Name: Maja Nikolic **Aufwand in h:** 12

Punkte: _____ **Kurzzeichen Tutor/in:** _____

Beispiel 1 (100 Punkte): Operatoren überladen

Schreiben Sie eine Klasse **`rational_t`**, die es ermöglicht, mit Bruchzahlen (über den ganzen Zahlen) zu rechnen.

Ein Beispiel: Das Programmfragment

```
1 rational_t r (-1, 2);
2
3 std::cout << r * -10                  << '\n'
4           << r * rational_t (20, -2) << '\n';
5
6 r = 7;
7
8 std::cout << r + rational_t (2, 3)          << '\n'
9           << 10 / r / 2 + rational_t (6, 5) << '\n';
```

führt zu dieser Ausgabe:

```
<5>
<5>
<23/3>
<67/35>
```

Beachten Sie diese Vorgaben und Implementierungshinweise:

1. Die Datenkomponenten für Zähler und Nenner sind vom Datentyp **`int`**.
2. Bei einer Division durch Null wird ein Fehler ausgegeben bzw. geworfen. Erstellen Sie hierfür eine eigene Exception-Klasse.
3. Werden vom Anwender der Klasse `rational_t` ungültige Zahlen übergeben, so wird ein Fehler ausgegeben bzw. geworfen.
4. Schreiben Sie ein privates Prädikat `is_consistent`, mit dessen Hilfe geprüft werden kann, ob `*this` konsistent (gültig, valide) ist. Verwenden Sie diese Methode an sinnvollen Stellen Ihrer Implementierung, um die Gültigkeit an verschiedenen Stellen im Code zu gewährleisten.
5. Schreiben Sie eine private Methode `normalize`, mit deren Hilfe eine rationale Zahl in ihren kanonischen Repräsentanten konvertiert werden kann. Verwenden Sie diese Methode in Ihren anderen Methoden.
6. Schreiben Sie eine Methode `print`, die eine rationale Zahl auf einem `std::ostream` ausgibt.
7. Schreiben Sie eine Methode `scan`, die eine rationale Zahl von einem `std::istream` einliest.

8. Schreiben Sie eine Methode `as_string`, die eine rationale Zahl als Zeichenkette vom Typ `std::string` liefert. (Verwenden Sie dazu die Funktion `std::to_string`.)
9. Verwenden Sie Referenzen und `const` so oft wie möglich und sinnvoll. Vergessen Sie nicht auf konstante Methoden.
10. Schreiben Sie die Methoden `get_numerator` und `get_denominator` mit der entsprechenden Semantik.
11. Schreiben Sie die Methoden `is_negative`, `is_positive` und `is_zero` mit der entsprechenden Semantik.
12. Schreiben Sie Konstruktoren ohne Argument (default constructor), mit einem Integer (Zähler) sowie mit zwei Integern (Zähler und Nenner) als Argument. Schreiben Sie auch einen Kopierkonstruktor (copy constructor, copy initialization).
13. Überladen Sie den Zuweisungsoperator (assignment operator, copy assignment), der bei Selbstzuweisung entsprechend reagiert.
14. Überladen Sie die Vergleichsoperatoren ==, !=, <, <=, > und >=. Implementieren Sie diese, indem Sie auch Delegation verwenden.
15. Überladen Sie die Operatoren +=, -=, *= und /= (compound assignment operators).
16. Überladen Sie die Operatoren +, -, * und /. Implementieren Sie diese, indem Sie Delegation verwenden. Denken Sie daran, dass der linke Operand auch vom Datentyp `int` sein können muss.
17. Überladen Sie die Operatoren << und >>, um rationale Zahlen „ganz normal“ auf Streams schreiben und von Streams einlesen zu können. Implementieren Sie diese, indem Sie Delegation verwenden.

