom Typ `Fraction` sollen jederzeit normalisiert sein (gekürzt und Vorzeichen im Zähler). Erstellen Sie dazu eine private Methode `void normalize()`, die eine weitere private Methode `int gcd(int const n, int const d)` (für greatest common divisor, deutsch größter gemeinsamer Teiler) für das Kürzen der Brüche verwendet. (1 Punkt)

**d)** Führen Sie das Hauptprogramm `FractionTest.cpp` aus und kopieren Sie die Ergebnisse als Block-Kommentar an das Ende dieser Datei. Das Ergebnis Ihrer Implementierung muss hierbei wie folgt aussehen:

```
1  /**** Ausgabe der Bruche ***
2  Bruch 1/0   : 1/1
3  Bruch 0/0   : 1/1
4  Bruch 5/-10 : -1/2
5  Bruch -5/10 : -1/2
6  Bruch -5/-10: 1/2
7  Bruch 5/9   : 5/9
8  Bruch 4/6   : 2/3
9
10 *** Rechnen mit Bruchen ***
11 Addition:       5/9 + 4/6 = 11/9
12 Subtraktion:    5/9 - 4/6 = -1/9
13 Multiplikation: 5/9 * 4/6 = 10/27
14 Division:       5/9 / 4/6 = 5/6*/
```

(0.1 Punkte)

**Hinweis:** Unterteilen Sie die Deklaration und die Definition der jeweiligen Klassen in die Dateien `Fraction.h` und `Fraction.cpp`.