Aerobus

Generated by Doxygen 1.9.4

Chapter 1

Concept Index

1.1 Concepts

Here is a list of all documented concepts with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Concept Documentation

4.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

`#include <lib.h>`

4.1.1 Concept definition

```
template<typename R>
concept aerobus::IsEuclideanDomain =  IsRing<R> && requires {
        typename R::template div_t<typename R::one, typename R::one>;
        typename R::template mod_t<typename R::one, typename R::one>;
        typename R::template gcd_t<typename R::one, typename R::one>;
        R::template pos_v<typename R::one> == true;
        R::template gt_v<typename R::one, typename R::zero> == true;
        R::is_euclidean_domain == true;
    }
```

4.1.2 Detailed Description

Concept to express R is an euclidean domain.

4.2 aerobus::IsField Concept Reference

Concept to express R is a field.

`#include <lib.h>`

4.2.1 Concept definition

```
template<typename R>
concept aerobus::IsField =  IsEuclideanDomain<R> && requires {
        R::is_field == true;
    }
```

4.2.2 Detailed Description

Concept to express R is a field.

4.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring (ordered)

`#include <lib.h>`

4.3.1 Concept definition

```
template<typename R>
concept aerobus::IsRing =  requires {
        typename R::one;
        typename R::zero;
        typename R::template add_t<typename R::one, typename R::one>;
        typename R::template sub_t<typename R::one, typename R::one>;
        typename R::template mul_t<typename R::one, typename R::one>;
        typename R::template minus_t<typename R::one>;
        R::template eq_v<typename R::one, typename R::one> == true;
    }
```

4.3.2 Detailed Description

Concept to express R is a Ring (ordered)

Chapter 5

Class Documentation

5.1 aerobus::bigint Struct Reference

Classes

- struct to_hex_helper
- struct to_hex_helper< x >
- struct val
- struct val< s, a0 >

Public Types

- enum **signs** { **positive** , **negative** }
- using **zero** = val< signs::positive, 0 >
- using **one** = val< signs::positive, 1 >
- template<string_literal S>
  using **from_hex_t** = typename from_hex_helper< S, internal::make_index_sequence_reverse<(S.len() - 1)/8+1 > >::type
- template<typename I >
  using **minus_t** = typename I::minus_t

  minus operator (-I)
- template<typename I >
  using **simplify_t** = typename simplify< I >::type

  trim leading zeros
- template<typename I1 , typename I2 >
  using **add_t** = typename add< I1, I2 >::type

  addition operator (I1 + I2)
- template<typename I1 , typename I2 >
  using **sub_t** = typename sub< I1, I2 >::type

  substraction operator (I1 - I2)
- template<typename I , size_t s>
  using **shift_left_t** = typename I::template shift_left< s >

  shift left operator (add zeros to the end)
- template<typename I , size_t s>
  using **shift_right_t** = typename shift_right_helper< I, s >::type

  shift right operator (get highest digits)

- template<typename I1 , typename I2 >
  using **mul_t** = typename mul< I1, I2 >::type
  *multiplication operator (I1 * I2)*
- template<typename... Is>
  using **vadd_t** = typename vadd< Is... >::type
  addition of multiple values
- template<typename I >
  using **div_2_t** = typename div_2< I >::type
  division by 2
- template<typename I1 , typename I2 >
  using **floor_t** = typename floor_helper< I1, I2 >::type
  floor(A/B)
- template<typename I1 , typename I2 >
  using **div_t** = typename div_helper< I1, I2 >::Q
  division operator (I1/I2)
- template<typename I1 , typename I2 >
  using **mod_t** = typename div_helper< I1, I2 >::R
  modulo (remainder) operator (I1 % I2)
- template<typename I1 , typename I2 >
  using **gcd_t** = gcd_t< bigint, I1, I2 >
  gcd operator
- template<typename I1 , typename I2 , typename I3 >
  using **fma_t** = add_t< mul_t< I1, I2 >, I3 >
  *fma operator (I1 * I2 + I3)*

Static Public Attributes

- static constexpr bool **is_euclidean_domain** = true
- template<typename I1 , typename I2 >
  static constexpr bool **eq_v** = eq<I1, I2>::value
  equality operator (I1 == I2)
- template<typename I >
  static constexpr bool **pos_v** = I::sign == signs::positive && !I::is_zero_v
  positivity operator (strict) (I > 0)
- template<typename I1 , typename I2 >
  static constexpr bool **gt_v** = gt_helper<I1, I2>::value
  greater operator (strict) (I1 > I2)
- template<typename I1 , typename I2 >
  static constexpr bool **ge_v** = eq_v<I1, I2> || gt_v<I1, I2>
  greater or equal operator (I1 >= I2)

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

- src/lib.h

5.2 aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, E > Struct Template Reference

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

- src/lib.h

5.3 aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0||index > 0)> > Struct Template Reference

Public Types

- using **type** = typename Ring::zero

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

- src/lib.h

5.4 aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference

Public Types

- using **type** = aN

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

- src/lib.h

5.5 aerobus::ContinuedFraction< values > Struct Template Reference

represents a continued fraction a0 + 1/(a1 + 1/(...))

`#include <lib.h>`

5.5.1 Detailed Description

template<int64_t... values>
struct aerobus::ContinuedFraction< values >

represents a continued fraction a0 + 1/(a1 + 1/(...))

Template Parameters

...values	

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

- src/lib.h

5.6 aerobus::ContinuedFraction< a0 > Struct Template Reference

Public Types

- using **type** = typename q64::template inject_constant_t< a0 >

Static Public Attributes

- static constexpr double **val** = type::template get<double>()

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

- src/lib.h

5.7 aerobus::ContinuedFraction< a0, rest... > Struct Template Reference

Public Types

- using **type** = q64::template add_t< typename q64::template inject_constant_t< a0 >, typename q64::template div_t< typename q64::one, typename ContinuedFraction< rest... >::type > >

Static Public Attributes

- static constexpr double **val** = type::template get<double>()

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

- src/lib.h

5.8 aerobus::bigint::val< s, an, as >::digit_at< index, E > Struct Template Reference

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

- src/lib.h

5.9 aerobus::bigint::val< s, a0 >::digit_at< index, E > Struct Template Reference

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

- src/lib.h

5.10 aerobus::bigint::val< s, a0 >::digit_at< index, std::enable_if_t< index !=0 > > Struct Template Reference

Static Public Attributes

- static constexpr uint32_t **value** = 0

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

- src/lib.h

5.11 aerobus::bigint::val< s, a0 >::digit_at< index, std::enable_if_t< index==0 > > Struct Template Reference

Static Public Attributes

- static constexpr uint32_t **value** = a0

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

- src/lib.h

5.12 aerobus::bigint::val< s, an, as >::digit_at< index, std::enable_if_t<(index > sizeof...(as))> > Struct Template Reference

Static Public Attributes

- static constexpr uint32_t **value** = 0

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

- src/lib.h

5.13 aerobus::bigint::val< s, an, as >::digit_at< index, std::enable_if_t<(index<=sizeof...(as))> > Struct Template Reference

Static Public Attributes

- static constexpr uint32_t **value** = internal::value_at<(sizeof...(as) - index), an, as...>::value

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

- src/lib.h

5.14 aerobus::i32 Struct Reference

32 bits signed integers, seen as a algebraic ring with related operations

`#include <lib.h>`

Classes

- struct val
  values in i32

Public Types

- using **inner_type** = int32_t
- using **zero** = val< 0 >
  constant zero
- using **one** = val< 1 >
  constant one
- template<auto x>
  using **inject_constant_t** = val< static_cast< int32_t >(x)>
- template<typename v >
  using **inject_ring_t** = v
- template<typename v1 , typename v2 >
  using **add_t** = typename add< v1, v2 >::type
  addition operator
- template<typename v1 >
  using **minus_t** = val<-v1::v >
  -v1
- template<typename v1 , typename v2 >
  using **sub_t** = typename sub< v1, v2 >::type
  substraction operator
- template<typename v1 , typename v2 >
  using **mul_t** = typename mul< v1, v2 >::type
  multiplication operator

- template<typename v1 , typename v2 >
  using **div_t** = typename div< v1, v2 >::type
  division operator
- template<typename v1 , typename v2 >
  using **mod_t** = typename remainder< v1, v2 >::type
  modulus operator
- template<typename v1 , typename v2 >
  using **lt_t** = typename lt< v1, v2 >::type
  strict less operator (v1 < v2)
- template<typename v1 , typename v2 >
  using **gcd_t** = gcd_t< i32, v1, v2 >
  greatest common divisor

Static Public Attributes

- static constexpr bool **is_field** = false
  integers are not a field
- static constexpr bool **is_euclidean_domain** = true
  integers are an euclidean domain
- template<typename v1 , typename v2 >
  static constexpr bool **gt_v** = gt<v1, v2>::type::value
  strictly greater operator (v1 > v2)
- template<typename v1 , typename v2 >
  static constexpr bool **eq_v** = eq<v1, v2>::type::value
  equality operator
- template<typename v1 >
  static constexpr bool **pos_v** = (v1::v > 0)
  positivity (v1 > 0)

