SWE3; WS 2022

Wolfgang Eder

S2110458039

Zeitaufwand: 18 Stunden

Programmierübung UE04

Inhaltsverzeichnis

Beispiel 1: „Klasse rational_t erweitern“

Code

Das Programm befindet sich in der Solution SWE3_EDER_UE04 unter der Solution: „Klasse_rational_t_erweitern“.

Lösungsansatz:

Es wird eine generische Klasse für rationale Zahlen implementiert, die es ermöglicht mit den Bruchzahlen zu rechnen, diese zu manipulieren und ein-/auszugeben.

Die Klasse beinhaltet zwei Datentypen(Default Integer) welche als Objekt „rational_t“ angelegt werden. Diese werden mit Hilfe von Überladungen von deren Operatoren gerechnet, von Hilfsfunktionen auf deren Integrität überprüft und wiederrum mit überladenen Operatoren aus- sowie eingegeben.

Dabei wird ein Konzept „numeric“ angelegt welches die Datentypen einschränkt und den Funktionsumfang wiederspiegelt. Dafür wurde eine Testklasse „numeric_t“(default Integer) angelegt welche die des Konzepts geforderten Operatoren wiederspiegelt und über die Klasse zur Verfügung stellt. Explizit wurden folgende Operatoren gewählt:

```
template<typename S>
concept numeric = requires(S t) {
    t + t;
    t += t;
    t - t;
    t -= t;
    t* t;
    t *= t;
    t / t;
    t /= t;
    t == t;
    t != t;
    t < t;
    t > t;
    t <= t;
    t >= t;
    std::cout << t;
    std::cin >> t;
};
```

Das Objekt wird über die Konstruktoren entweder mit einem oder zwei Argumenten erzeugt, ggf. „inline“ deklariert und auf ihre Integrität überprüft. Insbesondere bei rationalen Zahlen wird darauf geachtet das der Nenner keinen Nullwert annimmt. Sollte dem so sein wird dieser mit eins initialisiert oder ggf. eine eigens implementierte Ausnahme bei einer Division durch Null an den Compiler geschickt.

Im Allgemeinen wird darauf geachtet soviel wie möglich zu delegieren, konstant zu deklarieren und zu verstecken.

Zur Manipulation der Werte des Objektes werden „Get-a“ und „Set-a“ Funktionen verwendet.

Zur Sicherstellung der Integrität werden Hilfsfunktionen verwendet, welche auf den Zustand, also positiv-negativ und Nullwert, rückschließen lassen. Sollten Zahlen zum rechnen verwendet werden welche ohne einen gemeinsamen Teiler nicht gerechnet werden können, werden diese Zahlen normalisiert.

Die Normalisierung überprüft die rationale Zahl darauf, ob Zähler und Nenner ident sind, sie mittels ihres kleinsten gemeinsamen Nenners dargestellt sind und ob sich die Vorzeichen beider vereinigt darstellen lassen. Sollte das der Fall sein wird die Zahl nivelliert.

Die Funktion „is_consistent“ stellt sicher, dass der Nenner nicht null ist und wirft ggf. eine individuell implementierte Ausnahme.

Die Funktion „inverse“ bildet den Kehrwert des Bruches als auch gibt es eine Funktion „lowest common multiple“, welche den kleinsten Multiplikator zur Verwendung bereitstellt.

Die Operatoren werden überladen sowohl in einer Version zum rechnen von Ganzzahlen mit rationalen Zahlen als auch vice versa zum Rechnen von rationalen Zahlen mit Ganzzahlen.

