Uebung 03

Aufwand: 16 Stunden

Selina A

11-13-2022

Übung 03

BEISPIEL OPERATOREN ÜBERLADEN

LÖSUNGSIDEE:

Das Programm kann mit Bruchzahlen rechnen. Es wird eine Klasse erstellt mit Nenner und Zähler die ganzen Zahlen sein müssen.

Eine Division durch Null ist nicht möglich und bei einem Versuch wird eine Fehlermeldung ausgegeben. Außerdem werden die Eingaben des Anwenders überprüft, ob es valide Zahlen sind. Die Methode is_consistent soll zeigen, ob Nenner und Zähler gültig sind, indem überprüft wird, ob der Nenner nicht Null ist. Um die Ausgabe zu optimieren wird der größte gemeinsame Teiler von Nenner und Zähler gesucht um somit den Bruch zu vereinfachen (noramlize()). Um eine Ausgabe mithilfe der print Funktion auszuführen, werden die Ausgabeoperatoren überladet, um die Ausgabe zu vereinfachen. Außerdem wird der Bruch mithilfe der optimize Funktion optimiert. Es werden Vorzeichen überprüft und bei zwei Negativen Zahlen wird der Bruch wieder positiv. Bei der Ausgabe wird ebenfalls überprüft, ob der Nenner 1 oder -1 ist. Wenn ja wird nur der Zähler ausgegeben.

Es ist ebenfalls möglich eine Rationale Zahl (Bruch) aus einem File einzulesen. Es wird überprüft, ob es sich um ein gültiges File handelt, bevor die Zahl eingelesen wird. Es wird überprüft, ob die Eingabe korrekt ist, indem ein Bruchstrich verwendet wurde und links und rechts davon Zahlen stehen.

Eine weitere Methode der Klasse ist as string, hier werden die beiden Zahlen in einen String umgewandelt.

Um zugriff auf die Privaten integer Zähler und Nenner zu erhalten werden zwei Methoden erstellt, um auch von außerhalb auf diese Daten zugreffen zu können.

Mit den Hilfsfunktionen is_negative, is positiv und is_zero wird kontrolliert, ob eine Zahl negativ, positiv oder Null ist und die Antwort zurückgegeben.

Es werden auch die Vergleichsoperatoren überladen, um somit zwei Brüche vergleichen zu können, ob diese kleiner, kleiner gleich, größer gleich, größer, nicht gleich oder gleich sind.

Das Überladen der Operatoren +, +=, -, -=, *=, /= erleichtert das Rechnen.

Es wird auch der Eingabe Operator überladen.

QUELLTEXT

RATIONAL_T.H

```
#ifndef rational_t_h
#define rational_t_h
#include <iostream>
#include <string>

class rational_t
{
	int zaehler;
	int nenner;

public:
	rational_t(int _zaehler = 0, int nenner = 1);

	rational_t(rational_t const& other);
	rational_t();

	void add(rational_t const& other);
	void sub(rational_t const& other);
	void mul(rational_t const& other);
	void div(rational_t const& other);

	void scan(std::ifstream& in, int& n, int& z)const;
	void print(std::ostream& os)const;
	std::string as_string()const;
	int get_numerator()const;
	int get_denominator()const;
	bool is_negative(int& const x);
	bool is_positive(int& const x);
	bool is_zero(int& const x);
	void optimize();

	bool operator==(rational_t const& other);
	bool operator!=(rational_t const& other);
	bool operator<(rational_t const& other);
	bool operator<=(rational_t const& other);
	bool operator>(rational_t const& other);
	bool operator>=(rational_t const& other);

	rational_t operator+(rational_t const& other);
	rational_t operator-(rational_t const& other);
	rational_t operator*(rational_t const& other);
	rational_t operator/(rational_t const& other);

	rational_t& operator+=(rational_t const& other);
	rational_t& operator-=(rational_t const& other);
	rational_t& operator*=(rational_t const& other);
	rational_t& operator/=(rational_t const& other);


	rational_t &operator=(rational_t const& other);
private:
	bool is_consistent()const;
	void normalize(int& z, int& n);

};
std::ostream& operator<<(std::ostream& lhs, rational_t const& rhs);
std::ifstream& operator>>(std::ifstream& lhs, rational_t const& rhs);
void print_result(rational_t const& rational);
```

```
//non-Member-Operator
rational_t operator+(int const& lhs, rational_t const& rhs);
rational_t operator-(int const& lhs, rational_t const& rhs);
rational_t operator*(int const& lhs, rational_t const& rhs);
rational_t operator/(int const& lhs, rational_t const& rhs);
#endif // !rational_t_h
```

