Uebung 04

Aufwand: 14 Stunden

Selina Adlberger

12-11-2022

Übung 04

BEISPIEL OPERATOREN ÜBERLADEN

LÖSUNGSIDEE:

Das Programm kann mit Bruchzahlen rechnen. Es wird eine Klasse erstellt mit Nenner und Zähler. Der Datentyp wird mithilfe eines Templates übergeben.

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. Hierfür wird die Operationen und neutralen Elemente herangezogen.

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.

Mithilfe des Templates soll es auch möglich seine anderen Datentypen in die Klasse zu übergeben. Es wird ein datentyp number_t angelegt, der dies ermöglicht.

Anforderungen C++ Concept numeric:

v + v; //T requires operator +

- v - v;
- v* v;
- v / v;
- v += v;
- v -= v;
- v*= v;
- v /= v;

- v == v;
- v != v;
- v < v;
- v > v;
- v <= v;
- v >= v;
- std::cout << v;

Diese Operatoren müssen für number_t und für rational_t erstellt werden mit dem „Barton-Nackman Trick“. Bei diesem Trick wird ein Operator als friend mit zwei Parameter übergeben.

Wenn in einer der beiden Klassen eine Operation fehlt, funktioniert es nicht mehr.

Es werden neutrale Elemente erstellt, diese sind im gesamten Programm verwendbar. Außerdem werden einige Funktionen als Operation gespeichert um diese allgemeinen Funktionen in mehreren Programmen verwenden kann.

