SEK_UE04

1.Beispiel

Ansatz

Um dieses Problem zu lösen, muss eine Klasse *rational_t* so erweitert werden, sodass selbige als Template Klasse fungiert. Zusätzlich zur Template-Klasse muss außerdem noch ein Concept, also eine Sammlung von Operationen welche ein Typ unterstützen muss, um mit *rational_t* initialisiert werden zu können implementiert werden. Diese Anforderungen umfassen folgende Operatoren:

+, -, *, /, +=, -=, *=, /=, ==, !=, <, >, <=, >=, << und >>.

Alle diese Operatoren werden benötigt, damit alle Funktionen der *rational_t* Klasse reibungslos funktionieren. Zusätzlich sollen sofern möglich alle Operator-Funktionen mithilfe des Barton Nackmann Tricks implementiert werden. Somit werden Funktionen als non-member implementiert, aber der Zugriff auf private Klassenvariablen ist trotzdem möglich. Um das Programm effizient testen zu können, muss außerdem auch noch ein eigener numerischer Typ als Klasse entwickelt werden. Welcher alle Anforderungen von *rational_t* unterstützt. Zu guter Letzt wird noch eine Funktion *inverse* benötigt, welche den Kehrwert einer rationalen Zahl bildet.

Umsetzung

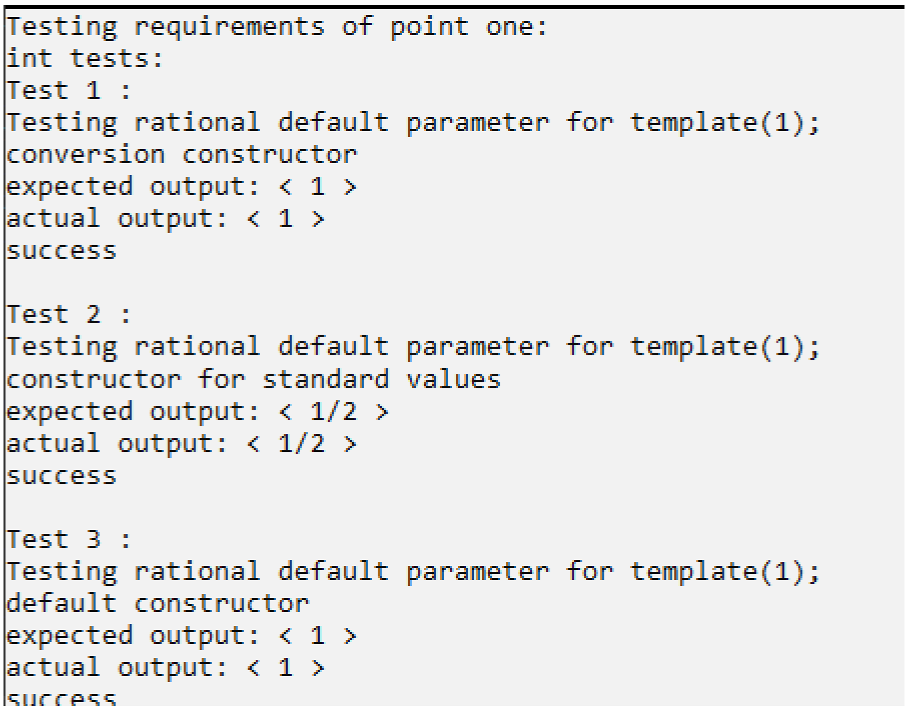
Zur Umsetzung ist in der Solution “UE04_Fallmann” im Projekt “T01” zu finden.
Der Bezug zu den jeweiligen Aufgabenunterpunkt wird jeweils in “(...)” angegeben.

Testcases

generic data-type rational_t initialisation

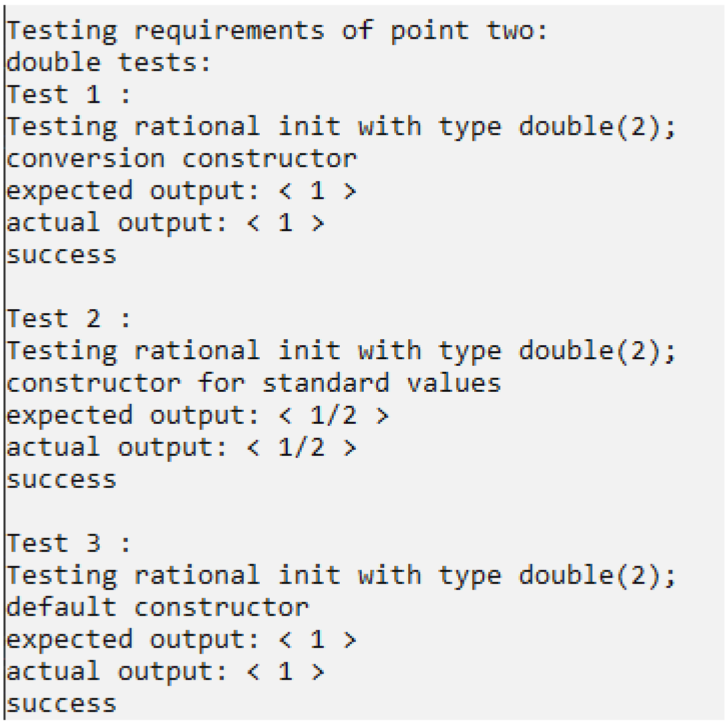
Anmerkung: Die Testcases der *rational_t* Klasse werden intern für jeden Datentyp extra gezählt.

- Test init with int and no parameters:

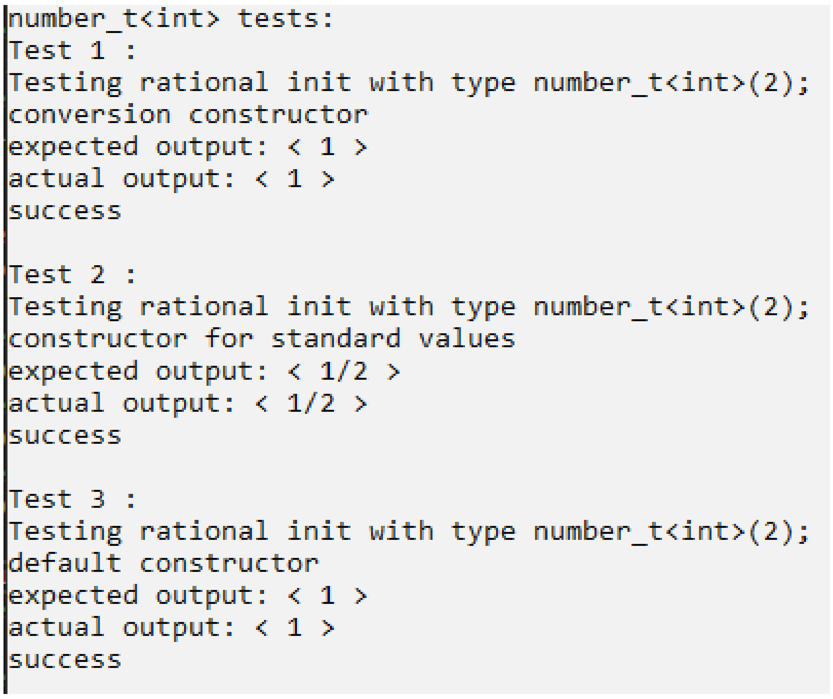


```
Testing requirements of point one:
int tests:
Test 1 :
Testing rational default parameter for template(1);
conversion constructor
expected output: < 1 >
actual output: < 1 >
success

Test 2 :
Testing rational default parameter for template(1);
constructor for standard values
expected output: < 1/2 >
actual output: < 1/2 >
success

Test 3 :
Testing rational default parameter for template(1);
default constructor
expected output: < 1 >
actual output: < 1 >
success
```

- <u>Test rational_t init with different data-types:</u>

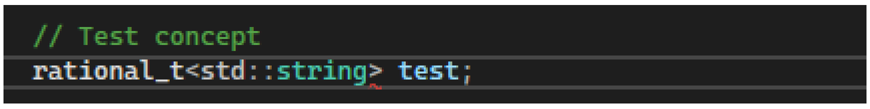


```
Testing requirements of point two:
double tests:
Test 1 :
Testing rational init with type double(2);
conversion constructor
expected output: < 1 >
actual output: < 1 >
success

Test 2 :
Testing rational init with type double(2);
constructor for standard values
expected output: < 1/2 >
actual output: < 1/2 >
success

Test 3 :
Testing rational init with type double(2);
default constructor
expected output: < 1 >
actual output: < 1 >
success
```

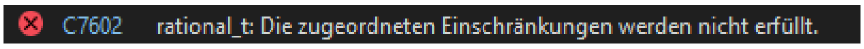


```
number_t<int> tests:
Test 1 :
Testing rational init with type number_t<int>(2);
conversion constructor
expected output: < 1 >
actual output: < 1 >
success

Test 2 :
Testing rational init with type number_t<int>(2);
constructor for standard values
expected output: < 1/2 >
actual output: < 1/2 >
success

Test 3 :
Testing rational init with type number_t<int>(2);
default constructor
expected output: < 1 >
actual output: < 1 >
success
```

- Test concept for rational_t:
  Bei diesem Test wurde versucht einen rational_t<string> anzulegen. Da dieser Typ nicht unterstützt wird gibt der Compiler einen Fehler aus. Numerische Datentypen werden unterstützt wie bereits aus den vorhergehenden Testfällen hervorgeht.(3)



```
// Test concept
rational_t<std::string> test;
```



```
C7602    rational_t: Die zugeordneten Einschränkungen werden nicht erfüllt.
```

generic data-type rational_t arithmetics

Bei allen arithmetischen Tests wird der Compound assign Operator der jeweiligen Operation verwendet, da jener im Hintergrund mit dem jeweiligen Operator(+, -, *, /) arbeitet.