RATIONAL_T.CPP

```
#include "rational_t.h"
#include <stdexcept>
#include"div_by_zero_error.h"
#include <fstream>

#include <vector>

rational_t::rational_t(int _zaehler, int _nenner) :zaehler{ _zaehler }, nenner{
_nenner } {
	if (nenner == 0) {
		std::cout << "not valied";

	}
}
rational_t::rational_t(rational_t const& other) {
	nenner = other.nenner;
	zaehler = other.zaehler;
}

bool rational_t::is_consistent()const {
	if (nenner != 0) {
		return true;
	}
	else {
		return false;
	}

}
void rational_t::normalize(int& z, int& n) {
	int low(0);
	int _n=n;
	int _z=n;
	//if the divisor or numerator are negative,
	//it doesn't work to find the greatest common
	//divisor because you're searching in the forward
	//loop to the smaller number away from 0
	if (is_negative(n))_n*= (-1);
	if (is_negative(z))_z*= (-1);
	if (z < n)low =  _z;
	else low = _n;
	//search divider
	int teiler = 1;
	for (int i(1); i <= low; ++i) {
		if (n % i == 0 && z % i == 0) {
			teiler = i;
		}
	}
	z = z / teiler;
	n = n / teiler;
}
void rational_t::print(std::ostream& os)const {
	//its 0
	if (zaehler == 0 ) {
		os << 0 << "";
	}
```

```
	//integer
	else if (nenner == 1) {
		os << zaehler << "";
	}
	else {
		os << zaehler << "/" << nenner << "";
	}
}
void rational_t::scan(std::ifstream& in, int& n,int& z)const {

	std::vector<std::string> input;
	std::string line;
	std::string d;
	int x = 0;
	//file is ok
	if (in.good()) {

		while (std::getline(in, line, '/'))
		{
			input.push_back(line);
		}
		std::getline(in, line);


		if (input.size() != 2) {
			std::cout << "\nincorrect input";
		}
		else {
			bool check(true);
			for (int j(0); j < input.size(); ++j) {
				for (int i(0); i < input[j].size() - 1; ++i) {
					if (!isdigit(input[j][i])) {
						check = false;
					}

				}
			}

			if (check) {

				z = std::stoi(input[0]);
				n = std::stoi(input[1]);


			}
			else {
				std::cout << "\nno digit";
			}
			std::cout << z << "/" << n;
		}
	}//file is not good
	else {
		std::cout << "\nfile is damaged";
	}

}

std::string rational_t::as_string()const {
	return std::to_string(zaehler) + "/" + std::to_string(nenner) ;
}
int rational_t::get_numerator()const {
	return zaehler;
}
```

```
int rational_t::get_denominator()const {
	return nenner;
}

bool rational_t:: is_negative(int& const x) {
	if (x < 0) return true;
	else return false;
}
bool rational_t:: is_positive(int& const x) {
	if (x > 0) return true;
	else return false;
}
bool rational_t::is_zero(int& const x) {
	if (x == 0)return true;
	else return false;
}

rational_t& rational_t::operator=(rational_t const& other) {
	if (this == &other) {
		return *this;
	}

	nenner = other.nenner;
	zaehler = other.zaehler;
	return *this;
}

bool rational_t::operator==(rational_t const& other) {
	//rational_t tmp(*this);
	if (nenner == other.nenner && zaehler == other.zaehler)return true;
	else return false;
}
bool rational_t:: operator!=(rational_t const& other) {
	//rational_t tmp(*this);
	if (nenner != other.nenner || zaehler != other.zaehler)return true;
	else return false;
}
bool  rational_t::operator<(rational_t const& other) {

	double erg1 = (double)zaehler / (double)nenner;
	double erg2=(double)other.zaehler / (double)other.nenner;
	if (erg1 < erg2)return true;
	else return false;

}
bool  rational_t::operator<=(rational_t const& other) {

