Inhaltsverzeichnis

Beispiel 1 – Operatoren überladen

- Lösungsidee
- Testfälle
- Aufwand ca. 10h

Lösungsidee:
normalize: es wird der größte gemeinsame Teiler ermittelt und anschließend sowohl Zähler als auch Nenner um diesen gekürzt.
is_consistent: Prüfung, ob Nenner 0 ist – wenn ja wird entsprechend mittels Exception-Klasse
print: gibt den Bruch auf dem gewählten Output-Stream aus – ist der Nenner 1, so wird der Bruch als ganze Zahl dargestellt → nur Zähler ausgeben
scan: liest den Bruch von gewähltem Input-Stream ein:
– gültiger Ausdruck = < evtl. negatives Vorzeichen Zahl1 / evtl. negatives Vorzeichen Zahl2 > (ohne Leerzeichen). Eingegebener Ausdruck muss dementsprechend überprüft und eingelesen werden.
as_string: baut aus den zwei Ziffern für den Bruch eine Zeichenkette mithilfe der Funktion std::to_string
get_numerator: liefert einen int mit dem Wert des Zählers
get_denominator: liefert einen int mit dem Wert des Nenners
is_negative: gibt an, ob Zähler oder Nenner ein negatives Vorzeichen haben
is_posititve: gibt an, ob Zähler und Nenner kein negatives Vorzeichen haben
is_zero: gibt an, ob Nenner 0 ist
Überladung Zuweisungsoperator: erfolgt durch Zuweisung der in der Schnittstellte übergebenen Klasse, zu den jeweiligen Teilen des Bruches
Überladung Vergleichsoperatoren: Funktionsdefinition durch entsprechende Vergleiche der Brüche, muss einen bool'schen Wert zurückliefern
Überladung der Rechenoperatoren: Definition mithilfe der ausimplementierten Funktionen der Grundrechenarten – Rückgabewert ist die berechnete Summe der Klasse
Überladung Compound Assignment Operators: Definition durch Verwenden der Funktionen der Grundrechenarten – Klasse wird einfach „dazugerechnet"

Testfälle:

Test 1 – Ausdruck auf der Konsole

```
void test_1_print_console() {
    std::cout << "Test 1: print on console:\n";
    rational_t number1(10, 5);
    number1.print();
    std::cout << "\n";
    rational_t number2(5,1);
    number2.print();
}
```

```
Microsoft Visual Studio-Debugging-Konsole
Test 1: print on console:
<10/5>
<5>
```

Test 2 – Ausdruck in File

```
void test_2_print_in_file() {
    std::cout << "Test 2: print in file:\n\tSee File in directory!";
    std::ofstream out{"output.txt"};
    rational_t number1(10, 5);
    number1.print(out);
    out << "\n";
    rational_t number2(5, 1);
    number2.print(out);
}
```

```
output - Editor
Datei Bearbeiten Format Ansicht Hilfe
<10/5>
<5>
```

Test 3 – gültigen Ausdruck von File input.txt scannen

```
void test_3_scan_from_file() {
    std::cout << "Test 3: scan from file input.txt\n";
    std::ifstream in("input.txt");
    rational_t number;
    number.scan(in);
    number.print();
}
```

```
Test 3: scan from file input.txt
<10/5>
C:\Users\laris\OneDrive\Documents\St
```

```
input - Editc
Datei Bearbeit
<10/5>
```

Test 4 – von leerem File scannen

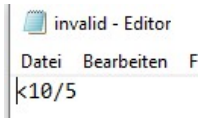
```
void test_4_scan_from_empty_file() {
    try {
        std::cout << "Test 4: scan from empty file empty.txt\n";
        std::ifstream in("empty.txt");
        rational_t number;
        number.scan(in);
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 4: scan from empty file empty.txt
Error! Invalid Data!
C:\Users\laris\OneDrive\Documents\Studiu
```

Test 5 – ungültige Zeichenkette in File

```
void test_5_invalid_input_from_file() {
    try {
        std::cout << "Test 5: scan from invalid file invalid.txt\n";
        std::ifstream in("invalid.txt");
        rational_t number;
        number.scan(in);
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

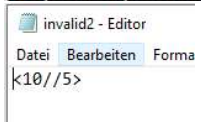
```
Test 5: scan from invalid file invalid.txt
Error! Invalid Data!
```



Test 6 – ungültige Zeichenkette in File

```
void test_6_invalid_input_from_file() {
    try {
        std::cout << "Test 6: scan from invalid file invalid2.txt\n";
        std::ifstream in("invalid2.txt");
        rational_t number;
        number.scan(in);
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

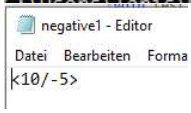
```
Test 6: scan from invalid file invalid2.txt
<10/5>
C:\Users\laris\OneDrive\Documents\Studium\WS
```