5.14.1 Detailed Description

32 bits signed integers, seen as a algebraic ring with related operations

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

- src/lib.h

5.15 aerobus::i64 Struct Reference

64 bits signed integers, seen as a algebraic ring with related operations

```
#include <lib.h>
```

Classes

- struct val
  values in i64

Public Types

- using **inner_type** = int64_t
- template<auto x>
  using **inject_constant_t** = val< static_cast< int64_t >(x)>
- template<typename v >
  using **inject_ring_t** = v
- using **zero** = val< 0 >

  constant zero
- using **one** = val< 1 >

  constant one
- template<typename v1 , typename v2 >
  using **add_t** = typename add< v1, v2 >::type

  addition operator
- template<typename v1 >
  using **minus_t** = val<-v1::v >

  -v1
- template<typename v1 , typename v2 >
  using **sub_t** = typename sub< v1, v2 >::type

  substraction operator
- template<typename v1 , typename v2 >
  using **mul_t** = typename mul< v1, v2 >::type

  multiplication operator
- template<typename v1 , typename v2 >
  using **div_t** = typename div< v1, v2 >::type

  division operator
- template<typename v1 , typename v2 >
  using **mod_t** = typename remainder< v1, v2 >::type

  modulus operator
- template<typename v1 , typename v2 >
  using **lt_t** = typename lt< v1, v2 >::type

  strict less operator (v1 < v2)
- template<typename v1 , typename v2 >
  using **gcd_t** = gcd_t< i64, v1, v2 >

  greatest common divisor

Static Public Attributes

- static constexpr bool **is_field** = false

  integers are not a field
- static constexpr bool **is_euclidean_domain** = true

  integers are an euclidean domain
- template<typename v1 , typename v2 >
  static constexpr bool **gt_v** = gt<v1, v2>::type::value

  strictly greater operator (v1 > v2)
- template<typename v1 , typename v2 >
  static constexpr bool **eq_v** = eq<v1, v2>::type::value

  equality operator
- template<typename v1 >
  static constexpr bool pos_v = (v1::v > 0)

  is v posititive

5.15.1 Detailed Description

64 bits signed integers, seen as a algebraic ring with related operations

5.15.2 Member Data Documentation

5.15.2.1 pos_v

```
template<typename v1 >
constexpr bool aerobus::i64::pos_v = (v1::v > 0)  [static], [constexpr]
```

is v posititive

weirdly enough, for clang, this must be declared before gcd_t

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

- src/lib.h

5.16 aerobus::bigint::floor_helper< A, B, std::enable_if_t< gt_helper< A, B >::value &&(A::digits !=1||B::digits !=1)> >::inner< lowerbound, upperbound, E > Struct Template Reference

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

- src/lib.h

5.17 aerobus::polynomial< Ring, variable_name >::eval_helper< valueRing, P >::inner< index, stop > Struct Template Reference

Static Public Member Functions

- static constexpr valueRing **func** (const valueRing &accum, const valueRing &x)

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

- src/lib.h

5.18 aerobus::bigint::floor_helper< A, B, std::enable_if_t< gt_helper< A, B >::value &&(A::digits !=1||B::digits !=1)> >::inner< lowerbound, upperbound, std::enable_if_t< eq< typename add< lowerbound, one >::type, upperbound >::value > > Struct Template Reference

Public Types

- using **type** = lowerbound

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

- src/lib.h

5.19 aerobus::bigint::floor_helper< A, B, std::enable_if_t< gt_helper< A, B >::value &&(A::digits !=1||B::digits !=1)> >::inner< lowerbound, upperbound, std::enable_if_t< gt_helper< upperbound, typename add< lowerbound, one >::type >::value &&!gt_helper< typename mul< average_t< upperbound, lowerbound >, B >::type, A >::value > > Struct Template Reference

Public Types

- using **type** = typename simplify< typename inner< average_t< upperbound, lowerbound >, upperbound >::type >::type

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

- src/lib.h

5.20 aerobus::bigint::floor_helper< A, B, std::enable_if_t< gt_helper< A, B >::value &&(A::digits !=1||B::digits !=1)> >::inner< lowerbound, upperbound, std::enable_if_t< gt_helper< upperbound, typename add< lowerbound, one >::type >::value &&gt_helper< typename mul< average_t< upperbound, lowerbound >, B >::type, A >::value > > Struct Template Reference

Public Types

- using **type** = typename simplify< typename inner< lowerbound, average_t< upperbound, lowerbound > >::type >::type

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

- src/lib.h

5.21 aerobus::polynomial< Ring, variable_name >::eval_helper< valueRing, P >::inner< stop, stop > Struct Template Reference

Static Public Member Functions

- static constexpr valueRing **func** (const valueRing &accum, const valueRing &x)

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

- src/lib.h

5.22 aerobus::is_prime< n > Struct Template Reference

checks if n is prime

`#include <lib.h>`

Static Public Attributes

- static constexpr bool **value** = internal::_is_prime<n, 5>::value
  true iff n is prime

5.22.1 Detailed Description

template<int32_t n>
struct aerobus::is_prime< n >

checks if n is prime

Template Parameters

n	

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

- src/lib.h

5.23 aerobus::polynomial< Ring, variable_name > Struct Template Reference

`#include <lib.h>`

Classes

- struct val
- struct val< coeffN >

Public Types

- using **zero** = val< typename Ring::zero >

  constant zero
- using **one** = val< typename Ring::one >

  constant one
- using **X** = val< typename Ring::one, typename Ring::zero >

  generator
- template<typename P >
  using simplify_t = typename simplify< P >::type

  simplifies a polynomial (deletes highest degree if null, do nothing otherwise)
- template<typename v1 , typename v2 >
  using add_t = typename add< v1, v2 >::type

  adds two polynomials
- template<typename v1 , typename v2 >
  using sub_t = typename sub< v1, v2 >::type

  substraction of two polynomials
- template<typename v1 >
  using **minus_t** = sub_t< zero, v1 >
- template<typename v1 , typename v2 >
  using mul_t = typename mul< v1, v2 >::type

  multiplication of two polynomials
- template<typename v1 , typename v2 >
  using lt_t = typename lt_helper< v1, v2 >::type

  strict less operator
- template<typename v1 , typename v2 >
  using div_t = typename div< v1, v2 >::q_type

  division operator
- template<typename v1 , typename v2 >
  using mod_t = typename div_helper< v1, v2, zero, v1 >::mod_type

  modulo operator
- template<typename coeff , size_t deg>
  using monomial_t = typename monomial< coeff, deg >::type

  monomial : coeff X^ deg
- template<typename v >
  using derive_t = typename derive_helper< v >::type

  derivation operator
- template<typename v1 , typename v2 >
  using gcd_t = std::conditional_t< Ring::is_euclidean_domain, typename make_unit< gcd_t< polynomial< Ring, variable_name >, v1, v2 > >::type, void >

  greatest common divisor of two polynomials
- template<auto x>
  using **inject_constant_t** = val< typename Ring::template inject_constant_t< x > >
- template<typename v >
  using **inject_ring_t** = val< v >

Static Public Attributes

- static constexpr bool **is_field** = false
- static constexpr bool **is_euclidean_domain** = Ring::is_euclidean_domain
- template<typename v1 , typename v2 >
  static constexpr bool eq_v = eq_helper<v1, v2>::value

  equality operator
- template<typename v1 , typename v2 >
  static constexpr bool gt_v = gt_helper<v1, v2>::type::value

  strict greater operator
- template<typename v >
  static constexpr bool pos_v = Ring::template pos_v<typename v::aN>

  checks for positivity (an > 0)

5.23.1 Detailed Description

template<typename Ring, char variable_name = 'x'>
requires IsEuclideanDomain<Ring>
struct aerobus::polynomial< Ring, variable_name >

polynomial with coefficients in Ring Ring must be an integral domain

5.23.2 Member Typedef Documentation

5.23.2.1 add_t

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring, variable_name >::add_t = typename add<v1, v2>::type
```

adds two polynomials

Template Parameters

v1	
v2	

5.23.2.2 derive_t

```
template<typename Ring , char variable_name = 'x'>
template<typename v >
using aerobus::polynomial< Ring, variable_name >::derive_t = typename derive_helper<v>::type
```

derivation operator

Template Parameters

v	

5.23.2.3 div_t

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring, variable_name >::div_t = typename div<v1, v2>::q_type
```

division operator

Template Parameters

v1	
v2	

5.23.2.4 gcd_t

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring, variable_name >::gcd_t = std::conditional_t< Ring::is_↩
euclidean_domain, typename make_unit<gcd_t<polynomial<Ring, variable_name>, v1, v2> >::type,
void>
```

greatest common divisor of two polynomials

Template Parameters

v1	
v2	

5.23.2.5 lt_t

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring, variable_name >::lt_t = typename lt_helper<v1, v2>::type
```

strict less operator

Template Parameters

v1	
v2	

5.23.2.6 mod_t

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring, variable_name >::mod_t = typename div_helper<v1, v2, zero,
v1>::mod_type
```

modulo operator

Template Parameters

v1	
v2	

5.23.2.7 monomial_t