QUELLTEXT

RATIONAL_T.H

```
#ifndef rational_t_h
#define rational_t_h
#include <iostream>
#include <fstream>
#include <string>
#include <stdexcept>
#include <vector>
#include"number_t.h"
#include"operations.h"

template<typename T>
concept Numeric = requires (T v) {
	v + v; //T requires operator +
	v - v;
	v* v;
	v / v;
	v += v;
	v -= v;
	v*= v;
	v /= v;
	v == v;
	v != v;
	v < v;
	v > v;
	v <= v;
	v >= v;
	std::cout << v;

};
template<Numeric T=int>
class rational_t {

	T zaehler;
	T nenner;

	friend inline std::ostream& operator<<(std::ostream& lhs, rational_t<T> const&
rhs) {
		rhs.print(lhs);
		return lhs;
	}
	friend inline bool operator==(rational_t<T> const& lhs, rational_t<T> const&
rhs) {
		//rational_t tmp(*this);
		if (lhs.nenner == rhs.nenner && lhs.zaehler == rhs.zaehler)return true;
		else return false;
	}
	friend inline bool operator!=(rational_t<T> const& lhs, rational_t<T> const&
rhs) {
		//rational_t tmp(*this);
		if (lhs.nenner != rhs.nenner || lhs.zaehler != rhs.zaehler)return true;
		else return false;
	}
	friend inline bool operator<(rational_t<T> const& lhs, rational_t<T> const&
rhs) {

		T erg1 = lhs.zaehler /  lhs.nenner;
		T erg2 =  rhs.zaehler /  rhs.nenner;
		if (erg1 < erg2)return true;
		else return false;
```

```
	}
	friend inline bool operator<=(rational_t<T> const& lhs, rational_t<T> const&
rhs) {

		T erg1 =  lhs.zaehler / lhs.nenner;
		T erg2 = rhs.zaehler / rhs.nenner;
		if (erg1 <= erg2)return true;
		else return false;
	}
	friend inline bool operator>(rational_t<T> const& lhs, rational_t<T> const&
rhs) {

		T erg1 = lhs.zaehler / lhs.nenner;
		T erg2 = rhs.zaehler / rhs.nenner;
		if (erg1 > erg2)return true;
		else return false;
	}
	friend inline bool operator>=(rational_t<T> const& lhs, rational_t<T> const&
rhs) {

		T erg1 = lhs.zaehler / lhs.nenner;
		T erg2 = rhs.zaehler / rhs.nenner;
		if (erg1 >= erg2)return true;
		else return false;
	}

	friend inline rational_t<T> operator+(rational_t<T> const& lhs, rational_t<T>
const& rhs) {

		rational_t<T> tmp(lhs);
		tmp.add(rhs);
		if (tmp.is_consistent())return tmp;
	}
	friend inline rational_t<T> operator-(rational_t<T> const& lhs, rational_t<T>
const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.sub(rhs);
		return tmp;
	}
	friend inline rational_t<T> operator*(rational_t<T> const& lhs, rational_t<T>
const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.mul(rhs);
		if (tmp.is_consistent())
			return tmp;
	}
	friend inline rational_t<T> operator/(rational_t<T> const& lhs, rational_t<T>
const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.div(rhs);
		if (tmp.is_consistent())
			return tmp;
	}


	friend inline std::ifstream& operator>>(std::ostream& lhs, rational_t<T> const&
rhs) {
		T z = rhs.get_denominator();
		T n = rhs.get_numerator();;
		rhs.scan(lhs, n, z);

		return lhs;
```

```
	}

	friend inline rational_t<T> operator+=(rational_t<T> const& lhs, rational_t<T>
const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.add(rhs);
		if (tmp.is_consistent())return tmp;
	}
	friend inline rational_t<T> operator-=(rational_t<T> const& lhs, rational_t<T>
const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.sub(rhs);
		return tmp;
	}
	friend inline rational_t<T> operator*=(rational_t<T> const& lhs, rational_t<T>
const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.mul(rhs);
		if (tmp.is_consistent())
			return tmp;
	}
	friend inline rational_t<T> operator/=(rational_t<T> const& lhs, rational_t<T>
const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.div(rhs);
		if (tmp.is_consistent())
			return tmp;
	}
	friend inline rational_t<T> operator+(int const& lhs, rational_t const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.add(rhs);
		return tmp;
	}
	friend inline rational_t<T> operator-(int const& lhs, rational_t const& rhs){
		rational_t<T> tmp(lhs);
		tmp.sub(rhs);
		return tmp;
	}
	friend inline rational_t<T> operator*(int const& lhs, rational_t const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.mul(rhs);
		return tmp;
	}
	friend inline rational_t<T> operator/(int const& lhs, rational_t const& rhs) {
		rational_t<T> tmp(lhs);
		tmp.div(rhs);
		return tmp;
	}
public:
	void inverse() {
		T tmp = nenner;
		nenner = zaehler;
		zaehler = tmp;
	}
	rational_t(T _zaehler = 0, T _nenner = 1) :zaehler{ _zaehler }, nenner{ _nenner
} {
		if (nenner == 0) {
			std::cout << "not valid";
		}
	}
	rational_t(rational_t<T> const& other) {
		nenner = other.nenner;
		zaehler = other.zaehler;
```

```
	}
	rational_t() {

	}
	int get_numerator()const {
		return zaehler;
	}
	int get_denominator()const {
		return nenner;
	}
	std::string as_string()const {
		return std::to_string(zaehler) + "/" + std::to_string(nenner);
	}
	bool is_negative(T& const x) {
		if (x < nelms::zero<T>() ) return true;
		else return false;
	}
	bool is_positive(T& const x) {
		if (x > nelms::zero<T>()) return true;
		else return false;
	}
	bool is_zero(T& const x) {
		if (ops::is_zero(x))return true;
		else return false;
	}
	bool is_consistent()const {
		if (nenner != nelms::zero<T>()) {
			return true;
		}
		else {
			return false;
		}

	}
	void 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;
		}
	}

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

	}
	void sub(rational_t<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 mul(rational_t<T> const& other) {
		zaehler = zaehler * other.zaehler;
```

```
		nenner = nenner * other.nenner;
		optimize();
	}
	void div(rational_t<T> const& other) {
		zaehler = zaehler * other.nenner;
		nenner = nenner * other.zaehler;
		optimize();
	}
	void print(std::ostream& os)const {
		//its 0
		if (zaehler == 0) {
			os << 0 << "";
		}
		//integer
		else if (nenner == 1) {
			os << zaehler << "";
		}
		else {
			os << zaehler << "/" << nenner << "";
		}
	}
	void scan(std::ifstream& in, T& n, T& 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";
		}

	}
private:
	void normalize(T& z, T& n) {
		T low(0);
		T _n = n;
		T _z = z;
		//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 =_n* (-1);
		if (is_negative(z))_z =_n* (-1);
		if (z < n)low = _z;
		else low = _n;
		//search divider

		//modulo does not work with datatyp T....it only works with int
		//it is not possible to find the divider
		/*int teiler = 1;
		for (int i(1); i < low+1; ++i) {
			if (n_int % i == 0 && z_int % i == 0) {
				teiler = i;
			}
		}
		z = z / teiler;
		n = n / teiler;*/
	}
};
template<typename T>
void print_result(rational_t<T> const& rational)
{
	std::cout << rational << "\n";
}
template<typename T>
std::ostream& operator<<(std::ostream& lhs, rational_t<T> const& rhs) {
	rhs.print(lhs);
	return lhs;
}
template<typename T>
std::ifstream& operator>>(std::ifstream& lhs, rational_t<T> const& rhs) {
	T z = rhs.get_denominator();
	T n = rhs.get_numerator();;
	rhs.scan(lhs, n, z);

	return lhs;
}

#endif
```