Test 7 – negative Zahlen von File

```
void test_7_negative_numbers_from_file() {
    try {
        std::cout << "Test 7: scan negative numbers from file negative1.txt\n";
        std::ifstream in("negative1.txt");
        rational_t number;
        number.scan(in);
        number.print();
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

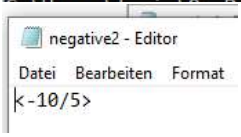
```
Test 7: scan negative numbers from file negative1.txt
<10/-5>
```



Test 8 – negative Zahlen von File

```
void test_8_negative_numbers_from_file() {
    try {
        std::cout << "Test 8: scan negative numbers from file negative2.txt\n";
        std::ifstream in("negative2.txt");
        rational_t number;
        number.scan(in);
        number.print();
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

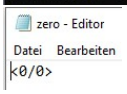
```
Test 8: scan negative numbers from file negative2.txt
<-10/5>
```



Test 9 – Division durch 0 von File

```
void test_9_division_by_zero_from_file() {
    try {
        std::cout << "Test 9: Division by Zero from File zero.txt\n";
        std::ifstream in("zero.txt");
        rational_t number;
        number.scan(in);
        number.print();
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 9: Division by Zero from File zero.txt
Error! Division by Zero
Dominator 0 ....
```



Test 10 – invalider Input von Konsole

```
void test_10_invalid_input_from_console() {
    try {
        std::cout << "Test 10: Invalid Input from Console\n";
        rational_t number;
        number.scan();
        number.print();
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 10: Invalid Input from Console
<10/*
Error! Invalid Data!<0>
C:\Users\laris\OneDrive\Documents\Stu
```

Test 11 – valider Input von Konsole

```
void test_11_valid_input_from_console() {
    try {
        std::cout << "Test 11: Valid Input from Console\n";
        rational_t number;
        number.scan();
        number.print();
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 11: Valid Input from Console
<11/13>
<11/13>
C:\Users\laris\OneDrive\Documents\Studiu
```

Test 12 – Division durch 0 von Konsole

```
void test_12_division_by_zero_from_console() {
    try {
        std::cout << "Test 12: Division by Zero from Console\n";
        rational_t number;
        number.scan();
        number.print();
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 12: Division by Zero from Console
<13/0>
Error! Division by Zero
Dominator 0 ....
C:\Users\laris\OneDrive\Documents\Studi
```

Test 13 – Normalisierung + Output auf Konsole

```
void test_13_normalize() {
    std::cout << "Test 13: Normalization\n";
    rational_t number(10,5);
    number.test_normalize();
    number.print();
}
```

```
Test 13: Normalization
<2>
C:\Users\laris\OneDrive\
```

Test 14 – Normalisierung + Output auf Konsole

```
void test_14_normalize() {
    std::cout << "Test 14: Normalization\n";
    rational_t number(9555,143);
    number.test_normalize();
    number.print();
}
```

```
est 14: Normalization
735/11>
:\Users\laris\OneDrive\Do
```

Test 15 – Normalisierung +Output auf Konsole (minus fällt weg)

```
void test_15_normalize() {
    std::cout << "Test 15: Normalization\n";
    rational_t number(-18,-3);
    number.test_normalize();
    number.print();
}
```

```
Test 15: Normalization
<6>
C:\Users\laris\OneDriv
```

Test 16 – Invalider Output auf Konsole

```
void test_16_invalid_output_on_console() {
    std::cout << "Test 16: Invalid Output on Console\n";
    rational_t number(4, 0);
    number.test_normalize();
    number.print();
}
```

```
Test 16: Invalid Output on Console
Error! Division by Zero
Dominator 0 ....
C:\Users\laris\OneDrive\Documents\S
```

Test 17 – to_string

```
void test_17_to_string() {
    std::cout << "Test 17: as_string\n";
    rational_t number(-4, 5);
    std::string number_as_string = number.as_string();
    std::cout << number_as_string;
}
```

```
Test 17: as_string
<-4/5>
```

Test 18 – get numerator / denominator

```
void test_18_get_numerator_denominator() {
    std::cout << "Test 18: get Numerator / Separator\n";
    rational_t number(4, 5);
    int _num = number.get_numerator();
    int _denom = number.get_denominator();
    std::cout << '<' << _num << '/' << _denom << '>';
}
```

```
Test 18: get Numerator / Separator
<4/5>
C:\Users\laris\OneDrive\Documents\S
```

Test 19 – Copy-Konstruktor

```
void test_19_copy_constructor() {
    std::cout << "Test 19: Copy-Constructor\n";
    rational_t other1(3,5);
    rational_t number1(other1);
    number1.print();
    std::cout << "\n";
    rational_t other2(3, 5);
    rational_t number2 = other2;
    number2.print();
}
```

```
Test 19: Copy-Constructor
<3/5>
<3/5>
C:\Users\laris\OneDrive\Docu
```