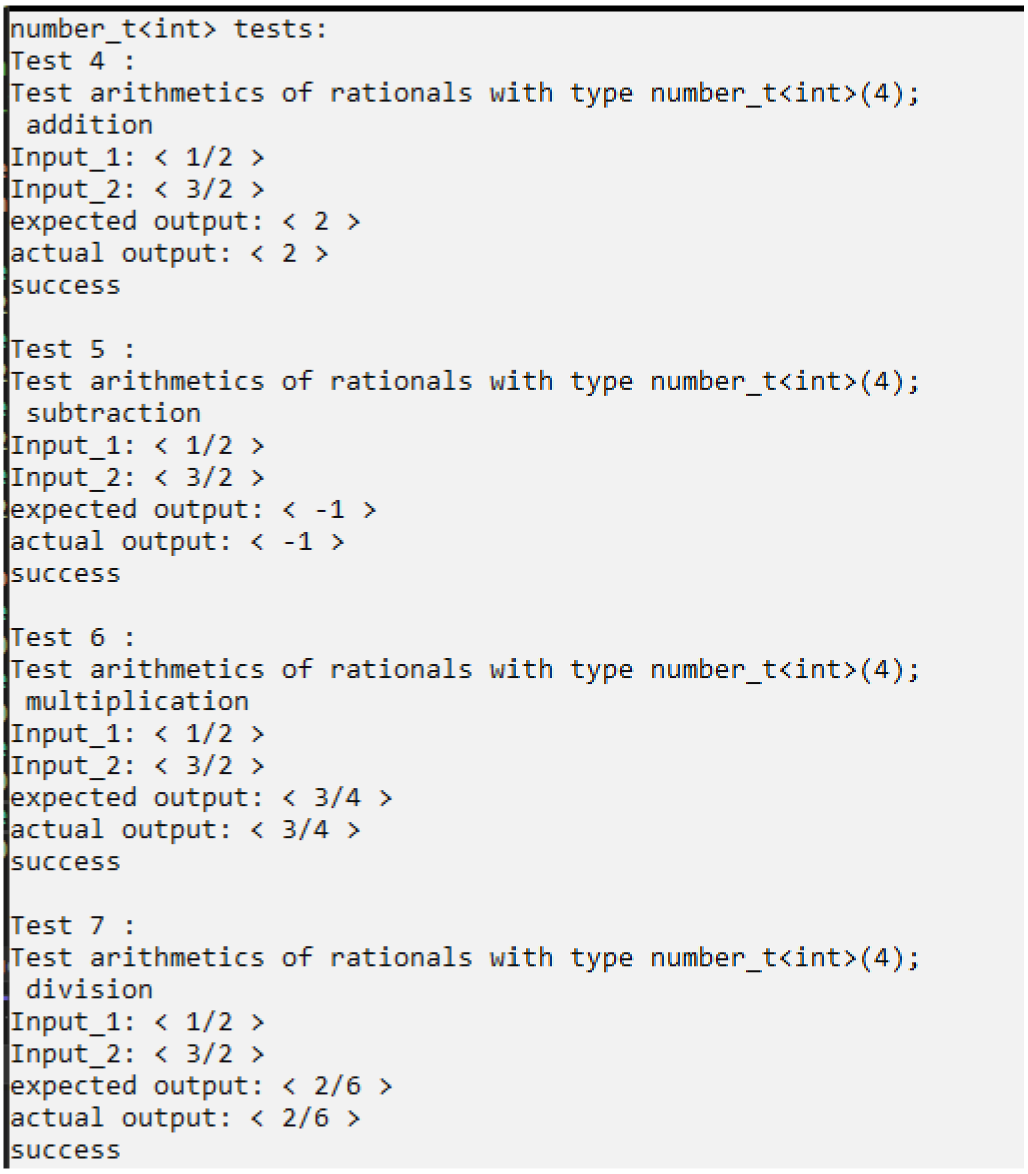
- Test arithmetics rational_t<int>:

```
Testing arithmetics of rational:
int tests:
Test 4 :
Test arithmetics of rationals with type int(4);
addition
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < 2 >
actual output: < 2 >
success

Test 5 :
Test arithmetics of rationals with type int(4);
subtraction
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < -1 >
actual output: < -1 >
success

Test 6 :
Test arithmetics of rationals with type int(4);
multiplication
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < 3/4 >
actual output: < 3/4 >
success

Test 7 :
Test arithmetics of rationals with type int(4);
division
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < 2/6 >
actual output: < 2/6 >
success
```

- Test arithmetics rational_t<double>:

```
double tests:
Test 4 :
Test arithmetics of rationals with type double(4);
addition
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < 2 >
actual output: < 2 >
success

Test 5 :
Test arithmetics of rationals with type double(4);
subtraction
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < -1 >
actual output: < -1 >
success

Test 6 :
Test arithmetics of rationals with type double(4);
multiplication
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < 3/4 >
actual output: < 3/4 >
success

Test 7 :
Test arithmetics of rationals with type double(4);
division
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < 2/6 >
actual output: < 2/6 >
success
```

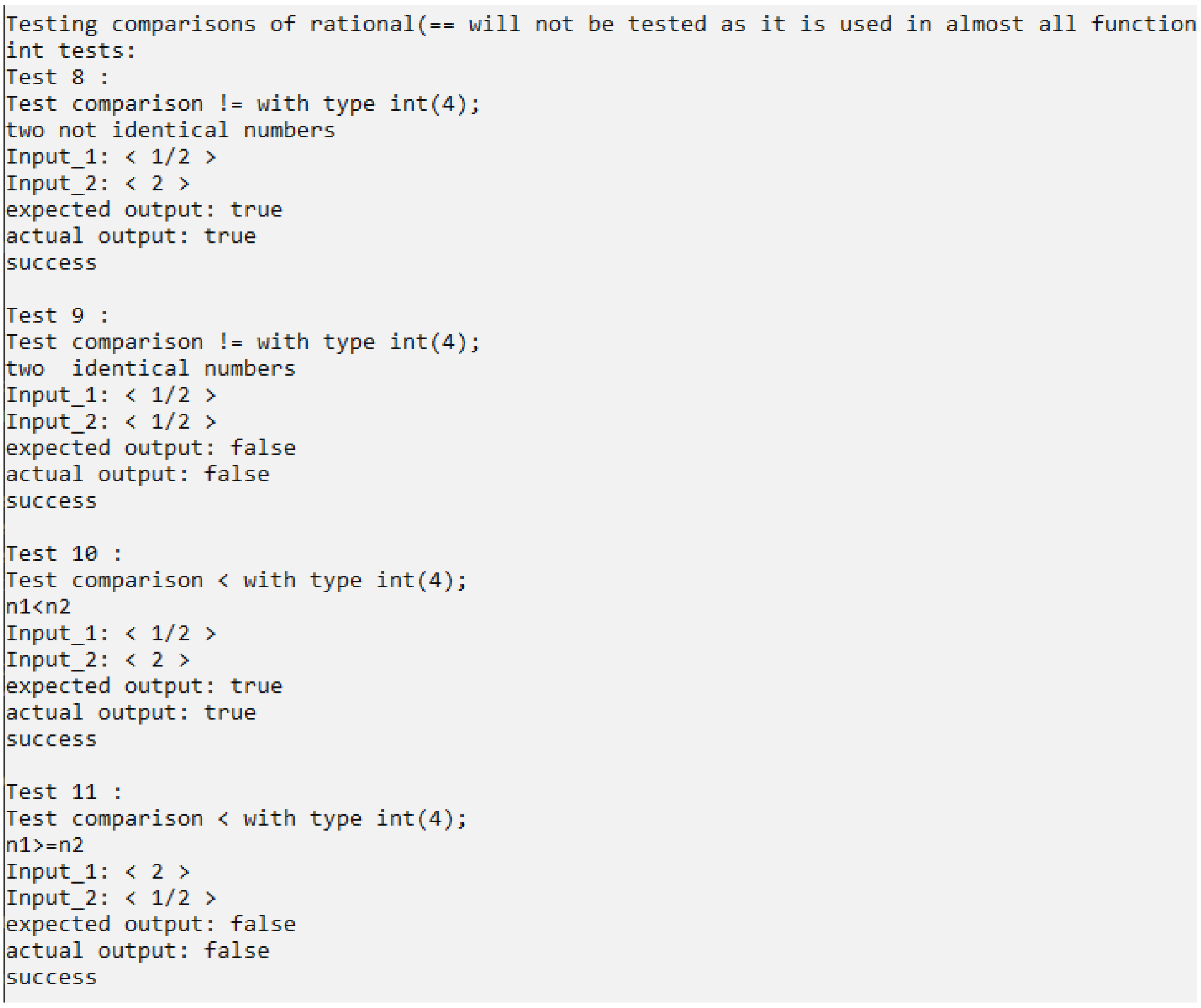
- Test arithmetics rational_t<number_t<int>>:



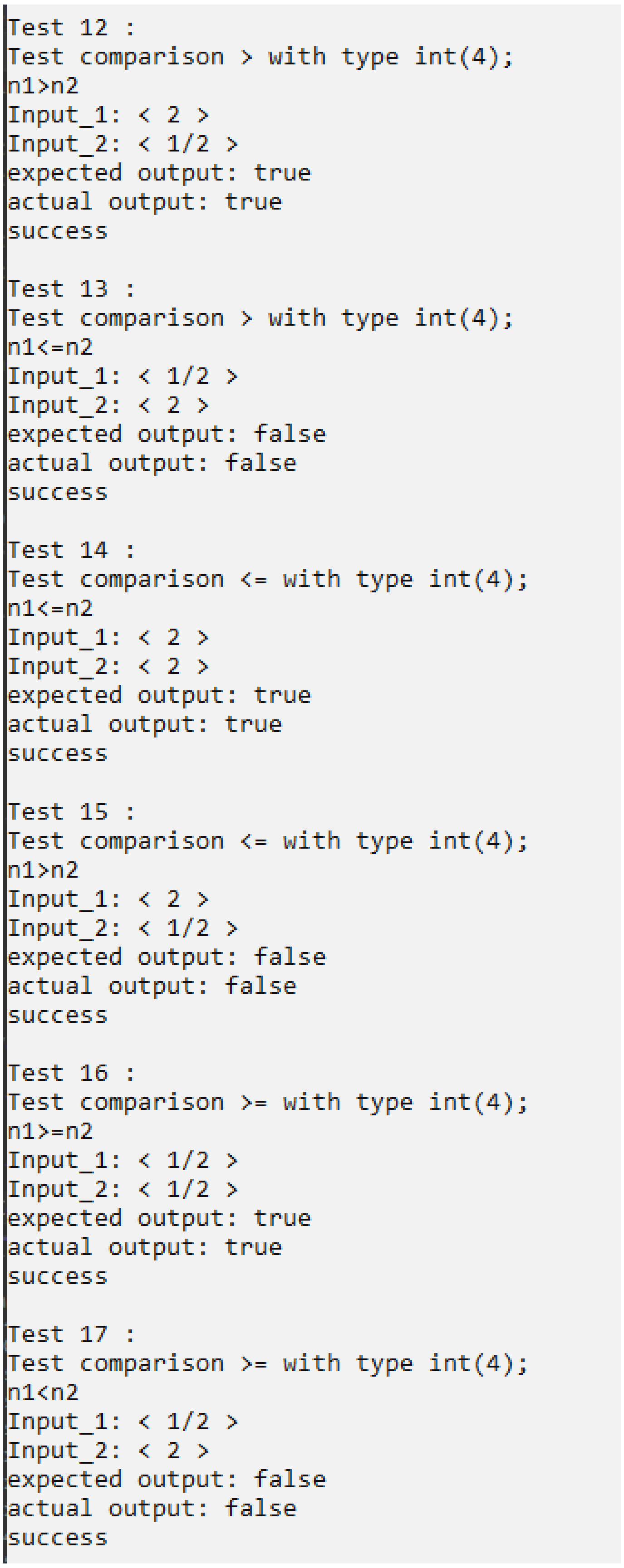
```
number_t<int> tests:
Test 4 :
Test arithmetics of rationals with type number_t<int>(4);
 addition
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < 2 >
actual output: < 2 >
success

Test 5 :
Test arithmetics of rationals with type number_t<int>(4);
 subtraction
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < -1 >
actual output: < -1 >
success

Test 6 :
Test arithmetics of rationals with type number_t<int>(4);
 multiplication
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < 3/4 >
actual output: < 3/4 >
success

Test 7 :
Test arithmetics of rationals with type number_t<int>(4);
 division
Input_1: < 1/2 >
Input_2: < 3/2 >
expected output: < 2/6 >
actual output: < 2/6 >
success
```

generic data-type rational_t comparisons

- Test comparisons rational_t<int>:

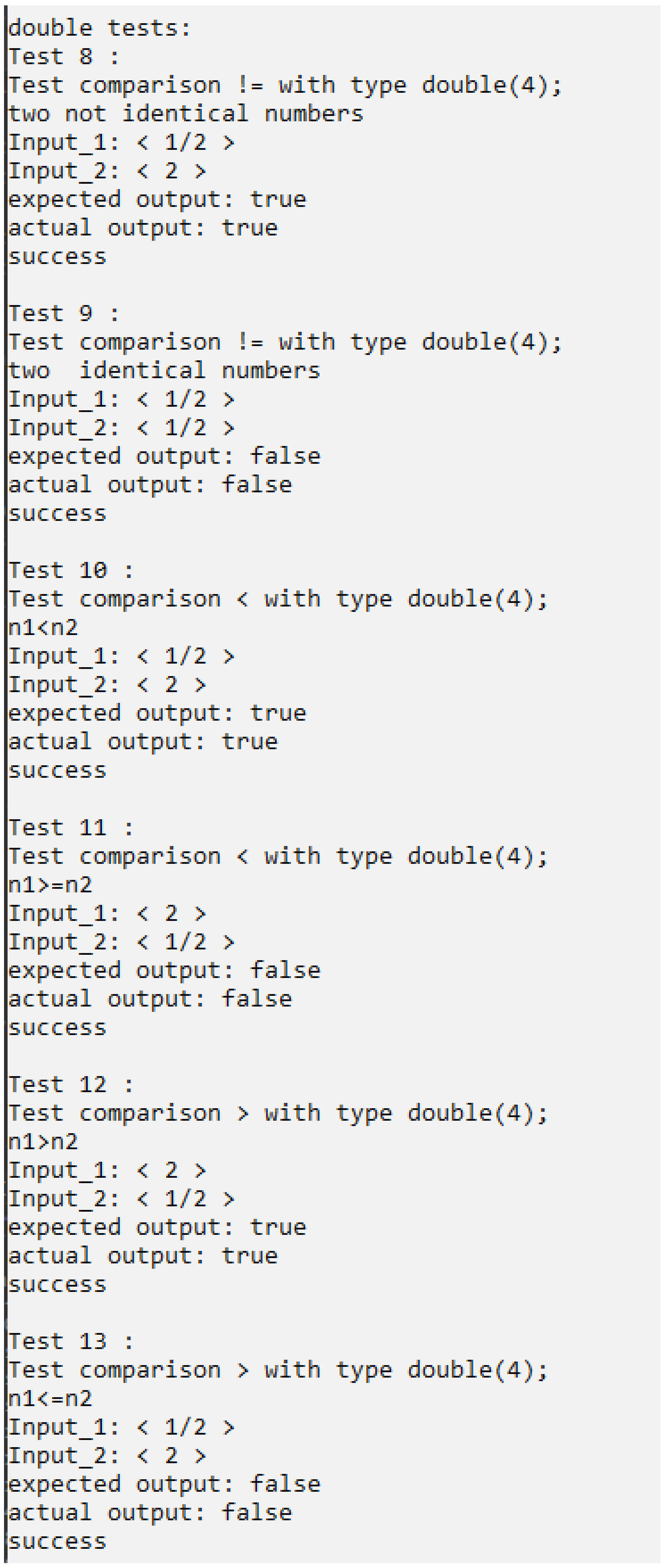


```
Testing comparisons of rational(== will not be tested as it is used in almost all function
int tests:
Test 8 :
Test comparison != with type int(4);
two not identical numbers
Input_1: < 1/2 >
Input_2: < 2 >
expected output: true
actual output: true
success

Test 9 :
Test comparison != with type int(4);
two  identical numbers
Input_1: < 1/2 >
Input_2: < 1/2 >
expected output: false
actual output: false
success

Test 10 :
Test comparison < with type int(4);
n1<n2
Input_1: < 1/2 >
Input_2: < 2 >
expected output: true
actual output: true
success

Test 11 :
Test comparison < with type int(4);
n1>=n2
Input_1: < 2 >
Input_2: < 1/2 >
expected output: false
actual output: false
success
```

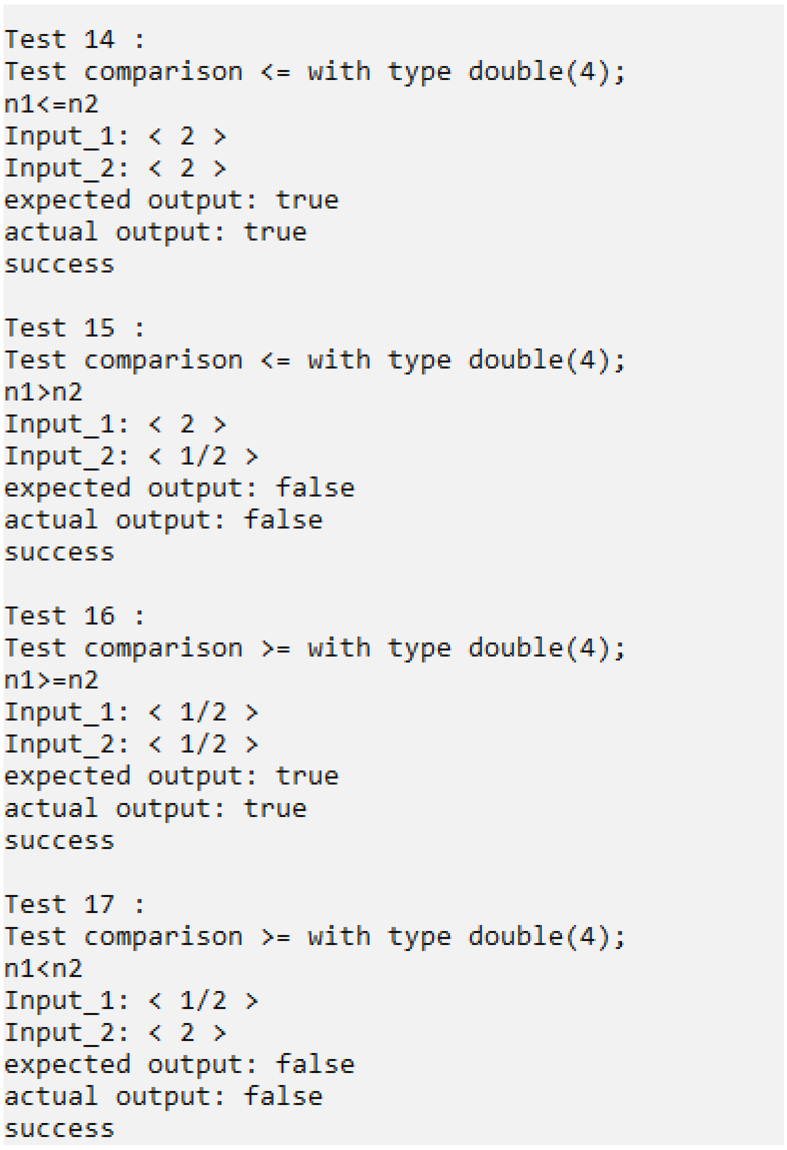


```
Test 12 :
Test comparison > with type int(4);
n1>n2
Input_1: < 2 >
Input_2: < 1/2 >
expected output: true
actual output: true
success

Test 13 :
Test comparison > with type int(4);
n1<=n2
Input_1: < 1/2 >
Input_2: < 2 >
expected output: false
actual output: false
success

Test 14 :
Test comparison <= with type int(4);
n1<=n2
Input_1: < 2 >
Input_2: < 2 >
expected output: true
actual output: true
success

Test 15 :
Test comparison <= with type int(4);
n1>n2
Input_1: < 2 >
Input_2: < 1/2 >
expected output: false
actual output: false
success

Test 16 :
Test comparison >= with type int(4);
n1>=n2
Input_1: < 1/2 >
Input_2: < 1/2 >
expected output: true
actual output: true
success

Test 17 :
Test comparison >= with type int(4);
n1<n2
Input_1: < 1/2 >
Input_2: < 2 >
expected output: false
actual output: false
success
```

- Test comparisons rational_t<double>:



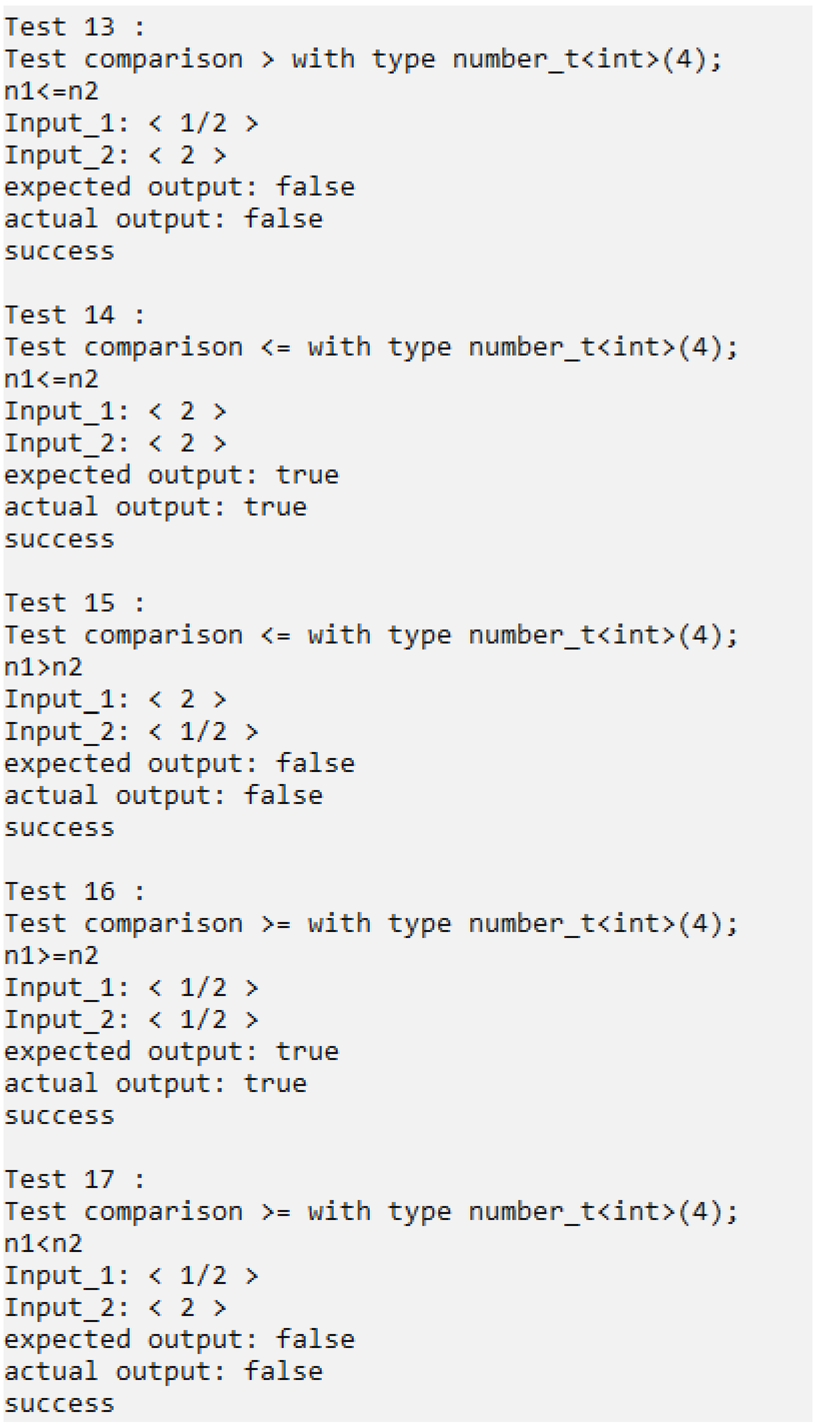
```
double tests:
Test 8 :
Test comparison != with type double(4);
two not identical numbers
Input_1: < 1/2 >
Input_2: < 2 >
expected output: true
actual output: true
success

Test 9 :
Test comparison != with type double(4);
two  identical numbers
Input_1: < 1/2 >
Input_2: < 1/2 >
expected output: false
actual output: false
success

Test 10 :
Test comparison < with type double(4);
n1<n2
Input_1: < 1/2 >
Input_2: < 2 >
expected output: true
actual output: true
success

Test 11 :
Test comparison < with type double(4);
n1>=n2
Input_1: < 2 >
Input_2: < 1/2 >
expected output: false
actual output: false
success

Test 12 :
Test comparison > with type double(4);
n1>n2
Input_1: < 2 >
Input_2: < 1/2 >
expected output: true
actual output: true
success

Test 13 :
Test comparison > with type double(4);
n1<=n2
Input_1: < 1/2 >
Input_2: < 2 >
expected output: false
actual output: false
success
```



```
Test 14 :
Test comparison <= with type double(4);
n1<=n2
Input_1: < 2 >
Input_2: < 2 >
expected output: true
actual output: true
success

Test 15 :
Test comparison <= with type double(4);
n1>n2
Input_1: < 2 >
Input_2: < 1/2 >
expected output: false
actual output: false
success

Test 16 :
Test comparison >= with type double(4);
n1>=n2
Input_1: < 1/2 >
Input_2: < 1/2 >
expected output: true
actual output: true
success

Test 17 :
Test comparison >= with type double(4);
n1<n2
Input_1: < 1/2 >
Input_2: < 2 >
expected output: false
actual output: false
success
```

- Test comparisons rational_t<number_t<int>>:

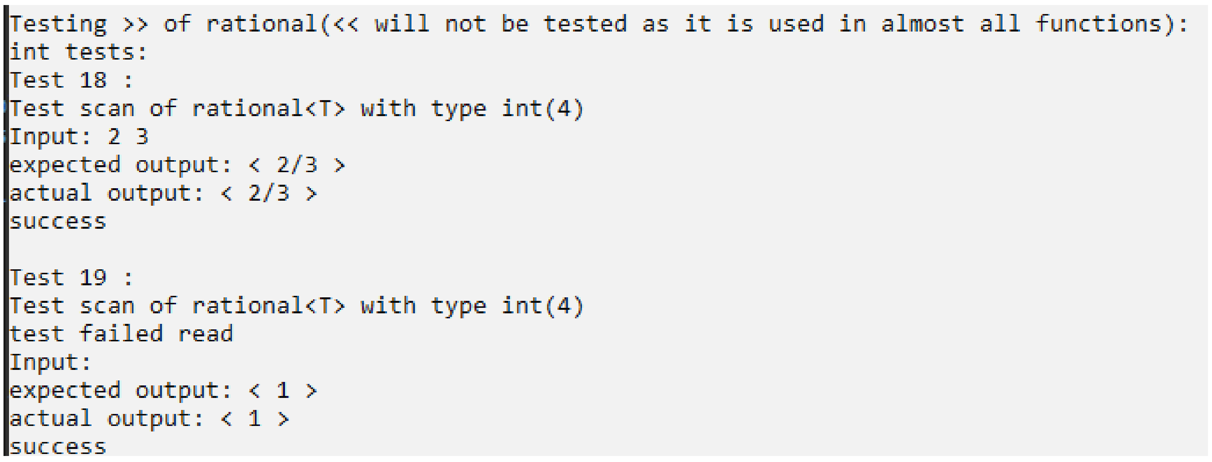
```
number_t<int> tests:
Test 8 :
Test comparison != with type number_t<int>(4);
two not identical numbers
Input_1: < 1/2 >
Input_2: < 2 >
expected output: true
actual output: true
success

Test 9 :
Test comparison != with type number_t<int>(4);
two  identical numbers
Input_1: < 1/2 >
Input_2: < 1/2 >
expected output: false
actual output: false
success

Test 10 :
Test comparison < with type number_t<int>(4);
n1<n2
Input_1: < 1/2 >
Input_2: < 2 >
expected output: true
actual output: true
success

Test 11 :
Test comparison < with type number_t<int>(4);
n1>=n2
Input_1: < 2 >
Input_2: < 1/2 >
expected output: false
actual output: false
success

Test 12 :
Test comparison > with type number_t<int>(4);
n1>n2
Input_1: < 2 >
Input_2: < 1/2 >
expected output: true
actual output: true
success
```



```
Test 13 :
Test comparison > with type number_t<int>(4);
n1<=n2
Input_1: < 1/2 >
Input_2: < 2 >
expected output: false
actual output: false
success

Test 14 :
Test comparison <= with type number_t<int>(4);
n1<=n2
Input_1: < 2 >
Input_2: < 2 >
expected output: true
actual output: true
success

Test 15 :
Test comparison <= with type number_t<int>(4);
n1>n2
Input_1: < 2 >
Input_2: < 1/2 >
expected output: false
actual output: false
success

Test 16 :
Test comparison >= with type number_t<int>(4);
n1>=n2
Input_1: < 1/2 >
Input_2: < 1/2 >
expected output: true
actual output: true
success

Test 17 :
Test comparison >= with type number_t<int>(4);
n1<n2
Input_1: < 1/2 >
Input_2: < 2 >
expected output: false
actual output: false
success
```

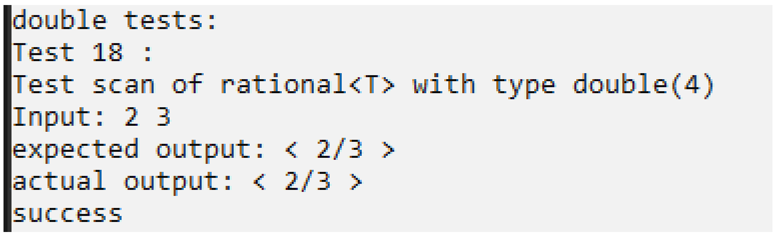
generic data-type rational_t scan tests

- Test scan with type int:



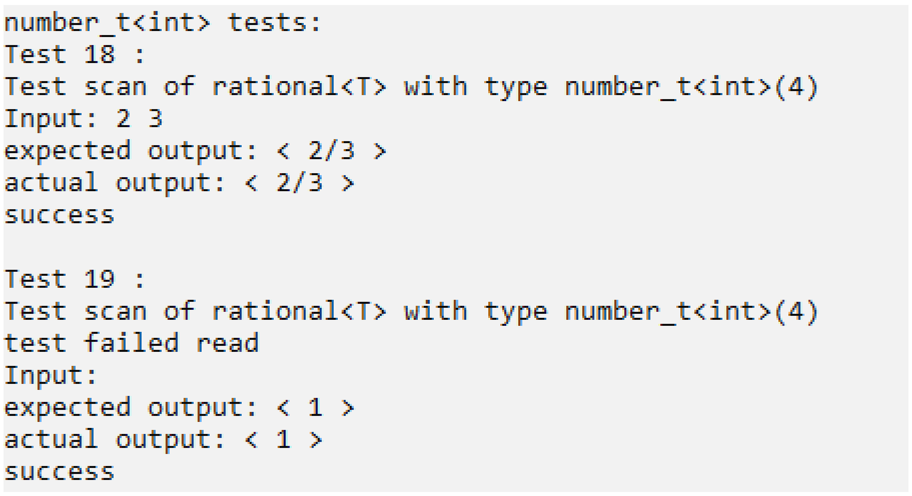
```
Testing >> of rational(<< will not be tested as it is used in almost all functions):
int tests:
Test 18 :
Test scan of rational<T> with type int(4)
Input: 2 3
expected output: < 2/3 >
actual output: < 2/3 >
success

Test 19 :
Test scan of rational<T> with type int(4)
test failed read
Input:
expected output: < 1 >
actual output: < 1 >
success
```

- Test scan with type double:



```
double tests:
Test 18 :
Test scan of rational<T> with type double(4)
Input: 2 3
expected output: < 2/3 >
actual output: < 2/3 >
success
```

- Test scan with type number_t<int>:

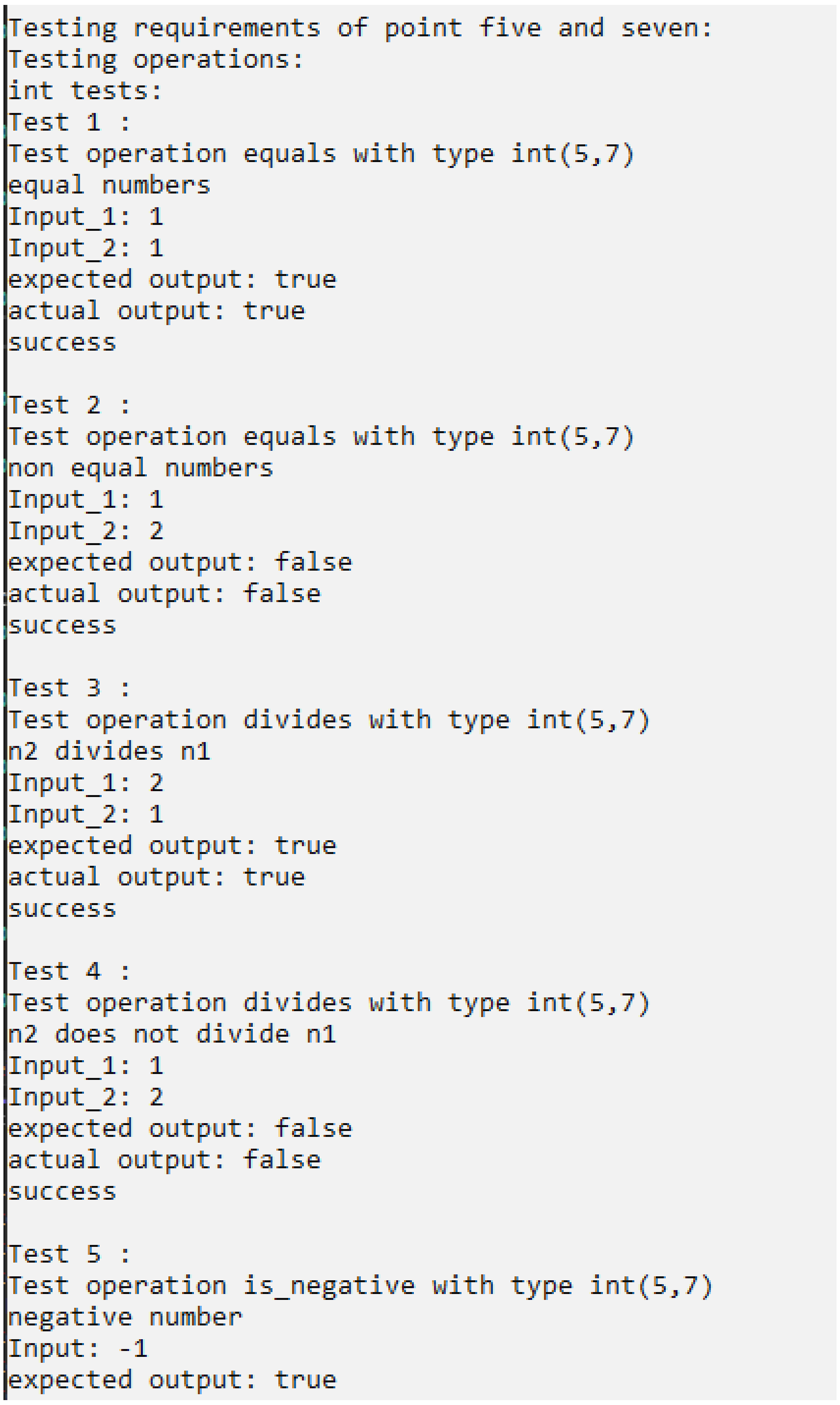


```
number_t<int> tests:
Test 18 :
Test scan of rational<T> with type number_t<int>(4)
Input: 2 3
expected output: < 2/3 >
actual output: < 2/3 >
success

Test 19 :
Test scan of rational<T> with type number_t<int>(4)
test failed read
Input:
expected output: < 1 >
actual output: < 1 >
success
```

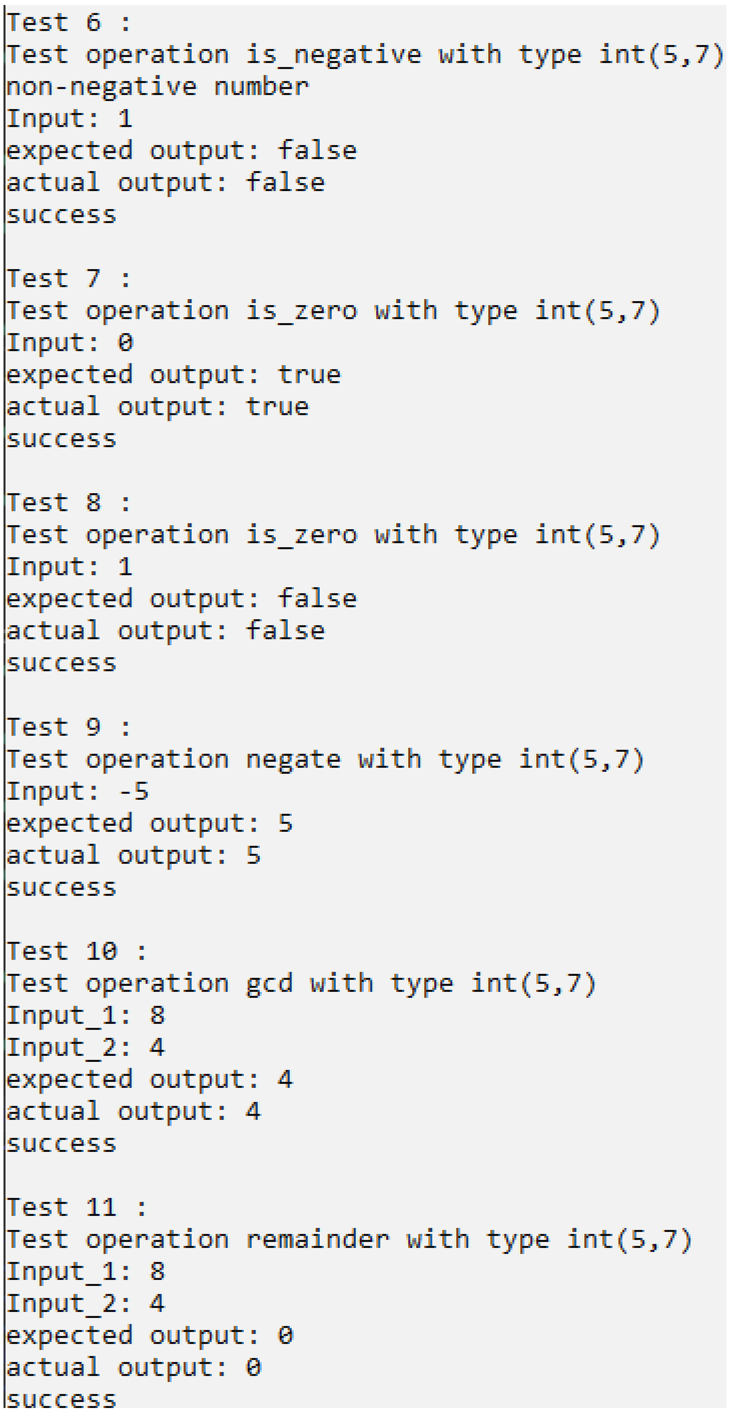
operations

Anmerkung: Die Testcases des *operations* namespaces werden intern für jeden Datentyp extra gezählt.

- Test operations int:



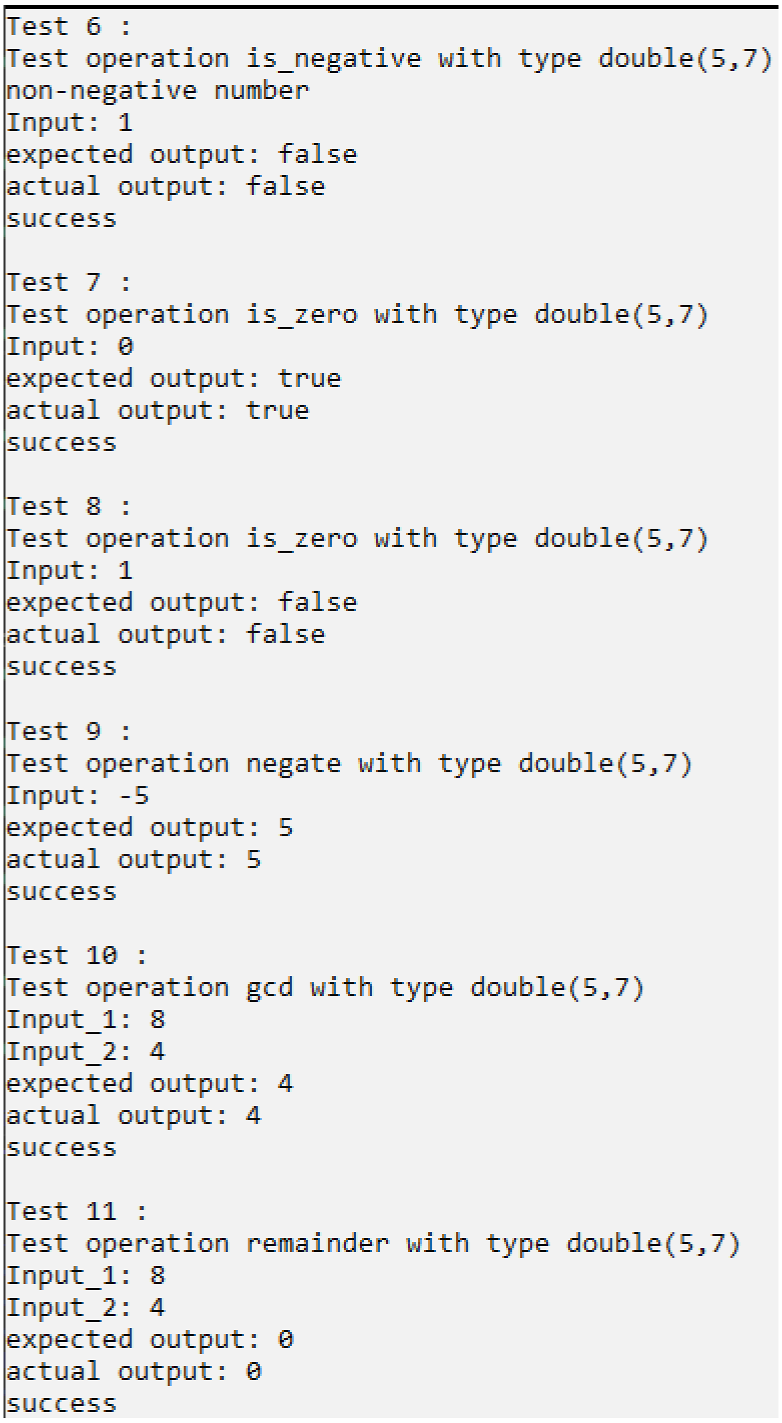
```
Testing requirements of point five and seven:
Testing operations:
int tests:
Test 1 :
Test operation equals with type int(5,7)
equal numbers
Input_1: 1
Input_2: 1
expected output: true
actual output: true
success

Test 2 :
Test operation equals with type int(5,7)
non equal numbers
Input_1: 1
Input_2: 2
expected output: false
actual output: false
success

Test 3 :
Test operation divides with type int(5,7)
n2 divides n1
Input_1: 2
Input_2: 1
expected output: true
actual output: true
success

Test 4 :
Test operation divides with type int(5,7)
n2 does not divide n1
Input_1: 1
Input_2: 2
expected output: false
actual output: false
success

Test 5 :
Test operation is_negative with type int(5,7)
negative number
Input: -1
expected output: true
```



```
Test 6 :
Test operation is_negative with type int(5,7)
non-negative number
Input: 1
expected output: false
actual output: false
success

Test 7 :
Test operation is_zero with type int(5,7)
Input: 0
expected output: true
actual output: true
success

Test 8 :
Test operation is_zero with type int(5,7)
Input: 1
expected output: false
actual output: false
success

Test 9 :
Test operation negate with type int(5,7)
Input: -5
expected output: 5
actual output: 5
success

Test 10 :
Test operation gcd with type int(5,7)
Input_1: 8
Input_2: 4
expected output: 4
actual output: 4
success

Test 11 :
Test operation remainder with type int(5,7)
Input_1: 8
Input_2: 4
expected output: 0
actual output: 0
success
```

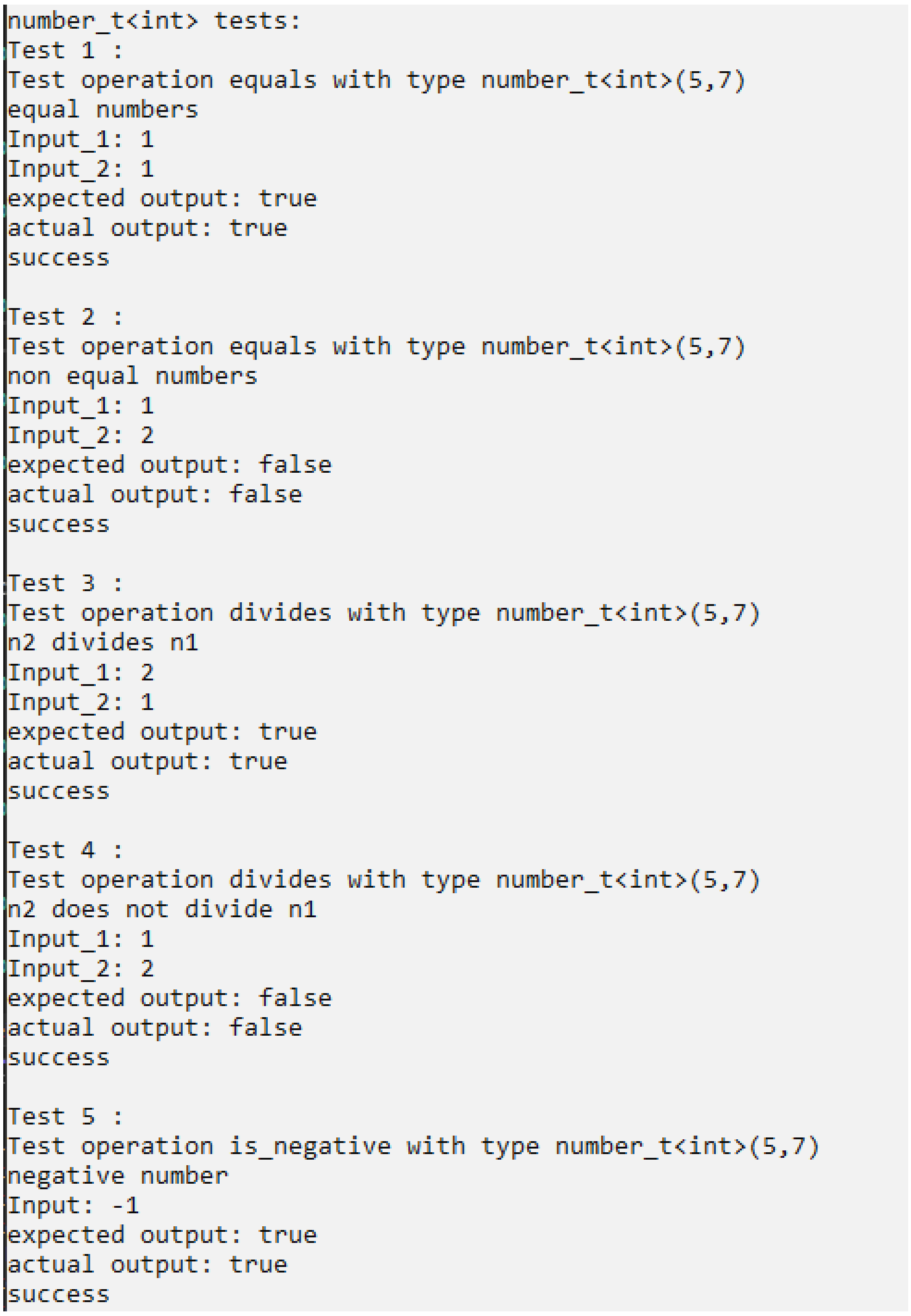
- Test operations double:

```
double tests:
Test 1 :
Test operation equals with type double(5,7)
equal numbers
Input_1: 1
Input_2: 1
expected output: true
actual output: true
success

Test 2 :
Test operation equals with type double(5,7)
non equal numbers
Input_1: 1
Input_2: 2
expected output: false
actual output: false
success

Test 3 :
Test operation divides with type double(5,7)
n2 divides n1
Input_1: 2
Input_2: 1
expected output: true
actual output: true
success

Test 4 :
Test operation divides with type double(5,7)
n2 does not divide n1
Input_1: 1
Input_2: 2
expected output: false
actual output: false
success

Test 5 :
Test operation is_negative with type double(5,7)
negative number
Input: -1
expected output: true
actual output: true
success
```



```
Test 6 :
Test operation is_negative with type double(5,7)
non-negative number
Input: 1
expected output: false
actual output: false
success

Test 7 :
Test operation is_zero with type double(5,7)
Input: 0
expected output: true
actual output: true
success

Test 8 :
Test operation is_zero with type double(5,7)
Input: 1
expected output: false
actual output: false
success

Test 9 :
Test operation negate with type double(5,7)
Input: -5
expected output: 5
actual output: 5
success

Test 10 :
Test operation gcd with type double(5,7)
Input_1: 8
Input_2: 4
expected output: 4
actual output: 4
success

Test 11 :
Test operation remainder with type double(5,7)
Input_1: 8
Input_2: 4
expected output: 0
actual output: 0
success
```

- Test operations<number_t<int>:

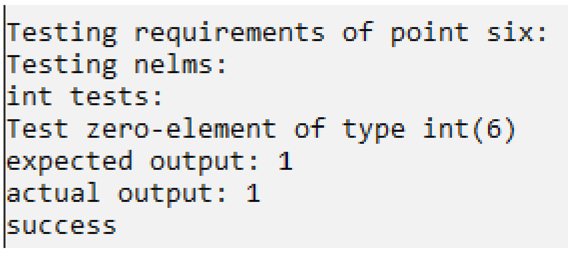


```
number_t<int> tests:
Test 1 :
Test operation equals with type number_t<int>(5,7)
equal numbers
Input_1: 1
Input_2: 1
expected output: true
actual output: true
success

Test 2 :
Test operation equals with type number_t<int>(5,7)
non equal numbers
Input_1: 1
Input_2: 2
expected output: false
actual output: false
success

Test 3 :
Test operation divides with type number_t<int>(5,7)
n2 divides n1
Input_1: 2
Input_2: 1
expected output: true
actual output: true
success

Test 4 :
Test operation divides with type number_t<int>(5,7)
n2 does not divide n1
Input_1: 1
Input_2: 2
expected output: false
actual output: false
success

Test 5 :
Test operation is_negative with type number_t<int>(5,7)
negative number
Input: -1
expected output: true
actual output: true
success
```

```
Test 6 :
Test operation is_negative with type number_t<int>(5,7)
non-negative number
Input: 1
expected output: false
actual output: false
success

Test 7 :
Test operation is_zero with type number_t<int>(5,7)
Input: 0
expected output: true
actual output: true
success

Test 8 :
Test operation is_zero with type number_t<int>(5,7)
Input: 1
expected output: false
actual output: false
success

Test 9 :
Test operation negate with type number_t<int>(5,7)
Input: -5
expected output: 5
actual output: 5
success

Test 10 :
Test operation gcd with type number_t<int>(5,7)
Input_1: 8
Input_2: 4
expected output: 4
actual output: 4
success

Test 11 :
Test operation remainder with type number_t<int>(5,7)
Input_1: 8
Input_2: 4
expected output: 0
actual output: 0
success
```

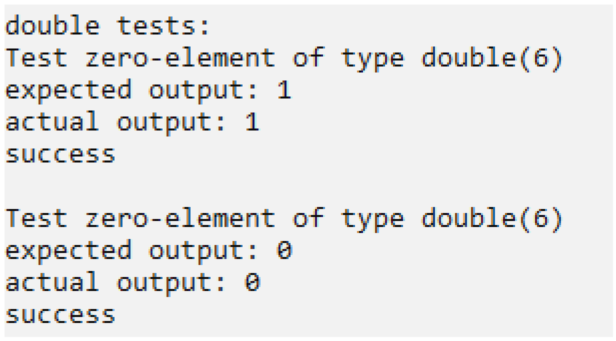
nelms

- Test nelms int:



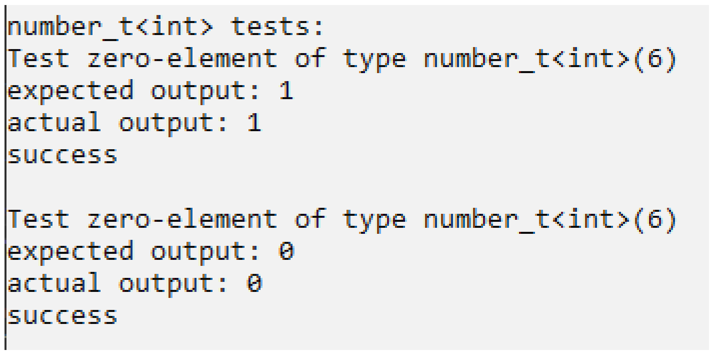
```
Testing requirements of point six:
Testing nelms:
int tests:
Test zero-element of type int(6)
expected output: 1
actual output: 1
success
```

- Test nelms double:



```
double tests:
Test zero-element of type double(6)
expected output: 1
actual output: 1
success

Test zero-element of type double(6)
expected output: 0
actual output: 0
success
```

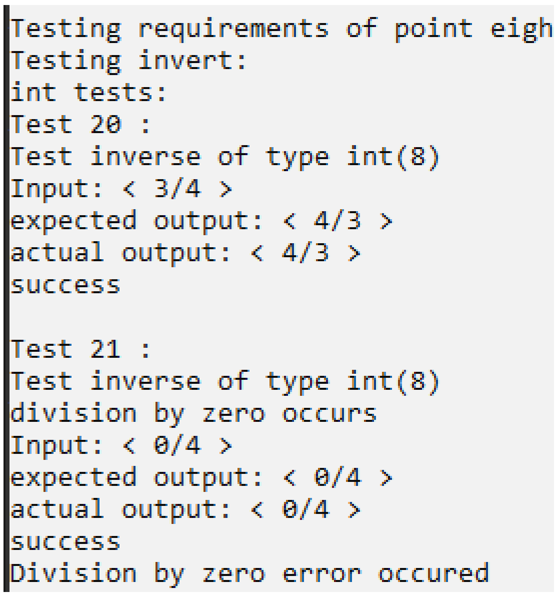
- Test nelms number_t<int>:



```
number_t<int> tests:
Test zero-element of type number_t<int>(6)
expected output: 1
actual output: 1
success

Test zero-element of type number_t<int>(6)
expected output: 0
actual output: 0
success
```

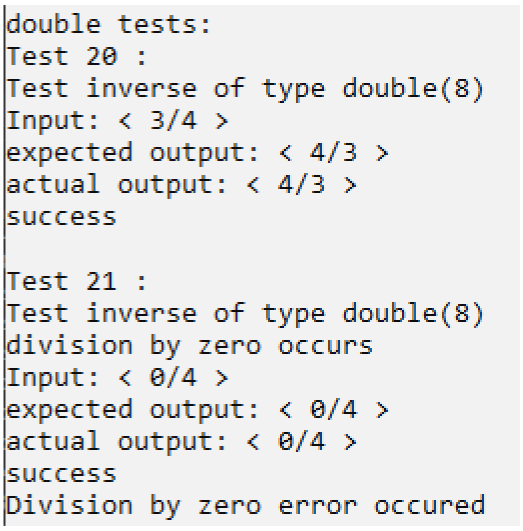
test inverse of rational_t

- Test inverse rational_t<int>:



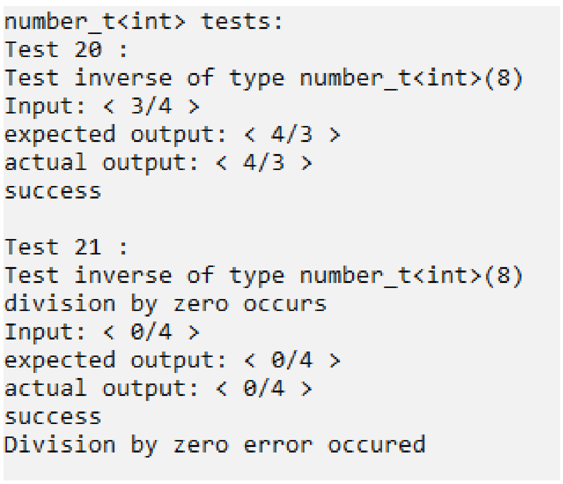
```
Testing requirements of point eigh
Testing invert:
int tests:
Test 20 :
Test inverse of type int(8)
Input: < 3/4 >
expected output: < 4/3 >
actual output: < 4/3 >
success

Test 21 :
Test inverse of type int(8)
division by zero occurs
Input: < 0/4 >
expected output: < 0/4 >
actual output: < 0/4 >
success
Division by zero error occured
```

- Test inverse rational_t<double>:



```
double tests:
Test 20 :
Test inverse of type double(8)
Input: < 3/4 >
expected output: < 4/3 >
actual output: < 4/3 >
success

Test 21 :
Test inverse of type double(8)
division by zero occurs
Input: < 0/4 >
expected output: < 0/4 >
actual output: < 0/4 >
success
Division by zero error occured
```

- Test inverse rational_t<number_t<int>>:



```
number_t<int> tests:
Test 20 :
Test inverse of type number_t<int>(8)
Input: < 3/4 >
expected output: < 4/3 >
actual output: < 4/3 >
success

Test 21 :
Test inverse of type number_t<int>(8)
division by zero occurs
Input: < 0/4 >
expected output: < 0/4 >
actual output: < 0/4 >
success
Division by zero error occured
```