NUMBER_T.H

```
#ifndef number_t_h
#define number_t_h

#include "operations.h"
#include "rational_t.h"
template<typename T>
```

```
class number_t {

private:
	T value;


	friend inline std::ostream& operator<<(std::ostream& lhs, number_t<T> const&
rhs) {
		lhs << rhs.value;
		return lhs;
	}
	friend inline std::ifstream& operator>>(std::ostream& lhs, number_t<T> const&
rhs) {
		lhs >> rhs.value;

		return lhs;
	}

	friend inline number_t<T> operator+(number_t<T> const& lhs, number_t<T> const&
rhs) {
		number_t<T> tmp(lhs); //copy
		tmp.value += rhs.value;
		return tmp;
	}
	friend inline number_t<T> operator-(number_t<T> const& lhs, number_t<T> const&
rhs) {
		number_t<T> tmp(lhs); //copy
		tmp.value -= rhs.value;
		return tmp;
	}
	friend inline number_t<T> operator*(number_t<T> const& lhs, number_t<T> const&
rhs) {
		number_t<T> tmp(lhs); //copy
		tmp.value *= rhs.value;
		return tmp;
	}
	friend inline number_t<T> operator/(number_t<T> const& lhs, number_t<T> const&
rhs) {
		number_t<T> tmp(lhs); //copy
		tmp.value /= rhs.value;
		return tmp;
	}

	friend inline number_t<T> operator+=(number_t<T> const& lhs, number_t<T> const&
rhs) {
		number_t<T> tmp(lhs); //copy
		tmp.value += rhs.value;
		return tmp;
	}
	friend inline number_t<T> operator-=(number_t<T> const& lhs, number_t<T> const&
rhs) {
		number_t<T> tmp(lhs); //copy
		tmp.value -= rhs.value;
		return tmp;
	}
	friend inline number_t<T> operator*=(number_t<T> const& lhs, number_t<T> const&
rhs) {
		number_t<T> tmp(lhs); //copy
		tmp.value *= rhs.value;
		return tmp;
	}
	friend inline number_t<T> operator/=(number_t<T> const& lhs, number_t<T> const&
rhs) {
```

```
		number_t<T> tmp(lhs); //copy
		tmp.value /= rhs.value;
		return tmp;
	}


	friend inline bool operator==(number_t<T> const& lhs, number_t<T> const& rhs) {
		return (lhs.value == rhs.value);
	}
	friend inline bool operator!=(number_t<T> const& lhs, number_t<T> const& rhs) {
		return lhs.value != rhs.value;
	}
	friend inline bool operator<(number_t<T> const& lhs, number_t<T> const& rhs) {
		return lhs.value < rhs.value;
	}
	friend inline bool operator<=(number_t<T> const& lhs, number_t<T> const& rhs) {
		return lhs.value <= rhs.value;
	}
	friend inline bool operator>(number_t<T> const& lhs, number_t<T> const& rhs) {
		return lhs.value > rhs.value;
	}
	friend inline bool operator>=(number_t<T> const& lhs, number_t<T> const& rhs) {
		return lhs.value >= rhs.value;
	}

public:
	number_t() {
	}
	number_t(T const& _value) :value(_value) {  //conversion destructor
		//std::cout << "Conversion Constructor\n";
	}
	number_t(number_t<T> const& other) : value(other.value) {
	}


};
namespace nelms {
	template<> inline number_t<int> zero() {
		return number_t(0);
	}
	template<> inline number_t<int> one() {
		return number_t(1);
	}
}
#endif // !number_t_h
```

