Rationals

0.1

Generated by Doxygen 1.9.5

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Concept Index

2.1 Concepts

Here is a list of all concepts with brief descriptions:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 rational Namespace Reference

Functions

- template<typename T >
  Rational< T > simplify (Rational< T > r)

  Simplify any Rational using std::gcd()
- template<typename T >
  Rational< T > inverse (Rational< T > r)

  Computes the inverse of any Rational.

5.1.1 Function Documentation

5.1.1.1 inverse()

```
template<typename T >
Rational< T > rational::inverse (
            Rational< T > r )
```

Computes the inverse of any Rational.

Returns a new Rational of the same type with the inverse numerator and denominator.

5.1.1.2 simplify()

```
template<typename T >
Rational< T > rational::simplify (
            Rational< T > r )
```

Simplify any Rational using std::gcd()

If the gcd of the numerator and denominator is different than 1, returns a new Rational which values are the numerator and denominator of the given Rational divided by their gcd. Otherwise returns the given Rational.

Chapter 6

Concept Documentation

6.1 Integer Concept Reference

concept which allows to filter types for the rationals, allowing only integer types

`#include <Rationals.hpp>`

6.1.1 Concept definition

```
template<typename T>
concept Integer =  std::integral<T>
```

6.1.2 Detailed Description

concept which allows to filter types for the rationals, allowing only integer types

Chapter 7

Data Structure Documentation

7.1 Rational< T > Class Template Reference

Class of rationals using int types.

`#include <Rationals.hpp>`

Public Member Functions

- Rational (T num=0, T denom=1)
  Constructor.
- T & num ()
  Getter of the numerator.
- T num () const
  Getter of the numerator.
- T & denom ()
  Getter of the denominator.
- T denom () const
  Getter of the denominator.
- bool operator== (Rational const &r) const
  operator ==
- bool operator== (T n) const
  operator ==
- bool operator< (Rational const &r) const
  operator <
- bool operator< (T n) const
  operator <
- bool operator> (Rational const &r) const
  operator >
- bool operator> (T n) const
  operator >
- bool operator<= (Rational const &r) const
  operator <=
- bool operator<= (T n) const
  operator <=

- bool operator>= (Rational const &r) const

  operator >=
- bool operator>= (T n) const

  operator >=
- bool operator!= (Rational const &r) const

  operator !=
- bool operator!= (T n) const

  operator !=
- Rational operator= (Rational const &r)

  operator =
- Rational operator= (T n)

  operator =
- Rational operator+ (Rational const &r) const

  operator +
- Rational operator+ (T n) const

  operator +
- Rational operator- (Rational const &r) const

  operator - (binary)
- Rational operator- (T n) const

  operator - (binary)
- Rational operator- () const

  operator - (unary)
- Rational operator* (Rational const &r) const

  *operator **
- Rational operator* (T n) const

  *operator **
- Rational operator/ (Rational const &r) const

  operator /
- Rational operator/ (T n) const

  operator /

7.1.1 Detailed Description

template<typename T>
requires Integer<T>
class Rational< T >

Class of rationals using int types.

Template Parameters

T	can be any int types from the std : int8_t, int16_t, int32_t, int64_t, uint8_t, uint16_t, uint32_t, uint64_t

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Rational()

```
template<typename T >
Rational< T >::Rational (
            T num = 0,
            T denom = 1 )
```

Constructor.

Constructor from two integers used as numerator and denominator. Defaults values are 0 for the numerator and 1 for the denominator.

Parameters

num	: Numerator
denom	: Denominator

7.1.3 Member Function Documentation

7.1.3.1 denom() [1/2]

```
template<typename T >
T & Rational< T >::denom
```

Getter of the denominator.

Returns a reference to denominator.

7.1.3.2 denom() [2/2]

```
template<typename T >
T Rational< T >::denom
```

Getter of the denominator.

Returns a copy of the denominator.

7.1.3.3 num() [1/2]

```
template<typename T >
T & Rational< T >::num
```

Getter of the numerator.

Returns a reference to numerator.

7.1.3.4 num() [2/2]

```
template<typename T >
T Rational< T >::num
```

Getter of the numerator.

Returns a copy of the numerator.

7.1.3.5 operator"!=() [1/2]

```
template<typename T >
bool Rational< T >::operator!= (
            Rational< T > const & r ) const
```

operator !=

Negates the == operator

7.1.3.6 operator"!=() [2/2]

```
template<typename T >
bool Rational< T >::operator!= (
            T n ) const
```

operator !=

Negates the == operator

7.1.3.7 operator∗() [1/2]

```
template<typename T >
Rational< T > Rational< T >::operator* (
            Rational< T > const & r ) const
```

operator ∗

Returns the product (simplified).

7.1.3.8 operator∗() [2/2]

```
template<typename T >
Rational< T > Rational< T >::operator* (
            T n ) const
```

operator ∗

Returns the product (simplified).

7.1.3.9 operator+() [1/2]

```
template<typename T >
Rational< T > Rational< T >::operator+ (
            Rational< T > const & r ) const
```

operator +

Return the sum (simplified).

7.1.3.10 operator+() [2/2]

```
template<typename T >
Rational< T > Rational< T >::operator+ (
            T n ) const
```

operator +

Return the sum (simplified).

7.1.3.11 operator-() [1/3]

```
template<typename T >
Rational< T > Rational< T >::operator-
```

operator - (unary)

Negates the numerator.

7.1.3.12 operator-() [2/3]

```
template<typename T >
Rational< T > Rational< T >::operator- (
            Rational< T > const & r ) const
```

operator - (binary)

Returns the difference (simplified).

7.1.3.13 operator-() [3/3]

```
template<typename T >
Rational< T > Rational< T >::operator- (
            T n ) const
```

operator - (binary)

Returns the difference (simplified).

7.1.3.14 operator/() [1/2]

```
template<typename T >
Rational< T > Rational< T >::operator/ (
            Rational< T > const & r ) const
```

operator /

Returns the division (simplified). Multiplies the inverse.

7.1.3.15 operator/() [2/2]

```
template<typename T >
Rational< T > Rational< T >::operator/ (
            T n ) const
```

operator /

Returns the division (simplified). Multiplies the inverse.

7.1.3.16 operator<() [1/2]

```
template<typename T >
bool Rational< T >::operator< (
            Rational< T > const & r ) const
```

operator <

Compares the numerators once the rationals are set on the same denominator.

7.1.3.17 operator<() [2/2]

```
template<typename T >
bool Rational< T >::operator< (
            T n ) const
```

operator <

Compares the numerator with n multiplied by the denominator.

7.1.3.18 operator<=() [1/2]

```
template<typename T >
bool Rational< T >::operator<= (
            Rational< T > const & r ) const
```

operator <=

Use the previously defined operators

7.1.3.19 operator<=() [2/2]

```
template<typename T >
bool Rational< T >::operator<= (
            T n ) const
```

operator <=

Use the previously defined operators

7.1.3.20 operator=() [1/2]

```
template<typename T >
Rational< T > Rational< T >::operator= (
            Rational< T > const & r )
```

operator =

Assigns field by field and returns also the parameter.

7.1.3.21 operator=() [2/2]

```
template<typename T >
Rational< T > Rational< T >::operator= (
            T n )
```

operator =

Assigns the given number to the numerator and 1 to the denominator.

7.1.3.22 operator==() [1/2]

```
template<typename T >
bool Rational< T >::operator== (
            Rational< T > const & r ) const
```

operator ==

Compares field by field.

7.1.3.23 operator==() [2/2]

```
template<typename T >
bool Rational< T >::operator== (
            T n ) const
```

operator ==

Compares if denominator is 1, then compares the numerator with the paramater.

7.1.3.24 operator>() [1/2]

```
template<typename T >
bool Rational< T >::operator> (
            Rational< T > const & r ) const
```

operator >

Compares the numerators once the rationals are set on the same denominator.

7.1.3.25 operator>() [2/2]

```
template<typename T >
bool Rational< T >::operator> (
            T n ) const
```

operator >

Compares the numerator with n multiplied by the denominator.

7.1.3.26 operator>=() [1/2]

```
template<typename T >
bool Rational< T >::operator>= (
            Rational< T > const & r ) const
```

operator >=

Use the previously defined operators

7.1.3.27 operator>=() [2/2]

```
template<typename T >
bool Rational< T >::operator>= (
            T n ) const
```

operator >=

Use the previously defined operators

The documentation for this class was generated from the following file:

- include/rationals/Rationals.hpp

Chapter 8

File Documentation

8.1 include/rationals/Rationals.hpp File Reference

```
#include <concepts>
#include <string>
#include <numeric>
#include <iostream>
#include <math.h>
```

Data Structures

- class Rational< T >

  Class of rationals using int types.

Namespaces

- namespace rational

Concepts

- concept Integer

  concept which allows to filter types for the rationals, allowing only integer types

Functions

- template<typename T >
  Rational< T > rational::simplify (Rational< T > r)

  Simplify any Rational using std::gcd()
- template<typename T >
  Rational< T > rational::inverse (Rational< T > r)

  Computes the inverse of any Rational.
- template<typename T >
  std::ostream & operator<< (std::ostream &stream, const Rational< T > &r)

- template<typename T >
  Rational< T > operator/ (const T x, const Rational< T > &r)

  Division operator when the Rational is the second operand.
- template<typename T >
  Rational< T > operator* (const T x, const Rational< T > &r)

  Multiplication operator when the Rational is the second operand.
- template<typename T >
  Rational< T > operator+ (const T x, const Rational< T > &r)
- template<typename T >
  Rational< T > operator- (const T x, const Rational< T > &r)
- template<typename T >
  Rational< T > operator== (const T x, const Rational< T > &r)
- template<typename T >
  Rational< T > operator!= (const T x, const Rational< T > &r)
- template<typename T >
  Rational< T > operator< (const T x, const Rational< T > &r)
- template<typename T >
  Rational< T > operator> (const T x, const Rational< T > &r)
- template<typename T >
  Rational< T > operator<= (const T x, const Rational< T > &r)
- template<typename T >
  Rational< T > operator>= (const T x, const Rational< T > &r)

8.1.1 Function Documentation

8.1.1.1 operator"!=()

```
template<typename T >
Rational< T > operator!= (
            const T x,
            const Rational< T > & r )
```

Reverses the operands to use the member operator.

8.1.1.2 operator*()

```
template<typename T >
Rational< T > operator* (
            const T x,
            const Rational< T > & r )
```

Multiplication operator when the Rational is the second operand.

8.1.1.3 operator+()

```
template<typename T >
Rational< T > operator+ (
            const T x,
            const Rational< T > & r )
```

Plus operator when the Rational is the second operand.

8.1.1.4 operator-()

```
template<typename T >
Rational< T > operator- (
            const T x,
            const Rational< T > & r )
```

Minus operator when the Rational is the second operand.

8.1.1.5 operator/()

```
template<typename T >
Rational< T > operator/ (
            const T x,
            const Rational< T > & r )
```

Division operator when the Rational is the second operand.

8.1.1.6 operator<()

```
template<typename T >
Rational< T > operator< (
            const T x,
            const Rational< T > & r )
```

Does the reverse comparison to use the member operator.

8.1.1.7 operator<<()

```
template<typename T >
std::ostream & operator<< (
            std::ostream & stream,
            const Rational< T > & r )
```

Send to stream operator.

8.1.1.8 operator<=()

```
template<typename T >
Rational< T > operator<= (
            const T x,
            const Rational< T > & r )
```

Does the reverse comparison to use the member operator.

8.1.1.9 operator==()

```
template<typename T >
Rational< T > operator== (
            const T x,
            const Rational< T > & r )
```

Reverses the operands to use the member operator.

8.1.1.10 operator>()

```
template<typename T >
Rational< T > operator> (
            const T x,
            const Rational< T > & r )
```

Does the reverse comparison to use the member operator.

8.1.1.11 operator>=()

```
template<typename T >
Rational< T > operator>= (
            const T x,
            const Rational< T > & r )
```

Does the reverse comparison to use the member operator.

8.2 Rationals.hpp

Go to the documentation of this file.

```
1 #ifndef RATIONALS_HPP
2 #define RATIONALS_HPP
3
4 #include <concepts>
5 #include <string>
6 #include <numeric>
7 #include <iostream>
8 #include <math.h>
9
11 template <typename T>
12 concept Integer = std::integral<T>;
13
14 /*-------------------------------------------*/
15 /*             Class definition              */
16
20 template <typename T>
21 requires Integer<T>
```

```
22 class Rational{
23 
28     private:
29 
31         T numerator;
33         T denominator;
34 
35     public:
36 
38         Rational(T num=0, T denom=1);
39 
41         T& num();
43         T num() const;
45         T& denom();
47         T denom() const;
48 
50         bool operator ==(Rational const & r) const;
52         bool operator ==(T n) const;
53 
55         bool operator <(Rational const & r) const;
57         bool operator <(T n) const;
58 
60         bool operator >(Rational const & r) const;
62         bool operator >(T n) const;
63 
65         bool operator <=(Rational const & r) const;
67         bool operator <=(T n) const;
68 
70         bool operator >=(Rational const & r) const;
72         bool operator >=(T n) const;
73 
75         bool operator !=(Rational const & r) const;
77         bool operator !=(T n) const;
78 
79 
81         Rational operator = (Rational const & r);
83         Rational operator = (T n);
84 
86         Rational operator + (Rational const & r) const;
88         Rational operator + (T n) const;
89 
91         Rational operator - (Rational const & r) const;
93         Rational operator - (T n) const;
94 
96         Rational operator - () const;
97 
99         Rational operator * (Rational const & r) const;
101         Rational operator * (T n) const;
102 
104         Rational operator / (Rational const & r) const;
106         Rational operator / (T n) const;
107 
108 };
109 
110 /*-------------------------------------------*/
111 /*  Rational namespace for non-member functions */
112 
113 namespace rational{
114 
116 
119     template <typename T>
120     Rational<T> simplify(Rational<T> r){
121         int pgcd = std::gcd(r.num(), r.denom());
122         if (pgcd != 1){
123             return Rational<T>(r.num()/pgcd, r.denom()/pgcd);
124         }
125         return r;
126     }
127 
129 
132     template <typename T>
133     Rational<T> inverse(Rational<T> r){
134         return Rational<T>(r.denom(), r.num());
135     }
136 
140     template<typename T, typename U>
141     static Rational<T> decimalToRational(U f){
142 
143         /*  a: continued fraction coefficients. */
144         T num, denom;
145         int a, h[3] = { 0, 1, 0 }, k[3] = { 1, 0, 0 };
146         int x, d, n = 1;
147         int i, neg = 0;
148         int md = 10000;
149 
150         if (md <= 1) {
151             denom = 1; num = (int) f;
```

```
152         } else {
153             if (f < 0) { neg = 1; f = -f; }
154 
155             while (f != floor(f)) { n <<= 1; f *= 2; }
156             d = f;
157 
158             /* continued fraction and check denominator each step */
159             for (i = 0; i < 64; i++) {
160                 a = n ? d / n : 0;
161                 if (i && !a) break;
162 
163                 x = d; d = n; n = x % n;
164 
165                 x = a;
166                 if (k[1] * a + k[0] >= md) {
167                     x = (md - k[0]) / k[1];
168                     if (x * 2 >= a || k[1] >= md)
169                         i = 65;
170                     else
171                         break;
172                 }
173 
174                 h[2] = x * h[1] + h[0]; h[0] = h[1]; h[1] = h[2];
175                 k[2] = x * k[1] + k[0]; k[0] = k[1]; k[1] = k[2];
176             }
177             denom = k[1];
178             num = neg ? -h[1] : h[1];
179         }
180 
181 
182         return Rational<T>(num, denom);
183     }
184 
186     template <typename T>
187     static const Rational<T> inf = Rational<T>(1, 0);
188 
189 }
190 
191 
192 
193 
194 /*-----------------------------------------------*/
195 /*                Constructor                    */
196 
202 template<typename T>
203 Rational<T>::Rational(T num, T denom){
204     if(denom < 0){
205         denom *= -1;
206         num *= -1;
207     }
208     if (denom == 0){
209         num = 1;
210     }
211     this->numerator = num;
212     this->denominator = denom;
213 }
214 
215 
216 
217 
218 
219 /*-----------------------------------------------*/
220 /*            Getters and Setters                */
221 
225 template<typename T>
226 T& Rational<T>::num(){
227     return this->numerator;
228 }
229 
233 template<typename T>
234 T  Rational<T>::num() const{
235     return this->numerator;
236 }
237 
241 template<typename T>
242 T& Rational<T>::denom(){
243     return this->denominator;
244 }
245 
249 template<typename T>
250 T  Rational<T>::denom() const{
251     return this->denominator;
252 }
253 
254 
255 
256 
```

```
257
258 /*-----------------------------------------------*/
259 /*             Operators (comparison)            */
260
264 template<typename T>
265 bool Rational<T>::operator ==(Rational<T> const & r) const{
266     return this->numerator == r.numerator && this->denominator == r.denominator;
267 }
268
272 template<typename T>
273 bool Rational<T>::operator ==(T n) const{
274     return this->denom() == 1 && this.num() == n;
275 }
276
280 template<typename T>
281 bool Rational<T>::operator <(Rational const & r) const{
282     return this.num()*r.denom() < r.num()*this->denom();
283 }
284
288 template<typename T>
289 bool Rational<T>::operator <(T n) const{
290     return this.num() < n*this->denom();
291 }
292
296 template<typename T>
297 bool Rational<T>::operator >(Rational const & r) const{
298     return this.num()*r.denom() > r.num()*this->denom();
299 }
300
304 template<typename T>
305 bool Rational<T>::operator >(T n) const{
306     return this.num() > n*this->denom();
307 }
308
312 template<typename T>
313 bool Rational<T>::operator <=(Rational const & r) const{
314     return *(this) < r || *(this) == r;
315 }
316
320 template<typename T>
321 bool Rational<T>::operator <=(T n) const{
322     return *(this) < n || *(this) == n;
323 }
324
328 template<typename T>
329 bool Rational<T>::operator >=(Rational const & r) const{
330     return *(this) > r || *(this) == r;
331 }
332
336 template<typename T>
337 bool Rational<T>::operator >=(T n) const{
338     return *(this) > n || *(this) == n;
339 }
340
344 template<typename T>
345 bool Rational<T>::operator !=(Rational const & r) const{
346     return ! (*(this)==r);
347 }
348
352 template<typename T>
353 bool Rational<T>::operator !=(T n) const{
354     return ! (*(this) == n);
355 }
356
357
358 /*-----------------------------------------------*/
359 /*   Operators (mathematical and assignative)    */
360
364 template<typename T>
365 Rational<T> Rational<T>::operator =(Rational<T> const & r){
366     this->numerator = r.numerator;
367     this->denominator = r.denominator;
368     return r;
369 }
370
374 template<typename T>
375 Rational<T> Rational<T>::operator =(T n){
376     this->numerator = n;
377     this->denominator = (T)1;
378     return *(this);
379 }
380
384 template<typename T>
385 Rational<T> Rational<T>::operator +(Rational<T> const & r) const{
386     if (this->denom() == 0 || r.denom() == 0){
387         return rational::inf<T>;
388     }
```

```
389     return rational::simplify(Rational<T>(this->numerator*r.denominator+r.numerator*this->denominator,
        this->denominator*r.denominator));
390 }
391
395 template<typename T>
396 Rational<T> Rational<T>::operator +(T n) const{
397     if (this->denom() == 0){
398         return rational::inf<T>;
399     }
400     Rational<T> r(n, 1);
401     return rational::simplify(Rational<T>(this->numerator*r.denominator+r.numerator*this->denominator,
        this->denominator*r.denominator));
402 }
403
407 template<typename T>
408 Rational<T> Rational<T>::operator -(Rational<T> const & r) const{
409     if (this->denom() == 0 || r.denom() == 0){
410         return rational::inf<T>;
411     }
412     return rational::simplify(Rational<T>(this->numerator*r.denominator-r.numerator*this->denominator,
        this->denominator*r.denominator));
413 }
414
418 template<typename T>
419 Rational<T> Rational<T>::operator -(T n) const{
420     if (this->denom() == 0){
421         return rational::inf<T>;
422     }
423     Rational<T> r(n, 1);
424     return rational::simplify(Rational<T>(this->numerator*r.denominator-r.numerator*this->denominator,
        this->denominator*r.denominator));
425 }
426
430 template<typename T>
431 Rational<T> Rational<T>::operator *(Rational<T> const & r) const{
432     if (this->denom() == 0 || r.denom() == 0){
433         return rational::inf<T>;
434     }
435     if (this->num() == 0 || r.num() == 0){
436         return Rational<T>(1,1);
437     }
438     return rational::simplify(Rational<T>(this->numerator*r.numerator,
        this->denominator*r.denominator));
439 }
440
444 template<typename T>
445 Rational<T> Rational<T>::operator *(T n) const{
446     if (this->denom() == 0 && n){
447         return rational::inf<T>;
448     }
449     Rational<T> r(n, 1);
450     return rational::simplify(Rational<T>(this->numerator*r.denominator+r.numerator*this->denominator,
        this->denominator*r.denominator));
451 }
452
456 template<typename T>
457 Rational<T> Rational<T>::operator /(Rational<T> const & r) const{
458     return (*this)*rational::inverse(r);
459 }
460
464 template<typename T>
465 Rational<T> Rational<T>::operator /(T n) const{
466     if (this->denom() == 0){
467         return rational::inf<T>;
468     }
469     Rational<T> r(n, 1);
470     return (*this)*rational::inverse(r);
471 }
472
476 template<typename T>
477 Rational<T> Rational<T>::operator -() const{
478     return Rational<T>(-this->numerator, this->denominator);
479 }
480
481
482
483
484
485 /*-------------------------------------------*/
486 /*            "Outside" Operators            */
487
491 template<typename T>
492 std::ostream& operator « (std::ostream& stream, const Rational<T> & r){
493     if (r.denom() == 0) {
494         stream « "inf";
495     } else {
496         stream « r.num() « "/" « r.denom();
```

```
497     }
498     return stream;
499 }
500 
501 
502 /*   Operators (mathematical)   */
503 
505 template<typename T>
506 Rational<T> operator / (const T x, const Rational<T> & r){
507     return rational::simplify(Rational<T>(r.denom()*x, r.num()));
508 }
509 
511 template<typename T>
512 Rational<T> operator * (const T x, const Rational<T> & r){
513     return rational::simplify(Rational<T>(r.num()*x, r.denom()));
514 }
515 
519 template<typename T>
520 Rational<T> operator + (const T x, const Rational<T> & r){
521     return rational::simplify(Rational<T>(x, 1)+r);
522 }
523 
527 template<typename T>
528 Rational<T> operator - (const T x, const Rational<T> & r){
529     return rational::simplify(Rational<T>(x, 1)-r);
530 }
531 
532 
533 /*   Operators (comparison)   */
534 
538 template<typename T>
539 Rational<T> operator == (const T x, const Rational<T> & r){
540     return r == x;
541 }
542 
546 template<typename T>
547 Rational<T> operator != (const T x, const Rational<T> & r){
548     return r != x;
549 }
550 
554 template<typename T>
555 Rational<T> operator < (const T x, const Rational<T> & r){
556     return r > x;
557 }
558 
562 template<typename T>
563 Rational<T> operator > (const T x, const Rational<T> & r){
564     return r < x;
565 }
566 
570 template<typename T>
571 Rational<T> operator <= (const T x, const Rational<T> & r){
572     return r >= x;
573 }
574 
578 template<typename T>
579 Rational<T> operator >= (const T x, const Rational<T> & r){
580     return r <= x;
581 }
582 
583 #endif
```

Index