Test 20 – Überladung Vergleichsoperatoren

```
void test_20_overload_compare_operators() {
    try {
        std::cout << "Test 20: Overloading Compare Operators\n";
        rational_t other(-1, 2);
        rational_t number(1, 3);
        // <
        if (number < other) {
            std::cout << "Number < Other\n";
        }
        else {
            std::cout << "Other < Number\n";
        }
        // >
        if (number > other) {
            std::cout << "Number > Other\n";
        }
        else {
            std::cout << "Other > Number\n";
        }
        // <=
        if (number <= other) {
            std::cout << "Number <= Other\n";
        }
        else {
            std::cout << "Other <= Number\n";
        }
        // >=
        if (number >= other) {
            std::cout << "Number >= Other\n";
        }
        else {
            std::cout << "Other >= Number\n";
        }
        //==
        if (number == other) {
            std::cout << "Number == Other\n";
        }
        else {
            std::cout << "Number!= Other\n";
        }
        //!=
        if (number != other) {
            std::cout << "Number != Other\n";
        }
        else {
            std::cout << "Number== Other\n";
        }
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 20: Overloading Compare Operators
Other < Number
Number > Other
Other <= Number
Number >= Other
Number!= Other
Number != Other
```

Test 21 – Überladen Vergleichsoperatoren

```
void test_21_overload_compare_operators() {
    try{
    std::cout << "Test 21: Overloading Compare Operators\n";
    rational_t other(3, 17);
    rational_t number(2, 9);
    rational_t other2(2, 3);
    rational_t number2(2, 3);
    rational_t number3(3, 0);
    // <
    if (number < other) {
        std::cout << "Number < Other\n";
    }
    else {
        std::cout << "Other < Number\n";
    }
    // >
    if (number > other) {
        std::cout << "Number > Other\n";
    }
    else {
        std::cout << "Other > Number\n";
    }
    // <=
    if (number <= other) {
        std::cout << "Number <= Other\n";
    }
    else {
        std::cout << "Other <= Number\n";
    }
    // >=
    if (number >= other) {
        std::cout << "Number >= Other\n";
    }
    else {
        std::cout << "Other >= Number\n";
    }
    //==
    if (number2 == other2) {
        std::cout << "Number == Other\n";
    }
    else {
        std::cout << "Number!= Other\n";
    }
    //!=
    if (number3 != other2) {
        std::cout << "Number != Other\n";
    }
    else {
        std::cout << "Number== Other\n";
    }
}
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 21: Overloading Compare Operators
Other < Number
Number > Other
Other <= Number
Number >= Other
Number == Other
Error! Division by Zero
Dominator 0 ....
C:\Users\laris\OneDrive\Documents\Studiu
```

Test 22 – Grundrechenarten

```
void test_22_basic_arithmetics() {
    try {
        std::cout << "Test 22: Basic Arithmetics\n";
        rational_t number1(1, 3);
        rational_t number2(1, 3);
        rational_t number3(1, 3);
        rational_t number4(1, 3);
        rational_t other(1, 5);
        number1.mul(other);
        number1.print();
        std::cout << "\n";
        number2.div(other);
        number2.print();
        std::cout << "\n";
        number3.add(other);
        number3.print();
        std::cout << "\n";
        number4.sub(other);
        number4.print();
        std::cout << "\n";
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 22: Basic Arithmetics
<1/15>
<5/3>
<8/15>
<2/15>

C:\Users\laris\OneDrive\Do
mit Code "0" beendet.
```

Test 23 – Grundrechenarten

```
void test_23_basic_arithmetics() {
    try {
        std::cout << "Test 23: Basic Arithmetics\n";
        rational_t number1(-2, 7);
        rational_t number2(-2, 7);
        rational_t number3(-2, 7);
        rational_t number4(-2, 7);
        rational_t other(2, 9);
        number1.mul(other);
        number1.print();
        std::cout << "\n";
        number2.div(other);
        number2.print();
        std::cout << "\n";
        number3.add(other);
        number3.print();
        std::cout << "\n";
        number4.sub(other);
        number4.print();
        std::cout << "\n";
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 23: Basic Arithmetics
<-4/63>
<-9/7>
<-4/63>
<-32/63>

C:\Users\laris\OneDrive\Docu
```