	double erg1 = (double)zaehler / (double)nenner;
	double erg2 = (double)other.zaehler / (double)other.nenner;
	if (erg1 <= erg2)return true;
	else return false;
}
bool  rational_t::operator>(rational_t const& other) {

	double erg1 = (double)zaehler / (double)nenner;
	double erg2 = (double)other.zaehler / (double)other.nenner;
	if (erg1 > erg2)return true;
	else return false;
}
bool  rational_t::operator>=(rational_t const& other) {

	double erg1 = (double)zaehler / (double)nenner;
	double erg2 = (double)other.zaehler / (double)other.nenner;
```

```
	if (erg1 >= erg2)return true;
	else return false;
}

void rational_t::add(rational_t const& other) {
	zaehler = zaehler * other.nenner + nenner * other.zaehler;
	nenner = nenner * other.nenner;
	optimize();

}
void rational_t::sub(rational_t const& other) {
	zaehler = (zaehler * other.nenner) - (nenner * other.zaehler);
	nenner = nenner * other.nenner;
	optimize();
	//wenn der  zaehler0 ist wird auch der nenner 0
	if (is_zero(zaehler)) {
		nenner = 0;
	}
}
void rational_t::mul(rational_t const& other) {
	zaehler = zaehler * other.zaehler;
	nenner = nenner * other.nenner;
	optimize();
}
void rational_t::div(rational_t const& other) {
	zaehler = zaehler * other.nenner;
	nenner = nenner * other.zaehler;
	optimize();
}

rational_t rational_t::operator+(rational_t const& other) {
	rational_t tmp(*this);
	tmp.add(other);
	if(tmp.is_consistent())return tmp;
}
rational_t rational_t:: operator-(rational_t const& other) {
	rational_t tmp(*this);
	tmp.sub(other);
	return tmp;


}
rational_t rational_t::operator*(rational_t const& other) {
	rational_t tmp(*this);
	tmp.mul(other);
	if (tmp.is_consistent())
		return tmp;
}
rational_t rational_t::operator/(rational_t const& other) {
	rational_t tmp(*this);
	tmp.div(other);
	if (tmp.is_consistent())
		return tmp;
}

rational_t operator+(int const& lhs, rational_t const& rhs) {
	rational_t tmp(lhs);
	tmp.add(rhs);
	return tmp;
}
rational_t operator-(int const& lhs, rational_t const& rhs) {
	rational_t tmp(lhs);
	tmp.sub(rhs);
```

```
	return tmp;
}
rational_t operator*(int const& lhs, rational_t const& rhs) {
	rational_t tmp(lhs);
	tmp.mul(rhs);
	return tmp;
}
rational_t operator/(int const& lhs, rational_t const& rhs) {
	rational_t tmp(lhs);
	tmp.div(rhs);
	return tmp;
}

rational_t& rational_t::operator+=(rational_t const& other) {
	add(other);
	return *this;
}
rational_t& rational_t::operator-=(rational_t const& other) {
	sub(other);
	return *this;
}
rational_t& rational_t::operator*=(rational_t const& other) {
	mul(other);
	return *this;
}
rational_t& rational_t::operator/=(rational_t const& other) {
	div(other);
	return *this;
}

std::ostream& operator<<(std::ostream& lhs, rational_t const& rhs) {
	rhs.print(lhs);

	return lhs;
}
std::ifstream& operator>>(std::ifstream& lhs, rational_t const& rhs) {
	int z = rhs.get_denominator();
	int n = rhs.get_numerator();;
	rhs.scan(lhs, n, z);

	return lhs;
}
void print_result(rational_t const& rational)
{
	std::cout << rational << "\n";
}
void rational_t:: optimize() {
	if (is_negative(zaehler)&& is_negative(nenner)) {
		nenner = nenner * -1;
		zaehler = zaehler * -1;
	}
	normalize(zaehler, nenner);
	if (nenner == -1) {
		zaehler = zaehler * (-1);
		nenner = nenner * -1;
	}

}
```

