SWE3 Übung 03

1 INHALTSVERZEICHNIS

2 Beispiel 1: Rational 3
2.1 Lösungsidee 3
2.1.1 Konstruktor ohne Parameter 3
2.1.2 Konstruktor mit einer Zahl als Parameter 3
2.1.3 Konstruktor mit 2 Zahlen als Parameter 3
2.1.4 Konstruktor mit einem rational_t object als Parameter 3
2.1.5 Get_numerator 3
2.1.6 Get_denominator 3
2.1.7 Is_negative 3
2.1.8 Is_positive 3
2.1.9 Is_zero 3
2.1.10 Print 3
2.1.11 Scan 3
2.1.12 As_string 3
2.1.13 Is_consistent 3
2.1.14 Normalize 3
2.1.15 Gcd 4
2.1.16 To_positive 4
2.1.17 Operator= 4
2.1.18 Operator== 4
2.1.19 Operator!= 4
2.1.20 Operator< 4
2.1.21 Operator> 4
2.1.22 Operator<= 4
2.1.23 Operator>= 4
2.1.24 Operator+= 4
2.1.25 Operator-= 4
2.1.26 Operator*= 4
2.1.27 Operator/= 4
2.1.28 Operator+ 4
2.1.29 Operator- 5
2.1.30 Operator* 5

2.1.31 Operator/ 5
2.1.32 Operatoren +,-,*,/ mit rational_t und int 5
2.1.33 Operator<< 5
2.1.34 Operator>> 5
2.2 Code 5
2.3 Testfälle 6

Zeitaufwand: 6h

2 BEISPIEL 1: RATIONAL

2.1 LÖSUNGSIDEE

2.1.1 Konstruktor ohne Parameter
Der Zähler wird mit 0 und der Nenner wird mit 1 belegt: 0/1

2.1.2 Konstruktor mit einer Zahl als Parameter
Der Zähler wird mit dem Parameter und der Nenner wird mit 1 belegt.

2.1.3 Konstruktor mit 2 Zahlen als Parameter
Zähler und Nenner werden mit den 2 übergebenen Zahlen belegt. Falls der Parameter für den Nenner 0 ist (Division durch 0) dann wird eine Exception geworfen.

2.1.4 Konstruktor mit einem rational_t object als Parameter
Zähler und Nenner werden mit Zähler und Nenner des übergebenen Objekts belegt.

2.1.5 Get_numerator
Der Zähler wird zurückgeliefert.

2.1.6 Get_denominator
Der Nenner wird zurückgeliefert.

2.1.7 Is_negative
Gibt True zurück, wenn die Bruchzahl negativ ist und False wenn sie positiv ist.

2.1.8 Is_positive
Macht das Gegenteil von is_negative

2.1.9 Is_zero
Gibt True zurück, wenn die Bruchzahl 0 ist (wenn der Zähler 0 ist) und False wenn sie nicht 0 ist.

2.1.10 Print
Gibt die Bruchzahl auf einem Output-Stream als String aus.

2.1.11 Scan
Liest die Bruchzahl von einem Input-Stream. Wirft eine Exception, falls ungültige Werte eingegeben werden.

2.1.12 As_string
Liefert die Bruchzahl als String zurück.

2.1.13 Is_consistent
Prüft, ob der Nenner 0 ist. Ist der Nenner nicht 0, wird True zurückgeliefert, ansonsten False.

2.1.14 Normalize
Konvertiert die Bruchzahl in ihre kanonischen Repräsentanten. Dafür wird der größte gemeinsame Teiler von Zähler und Nenner ermittelt und die beiden werden durch diesen dividiert. Falls sowohl Zähler als auch Nenner negativ sind, werden die Minus weggelassen.

2.1.15 Gcd
Ermittelt den größten gemeinsamen Teiler zweier Zahlen.

2.1.16 To_positive
Wandelt eine Zahl in ihr positives Gegenstück um, falls diese negativ ist.

2.1.17 Operator=
Überlädt den Zuweisungsoperator und erlaubt damit eine rationale Zahl nur mithilfe eines „=“ mit einer anderen rationalen Zahl zu initialisieren.

2.1.18 Operator==
Zwei Bruchzahlen können über „==“ auf Gleichheit überprüft werden.

2.1.19 Operator!=
Zwei Bruchzahlen können über „!=“ auf Ungleichheit überprüft werden.

2.1.20 Operator<
Über „<“ kann überprüft werden, ob eine Bruchzahl kleiner ist als die andere.

2.1.21 Operator>
Über „>“ kann überprüft werden, ob eine Bruchzahl größer ist als die andere.

2.1.22 Operator<=
Über „<=“ kann überprüft werden, ob eine Bruchzahl kleiner oder gleich ist als die andere.

2.1.23 Operator>=
Über „>=“ kann überprüft werden, ob eine Bruchzahl größer oder gleich ist als die andere.

2.1.24 Operator+=
Zum Addieren zweier Brüche werden folgende Formeln angewandt:

Zähler = Zähler1 * Nenner2 + Zähler2 * Nenner1

Nenner = Nenner1 * Nenner2;

2.1.25 Operator-=
Beim Subtrahieren zweier Brüche, wird das Vorzeichen des Zählers der rechten Zahl umgekehrt und anschließend werden die Brüche addiert.

2/3 – 7/8 = 2/3 + (-7/8)

2.1.26 Operator*=
Falls einer der beiden Brüche 0 ist, wird auch das Ergebnis zu 0 (0/1). Anderenfalls werden sowohl die beiden Zähler als auch die beiden Nenner multipliziert.