Test 24 – Grundrechenarten, Bruch durch 0

```
void test_24_basic_arithmetics() {
    try {
        std::cout << "Test 24: Basic Arithmetics\n";
        rational_t number1(-2, 7);
        rational_t number2(-2, 7);
        rational_t number3(-2, 7);
        rational_t number4(-2, 7);
        rational_t other(2, 0);
        number1.mul(other);
        number1.print();
        std::cout << "\n";
        number2.div(other);
        number2.print();
        std::cout << "\n";
        number3.add(other);
        number3.print();
        std::cout << "\n";
        number4.sub(other);
        number4.print();
        std::cout << "\n";
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 24: Basic Arithmetics
Error! Division by Zero
Dominator 0 ....
C:\Users\laris\OneDrive\Docu
```

Test 25 – Überladung Grundrechenarten

```
void test_25_basic_arithmetics_overload() {
    try {
        rational_t number(-2, 7);
        rational_t other(2, 0);
        rational_t result1 = number + other;
        result1.print(); std::cout << "\n";
        rational_t result2 = number - other;
        result2.print(); std::cout << "\n";
        rational_t result3 = number / other;
        result3.print(); std::cout << "\n";
        rational_t result4 = number * other;
        result4.print(); std::cout << "\n";
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Error! Division by Zero
Dominator 0 ....
C:\Users\laris\OneDrive\
```

Test 26 – Überladung Grundrechenarten

```
void test_26_basic_arithmetics_overload() {
    try {
        rational_t number(-2, 7);
        rational_t other(2, 9);
        rational_t result1 = number + other;
        result1.print(); std::cout << "\n";
        rational_t result2 = number - other;
        result2.print(); std::cout << "\n";
        rational_t result3 = number / other;
        result3.print(); std::cout << "\n";
        rational_t result4 = number * other;
        result4.print(); std::cout << "\n";
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 26: Basic Arithmetics Overload
<-4/63>
<-32/63>
<-9/7>
<-4/63>

C:\Users\laris\OneDrive\Documents\Stud
```

Test 27 – Grundrechenarten (compound assignment operators)

```
void test_27_basic_arithmetic_compound_assignment() {
    std::cout << "Test 27: Basic Arithmetics Compound Assignment\n";
    try {
        rational_t number1(-2, 7);
        rational_t number2(-2, 7);
        rational_t number3(-2, 7);
        rational_t number4(-2, 7);
        rational_t other(2, 9);
        number1 += other;
        number1.print(); std::cout << "\n";
        number2 -= other;
        number2.print(); std::cout << "\n";
        number3 /= other;
        number3.print(); std::cout << "\n";
        number4 *= other;
        number4.print(); std::cout << "\n";
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 27: Basic Arithmetics Compound Assignment
<-4/63>
<-32/63>
<-9/7>
<-4/63>

C:\Users\laris\OneDrive\Documents\Studium\WS 2022\
```

Test 28 – Grundrechenarten (compound assignment operators)

```
void test_28_basic_arithmetic_compound_assignment() {
    std::cout << "Test 28: Basic Arithmetics Compound Assignment\n";
    try {
        rational_t number1(-2, 7);
        rational_t number2(-2, 7);
        rational_t number3(-2, 7);
        rational_t number4(-2, 7);
        rational_t other(2, 0);
        number1 += other;
        number1.print(); std::cout << "\n";
        number2 -= other;
        number2.print(); std::cout << "\n";
        number3 /= other;
        number3.print(); std::cout << "\n";
        number4 *= other;
        number4.print(); std::cout << "\n";
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 28: Basic Arithmetics Compound Assignment
Error! Division by Zero
Dominator 0 ....
C:\Users\laris\OneDrive\Documents\Studium\WS 2
```

Test 29 - Überladung Operator <<

```
void test_29_overloading_output() {
    std::cout << "Test 29: Overloading Output Operator\n";
    try {
        rational_t number1(-2, 7);
        std::cout << number1 << "\n\n";
        rational_t number2(3, 5);
        std::cout << number2 << "\n\n";
        rational_t number3(3, 0);
        std::cout << number3 << "\n\n";
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 29: Overloading Output Operator
<-2/7>

<3/5>

Error! Division by Zero
Dominator 0 ....
C:\Users\laris\OneDrive\Documents\Stu
```

Test 30 – Überladung Operator >>

```
void test_30_overloading_input() {
    std::cout << "Test 30: Overloading Input Operator\n";
    try {
        rational_t number1(2,1);
        std::cin >> number1;
        rational_t number2(3, 5);
        std::cin >> number2;
        rational_t number3(3, 0);
        std::cin >> number3;
        std::cout << number1 << "  " << number2 << "  " << number3 << "\n";
    }
    catch (DivideByZeroError& e) {
        std::cout << e.what() << "\n";
        std::cout << "Dominator 0 ....";
    }
}
```

```
Test 30: Overloading Input Operator
<1/3>
<5/6>
<-3/8>
<1/3>  <5/6>  <-3/8>

C:\Users\laris\OneDrive\Documents\Stu
mit Code "0" beendet
```