DIV_BY_ZERO_ERROR_H

```
#ifndef div_by_zero_error_h
```

```
#define div_by_zero_error_h

#include<exception>

class DivideByZeroError :public std::exception {
public:
	virtual const char* what() const throw() {
		return "Divide by Zero Error";
	}
};
#endif // !div_by_zero_error_h
```

TEST.H

```
#ifndef test_h
#define div_by zero_h
#include "rational_t.h"
#include"div_by_zero_error.h"

//test _invoice types
void test_addition();
void test_sub();
void test_sub_is_null();
void test_multiplication();
void test_division();
//op
void test_addition_op();
void test_subtraction_op();
void test_multiplication_op();
void test_division_op();

//op assign
void test_addition_op_ass();
void test_sub_op_ass();
void test_multiplication_op_ass();
void test_division_op_ass();

//test with int
void test_addition_op_int();
void test_sub_op_int();
void test_mul_op_int();
void test_div_op_int();



void test_divide_by_zero();
//test compare op
void test_same();
void test_different();
void test_smaller();
void test_smaller_same();
void test_bigger();
void test_bigger_same();

//test read
void test_read_file_not_valid_file();
void test_read_file_valid();
//letter in file
void test_read_file_false1();
// not / in file
void test_read_file_false2();
// test ausgabe
//denominator and numerator are negative
```

```
void test_output_negativ();
// fracture must be shortened
void test_output_normalize();
//can be output as an integervoid test_integer_1();
void test_integer_2();
//denominator is 0
void test_output_null1();

//numerator is 0
void test_output_null2();
#endif
```