2.1.27 Operator/=
Fall die rechte Zahl den Wert 0 hat, wird eine Exception geworfen. Ansonsten wird folgende Formel angewendet:

Zähler = Zähler1 * Nenner2

Nenner = Nenner1 * Zähler2

2.1.28 Operator+
Der += Operator wird aufgerufen.

2.1.29 Operator-
Der -= Operator wird aufgerufen.

2.1.30 Operator*
Der *= Operator wird aufgerufen.

2.1.31 Operator/
Der /= Operator wird aufgerufen.

2.1.32 Operatoren +,-,*,/ mit rational_t und int
Diese Operatoren überladen die Addition, Subtraktion, Multiplikation und Division einer ganzen Zahl mit einer Bruchzahl. Hierbei wird aus der ganzen Zahl ein Bruch (5 = 5/1) und dann wird wie oben beschrieben gerechnet.

2.1.33 Operator<<
Print wird aufgerufen, der Output-Stream-Parameter wird mitübergeben.

2.1.34 Operator>>
Scan wird aufgerufen, der Input-Stream-Parameter wird mitübergeben.

2.2 CODE
siehe SWE3_Huspek_Ue03

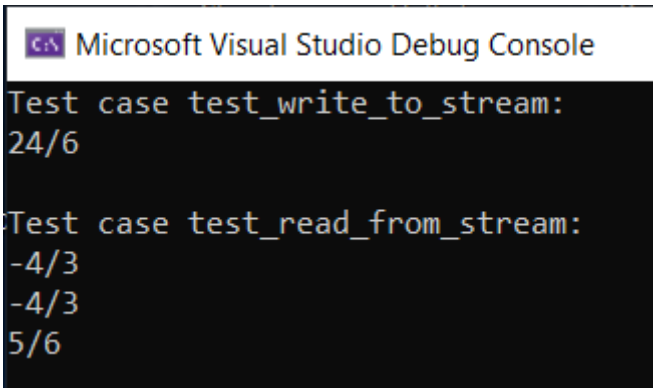
2.3 TESTFÄLLE

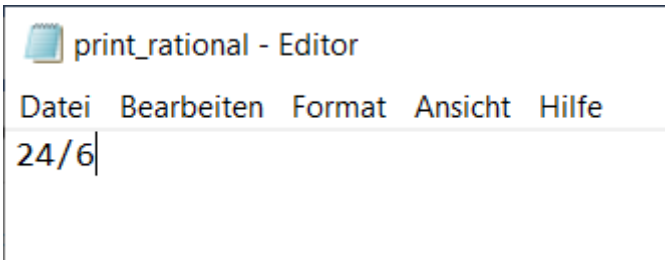
Microsoft Visual Studio Debug Console

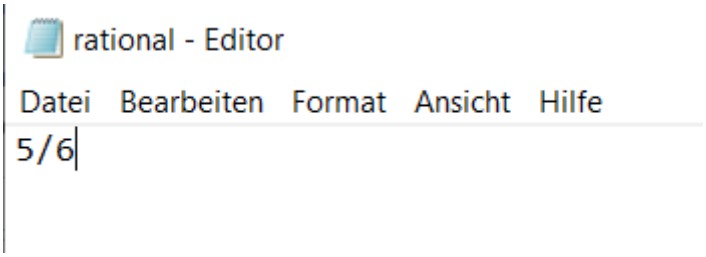
```
Test case test_constructor1:
0/1
Test case test_constructor2:
-5/1
Test case test_constructor3:
12/5
Test case test_constructor3_invalid_input:
Divide by Zero Custom Error!

Test case test_copy_constructor:
6/76/7
Test case test_normalize:
24/83/1
Test case test_scan:
Please enter a rational: 7/6
7/6
Test case test_scan:
Please enter a rational: 8/a
Invalid input Custom Error!

Test case test_scan:
Please enter a rational: 5(9
Invalid input Custom Error!

Test case test_get_numerator:
6/20 Numerator: 6

Test case test_get_denominator:
6/20 Denominator: 20

Test case test_is_negative:
3/4 is not negative.

Test case test_is_negative:
-3/4 is negative.

Test case test_is_negative:
3/-4 is negative.

Test case test_is_negative:
-3/-4 is not negative.

Test case test_is_positive:
3/4 is positive.

Test case test_is_positive:
-3/4 is not positive.
```

Microsoft Visual Studio Debug Console

```
Test case test_is_positive:
3/-4 is not positive.

Test case test_is_positive:
-3/-4 is positive.

Test case test_is_zero:
3/4 is not zero.

Test case test_is_zero:
0/5 is zero.

Test case test_add:
4/5 + -2/3 = 2/15
2/3 + -2/3 = 0/1
-2/3 + 0/3 = -2/3
0/3 + -2/3 = -2/3
5 + 4/5 = 29/5

Test case test_sub:
4/5 - -2/3 = 22/15
2/3 - 2/3 = 0/1
-2/3 - 0/3 = -2/3
0/3 - 5/6 = -5/6
5 - 4/5 = 21/5

Test case test_mul:
12/4 * 3/2 = 9/2
12/4 * 0/1 = 0/1
0/1 * 12/4 = 0/1
7/-8 * -5/3 = 35/24
13 * -5/3 = -65/3

Test case test_div:
12/4 / 3/2 = 2/1
12/4 / 0/1 = Divide by Zero Custom Error!
0/1 / 12/4 = 0/1
7/-8 / -5/3 = 21/40
13 / -5/3 = 39/-5
```


Microsoft Visual Studio Debug Console
Test case test_write_to_stream:
24/6
Test case test_read_from_stream:
-4/3
-4/3
5/6


print_rational - Editor
Datei Bearbeiten Format Ansicht Hilfe
24/6


rational - Editor
Datei Bearbeiten Format Ansicht Hilfe
5/6