OPERATIONS.H

```
#ifndef operation_h
#define operation_h

#include<type_traits>

namespace nelms { //neutral elements for rational_t construction
	template<typename T> inline T zero() {
		return T();
	}
	template<> inline int zero() {
		return 0;
	}
	template<> inline double zero() {
		return 0.0;
```

```
	}

	template<typename T> inline T one() {
		return std::integral_constant<T, 1>::value;
	}
	template<> inline int one() {
		return 1;
	}
	template<> inline double one() {
		return 1.0;
	}
}

namespace ops {

	template<typename T>
	inline bool is_negative(T const& a) {
		return a < nelms::zero<T>();  // vergleicht mit nullelement aus neutralem
Element
	}
	template<typename T>
	inline bool is_zero(T const& a) {
		return a == nelms::zero<T>();
	}
	template<typename T>
	inline T negate(T const& a) {
		return nelms::zero<T> -a;  //bekommen das negative udn wenn negativ dann
positiv
	}
	template<typename T>
	inline T abs(T const& a) {
		if (a < 0)return a * (-1);
		else return a;
	}
	template<typename T>
	inline bool divides(T const& a, T const& b) {
		if (a % b == 0) {
			return true;
		}
		else return false;
	}
	template<typename T>
	inline bool equals(T const& a, T const& b) {
		if (a == b)return true;
		else return false;
	}
	template<typename T>
	inline T gcd(T a, T b) {

		while (a != b) {
			if (a > b)
				a -= b;
			else
				b -= a;
		}
		return a;
	}
	template<typename T>
	inline T remainder(T const& a, T const& b) {
		return a / b;
	}
}
```

```
#endif // !operation_h
```

TEST.H

```
#ifndef test_h
#define test_h

//test defaultvalue int for template
void test_addition();
void test_sub();
void test_multiplication();
void test_division();
//test number-t in rational_t
void test_number_t();

//test numeric_t_class()
void test_number_class();
//test operators
void test_operators();
//test inverse
void test_inverse();


//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();

#endif // !1
```

TEST.CPP

```
#include "test.h"
#include "rational_t.h"
#include "number_t.h"
#include <string>


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_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);
}


//number_t <int> tests
void test_addition_num() {
	rational_t<number_t<int>>a(3, 7);
	rational_t<number_t<int>>b(-5, -6);
	a.add(b);

	print_result(a);

}
void test_subtraction_num() {
	rational_t<number_t<int>>a(3, 7);
	rational_t<number_t<int>>b(5, -6);
	a.sub(b);
	print_result(a);

}
void test_multiplication_num() {
	rational_t<number_t<int>>a(5, -2);
	rational_t<number_t<int>>b(3, 4);
	a.mul(b);
	print_result(a);
}

void test_division_num() {
	rational_t<number_t<int>>a(2, 1);
	rational_t<number_t<int>>b(1, -2);
	a.div(b);
	print_result(a);
}
void test_number_t() {
	test_addition_num();
	test_subtraction_num();
	test_multiplication_num();
	test_division_num();
}

void test_concept() {

//	rational_t<std::string> a('s', 't');

}

void test_same() {
	// same
	rational_t <number_t<int>> a(1, 3);
	rational_t<number_t<int>> 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<number_t<int>> c(1, 3);
		rational_t<number_t<int>> 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<number_t<int>> a(1, 3);
		rational_t<number_t<int>> 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<number_t<int>> c(1, 3);
		rational_t<number_t<int>> 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<number_t<int>> a(1, 3);
	rational_t<number_t<int>> 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<number_t<int>> c(1, 3);
	rational_t<number_t<int>> 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<number_t<int>> a(1, 3);
	rational_t<number_t<int>> 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<number_t<int>> c(1, 3);
	rational_t<number_t<int>> 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<number_t<int>> a(1, 3);
	rational_t<number_t<int>> 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<number_t<int>> c(1, 3);
	rational_t<number_t<int>> 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<number_t<int>> a(1, 3);
	rational_t<number_t<int>> 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<number_t<int>> c(1, 3);
```

```
	rational_t<number_t<int>> 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_number_class() {
	 test_same();
	 test_different();
	 test_smaller();
	 test_smaller_same();
	 test_bigger();
	 test_bigger_same();


}

void test_addition_op_ass() {
	rational_t<> a(3, 7);
	rational_t<> b(-5, -6);
	a += 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_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<number_t<int>> a(2, 1);
	int b(21);  //convert
	rational_t<number_t<int>> c = b+ a;
	//c = b + a;
	print_result(c);
}
void test_sub_op_int() {
	rational_t<number_t<int>> a(2, 1);
	int b(1);  //convert
	rational_t<number_t<int>> c = b - a;

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

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

	print_result(c);
}
void test_operators() {
	 test_addition_op();
	 test_subtraction_op();
	 test_multiplication_op();
	 test_division_op();
	 std::cout << "op assign:\n";
	//op assign
	 test_addition_op_ass();
	 test_sub_op_ass();
	 test_multiplication_op_ass();
	 test_division_op_ass();
	 //with int
	 std::cout << "op with int:\n";
	  test_addition_op_int();
	  test_sub_op_int();
	  test_mul_op_int();
	  test_div_op_int();
}
```

```
void test_inverse() {
	rational_t<number_t<int>> b(1, 3);
	print_result(b);
	std::cout << "--->";
	b.inverse();
	print_result(b);
}



//scan
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);
}
```