TEST.CPP

```
#include "test.h"
#include<fstream>

void test_addition() {
	rational_t a(3, 7);
	rational_t b(-5, -6);
	a.add(b);
	print_result(a);
}
void test_sub() {
	rational_t a(3, 7);
	rational_t b(5, -6);
	a.sub(b);
	print_result(a);
}
void test_sub_is_null() {
	rational_t a(3, 7);
	rational_t b(3, 7);
	a.sub(b);
	print_result(a);
}
void test_multiplication() {
	rational_t a(5, -2);
	rational_t b(3, 4);
	a.mul(b);
	print_result(a);
}
void test_division() {
	rational_t a(2, 1);
	rational_t b(1, -2);
	a.div(b);
	print_result(a);
}

void test_addition_op() {
	rational_t a(3, 7);
	rational_t b(-5, -6);
	rational_t c = a + b;
	print_result(c);
}
void test_subtraction_op() {
	rational_t a(3, 7);
	rational_t b(5, -6);
	rational_t c = a - b;
	print_result(c);
}
void test_multiplication_op() {
	rational_t a(5, -2);
	rational_t b(3, 4);
```

```
	rational_t c = a * b;
	print_result(c);

}
void test_division_op() {
	rational_t a(2, 1);
	rational_t b(1, -2);
	rational_t c = a / b;
	print_result(c);

}

void test_addition_op_ass() {
	rational_t a(3, 7);
	rational_t b(-5, -6);
	a += b;
	print_result(a);
}
void test_sub_op_ass() {
	rational_t a(3, 7);
	rational_t b(5, -6);
	a -= b;
	print_result(a);
}
void test_multiplication_op_ass() {
	rational_t a(5, -2);
	rational_t b(3, 4);
	a *= b;
	print_result(a);
}
void test_division_op_ass() {
	rational_t a(2, 1);
	rational_t b(1, -2);
	a /= b;
	print_result(a);
}

void test_addition_op_int() {
	rational_t a(2, 1);
	int b(21);  //convert
	rational_t c = a + b;
	//c = b + a;
	print_result(c);
}
void test_sub_op_int() {
	rational_t a(2, 1);
	int b(1);  //convert
	rational_t c = a - b;

	print_result(c);
}
void test_mul_op_int() {
	rational_t a(2, 1);
	int b(2);  //convert
	rational_t c = a *b;

	print_result(c);
}
void test_div_op_int() {
	rational_t a(2, 1);
	int b(2);  //convert
	rational_t c = a / b;
```

```
	print_result(c);
}
void test_divide_by_zero() {
	rational_t a(2, 1);
	rational_t b(0, 0);

	try {
		rational_t c = a / b;
		print_result(c);
	}
	catch (DivideByZeroError& e) {
		std::cout << "Div by zero:" << e.what();
	}
	std::cout << "PRoceeding with execution...\n\n";

}
void test_same() {
	// same
	rational_t a(1, 3);
	rational_t b(1, 3);
	if (a == b) {
		print_result(a);
		std::cout << " Is the same as";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	else {
		print_result(a);
		std::cout << " Is not same as";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	//not the same
	rational_t c(1, 3);
	rational_t d(-1, 3);
	if (c == d) {
		print_result(c);
		std::cout << " Is the same as\n";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";

	}

	else {
		print_result(c);
		std::cout << " Is not same as";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";
	}
}
void test_different() {
	// different
	rational_t a(1, 3);
	rational_t b(1, 3);
	if (a != b) {
		print_result(a);
		std::cout << " Is different as";
		std::cout << "\n";
		print_result(b);
```

```
		std::cout << "\n";
	}
	else {
		print_result(a);
		std::cout << " Is not different as";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	//not the same
	rational_t c(1, 3);
	rational_t d(-1, 3);
	if (c != d) {
		print_result(c);
		std::cout << " Is  different as\n";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";

	}

	else {
		print_result(c);
		std::cout << " Is not different as";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";
	}

}
void test_smaller() {
	rational_t a(1, 3);
	rational_t b(1, 2);
	if (a < b) {
		print_result(a);
		std::cout << " Is smaller than";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	else {
		print_result(a);
		std::cout << " Is not smaller than";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	//not smaller
	rational_t c(1, 3);
	rational_t d(-1, 3);
	if (c < d) {
		print_result(c);
		std::cout << " Is smaller than";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";
	}
	else {
		print_result(c);
		std::cout << " Is not smaller than";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";
```

```
	}

}
void test_smaller_same() {
	rational_t a(1, 3);
	rational_t b(1, 3);
	if (a <= b) {
		print_result(a);
		std::cout << " Is <=";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	else {
		print_result(a);
		std::cout << " Is <=";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	//smaller/same
	rational_t c(1, 3);
	rational_t d(-1, 3);
	if (c <= d) {
		print_result(c);
		std::cout << " Is <=";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";
	}
	else {
		print_result(c);
		std::cout << " Is not  <= ";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";
	}
}
void test_bigger() {
	rational_t a(1, 3);
	rational_t b(1, 2);
	if (a > b) {
		print_result(a);
		std::cout << " Is bigger than";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	else {
		print_result(a);
		std::cout << " Is not bigger than";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	//bigger
	rational_t c(1, 3);
	rational_t d(-1, 3);
	if (c > d) {
		print_result(c);
		std::cout << " Is bigger than";
		std::cout << "\n";
		print_result(d);
```

```
		std::cout << "\n";
	}
	else {
		print_result(c);
		std::cout << " Is not bigger than";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";
	}
}
void test_bigger_same() {
	rational_t a(1, 3);
	rational_t b(1, 3);
	if (a >= b) {
		print_result(a);
		std::cout << " Is >= ";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	else {
		print_result(a);
		std::cout << " Is not >= ";
		std::cout << "\n";
		print_result(b);
		std::cout << "\n";
	}
	//not  bigger/same
	rational_t c(1, 3);
	rational_t d(1, 2);
	if (c >= d) {
		print_result(c);
		std::cout << "  Is  >= ";;
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";
	}
	else {
		print_result(c);
		std::cout << "  Is not >= ";
		std::cout << "\n";
		print_result(d);
		std::cout << "\n";
	}
}

void test_read_file_not_valid_file() {
	rational_t a(1,1);

	std::ifstream input("new.tt");
	input >> a;
	print_result(a);
}
void test_read_file_valid() {
	rational_t a(1, 1);

	std::ifstream input("new.txt");
	input >> a;
	print_result(a);
}
void test_read_file_false1() {
	rational_t a(1, 1);
```

```
	std::ifstream input("false1.txt");
	input >> a;
	print_result(a);
}
void test_read_file_false2() {
	rational_t a(1, 1);

	std::ifstream input("false2.txt");
	input >> a;
	print_result(a);
}
void test_output_negativ() {
	rational_t a(-1, -1);
	print_result(a);
	std::cout << "optimized:";
	a.optimize();
	print_result(a);
}
void test_output_normalize() {
	rational_t a(6, 3);
	print_result(a);
	std::cout << "optimized:";
	a.optimize();
	print_result(a);
}
void test_output_null1() {
	rational_t a(1, 0);
	std::cout << "\n";
	print_result(a);
	std::cout << "optimized:";
	a.optimize();
	print_result(a);
}
void test_output_null2() {
	rational_t a(0, 1);
	print_result(a);
	std::cout << "optimized:";
	a.optimize();
	print_result(a);

}
void test_integer_1() {
	rational_t a(4, 2);
	print_result(a);
	std::cout << "optimized:";
	a.optimize();
	print_result(a);
}
void test_integer_2() {
	rational_t a(5, -5);
	print_result(a);
	std::cout << "optimized:";
	a.optimize();
	print_result(a);

}
```

MAIN.PP

```
#include "rational_t.h"
#include "test.h"
#include <stdexcept>
#include"div_by_zero_error.h"
#include <fstream>
```

```
#include<iostream>

int main() {

	std::cout << "Testing number operations: \n";
	test_addition();
	test_sub();
	test_multiplication();
	test_division();
	std::cout << "subtraction is null\n";
	test_sub_is_null();

	std::cout << "\n Testing operator overloading: \n";
	test_addition_op();
	test_subtraction_op();
	test_multiplication_op();
	test_division_op();
	std::cout << "\n Testing operator overloading (incl. assign): \n";
	test_addition_op_ass();
	test_sub_op_ass();
	test_multiplication_op_ass();
	test_division_op_ass();

	std::cout << "\n Testing operator overloading with int: \n";
	test_addition_op_int();
	test_sub_op_int();
	test_mul_op_int();
	test_div_op_int();
	std::cout << "\nTest by division zero:\n";
	test_divide_by_zero();
	std::cout << "\n Testing operator overloading with comparison: \n";
	test_same();
	std::cout << "\n";
	test_different();
	std::cout << "\n";
	test_smaller();
	std::cout << "\n";
	test_smaller_same();
	std::cout << "\n";
	test_bigger();
	std::cout << "\n";
	test_bigger_same();

	std::cout << "Testing Read file:";
	std::cout << "\nNot valid:";
	 test_read_file_not_valid_file();
	std::cout << "\nValid:";
	test_read_file_valid();
	std::cout << "Wrong rational number in file:\n";
	std::cout << "letter in file\n\n";
	//letter in file
	void test_read_file_false1();
	std::cout << "\n\nNo '/' in file\n\n";
	// not / in file
	void test_read_file_false2();

	std::cout << "Testing output:\n";
	std::cout << "\ndenominator and numerator are negative :\n";
	test_output_negativ();

	std::cout << "\nnormalize output:\n";
	test_output_normalize();
	std::cout << "\ndenominator is null:\n";
```

```
        test_output_null1();
        std::cout << "\nnumerator is null:\n";
        test_output_null2();
        std::cout << "\nTest integer:\n";
         test_integer_1();
         std::cout << "\n";
         test_integer_2();
}
```

TESTFÄLLE:

1. TEST MIT DEN OPERATOREN + SUBTRAKTION ERGIBT 0

```
Testing number operations:
53/42
53/42
15/-8
-4
subtraction is null
0
```

2. TEST MIT ÜBERLADEN DER OPERATOREN

```
 Testing operator overloading:
53/42
53/42
15/-8
-4
```

3. TEST MIT ÜBERLADEN DER OPERATOREN MIT INTEGER

```
 Testing operator overloading with int:
23
1
4
1
```

4. TEST BEI NULL DIVISION

```
Test by division zero:
not valied-858993460/-858993460
PRoceeding with execution...
```

5. TEST VOM ÜBERLADEN DES VERGLEICHSOPRATOR==

```
 Testing operator overloading with comparison:
1/3
 Is the same as
1/3

1/3
 Is not same as
-1/3
```

6. TEST VOM ÜBERLADEN DES VERGLEICHSOPRATOR !=

```
1/3
 Is not different as
1/3

1/3
 Is  different as

-1/3
```

7. TEST VOM ÜBERLADEN DES VERGLEICHSOPRATOR <

```
1/3
 Is smaller than
1/2

1/3
 Is not smaller than
-1/3
```

8. TEST VOM ÜBERLADEN DES VERGLEICHSOPRATOR <=

```
1/3
 Is <=
1/3

1/3
 Is not  <=
-1/3
```

9. TEST VOM ÜBERLADEN DES VERGLEICHSOPRATOR >

```
1/3
 Is not bigger than
1/2

1/3
 Is bigger than
-1/3
```

10. TEST VOM ÜBERLADEN DES VERGLEICHSOPRATOR <=

```
1/3
 Is >=
1/3

1/3
  Is not >=
1/2
```

11. EINLESEN AUS FILE

- File ist nicht vorhanden
- File wird richtig eingelesen
- Inhalt des Files enthält Buchstaben
- Inhalt des Files enthält kein „/“

```
Testing Read file:
Not valid:
file is damaged1

Valid:4/51
Wrong rational number in file:
letter in file



No '/' in file
```

TEST DES OUTPUTS: OPTIMIEREN DER AUSGABE

```
Testing output:

denominator and numerator are negative :
-1/-1
optimized:1

normalize output:
6/3
optimized:2

denominator is null:
not valied
1/0
optimized:1/0

numerator is null:
0
optimized:0

Test integer:
4/2
optimized:2

5/-5
optimized:-1
```