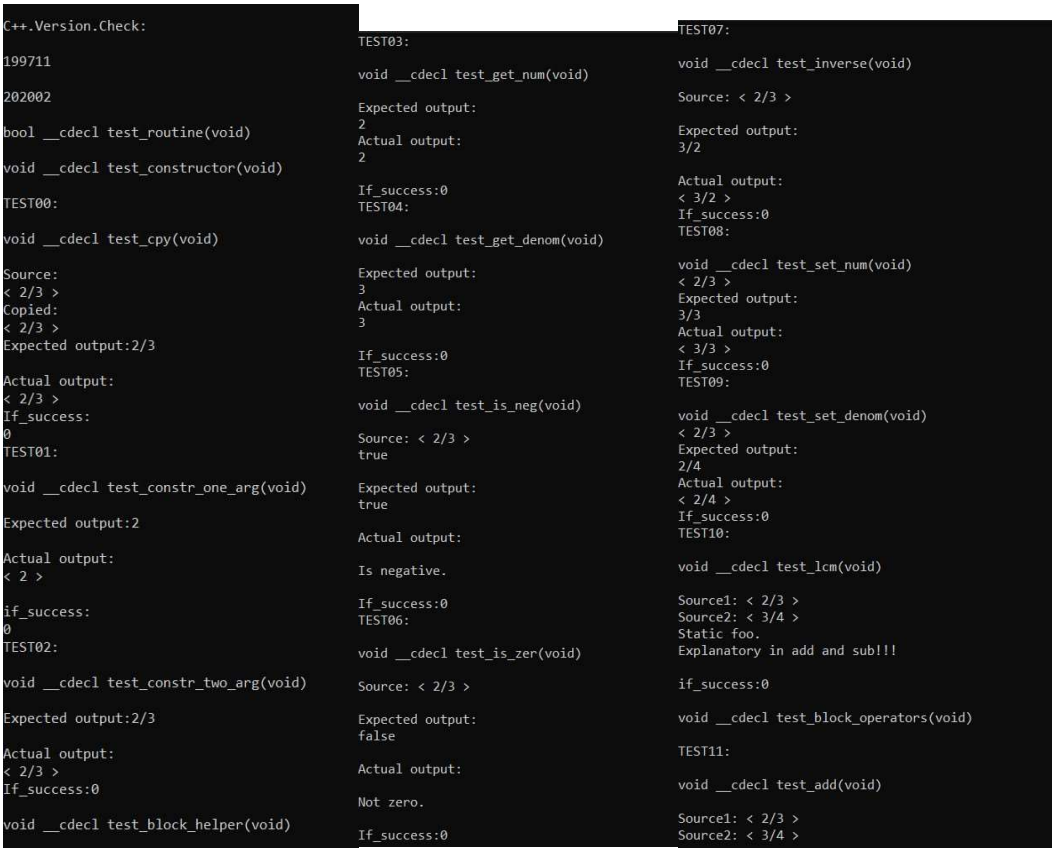
Zur Ausgabe werden die Zahlen über den überladenen Ausgabe Operator ausgegeben.

Zur Eingabe gibt es vice versa eine Funktion sowohl als auch einen überladenen Operator. Zur technischen Umsetzung werden in der Klasse Memberfunktionen sowie Non-Memberfunktionen verwendet und ein Friending bedingt diese, wobei mit dem Schlüsselwort den Non-member-functions Zugriff auf die privaten und geschützten Blöcke gewehrt wird. Einfach gesagt wird der sogenannte „Barton-Nackmann-Trick“ angewandt um die zweistelligen Operatoren inline und als friend-Funktionen darzustellen.

Zusätzlich bedingt die generische Programmierung noch weitere Definitionen für ihre Datentypen, welche in der Datei „operations.h“ abgebildet sind. Dort werden für die einzelnen Datentypen im ersten Namensraum, deren Nullwerte, Einserwerte bzw. Neutralen Elemente definiert. Zusätzlich werden noch Funktionen definiert die in einem eigenen Namensraum die Evaluierung der verschiedenen Datentypen erlaubt. Somit erlauben dies Funktionen die Ermittlung der absoluten Werte des Bruches, die Dividierbarkeit, die Gleichheit, der größte gemeinsame Teiler, das Vorzeichen, der Rest und die Negierbarkeit zu ermitteln.

Zur Testung wurde ein Test-Modul implementiert, welche alle Funktionen einzeln testet und inhaltlich in Blöcke gekapselt sind. Insgesamt wurden 38 Testfälle generiert welche sich folglich zeigen:

Testfälle:



```
C++.Version.Check:

199711

202002

bool __cdecl test_routine(void)

void __cdecl test_constructor(void)

TEST00:

void __cdecl test_cpy(void)

Source:
< 2/3 >
Copied:
< 2/3 >
Expected output:2/3

Actual output:
< 2/3 >
If_success:
0
TEST01:

void __cdecl test_constr_one_arg(void)

Expected output:2

Actual output:
< 2 >

if_success:
0
TEST02:

void __cdecl test_constr_two_arg(void)

Expected output:2/3

Actual output:
< 2/3 >
If_success:0

void __cdecl test_block_helper(void)

TEST03:

void __cdecl test_get_num(void)

Expected output:
2
Actual output:
2

If_success:0
TEST04:

void __cdecl test_get_denom(void)

Expected output:
3
Actual output:
3

If_success:0
TEST05:

void __cdecl test_is_neg(void)

Source: < 2/3 >
true

Expected output:
true

Actual output:

Is negative.

If_success:0
TEST06:

void __cdecl test_is_zer(void)

Source: < 2/3 >

Expected output:
false

Actual output:

Not zero.

If_success:0

TEST07:

void __cdecl test_inverse(void)

Source: < 2/3 >

Expected output:
3/2

Actual output:
< 3/2 >
If_success:0
TEST08:

void __cdecl test_set_num(void)
< 2/3 >
Expected output:
3/3
Actual output:
< 3/3 >
If_success:0
TEST09:

void __cdecl test_set_denom(void)
< 2/3 >
Expected output:
2/4
Actual output:
< 2/4 >
If_success:0
TEST10:

void __cdecl test_lcm(void)

Source1: < 2/3 >
Source2: < 3/4 >
Static foo.
Explanatory in add and sub!!!

if_success:0

void __cdecl test_block_operators(void)

TEST11:

void __cdecl test_add(void)

Source1: < 2/3 >
Source2: < 3/4 >
```

```
Source1: < 2/3 >
Source2: < 3/4 >

Expected output:
17/12

Actual output:

Result: < 17/12 >
if_success:0
TEST12:

void __cdecl test_sub(void)

Source1: < 2/3 >
Source2: < 3/4 >


Expected output:
-1/12

Actual output:

Result: < -7/3 >Unknown error.Sub funkt nicht ganS.Ongoing in -= etc.
if_success:1

TEST13:

void __cdecl test_mul(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
6/12

Actual output:

Result: < 6/12 >
if_success:0

TEST14:

void __cdecl test_div(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
8/9

Actual output:

Result: < 8/9 >
if_success:0


void __cdecl test_block_overloading(void)

TEST15:

void __cdecl test_add_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
17/12

Actual output:

Result: < 17/12 >
if_success:0
TEST16:

void __cdecl test_sub_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
-1/2

Actual output:

Result: < -7/3 >
if_success:0
TEST17:

void __cdecl test_mul_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:6/12


Actual output:

Result: < 6/12 >
if_success:0
```

```
TEST18:

void __cdecl test_div_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
8/9

Actual output:

Result: < 8/9 >
if_success:0

void __cdecl test_block_cmpnd_assgn_op(void)

TEST19:

void __cdecl test_add_op_ass(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
17/12

Actual output:

Result: < 17/12 >
if_success:0
TEST20:

void __cdecl test_sub_op_ass(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
-1/2

Actual output:

Result: < -7/3 >
if_success:0
```

```
TEST21:

void __cdecl test_mul_op_ass(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
6/12

Actual output:

Result: < 6/12 >
if_success:0
TEST22:

void __cdecl test_div_op_ass(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
8/9

Actual output:

Result: < 8/9 >
if_success:0

void __cdecl test_block_cmpr_op(void)

TEST23:

void __cdecl test_same_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
Same.

Actual output:

Result:
Same.

if_success:0
```

```
TEST24:

void __cdecl test_not_same_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
Not same.

Actual output:

Result:
Not same.

if_success:0
TEST25:

void __cdecl test_smaller_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
Not smaller

Actual output:

Result '<':
Not smaller.

if_success:0
TEST26:

void __cdecl test_smaller_same_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
Smaller-same.

Actual output:

Result:
Smaller-same.

if_success:0
```

```
TEST27:

void __cdecl test_bigger_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
Not Bigger.

Actual output:

Result:
Not bigger.

if_success:0
TEST28:

void __cdecl test_bigger_same_op(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
Bigger-same.

Actual output:

Result:
Bigger-same.

if_success:0

void __cdecl test_block_etc(void)

TEST29:

void __cdecl test_cpy_swap_idiom(void)

Source1: < 2/3 >
Source2: < 3/4 >
Expected output:
2/3

Actual output:

Result:
Copy-swaped-idiom:
< 2/3 >
if_success:0
```

```
TEST30:

void __cdecl test_is_consis(void)

Source: <2/0>

Expected output:
Exception:

Actual output:

Result:
Denominator is zero.Not consistent.

TEST31:

void __cdecl test_gcd(void)

Source1: < 2/3 >
Source2: < 3/4 >
Static foo.
Explanatory in normalize()!!!

if_success:0
TEST32:

void __cdecl test_norm(void)

Source1:20/30< 20/30 >
Source2: 12/4< 12/4 >
Normalized through print_foos.

Expected output:
S1:2/3; S2:3

if_success:0

void __cdecl test_block_lhs_rhs_op(void)

TEST33:

void __cdecl test_lhs_rhs_add(void)

Source1: < 2/3 >
Source2: < 3/4 >
Source3: 1
Source4: 2
Results:
LHS-RHS
RHS-LHS
Int-Rational:
< 1 >< 11/4 >
if_success:0
TEST34:

void __cdecl test_lhs_rhs_sub(void)

Source1: < 2/3 >
Source2: < 3/4 >
Source3: 1
Source4: 2
Results:
LHS-RHS
RHS-LHS
Int-Rational:
< -1 >< -1 >
if_success:0
TEST35:

void __cdecl test_lhs_rhs_mul(void)

Source1: < 2/3 >
Source2: < 3/4 >
Source3: 1
Source4: 2
Results:
LHS-RHS
RHS-LHS
Int-Rational:
< 2/3 >< 3/2 >
if_success:0
TEST36:

void __cdecl test_lhs_rhs_div(void)

Source1: < 2/3 >
Source2: < 3/4 >
Source3: 1
Source4: 2
Results:
LHS-RHS
RHS-LHS
Int-Rational:
< 1 >< 8/3 >
if_success:0
```

```
TEST37:

void __cdecl test_div_by_zero_exc(void)

Source1: < 1 >
Source2: < 0 >
Expected output: Exception!.
Actual output:
Divide By Zero Error.
if_success:0
TEST38:

void __cdecl test_scan(void)

Stringstream input:

Expected output:2 3
Actual output:< 2/3 >
if_success:0
```