Anmerkungen: (1) Geben Sie für Ihre Problemlösungen auch Lösungsideen an. (2) Kommentieren Sie Ihre Algorithmen ausführlich. (3) Strukturieren Sie Ihre Programme sauber. (4) Geben Sie ausreichend Testfälle ab und prüfen Sie alle Eingabedaten auf ihre Gültigkeit.

Inhaltsverzeichnis

Beispiel 1 Operatoren überladen

Lösungsidee

Um einen Bruch in die kanonische Repräsentation zu konvertieren, wird erst der größte gemeinsame Teiler von Zähler und Nenner gesucht. Anschließend werden beide durch diesen gemeinsamen Nenner geteilt. Falls der Bruch positiv ist, werden sowohl Nenner als auch Zähler ihre absoluten Beträge zugewiesen.

Um Brüche zu addieren/subtrahieren werden die Brüche erst durch Multiplizieren auf den gleichen Nenner gebracht, anschließend addiert/subtrahiert und gekürzt.

Für die Multiplikation werden die Nenner multipliziert und die Zähler multipliziert, anschließend gekürzt.

Um Brüche zu dividieren wird der Zähler mit dem Nenner, und der Nenner mit dem Zähler multipliziert. Anschließend gekürzt.

Die scan Methode liest einen String bis zum nächsten Leerzeichen ein und trennt diesen dann anhand eines Schrägstrichs. Der erste Teil wird zum Zähler und der andere demnach zum Teiler.

Um die rationale Zahl in einen String zu konvertieren wird nur der Zähler ausgegeben falls es sich um eine ganze Zahl handelt (Nenner=1 oder -1). Andernfalls wird erst der Zähler, dann der Nenner durch einen Schrägstrich getrennt ausgegeben.

Ein Bruch ist positiv wenn sowohl Nenner und Zähler dasselbe Vorzeichen haben und beträgt 0 wenn der Zähler 0 ist.

Um Brüche zu vergleichen werden diese durch Dividieren in Gleitkommazahlen umgewandelt, welche anschließend verglichen werden.

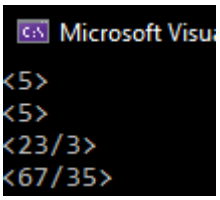
Quelltext: Rational

Tests

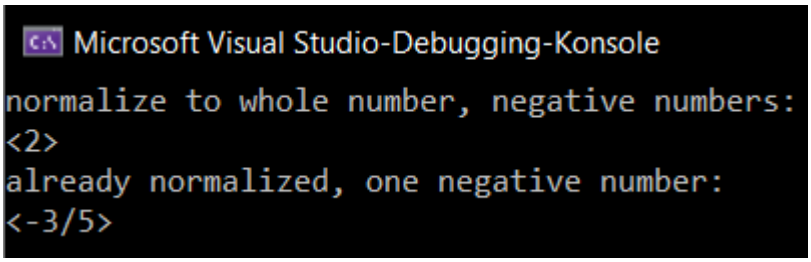
Da in jedem Test print bzw. as_string ausgeführt wird, wird dies nicht explizit getestet.

Offen: , calculations

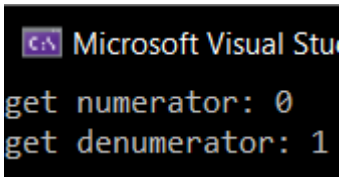
Beispiel von Übungszettel:



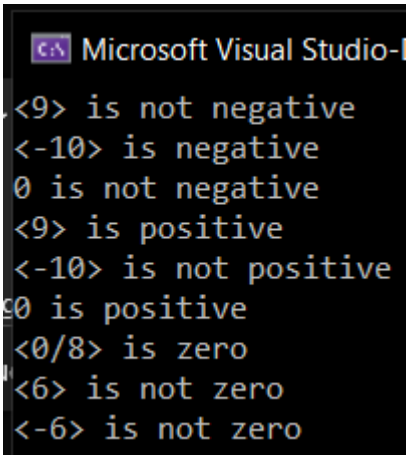
Normalize:



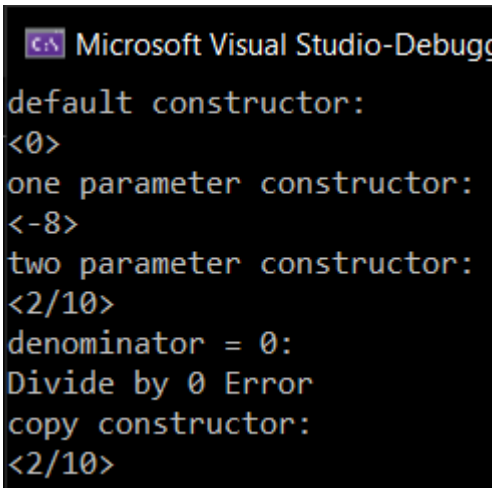
Get_nominator und get_denominator:



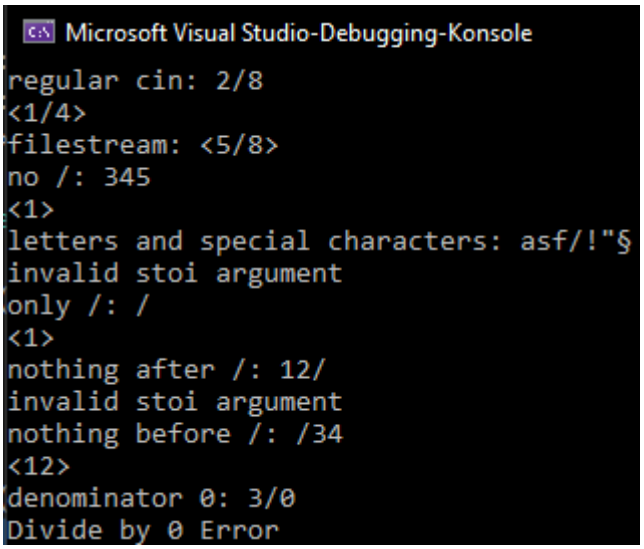
Is_negative, is_positive, is_zero:



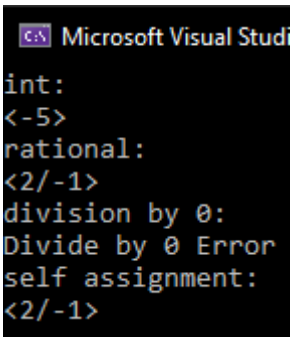
Konstruktoren:



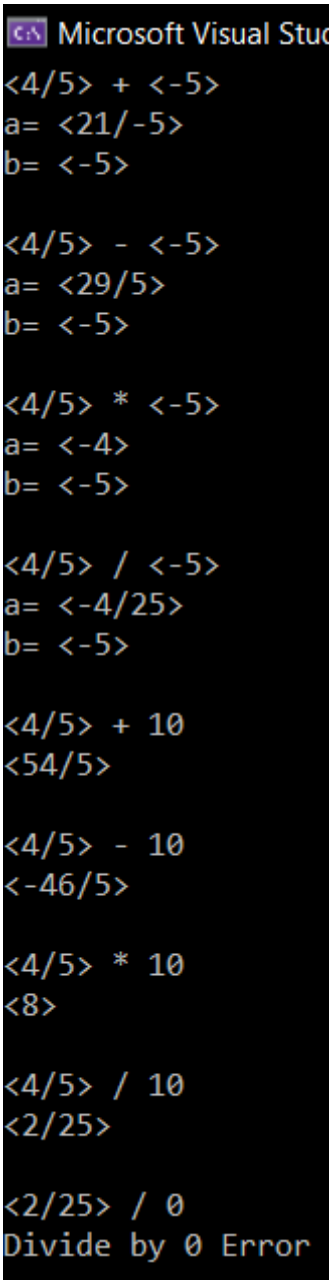
Scan:



assign:



Compound_assign:



calculations: