

Programmieren Übung: Klasse mit Operatoren: Bruchrechnen

Klaus Kusche

Wir wollen die Aufgabe „einfacher Taschenrechner“ aus dem ersten Jahr nochmals lösen, aber diesmal soll das Programm mit Brüchen statt mit **double**-Zahlen rechnen.

Wir brauchen also eine Klasse für Brüche, die zwei int's enthält (Zähler und Nenner), und zwar ohne direkte Zugriffsmöglichkeit von außen:

- Die Klasse hat keinen Standard-Konstruktor, sondern nur einen Konstruktor, der mit zwei **int's** (Zähler und Nenner) aufgerufen wird. Lässt man den Nenner beim Aufruf weg und gibt nur eine Zahl an, soll **1** als Nenner verwendet werden.
- Weiters hat die Klasse einen Konstruktor, der mit einem String aufgerufen wird (C-String, nicht C++-String, also **const char ***). Dieser String kann entweder einen Bruch oder nur eine ganze Zahl enthalten. Die beiden Fälle werden wie folgt unterschieden und behandelt:
 - Suche im String das Zeichen **'/'** (wie heißt die Stringfunktion, die ein Zeichen in einem String sucht, welchen Header braucht man dafür?) und merke dir die Fundstelle.
 - Wenn kein **'/'** gefunden wird, wird die Eingabe als ganze Zahl behandelt: Der Zähler des Ergebnisses ist der in einen **int** umgewandelte String (**atoi** verwenden), der Nenner ist **1**.
 - Wird ein **'/'** gefunden, so ist der Zähler wie bisher das **atoi** des Strings (das **atoi** hört mit dem Umwandeln automatisch vor dem **'/'** auf). Der Nenner ist das **atoi** des Strings ab einem Zeichen hinter der Fundstelle des **'/'** (wenn du den gemerkten Pointer auf die Fundstelle hast, wie rufst du **atoi** mit dem String ab dem Zeichen dahinter auf?).

Der Konstruktor ist etwas länger, schreib ihn außerhalb des **class**!

- Die Klasse soll die üblichen Operatoren zum Addieren, Subtrahieren, Multiplizieren und Dividieren zweier Brüche definieren (Potenzieren lassen wir weg).

Intern sehen die 4 Operatoren alle gleich aus: Sie bestehen nur aus dem **return** mit einem Aufruf des Konstruktors für das Ergebnis-Objekt mit den richtigen Werten für Zähler und Nenner.

Zur Auffrischung der Mathematik:

$$z1/n1 + z2/n2 = (z1*n2 + z2*n1)/(n1*n2)$$

$$z1/n1 - z2/n2 = (z1*n2 - z2*n1)/(n1*n2)$$

$$z1/n1 * z2/n2 = (z1*z2)/(n1*n2)$$

$$z1/n1 / z2/n2 = (z1*n2)/(n1*z2)$$

- Weiters brauchen wir den üblichen Ausgabe-Operator << für unsere Brüche (als globale Funktion, nicht als Methode unserer Klasse, aber mit Zugriffsrecht auf die beiden Member).

Wenn der Nenner des Bruches ungleich 1 ist, soll der ganze Bruch ausgegeben werden (zähler/nenner), sonst eine ganze Zahl (nur der Zähler).

Das **main** hat im Wesentlichen dieselbe Struktur wie in der Lösung der Übung des ersten Jahres, du kannst die Musterlösung als Ausgangsbasis nehmen.

- Aus den beiden **double**-Variablen werden zwei Bruch-Objekte, sie werden jeweils direkt an der Stelle deklariert, wo die Eingabewandlung erfolgt: Statt dem **atof** wird unser Konstruktor mit dem String-Argument aufgerufen. Der Bruch **input** ist also ein lokales Objekt innerhalb der **for**-Schleife.
- Alle **printf**'s werden auf C++-Ausgabe-Operatoren umgebaut (wohin schreibst du Fehlermeldungen?).
- Der Fall für das Potenzieren (und damit auch **math.h**) fällt weg, und die Prüfung auf Division durch 0 vorläufig auch.
- In den 4 Rechnungen müssen statt den kombinierten Rechen-Zuweisungs-Operatoren unsere Rechen-Operatoren mit separater Zuweisung verwendet werden (du brauchst keinen Zuweisungsoperator schreiben, der automatische genügt).

Tipp: Da ein auf der Befehlszeile eingegebener '*' nicht funktioniert, verwende in der Eingabe stattdessen ein 'x' für die Multiplikation.

Zusatzaufgabe 1:

Unser Programm erkennt bisher keine Brüche mit 0 im Nenner (und keine Divisionen durch 0) und liefert ungekürzte Brüche mit unnötig großem Zähler und Nenner (und eventuell mit negativem Nenner, was auch nicht schön ist).

Wir wollen daher unsere Bruch-Klasse um eine private Methode Kuerze erweitern. Sie soll weder Argumente noch einen Returnwert haben, sondern den Bruch, für den die Methode aufgerufen wird (d.h. das eigene Objekt), prüfen und kürzen.

Aufgerufen wird die Methode ganz am Ende der beiden Konstruktoren (in den Operatoren brauchen wir sie nicht aufrufen, weil diese ihr Ergebnis ohnehin mittels Konstruktor-Aufruf erzeugen und **Kuerze** dadurch indirekt aufgerufen wird).

Wir brauchen zuerst einmal eine Hilfsfunktion, die den ggT berechnet:

Zwei **int**'s gehen hinein, ein **int** soll zurückkommen.

Wie man den ggT zweier Zahlen ausrechnet, findest du auch in den Musterlösungen des Vorjahres (Verfahren von Euklid: **while**-Schleife mit %-Rechnung).

Vorsichtshalber solltest du von beiden Parametern den Absolutbetrag nehmen, bevor du mit der **while**-Schleife beginnst (mit negativen Zahlen funktioniert sie nämlich nicht).

Die Methode **Kuerze** (außerhalb des **class** implementieren!) besteht aus 3 Schritten:

- Ist der Nenner 0, so soll eine Fehlermeldung ausgegeben und das Programm beendet werden.
- Ist der Nenner negativ, soll bei Zähler und Nenner das Vorzeichen umgedreht werden (Minus nach oben).
- Danach wird der ggT von Zähler und Nenner berechnet (mit der Hilfsfunktion) und aus Zähler und Nenner herausdividiert.

Zusatzaufgabe 2, nur für Informatik-Studenten:

Schreib auch einen Typumwandlungs-Operator von einem Bruch-Objekt auf **double** (Achtung: **double**-Division, nicht **int**-Division!), und rufe ihn im **main** auf, um das Ergebnis zusätzlich auch als Kommazahl auszugeben.