MAIN.CPP

```
#include "rational_t.h"
#include "test.h"

int main() {

	std::cout << "Test template default value:\n";
	 test_addition();
	 test_sub();
	 test_multiplication();
	 test_division();

	 std::cout << "Test number_t in rational_t:\n";
	  test_number_t();

	  std::cout << "Test number_t clas in rational_t:\n";
	  test_number_class();

	  std::cout << "Test inverse: \n";
```

```
        test_inverse();


        std::cout << " Test Operatoren: \n ";
         test_operators();
        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();


    return 0;
}
```

TESTFÄLLE:

1. TEST_DEFAULT_INT():

Tests mit Default des templates ist int.
Erwartung: Wenn kein Datentyp mitgegeben wird, wird automatisch int als Default wert verwendet. Und somit eine rationale Zahl mit int ausgegeben

Output:

```
Test template default value:
53/42
53/42
15/-8
-4
```

2. TEST_NUMBER_T():

Test mit dem Datentyp number_t der zuvor erstellt wurde.
Erwartung: Gleiche Ausgabe wie mit Default int da der number_t typ ebenfalls int ist

Output:

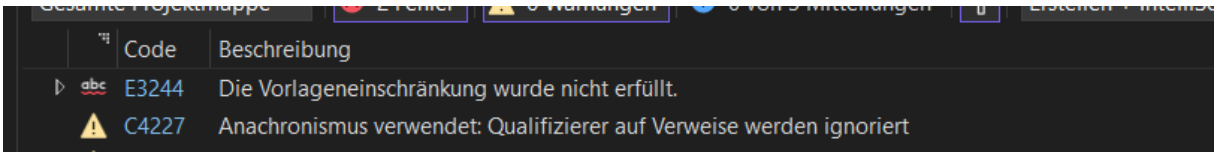
```
Test number_t in rational_t:
53/42
53/42
15/-8
-4
```

3. TEST_CONSEPT()

Test mit consept numeric()
Erwartung: Funktioniert auch mit numeric, da alle notwendigen Operatoren in beiden Klassen enthalten sind

Output:

Erwartung: string kann nicht übergeben werden → Fehlermeldung



4. TEST NUMBER_T_CLASS()

Test der number_t Klasse
Erwartung: Operationen funktionieren mit number_t<int> als Datentyp

Output:

```
 Is the same as
1/3

1/3
 Is not same as
-1/3

1/3
 Is not different as
1/3

1/3
 Is  different as

-1/3

1/3
 Is not smaller than
1/2

1/3
 Is not smaller than
-1/3

1/3
 Is <=
1/3

1/3
 Is <=
-1/3

1/3
 Is not bigger than
1/2

1/3
 Is not bigger than
-1/3

1/3
 Is >=
1/3

1/3
  Is  >=
1/2
```

5. TEST_INVERSE()

Es wird der Kehrwert berechent

Erwartung: Es wird der Kehrwert ausgegeben. Zuvor wird dieser noch vereinfacht

Output:

```
Test inverse:
1/3
--->3
```

6. TEST OPERATOREN()

Es werden die Operatoren (+,+=,-,-=,*,*=,/,/=)getestet, die Vergleichsoperatoren wurden bereits bei Test 4 getestet. Ebenfalls werden Operatoren mit int getestet.

Output:

```
 Test Operatoren:
 53/42
53/42
15/-8
-4
op assign:
3/7
3/7
5/-2
2
op with int:
23
-1
4
2/2
```

7. TEST_FILE()

- 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
```