SWE3; WS 2022
Wolfgang Eder
S2110458039

Zeitaufwand: 13 Stunden

Programmierübung UE03

Inhaltsverzeichnis

Beispiel 1: „Operatoren überladen“

Code

Das Programm befindet sich in der Solution SWE3_EDER_UE03 unter der Solution: „Operatoren überladen“.

Lösungsansatz:

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

Die Klasse beinhaltet zwei integer Typen 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.

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 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 können die Zahlen als Stringobjekt verwendet werden und können über den überladenen Ausgabe Operator ausgegeben werden, als auch über eine eigens implementierte Druckfunktion.

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.

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:

```
Visual Leak Detector read settings from: C:\Program
Visual Leak Detector Version 2.5.1 installed.

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>

TEST01:

void __cdecl test_constr_one_arg(void)
<2/1>

TEST02:

void __cdecl test_constr_two_arg(void)
<2/3>

void __cdecl test_block_helper(void)

TEST03:

void __cdecl test_get_num(void)
Get_Numerator: 2

TEST04:

void __cdecl test_get_denom(void)
Get_Denominator: 3

TEST05:

void __cdecl test_set_num(void)
Source: <2/3>
Input: 3
Setted: <3/3>

TEST06:

void __cdecl test_set_denom(void)
Source: <2/3>
Input: 4
Setted: <2/4>

TEST07:

void __cdecl test_asstr(void)
Variable conversion typeID:
class std::basic_string<char,struct std::char_traits<char>,class std::allocator<char> >
<2/3>

TEST08:

void __cdecl test_is_neg(void)
Source: <2/3>

Not negative.

TEST09:

void __cdecl test_is_pos(void)
Source: <2/3>

Is positive.

TEST10:

void __cdecl test_is_zer(void)
Source: <2/3>

Not zero.

void __cdecl test_block_operators(void)
```

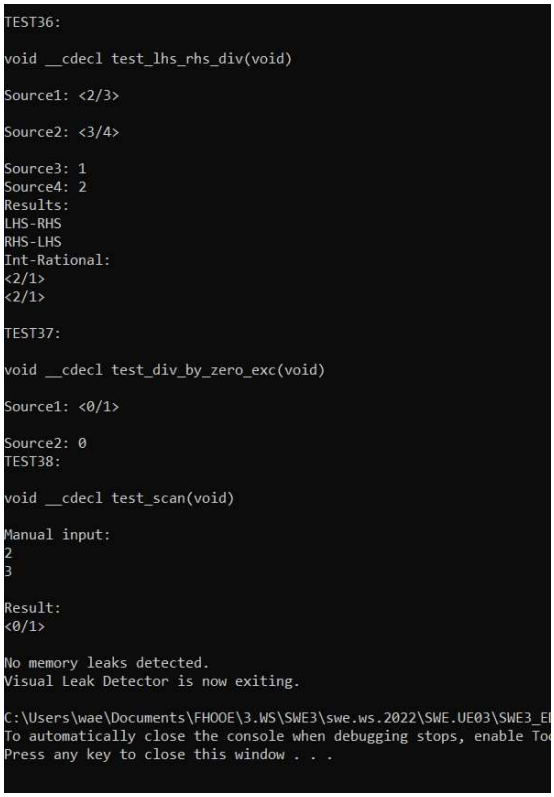
```
TEST11:

void __cdecl test_add(void)
Source1: <2/3>

Source2: <3/4>

Result: <5/7>

TEST12:

void __cdecl test_sub(void)
Source1: <2/3>

Source2: <3/4>

Result: <1/1>

TEST13:

void __cdecl test_mul(void)
Source1: <2/3>

Source2: <3/4>

Result: <6/-17>

TEST14:

void __cdecl test_div(void)
Source1: <2/3>

Source2: <3/4>

Result: <0/1>

void __cdecl test_block_overloading(void)

TEST15:

void __cdecl test_add_op(void)
Source1: <2/3>

Source2: <3/4>
<5/7>

TEST16:

void __cdecl test_sub_op(void)
Source1: <2/3>

Source2: <3/4>
<1/1>

TEST17:

void __cdecl test_mul_op(void)
Source1: <2/3>

Source2: <3/4>
<6/-17>

TEST18:

void __cdecl test_div_op(void)
Source1: <2/3>

Source2: <3/4>
<0/1>

void __cdecl test_block_cmpnd_assgn_op(void)

TEST19:

void __cdecl test_add_op_ass(void)
Source1: <2/3>

Source2: <3/4>
<5/7>
```

```
TEST20:

void __cdecl test_sub_op_ass(void)
Source1: <2/3>

Source2: <3/4>
<1/1>


TEST21:

void __cdecl test_mul_op_ass(void)
Source1: <2/3>

Source2: <3/4>
<6/-17>


TEST22:

void __cdecl test_div_op_ass(void)
Source1: <2/3>

Source2: <3/4>
<0/1>



void __cdecl test_block_cmpr_op(void)

TEST23:

void __cdecl test_same_op(void)
Source1: <2/3>

Source2: <3/4>
Not same.

TEST24:

void __cdecl test_not_same_op(void)
Source1: <2/3>

Source2: <3/4>
Same.
```


```
TEST20:

void __cdecl test_sub_op_ass(void)
Source1: <2/3>

Source2: <3/4>
<1/1>


TEST21:

void __cdecl test_mul_op_ass(void)
Source1: <2/3>

Source2: <3/4>
<6/-17>


TEST22:

void __cdecl test_div_op_ass(void)
Source1: <2/3>

Source2: <3/4>
<0/1>



void __cdecl test_block_cmpr_op(void)

TEST23:

void __cdecl test_same_op(void)
Source1: <2/3>

Source2: <3/4>
Not same.

TEST24:

void __cdecl test_not_same_op(void)
Source1: <2/3>

Source2: <3/4>
Same.
```


```
TEST30:

void __cdecl test_is_consis(void)
Source: <2/0>

TEST31:

void __cdecl test_gcd(void)
Source1: <2/3>

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

TEST32:

void __cdecl test_norm(void)
Source1: <2/3>
<2/3>

Source2: <3/1>
<3/1>

Normalized through print_foos.


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:
<3/4>
<1/1>
```

```
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:
<1/2>
<1/3>

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:
<1/-5>
<2/11>

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:
```


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:
<2/1>
<2/1>
TEST37:
void __cdecl test_div_by_zero_exc(void)
Source1: <0/1>
Source2: 0
TEST38:
void __cdecl test_scan(void)
Manual input:
2
3
Result:
<0/1>
No memory leaks detected.
Visual Leak Detector is now exiting.
C:\Users\wae\Documents\FHOOE\3.WS\SWE3\swe.ws.2022\SWE.UE03\SWE3_E
To automatically close the console when debugging stops, enable To
Press any key to close this window . . .