```
template<typename Ring , char variable_name = 'x'>
template<typename coeff , size_t deg>
using aerobus::polynomial< Ring, variable_name >::monomial_t = typename monomial<coeff, deg>↩
::type
```

monomial : coeff X^deg

Template Parameters

coeff	
deg	

5.23.2.8 mul_t

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring, variable_name >::mul_t = typename mul<v1, v2>::type
```

multiplication of two polynomials

Template Parameters

v1	
v2	

5.23.2.9 simplify_t

```
template<typename Ring , char variable_name = 'x'>
template<typename P >
using aerobus::polynomial< Ring, variable_name >::simplify_t = typename simplify<P>::type
```

simplifies a polynomial (deletes highest degree if null, do nothing otherwise)

Template Parameters

P	

5.23.2.10 sub_t

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring, variable_name >::sub_t = typename sub<v1, v2>::type
```

substraction of two polynomials

Template Parameters

v1	
v2	

5.23.3 Member Data Documentation

5.23.3.1 eq_v

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
constexpr bool aerobus::polynomial< Ring, variable_name >::eq_v = eq_helper<v1, v2>::value
[static], [constexpr]
```

equality operator

Template Parameters

v1	
v2	

5.23.3.2 gt_v

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
constexpr bool aerobus::polynomial< Ring, variable_name >::gt_v = gt_helper<v1, v2>::type↩
::value  [static], [constexpr]
```

strict greater operator

Template Parameters

v1	
v2	

5.23.3.3 pos_v

```
template<typename Ring , char variable_name = 'x'>
template<typename v >
constexpr bool aerobus::polynomial< Ring, variable_name >::pos_v = Ring::template pos_v<typename
v::aN>  [static], [constexpr]
```

checks for positivity (an > 0)

Template Parameters

v	

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

- src/lib.h

5.24 aerobus::type_list< Ts >::pop_front Struct Reference

Public Types

- using **type** = typename internal::pop_front_h< Ts... >::head

- using **tail** = typename internal::pop_front_h< Ts... >::tail

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

- src/lib.h

5.25 aerobus::Quotient< Ring, X > Struct Template Reference

Classes

- struct val

Public Types

- using **zero** = val< typename Ring::zero >
- using **one** = val< typename Ring::one >
- template<typename v1 , typename v2 >
  using **add_t** = val< typename Ring::template add_t< typename v1::type, typename v2::type > >
- template<typename v1 , typename v2 >
  using **mul_t** = val< typename Ring::template mul_t< typename v1::type, typename v2::type > >
- template<typename v1 , typename v2 >
  using **div_t** = val< typename Ring::template div_t< typename v1::type, typename v2::type > >
- template<typename v1 , typename v2 >
  using **mod_t** = val< typename Ring::template mod_t< typename v1::type, typename v2::type > >
- template<auto x>
  using **inject_constant_t** = val< typename Ring::template inject_constant_t< x > >
- template<typename v >
  using **inject_ring_t** = val< v >

Static Public Attributes

- template<typename v1 , typename v2 >
  static constexpr bool **eq_v** = Ring::template eq_v<typename v1::type, typename v2::type>
- template<typename v >
  static constexpr bool **pos_v** = true
- static constexpr bool **is_euclidean_domain** = true

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

- src/lib.h

5.26 aerobus::type_list< Ts >::split< index > Struct Template Reference

Public Types

- using **head** = typename inner::head
- using **tail** = typename inner::tail

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

- src/lib.h

5.27 aerobus::string_literal< N > Struct Template Reference

Public Member Functions

- constexpr **string_literal** (const char(&str)[N])
- template<size_t i>
  constexpr char **char_at** () const
- constexpr size_t **len** () const

Public Attributes

- char **value** [N]

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

- src/lib.h

5.28 aerobus::bigint::to_hex_helper< an, as > Struct Template Reference

Static Public Member Functions

- static std::string **func** ()

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

- src/lib.h

5.29 aerobus::bigint::to_hex_helper< x > Struct Template Reference

Static Public Member Functions

- static std::string **func** ()

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

- src/lib.h

5.30 aerobus::type_list< Ts > Struct Template Reference

Empty pure template struct to handle type list.

Classes

- struct pop_front
- struct split

Public Types

- template<typename T >
  using **push_front** = type_list< T, Ts... >
- template<uint64_t index>
  using **at** = internal::type_at_t< index, Ts... >
- template<typename T >
  using **push_back** = type_list< Ts..., T >
- template<typename U >
  using **concat** = typename concat_h< U >::type
- template<uint64_t index, typename T >
  using **insert** = typename internal::insert_h< index, type_list< Ts... >, T >::type
- template<uint64_t index>
  using **remove** = typename internal::remove_h< index, type_list< Ts... > >::type

Static Public Attributes

- static constexpr size_t **length** = sizeof...(Ts)

5.30.1 Detailed Description

template<typename... Ts>
struct aerobus::type_list< Ts >

Empty pure template struct to handle type list.

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

- src/lib.h

5.31 aerobus::type_list<> Struct Reference

Public Types

- template<typename T >
  using **push_front** = type_list< T >
- template<typename T >
  using **push_back** = type_list< T >
- template<typename U >
  using **concat** = U
- template<uint64_t index, typename T >
  using **insert** = type_list< T >

Static Public Attributes

- static constexpr size_t **length** = 0

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

- src/lib.h

5.32 aerobus::bigint::val< s, an, as > Struct Template Reference

Classes

- struct digit_at
- struct digit_at< index, std::enable_if_t<(index > sizeof...(as))> >
- struct digit_at< index, std::enable_if_t<(index<=sizeof...(as))> >

Public Types

- template<size_t ss>
  using **shift_left** = typename shift_left_helper< ss, s, an, as... >::type
- using **strip** = val< s, as... >
- using **minus_t** = val< opposite_v< s >, an, as... >

Static Public Member Functions

- static std::string **to_string** ()
- static std::string **to_hex** ()

Static Public Attributes

- static constexpr signs **sign** = s
- static constexpr uint32_t **aN** = an
- static constexpr size_t **digits** = sizeof...(as) + 1
- static constexpr bool **is_zero_v** = sizeof...(as) == 0 && an == 0

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

- src/lib.h

5.33 aerobus::i32::val< x > Struct Template Reference

values in i32

`#include <lib.h>`

Static Public Member Functions

- template<typename valueType >
  static constexpr valueType get ()

  cast x into valueType
- static std::string **to_string** ()

  string representation of value
- template<typename valueRing >
  static constexpr valueRing eval (const valueRing &v)

  cast x into valueRing

Static Public Attributes

- static constexpr int32_t **v** = x
- static constexpr bool **is_zero_v** = x == 0

  is value zero

5.33.1 Detailed Description

template<int32_t x>
struct aerobus::i32::val< x >

values in i32

Template Parameters

x	an actual integer

5.33.2 Member Function Documentation

5.33.2.1 eval()

```
template<int32_t x>
template<typename valueRing >
static constexpr valueRing aerobus::i32::val< x >::eval (
            const valueRing & v )  [inline], [static], [constexpr]
```

cast x into valueRing

Template Parameters

valueRing	double for example

5.33.2.2 get()

```
template<int32_t x>
template<typename valueType >
static constexpr valueType aerobus::i32::val< x >::get ( )  [inline], [static], [constexpr]
```

cast x into valueType

Template Parameters

valueType	double for example

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

- src/lib.h

5.34 aerobus::i64::val< x > Struct Template Reference

values in i64

```
#include <lib.h>
```

Static Public Member Functions

- template<typename valueType >
  static constexpr valueType get ()
  cast value in valueType
- static std::string **to_string** ()
  string representation
- template<typename valueRing >
  static constexpr valueRing eval (const valueRing &v)
  cast value in valueRing

Static Public Attributes

- static constexpr int64_t **v** = x
- static constexpr bool **is_zero_v** = x == 0
  is value zero

5.34.1 Detailed Description

template<int64_t x>
struct aerobus::i64::val< x >

values in i64

Template Parameters

x	an actual integer

5.34.2 Member Function Documentation

5.34.2.1 eval()

```
template<int64_t x>
template<typename valueRing >
static constexpr valueRing aerobus::i64::val< x >::eval (
            const valueRing & v )  [inline], [static], [constexpr]
```

cast value in valueRing

Template Parameters

valueRing	(double for example)

5.34.2.2 get()

```
template<int64_t x>
template<typename valueType >
static constexpr valueType aerobus::i64::val< x >::get ( )  [inline], [static], [constexpr]
```

cast value in valueType

Template Parameters

valueType	(double for example)

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

- src/lib.h

5.35 aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs > Struct Template Reference

Public Types

- using **aN** = coeffN

heavy weight coefficient (non zero)

- using **strip** = val< coeffs... >

  remove largest coefficient

- template<size_t index>
  using coeff_at_t = typename coeff_at< index >::type

  coefficient at index

Static Public Member Functions

- static std::string to_string ()

  get a string representation of polynomial

- template<typename valueRing >
  static constexpr valueRing eval (const valueRing &x)

  evaluates polynomial seen as a function operating on ValueRing

Static Public Attributes

- static constexpr size_t **degree** = sizeof...(coeffs)

  degree of the polynomial

- static constexpr bool **is_zero_v** = degree == 0 && aN::is_zero_v

  true if polynomial is constant zero

5.35.1 Member Typedef Documentation

5.35.1.1 coeff_at_t

```
template<typename Ring , char variable_name = 'x'>
template<typename coeffN , typename...  coeffs>
template<size_t index>
using aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs >::coeff_at_t = typename
coeff_at<index>::type
```

coefficient at index

Template Parameters

index	

5.35.2 Member Function Documentation

5.35.2.1 eval()

```
template<typename Ring , char variable_name = 'x'>
template<typename coeffN , typename...  coeffs>
template<typename valueRing >
static constexpr valueRing aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs
>::eval (
            const valueRing & x )  [inline], [static], [constexpr]
```

evaluates polynomial seen as a function operating on ValueRing

Template Parameters

valueRing	usually float or double

Parameters

x	value

Returns

P(x)

5.35.2.2 to_string()

```
template<typename Ring , char variable_name = 'x'>
template<typename coeffN , typename...  coeffs>
static std::string aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs >::to_↩
string ( )  [inline], [static]
```

get a string representation of polynomial

Returns

something like a_n X^n + ... + a_1 X + a_0

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

- src/lib.h

5.36 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference

Public Types

- using **type** = std::conditional_t< Ring::template pos_v< tmp >, tmp, typename Ring::template minus_t< tmp > >

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

- src/lib.h

5.37 aerobus::zpz< p >::val< x > Struct Template Reference

Static Public Member Functions

- template<typename valueType >
  static constexpr valueType **get** ()
- static std::string **to_string** ()
- template<typename valueRing >
  static constexpr valueRing **eval** (const valueRing &v)

Static Public Attributes

- static constexpr int32_t **v** = x % p
- static constexpr bool **is_zero_v** = v == 0

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

- src/lib.h

5.38 aerobus::polynomial< Ring, variable_name >::val< coeffN > Struct Template Reference

Classes

- struct coeff_at
- struct coeff_at< index, std::enable_if_t<(index< 0||index > 0)> >
- struct coeff_at< index, std::enable_if_t<(index==0)> >

Public Types

- using **aN** = coeffN
- using **strip** = val< coeffN >
- template<size_t index>
  using **coeff_at_t** = typename coeff_at< index >::type

Static Public Member Functions

- static std::string **to_string** ()
- template<typename valueRing >
  static constexpr valueRing **eval** (const valueRing &x)

Static Public Attributes

- static constexpr size_t **degree** = 0
- static constexpr bool **is_zero_v** = coeffN::is_zero_v

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

- src/lib.h

5.39 aerobus::bigint::val< s, a0 > Struct Template Reference

Classes

- struct digit_at
- struct digit_at< index, std::enable_if_t< index !=0 > >
- struct digit_at< index, std::enable_if_t< index==0 > >

Public Types

- template<size_t ss>
  using **shift_left** = typename shift_left_helper< ss, s, a0 >::type
- using **minus_t** = val< opposite_v< s >, a0 >

Static Public Member Functions

- static std::string **to_string** ()
- static std::string **to_hex** ()

Static Public Attributes

- static constexpr signs **sign** = s
- static constexpr uint32_t **aN** = a0
- static constexpr size_t **digits** = 1
- static constexpr bool **is_zero_v** = a0 == 0

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

- src/lib.h

5.40 aerobus::zpz< p > Struct Template Reference

```
#include <lib.h>
```

Classes

- struct val

Public Types

- using **inner_type** = int32_t
- template<auto x>
  using **inject_constant_t** = val< static_cast< int32_t >(x)>
- using **zero** = val< 0 >
- using **one** = val< 1 >
- template<typename v1 >
  using **minus_t** = val<-v1::v >

  -v1
- template<typename v1 , typename v2 >
  using **add_t** = typename add< v1, v2 >::type
- template<typename v1 , typename v2 >
  using **sub_t** = typename sub< v1, v2 >::type
- template<typename v1 , typename v2 >
  using **mul_t** = typename mul< v1, v2 >::type
- template<typename v1 , typename v2 >
  using **div_t** = typename div< v1, v2 >::type
- template<typename v1 , typename v2 >
  using **mod_t** = typename remainder< v1, v2 >::type
- template<typename v1 , typename v2 >
  using **lt_t** = typename lt< v1, v2 >::type
- template<typename v1 , typename v2 >
  using **gcd_t** = gcd_t< i32, v1, v2 >

Static Public Attributes

- static constexpr bool **is_field** = is_prime<p>::value
- static constexpr bool **is_euclidean_domain** = true
- template<typename v1 , typename v2 >
  static constexpr bool **gt_v** = gt<v1, v2>::type::value
- template<typename v1 , typename v2 >
  static constexpr bool **eq_v** = eq<v1, v2>::type::value
- template<typename v >
  static constexpr bool **pos_v** = pos<v>::type::value

5.40.1 Detailed Description

template<int32_t p>
struct aerobus::zpz< p >

congruence classes of integers for a modulus if p is prime, zpz is a field, otherwise an integral domain with all related operations

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

- src/lib.h

Chapter 6

File Documentation

6.1 lib.h

```
1 // -*- lsst-c++ -*-
2
3 #include <cstdint> // NOLINT(clang-diagnostic-pragma-pack)
4 #include <cstddef>
5 #include <cstring>
6 #include <type_traits>
7 #include <utility>
8 #include <algorithm>
9 #include <functional>
10 #include <string>
11 #include <concepts>
12 #include <array>
13 #include <format>
14
15
16 #ifdef _MSC_VER
17 #define ALIGNED(x) __declspec(align(x))
18 #define INLINED __forceinline
19 #else
20 #define ALIGNED(x) __attribute__((aligned(x)))
21 #define INLINED __attribute__((always_inline)) inline
22 #endif
23
24 // aligned allocation
25 namespace aerobus {
32     template<typename T>
33     T* aligned_malloc(size_t count, size_t alignment) {
34 #ifdef _MSC_VER
35         return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
36 #else
37         return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
38 #endif
39     }
40
41     constexpr std::array<int32_t, 1000> primes = { { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
      47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
      157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263,
      269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383,
      389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503,
      509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641,
      643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769,
      773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911,
      919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039,
      1049, 1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163,
      1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289,
      1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429,
      1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511, 1523, 1531, 1543,
      1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619, 1621, 1627, 1637, 1657,
      1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759, 1777, 1783, 1787,
      1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 1913, 1931,
      1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063,
      2069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203,
      2207, 2213, 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333,
      2339, 2341, 2347, 2351, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441,
      2447, 2459, 2467, 2473, 2477, 2503, 2521, 2531, 2539, 2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609,
      2617, 2621, 2633, 2647, 2657, 2659, 2663, 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713,
      2719, 2729, 2731, 2741, 2749, 2753, 2767, 2777, 2789, 2791, 2797, 2801, 2803, 2819, 2833, 2837, 2843,
      2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917, 2927, 2939, 2953, 2957, 2963, 2969, 2971, 2999,
```

```
    3001, 3011, 3019, 3023, 3037, 3041, 3049, 3061, 3067, 3079, 3083, 3089, 3109, 3119, 3121, 3137, 3163,
    3167, 3169, 3181, 3187, 3191, 3203, 3209, 3217, 3221, 3229, 3251, 3253, 3257, 3259, 3271, 3299, 3301,
    3307, 3313, 3319, 3323, 3329, 3331, 3343, 3347, 3359, 3361, 3371, 3373, 3389, 3391, 3407, 3413, 3433,
    3449, 3457, 3461, 3463, 3467, 3469, 3491, 3499, 3511, 3517, 3527, 3529, 3533, 3539, 3541, 3547, 3557,
    3559, 3571, 3581, 3583, 3593, 3607, 3613, 3617, 3623, 3631, 3637, 3643, 3659, 3671, 3673, 3677, 3691,
    3697, 3701, 3709, 3719, 3727, 3733, 3739, 3761, 3767, 3769, 3779, 3793, 3797, 3803, 3821, 3823, 3833,
    3847, 3851, 3853, 3863, 3877, 3881, 3889, 3907, 3911, 3917, 3919, 3923, 3929, 3931, 3943, 3947, 3967,
    3989, 4001, 4003, 4007, 4013, 4019, 4021, 4027, 4049, 4051, 4057, 4073, 4079, 4091, 4093, 4099, 4111,
    4127, 4129, 4133, 4139, 4153, 4157, 4159, 4177, 4201, 4211, 4217, 4219, 4229, 4231, 4241, 4243, 4253,
    4259, 4261, 4271, 4273, 4283, 4289, 4297, 4327, 4337, 4339, 4349, 4357, 4363, 4373, 4391, 4397, 4409,
    4421, 4423, 4441, 4447, 4451, 4457, 4463, 4481, 4483, 4493, 4507, 4513, 4517, 4519, 4523, 4547, 4549,
    4561, 4567, 4583, 4591, 4597, 4603, 4621, 4637, 4639, 4643, 4649, 4651, 4657, 4663, 4673, 4679, 4691,
    4703, 4721, 4723, 4729, 4733, 4751, 4759, 4783, 4787, 4789, 4793, 4799, 4801, 4813, 4817, 4831, 4861,
    4871, 4877, 4889, 4903, 4909, 4919, 4931, 4933, 4937, 4943, 4951, 4957, 4967, 4969, 4973, 4987, 4993,
    4999, 5003, 5009, 5011, 5021, 5023, 5039, 5051, 5059, 5077, 5081, 5087, 5099, 5101, 5107, 5113, 5119,
    5147, 5153, 5167, 5171, 5179, 5189, 5197, 5209, 5227, 5231, 5233, 5237, 5261, 5273, 5279, 5281, 5297,
    5303, 5309, 5323, 5333, 5347, 5351, 5381, 5387, 5393, 5399, 5407, 5413, 5417, 5419, 5431, 5437, 5441,
    5443, 5449, 5471, 5477, 5479, 5483, 5501, 5503, 5507, 5519, 5521, 5527, 5531, 5557, 5563, 5569, 5573,
    5581, 5591, 5623, 5639, 5641, 5647, 5651, 5653, 5657, 5659, 5669, 5683, 5689, 5693, 5701, 5711, 5717,
    5737, 5741, 5743, 5749, 5779, 5783, 5791, 5801, 5807, 5813, 5821, 5827, 5839, 5843, 5849, 5851, 5857,
    5861, 5867, 5869, 5879, 5881, 5897, 5903, 5923, 5927, 5939, 5953, 5981, 5987, 6007, 6011, 6029, 6037,
    6043, 6047, 6053, 6067, 6073, 6079, 6089, 6091, 6101, 6113, 6121, 6131, 6133, 6143, 6151, 6163, 6173,
    6197, 6199, 6203, 6211, 6217, 6221, 6229, 6247, 6257, 6263, 6269, 6271, 6277, 6287, 6299, 6301, 6311,
    6317, 6323, 6329, 6337, 6343, 6353, 6359, 6361, 6367, 6373, 6379, 6389, 6397, 6421, 6427, 6449, 6451,
    6469, 6473, 6481, 6491, 6521, 6529, 6547, 6551, 6553, 6563, 6569, 6571, 6577, 6581, 6599, 6607, 6619,
    6637, 6653, 6659, 6661, 6673, 6679, 6689, 6691, 6701, 6703, 6709, 6719, 6733, 6737, 6761, 6763, 6779,
    6781, 6791, 6793, 6803, 6823, 6827, 6829, 6833, 6841, 6857, 6863, 6869, 6871, 6883, 6899, 6907, 6911,
    6917, 6947, 6949, 6959, 6961, 6967, 6971, 6977, 6983, 6991, 6997, 7001, 7013, 7019, 7027, 7039, 7043,
    7057, 7069, 7079, 7103, 7109, 7121, 7127, 7129, 7151, 7159, 7177, 7187, 7193, 7207, 7211, 7213, 7219,
    7229, 7237, 7243, 7247, 7253, 7283, 7297, 7307, 7309, 7321, 7331, 7333, 7349, 7351, 7369, 7393, 7411,
    7417, 7433, 7451, 7457, 7459, 7477, 7481, 7487, 7489, 7499, 7507, 7517, 7523, 7529, 7537, 7541, 7547,
    7549, 7559, 7561, 7573, 7577, 7583, 7589, 7591, 7603, 7607, 7621, 7639, 7643, 7649, 7669, 7673, 7681,
    7687, 7691, 7699, 7703, 7717, 7723, 7727, 7741, 7753, 7757, 7759, 7789, 7793, 7817, 7823, 7829, 7841,
    7853, 7867, 7873, 7877, 7879, 7883, 7901, 7907, 7919 } };
42
51      template<typename T, size_t N>
52      constexpr bool contains(const std::array<T, N>& arr, const T& v) {
53          for (const auto& vv :  arr) {
54              if (v == vv) {
55                  return true;
56              }
57          }
58
59          return false;
60      }
61
62      template <size_t N>
63      struct string_literal {
64          constexpr string_literal(const char (&str)[N]) {
65              std::reverse_copy(str, str + N, value);
66          }
67
68          template<size_t i>
69          constexpr char char_at()const {
70              if constexpr (i < N)  {
71                  return this->value[i];
72              }
73              return 0;
74          }
75
76          constexpr size_t len()const { return N; }
77
78          char value[N];
79      };
80  }
81
82  // concepts
83  namespace aerobus
84  {
86      template <typename R>
87      concept IsRing = requires {
88          typename R::one;
89          typename R::zero;
90          typename R::template add_t<typename R::one, typename R::one>;
91          typename R::template sub_t<typename R::one, typename R::one>;
92          typename R::template mul_t<typename R::one, typename R::one>;
93          typename R::template minus_t<typename R::one>;
94          R::template eq_v<typename R::one, typename R::one> == true;
95      };
96
98      template <typename R>
99      concept IsEuclideanDomain = IsRing<R> && requires {
100          typename R::template div_t<typename R::one, typename R::one>;
101          typename R::template mod_t<typename R::one, typename R::one>;
102          typename R::template gcd_t<typename R::one, typename R::one>;
103
104          R::template pos_v<typename R::one> == true;
```

```
105         R::template gt_v<typename R::one, typename R::zero> == true;
106         R::is_euclidean_domain == true;
107     };
108 
110     template<typename R>
111     concept IsField = IsEuclideanDomain<R> && requires {
112         R::is_field == true;
113     };
114 }
115 
116 // utilities
117 namespace aerobus {
118     namespace internal
119     {
120         template<template<typename...> typename TT, typename T>
121         struct is_instantiation_of :  std::false_type { };
122 
123         template<template<typename...> typename TT, typename...  Ts>
124         struct is_instantiation_of<TT, TT<Ts...>> :  std::true_type { };
125 
126         template<template<typename...> typename TT, typename T>
127         inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
128 
129         template <size_t i, typename T, typename...  Ts>
130         struct type_at
131         {
132             static_assert(i < sizeof...(Ts) + 1, "index out of range");
133             using type = typename type_at<i - 1, Ts...>::type;
134         };
135 
136         template <typename T, typename...  Ts> struct type_at<0, T, Ts...> {
137             using type = T;
138         };
139 
140         template <size_t i, typename...  Ts>
141         using type_at_t = typename type_at<i, Ts...>::type;
142 
143         template<size_t i, auto x, auto...  xs>
144         struct value_at {
145             static_assert(i < sizeof...(xs) + 1, "index out of range");
146             static constexpr auto value = value_at<i-1, xs...>::value;
147         };
148 
149         template<auto x, auto...  xs>
150         struct value_at<0, x, xs...> {
151             static constexpr auto value = x;
152         };
153 
154 
155         template<int32_t n, int32_t i, typename E = void>
156         struct _is_prime {};
157 
158         // first 1000 primes are precomputed and stored in a table
159         template<int32_t n, int32_t i>
160         struct _is_prime<n, i, std::enable_if_t<(n < 7920) && (contains<int32_t, 1000>(primes, n))» :
    std::true_type {};
161 
162         // first 1000 primes are precomputed and stored in a table
163         template<int32_t n, int32_t i>
164         struct _is_prime<n, i, std::enable_if_t<(n < 7920) && (!contains<int32_t, 1000>(primes, n))» :
    std::false_type {};
165 
166         template<int32_t n, int32_t i>
167         struct _is_prime<n, i, std::enable_if_t<
168             (n >= 7920) &&
169             (i >= 5 && i * i <= n) &&
170             (n % i == 0 || n % (i + 2) == 0)» :  std::false_type {};
171 
172 
173         template<int32_t n, int32_t i>
174         struct _is_prime<n, i, std::enable_if_t<
175             (n >= 7920) &&
176             (i >= 5 && i * i <= n) &&
177             (n % i != 0 && n % (i + 2) != 0)» {
178             static constexpr bool value = _is_prime<n, i + 6>::value;
179         };
180 
181         template<int32_t n, int32_t i>
182         struct _is_prime<n, i, std::enable_if_t<
183             (n >= 7920) &&
184             (i >= 5 && i * i > n)» :  std::true_type {};
185     }
186 
189     template<int32_t n>
190     struct is_prime {
192         static constexpr bool value = internal::_is_prime<n, 5>::value;
193     };
```

```
194
195     namespace internal {
196         template <std::size_t...  Is>
197         constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
198             -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
199
200         template <std::size_t N>
201         using make_index_sequence_reverse
202             = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
203
209         template<typename Ring, typename E = void>
210         struct gcd;
211
212         template<typename Ring>
213         struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
214             template<typename A, typename B, typename E = void>
215             struct gcd_helper {};
216
217             // B = 0, A > 0
218             template<typename A, typename B>
219             struct gcd_helper<A, B, std::enable_if_t<
220                 B::is_zero_v && Ring::template pos_v<A»>
221             {
222                 using type = A;
223             };
224
225             // B = 0, A < 0
226             template<typename A, typename B>
227             struct gcd_helper<A, B, std::enable_if_t<
228                 B::is_zero_v && !Ring::template pos_v<A»>
229             {
230                 using type = typename Ring::template minus_t<A>;
231             };
232
233             // B != 0
234             template<typename A, typename B>
235             struct gcd_helper<A, B, std::enable_if_t<
236                 (!B::is_zero_v)
237                 » {
238             private:
239                 // A / B
240                 using k = typename Ring::template div_t<A, B>;
241                 // A - (A/B)*B = A % B
242                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
243             public:
244                 using type = typename gcd_helper<B, m>::type;
245             };
246
247             template<typename A, typename B>
248             using type = typename gcd_helper<A, B>::type;
249         };
250     }
251
254     template<typename T, typename A, typename B>
255     using gcd_t = typename internal::gcd<T>::template type<A, B>;
256 }
257
258 // quotient ring by the principal ideal generated by X
259 namespace aerobus {
260     template<typename Ring, typename X>
261     requires IsRing<Ring>
262     struct Quotient {
263         template <typename V>
264         struct val {
265         private:
266             using tmp = typename Ring::template mod_t<V, X>;
267         public:
268             using type = std::conditional_t<
269                 Ring::template pos_v<tmp>,
270                 tmp,
271                 typename Ring::template minus_t<tmp>
272             >;
273         };
274
275         using zero = val<typename Ring::zero>;
276         using one = val<typename Ring::one>;
277
278         template<typename v1, typename v2>
279         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
280         template<typename v1, typename v2>
281         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
282         template<typename v1, typename v2>
283         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
284         template<typename v1, typename v2>
285         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
286
287         template<typename v1, typename v2>
```

```
288         static constexpr bool eq_v = Ring::template eq_v<typename v1::type, typename v2::type>;
289
290         template<typename v>
291         static constexpr bool pos_v = true;
292
293         static constexpr bool is_euclidean_domain = true;
294
295         template<auto x>
296         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
297
298         template<typename v>
299         using inject_ring_t = val<v>;
300     };
301 }
302
303 // type_list
304 namespace aerobus
305 {
307     template <typename...  Ts>
308     struct type_list;
309
310     namespace internal
311     {
312         template <typename T, typename...  Us>
313         struct pop_front_h
314         {
315             using tail = type_list<Us...>;
316             using head = T;
317         };
318
319         template <uint64_t index, typename L1, typename L2>
320         struct split_h
321         {
322         private:
323             static_assert(index <= L2::length, "index ouf of bounds");
324             using a = typename L2::pop_front::type;
325             using b = typename L2::pop_front::tail;
326             using c = typename L1::template push_back<a>;
327
328         public:
329             using head = typename split_h<index - 1, c, b>::head;
330             using tail = typename split_h<index - 1, c, b>::tail;
331         };
332
333         template <typename L1, typename L2>
334         struct split_h<0, L1, L2>
335         {
336             using head = L1;
337             using tail = L2;
338         };
339
340         template <uint64_t index, typename L, typename T>
341         struct insert_h
342         {
343             static_assert(index <= L::length, "index ouf of bounds");
344             using s = typename L::template split<index>;
345             using left = typename s::head;
346             using right = typename s::tail;
347             using ll = typename left::template push_back<T>;
348             using type = typename ll::template concat<right>;
349         };
350
351         template <uint64_t index, typename L>
352         struct remove_h
353         {
354             using s = typename L::template split<index>;
355             using left = typename s::head;
356             using right = typename s::tail;
357             using rr = typename right::pop_front::tail;
358             using type = typename left::template concat<rr>;
359         };
360     }
361
362     template <typename...  Ts>
363     struct type_list
364     {
365     private:
366         template <typename T>
367         struct concat_h;
368
369         template <typename...  Us>
370         struct concat_h<type_list<Us...»
371         {
372             using type = type_list<Ts..., Us...>;
373         };
374
375     public:
```

```
376         static constexpr size_t length = sizeof...(Ts);
377
378         template <typename T>
379         using push_front = type_list<T, Ts...>;
380
381         template <uint64_t index>
382         using at = internal::type_at_t<index, Ts...>;
383
384         struct pop_front
385         {
386             using type = typename internal::pop_front_h<Ts...>::head;
387             using tail = typename internal::pop_front_h<Ts...>::tail;
388         };
389
390         template <typename T>
391         using push_back = type_list<Ts..., T>;
392
393         template <typename U>
394         using concat = typename concat_h<U>::type;
395
396         template <uint64_t index>
397         struct split
398         {
399         private:
400             using inner = internal::split_h<index, type_list<>, type_list<Ts...»;
401
402         public:
403             using head = typename inner::head;
404             using tail = typename inner::tail;
405         };
406
407         template <uint64_t index, typename T>
408         using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
409
410         template <uint64_t index>
411         using remove = typename internal::remove_h<index, type_list<Ts...»::type;
412     };
413
414     template <>
415     struct type_list<>
416     {
417         static constexpr size_t length = 0;
418
419         template <typename T>
420         using push_front = type_list<T>;
421
422         template <typename T>
423         using push_back = type_list<T>;
424
425         template <typename U>
426         using concat = U;
427
428         // TODO: assert index == 0
429         template <uint64_t index, typename T>
430         using insert = type_list<T>;
431     };
432 }
433
434 // i32
435 namespace aerobus {
437     struct i32 {
438         using inner_type = int32_t;
441         template<int32_t x>
442         struct val {
443             static constexpr int32_t v = x;
444
447             template<typename valueType>
448             static constexpr valueType get() { return static_cast<valueType>(x); }
449
451             static constexpr bool is_zero_v = x == 0;
452
454             static std::string to_string() {
455                 return std::to_string(x);
456             }
457
460             template<typename valueRing>
461             static constexpr valueRing eval(const valueRing& v) {
462                 return static_cast<valueRing>(x);
463             }
464         };
465
467         using zero = val<0>;
469         using one = val<1>;
471         static constexpr bool is_field = false;
473         static constexpr bool is_euclidean_domain = true;
477         template<auto x>
478         using inject_constant_t = val<static_cast<int32_t>(x)>;
```

```
479
480         template<typename v>
481         using inject_ring_t = v;
482
483     private:
484         template<typename v1, typename v2>
485         struct add {
486             using type = val<v1::v + v2::v>;
487         };
488
489         template<typename v1, typename v2>
490         struct sub {
491             using type = val<v1::v - v2::v>;
492         };
493
494         template<typename v1, typename v2>
495         struct mul {
496             using type = val<v1::v* v2::v>;
497         };
498
499         template<typename v1, typename v2>
500         struct div {
501             using type = val<v1::v / v2::v>;
502         };
503
504         template<typename v1, typename v2>
505         struct remainder {
506             using type = val<v1::v % v2::v>;
507         };
508
509         template<typename v1, typename v2>
510         struct gt {
511             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
512         };
513
514         template<typename v1, typename v2>
515         struct lt {
516             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
517         };
518
519         template<typename v1, typename v2>
520         struct eq {
521             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
522         };
523
524     public:
526         template<typename v1, typename v2>
527         using add_t = typename add<v1, v2>::type;
528
530         template<typename v1>
531         using minus_t = val<-v1::v>;
532
534         template<typename v1, typename v2>
535         using sub_t = typename sub<v1, v2>::type;
536
538         template<typename v1, typename v2>
539         using mul_t = typename mul<v1, v2>::type;
540
542         template<typename v1, typename v2>
543         using div_t = typename div<v1, v2>::type;
544
546         template<typename v1, typename v2>
547         using mod_t = typename remainder<v1, v2>::type;
548
550         template<typename v1, typename v2>
551         static constexpr bool gt_v = gt<v1, v2>::type::value;
552
554         template<typename v1, typename v2>
555         using lt_t = typename lt<v1, v2>::type;
556
558         template<typename v1, typename v2>
559         static constexpr bool eq_v = eq<v1, v2>::type::value;
560
562         template<typename v1>
563         static constexpr bool pos_v = (v1::v > 0);
564
566         template<typename v1, typename v2>
567         using gcd_t = gcd_t<i32, v1, v2>;
568     };
569 }
570
571 // i64
572 namespace aerobus {
574     struct i64 {
575         using inner_type = int64_t;
578         template<int64_t x>
579         struct val {
```

```
580             static constexpr int64_t v = x;
581
584             template<typename valueType>
585             static constexpr valueType get() { return static_cast<valueType>(x); }
586
588             static constexpr bool is_zero_v = x == 0;
589
591             static std::string to_string() {
592                 return std::to_string(x);
593             }
594
597             template<typename valueRing>
598             static constexpr valueRing eval(const valueRing& v) {
599                 return static_cast<valueRing>(x);
600             }
601         };
602
606         template<auto x>
607         using inject_constant_t = val<static_cast<int64_t>(x)>;
608
609         template<typename v>
610         using inject_ring_t = v;
611
613         using zero = val<0>;
615         using one = val<1>;
617         static constexpr bool is_field = false;
619         static constexpr bool is_euclidean_domain = true;
620
621     private:
622         template<typename v1, typename v2>
623         struct add {
624             using type = val<v1::v + v2::v>;
625         };
626
627         template<typename v1, typename v2>
628         struct sub {
629             using type = val<v1::v - v2::v>;
630         };
631
632         template<typename v1, typename v2>
633         struct mul {
634             using type = val<v1::v* v2::v>;
635         };
636
637         template<typename v1, typename v2>
638         struct div {
639             using type = val<v1::v / v2::v>;
640         };
641
642         template<typename v1, typename v2>
643         struct remainder {
644             using type = val<v1::v% v2::v>;
645         };
646
647         template<typename v1, typename v2>
648         struct gt {
649             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
650         };
651
652         template<typename v1, typename v2>
653         struct lt {
654             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
655         };
656
657         template<typename v1, typename v2>
658         struct eq {
659             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
660         };
661
662     public:
664         template<typename v1, typename v2>
665         using add_t = typename add<v1, v2>::type;
666
668         template<typename v1>
669         using minus_t = val<-v1::v>;
670
672         template<typename v1, typename v2>
673         using sub_t = typename sub<v1, v2>::type;
674
676         template<typename v1, typename v2>
677         using mul_t = typename mul<v1, v2>::type;
678
680         template<typename v1, typename v2>
681         using div_t = typename div<v1, v2>::type;
682
684         template<typename v1, typename v2>
685         using mod_t = typename remainder<v1, v2>::type;
```

```
686
688         template<typename v1, typename v2>
689         static constexpr bool gt_v = gt<v1, v2>::type::value;
690
692         template<typename v1, typename v2>
693         using lt_t = typename lt<v1, v2>::type;
694
696         template<typename v1, typename v2>
697         static constexpr bool eq_v = eq<v1, v2>::type::value;
698
701         template<typename v1>
702         static constexpr bool pos_v = (v1::v > 0);
703
705         template<typename v1, typename v2>
706         using gcd_t = gcd_t<i64, v1, v2>;
707     };
708 }
709
710 // z/pz
711 namespace aerobus {
716     template<int32_t p>
717     struct zpz {
718         using inner_type = int32_t;
719         template<int32_t x>
720         struct val {
721             static constexpr int32_t v = x % p;
722
723             template<typename valueType>
724             static constexpr valueType get() { return static_cast<valueType>(x % p); }
725
726             static constexpr bool is_zero_v = v == 0;
727             static std::string to_string() {
728                 return std::to_string(x % p);
729             }
730
731             template<typename valueRing>
732             static constexpr valueRing eval(const valueRing& v) {
733                 return static_cast<valueRing>(x % p);
734             }
735         };
736
737         template<auto x>
738         using inject_constant_t = val<static_cast<int32_t>(x)>;
739
740         using zero = val<0>;
741         using one = val<1>;
742         static constexpr bool is_field = is_prime<p>::value;
743         static constexpr bool is_euclidean_domain = true;
744
745     private:
746         template<typename v1, typename v2>
747         struct add {
748             using type = val<(v1::v + v2::v) % p>;
749         };
750
751         template<typename v1, typename v2>
752         struct sub {
753             using type = val<(v1::v - v2::v) % p>;
754         };
755
756         template<typename v1, typename v2>
757         struct mul {
758             using type = val<(v1::v* v2::v) % p>;
759         };
760
761         template<typename v1, typename v2>
762         struct div {
763             using type = val<(v1::v% p) / (v2::v % p)>;
764         };
765
766         template<typename v1, typename v2>
767         struct remainder {
768             using type = val<(v1::v% v2::v) % p>;
769         };
770
771         template<typename v1, typename v2>
772         struct gt {
773             using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
774         };
775
776         template<typename v1, typename v2>
777         struct lt {
778             using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
779         };
780
781         template<typename v1, typename v2>
782         struct eq {
```

```
783             using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
784         };
785 
786         template<typename v1>
787         struct pos {
788             using type = std::bool_constant<(v1::v > 0)>;
789         };
790 
791     public:
793         template<typename v1>
794         using minus_t = val<-v1::v>;
795 
796         template<typename v1, typename v2>
797         using add_t = typename add<v1, v2>::type;
798 
799         template<typename v1, typename v2>
800         using sub_t = typename sub<v1, v2>::type;
801 
802         template<typename v1, typename v2>
803         using mul_t = typename mul<v1, v2>::type;
804 
805         template<typename v1, typename v2>
806         using div_t = typename div<v1, v2>::type;
807 
808         template<typename v1, typename v2>
809         using mod_t = typename remainder<v1, v2>::type;
810 
811         template<typename v1, typename v2>
812         static constexpr bool gt_v = gt<v1, v2>::type::value;
813 
814         template<typename v1, typename v2>
815         using lt_t = typename lt<v1, v2>::type;
816 
817         template<typename v1, typename v2>
818         static constexpr bool eq_v = eq<v1, v2>::type::value;
819 
820         template<typename v1, typename v2>
821         using gcd_t = gcd_t<i32, v1, v2>;
822 
823         template<typename v>
824         static constexpr bool pos_v = pos<v>::type::value;
825     };
826 }
827 
828 // polynomial
829 namespace aerobus {
830     // coeffN x^N + ...
835     template<typename Ring, char variable_name = 'x'>
836     requires IsEuclideanDomain<Ring>
837     struct polynomial {
838         static constexpr bool is_field = false;
839         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
840 
841         template<typename coeffN, typename...  coeffs>
842         struct val {
844             static constexpr size_t degree = sizeof...(coeffs);
846             using aN = coeffN;
848             using strip = val<coeffs...>;
850             static constexpr bool is_zero_v = degree == 0 && aN::is_zero_v;
851 
852             private:
853             template<size_t index, typename E = void>
854             struct coeff_at {};
855 
856             template<size_t index>
857             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))>> {
858                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
859             };
860 
861             template<size_t index>
862             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))>> {
863                 using type = typename Ring::zero;
864             };
865 
866             public:
869             template<size_t index>
870             using coeff_at_t = typename coeff_at<index>::type;
871 
874             static std::string to_string() {
875                 return string_helper<coeffN, coeffs...>::func();
876             }
877 
882             template<typename valueRing>
883             static constexpr valueRing eval(const valueRing& x) {
884                 return eval_helper<valueRing, val>::template inner<0, degree +
    1>::func(static_cast<valueRing>(0), x);
885             }
```

```
886         };
887
888         // specialization for constants
889         template<typename coeffN>
890         struct val<coeffN> {
891             static constexpr size_t degree = 0;
892             using aN = coeffN;
893             using strip = val<coeffN>;
894             static constexpr bool is_zero_v = coeffN::is_zero_v;
895
896             template<size_t index, typename E = void>
897             struct coeff_at {};
898
899             template<size_t index>
900             struct coeff_at<index, std::enable_if_t<(index == 0)» {
901                 using type = aN;
902             };
903
904             template<size_t index>
905             struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
906                 using type = typename Ring::zero;
907             };
908
909             template<size_t index>
910             using coeff_at_t = typename coeff_at<index>::type;
911
912             static std::string to_string() {
913                 return string_helper<coeffN>::func();
914             }
915
916             template<typename valueRing>
917             static constexpr valueRing eval(const valueRing& x) {
918                 return static_cast<valueRing>(aN::template get<valueRing>());
919             }
920         };
921
923         using zero = val<typename Ring::zero>;
925         using one = val<typename Ring::one>;
927         using X = val<typename Ring::one, typename Ring::zero>;
928
929     private:
930         template<typename P, typename E = void>
931         struct simplify;
932
933         template <typename P1, typename P2, typename I>
934         struct add_low;
935
936         template<typename P1, typename P2>
937         struct add {
938             using type = typename simplify<typename add_low<
939             P1,
940             P2,
941             internal::make_index_sequence_reverse<
942             std::max(P1::degree, P2::degree) + 1
943             »::type>::type;
944         };
945
946         template <typename P1, typename P2, typename I>
947         struct sub_low;
948
949         template <typename P1, typename P2, typename I>
950         struct mul_low;
951
952         template<typename v1, typename v2>
953         struct mul {
954                 using type = typename mul_low<
955                     v1,
956                     v2,
957                     internal::make_index_sequence_reverse<
958                     v1::degree + v2::degree + 1
959                     »::type;
960         };
961
962         template<typename coeff, size_t deg>
963         struct monomial;
964
965         template<typename v, typename E = void>
966         struct derive_helper {};
967
968         template<typename v>
969         struct derive_helper<v, std::enable_if_t<v::degree == 0» {
970             using type = zero;
971         };
972
973         template<typename v>
974         struct derive_helper<v, std::enable_if_t<v::degree != 0» {
975             using type = typename add<
```

```
976                 typename derive_helper<typename simplify<typename v::strip>::type>::type,
977                 typename monomial<
978                     typename Ring::template mul_t<
979                         typename v::aN,
980                         typename Ring::template inject_constant_t<(v::degree)>
981                     >,
982                     v::degree - 1
983                 >::type
984             >::type;
985         };
986 
987         template<typename v1, typename v2, typename E = void>
988         struct eq_helper {};
989 
990         template<typename v1, typename v2>
991         struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
992             static constexpr bool value = false;
993         };
994 
995 
996         template<typename v1, typename v2>
997         struct eq_helper<v1, v2, std::enable_if_t<
998             v1::degree == v2::degree &&
999             (v1::degree != 0 || v2::degree != 0) &&
1000              (!Ring::template eq_v<typename v1::aN, typename v2::aN>)
1001          » {
1002              static constexpr bool value = false;
1003          };
1004 
1005          template<typename v1, typename v2>
1006          struct eq_helper<v1, v2, std::enable_if_t<
1007              v1::degree == v2::degree &&
1008              (v1::degree != 0 || v2::degree != 0) &&
1009              (Ring::template eq_v<typename v1::aN, typename v2::aN>)
1010          » {
1011              static constexpr bool value = eq_helper<typename v1::strip, typename v2::strip>::value;
1012          };
1013 
1014          template<typename v1, typename v2>
1015          struct eq_helper<v1, v2, std::enable_if_t<
1016              v1::degree == v2::degree &&
1017              (v1::degree == 0)
1018          » {
1019              static constexpr bool value = Ring::template eq_v<typename v1::aN, typename v2::aN>;
1020          };
1021 
1022          template<typename v1, typename v2, typename E = void>
1023          struct lt_helper {};
1024 
1025          template<typename v1, typename v2>
1026          struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1027              using type = std::true_type;
1028          };
1029 
1030          template<typename v1, typename v2>
1031          struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1032              using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
1033          };
1034 
1035          template<typename v1, typename v2>
1036          struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1037              using type = std::false_type;
1038          };
1039 
1040          template<typename v1, typename v2, typename E = void>
1041          struct gt_helper {};
1042 
1043          template<typename v1, typename v2>
1044          struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1045              using type = std::true_type;
1046          };
1047 
1048          template<typename v1, typename v2>
1049          struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1050              using type = std::false_type;
1051          };
1052 
1053          template<typename v1, typename v2>
1054          struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1055              using type = std::false_type;
1056          };
1057 
1058          // when high power is zero :  strip
1059          template<typename P>
1060          struct simplify<P, std::enable_if_t<
1061              std::is_same<
1062              typename Ring::zero,
```

```
1063            typename P::aN
1064            >::value && (P::degree > 0)
1065        »
1066        {
1067            using type = typename simplify<typename P::strip>::type;
1068        };
1069
1070        // otherwise :  do nothing
1071        template<typename P>
1072        struct simplify<P, std::enable_if_t<
1073            !std::is_same<
1074            typename Ring::zero,
1075            typename P::aN
1076            >::value && (P::degree > 0)
1077        »
1078        {
1079            using type = P;
1080        };
1081
1082        // do not simplify constants
1083        template<typename P>
1084        struct simplify<P, std::enable_if_t<P::degree == 0» {
1085            using type = P;
1086        };
1087
1088        // addition at
1089        template<typename P1, typename P2, size_t index>
1090        struct add_at {
1091            using type =
1092                typename Ring::template add_t<typename P1::template coeff_at_t<index>, typename
      P2::template coeff_at_t<index»;
1093        };
1094
1095        template<typename P1, typename P2, size_t index>
1096        using add_at_t = typename add_at<P1, P2, index>::type;
1097
1098        template<typename P1, typename P2, std::size_t...  I>
1099        struct add_low<P1, P2, std::index_sequence<I...» {
1100            using type = val<add_at_t<P1, P2, I>...>;
1101        };
1102
1103        // substraction at
1104        template<typename P1, typename P2, size_t index>
1105        struct sub_at {
1106            using type =
1107                typename Ring::template sub_t<typename P1::template coeff_at_t<index>, typename
      P2::template coeff_at_t<index»;
1108        };
1109
1110        template<typename P1, typename P2, size_t index>
1111        using sub_at_t = typename sub_at<P1, P2, index>::type;
1112
1113        template<typename P1, typename P2, std::size_t...  I>
1114        struct sub_low<P1, P2, std::index_sequence<I...» {
1115            using type = val<sub_at_t<P1, P2, I>...>;
1116        };
1117
1118        template<typename P1, typename P2>
1119        struct sub {
1120            using type = typename simplify<typename sub_low<
1121            P1,
1122            P2,
1123            internal::make_index_sequence_reverse<
1124            std::max(P1::degree, P2::degree) + 1
1125            »::type>::type;
1126        };
1127
1128        // multiplication at
1129        template<typename v1, typename v2, size_t k, size_t index, size_t stop>
1130        struct mul_at_loop_helper {
1131            using type = typename Ring::template add_t<
1132                typename Ring::template mul_t<
1133                typename v1::template coeff_at_t<index>,
1134                typename v2::template coeff_at_t<k - index>
1135                >,
1136                typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
1137            >;
1138        };
1139
1140        template<typename v1, typename v2, size_t k, size_t stop>
1141        struct mul_at_loop_helper<v1, v2, k, stop, stop> {
1142            using type = typename Ring::template mul_t<typename v1::template coeff_at_t<stop>, typename
      v2::template coeff_at_t<0»;
1143        };
1144
1145        template <typename v1, typename v2, size_t k, typename E = void>
1146        struct mul_at {};
```

```
1147
1148        template<typename v1, typename v2, size_t k>
1149        struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)» {
1150            using type = typename Ring::zero;
1151        };
1152
1153        template<typename v1, typename v2, size_t k>
1154        struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)» {
1155            using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
1156        };
1157
1158        template<typename P1, typename P2, size_t index>
1159        using mul_at_t = typename mul_at<P1, P2, index>::type;
1160
1161        template<typename P1, typename P2, std::size_t...  I>
1162        struct mul_low<P1, P2, std::index_sequence<I...» {
1163            using type = val<mul_at_t<P1, P2, I>...>;
1164        };
1165
1166        // division helper
1167        template< typename A, typename B, typename Q, typename R, typename E = void>
1168        struct div_helper {};
1169
1170        template<typename A, typename B, typename Q, typename R>
1171        struct div_helper<A, B, Q, R, std::enable_if_t<
1172            (R::degree < B::degree) ||
1173            (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
1174            using q_type = Q;
1175            using mod_type = R;
1176            using gcd_type = B;
1177        };
1178
1179        template<typename A, typename B, typename Q, typename R>
1180        struct div_helper<A, B, Q, R, std::enable_if_t<
1181            (R::degree >= B::degree) &&
1182            !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
1183        private:
1184            using rN = typename R::aN;
1185            using bN = typename B::aN;
1186            using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
    B::degree>::type;
1187            using rr = typename sub<R, typename mul<pT, B>::type>::type;
1188            using qq = typename add<Q, pT>::type;
1189
1190        public:
1191            using q_type = typename div_helper<A, B, qq, rr>::q_type;
1192            using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
1193            using gcd_type = rr;
1194        };
1195
1196        template<typename A, typename B>
1197        struct div {
1198            static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
1199            using q_type = typename div_helper<A, B, zero, A>::q_type;
1200            using m_type = typename div_helper<A, B, zero, A>::mod_type;
1201        };
1202
1203
1204        template<typename P>
1205        struct make_unit {
1206            using type = typename div<P, val<typename P::aN»::q_type;
1207        };
1208
1209        template<typename coeff, size_t deg>
1210        struct monomial {
1211            using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
1212        };
1213
1214        template<typename coeff>
1215        struct monomial<coeff, 0> {
1216            using type = val<coeff>;
1217        };
1218
1219        template<typename valueRing, typename P>
1220        struct eval_helper
1221        {
1222            template<size_t index, size_t stop>
1223            struct inner {
1224                static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
1225                    constexpr valueRing coeff = static_cast<valueRing>(P::template coeff_at_t<P::degree
    - index>::template get<valueRing>());
1226                    return eval_helper<valueRing, P>::template inner<index + 1, stop>::func(x * accum +
    coeff, x);
1227                }
1228            };
1229
1230            template<size_t stop>
```

```
1231            struct inner<stop, stop> {
1232                static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
1233                    return accum;
1234                }
1235            };
1236        };
1237
1238        template<typename coeff, typename...  coeffs>
1239        struct string_helper {
1240            static std::string func() {
1241                std::string tail = string_helper<coeffs...>::func();
1242                std::string result = "";
1243                if (Ring::template eq_v<coeff, typename Ring::zero>) {
1244                    return tail;
1245                }
1246                else if (Ring::template eq_v<coeff, typename Ring::one>) {
1247                    if (sizeof...(coeffs) == 1) {
1248                        result += std::string(1, variable_name);
1249                    }
1250                    else {
1251                        result += std::string(1, variable_name) + "^" +
    std::to_string(sizeof...(coeffs));
1252                    }
1253                }
1254                else {
1255                    if (sizeof...(coeffs) == 1) {
1256                        result += coeff::to_string() + " " + std::string(1, variable_name);
1257                    }
1258                    else {
1259                        result += coeff::to_string() + " " + std::string(1, variable_name) + "^" +
    std::to_string(sizeof...(coeffs));
1260                    }
1261                }
1262
1263                if(!tail.empty()) {
1264                    result += " + " + tail;
1265                }
1266
1267                return result;
1268            }
1269        };
1270
1271        template<typename coeff>
1272        struct string_helper<coeff> {
1273            static std::string func() {
1274                if(!std::is_same<coeff, typename Ring::zero>::value) {
1275                    return coeff::to_string();
1276                } else {
1277                    return "";
1278                }
1279            }
1280        };
1281
1282    public:
1285        template<typename P>
1286        using simplify_t = typename simplify<P>::type;
1287
1291        template<typename v1, typename v2>
1292        using add_t = typename add<v1, v2>::type;
1293
1297        template<typename v1, typename v2>
1298        using sub_t = typename sub<v1, v2>::type;
1299
1300        template<typename v1>
1301        using minus_t = sub_t<zero, v1>;
1302
1306        template<typename v1, typename v2>
1307        using mul_t = typename mul<v1, v2>::type;
1308
1312        template<typename v1, typename v2>
1313        static constexpr bool eq_v = eq_helper<v1, v2>::value;
1314
1318        template<typename v1, typename v2>
1319        using lt_t = typename lt_helper<v1, v2>::type;
1320
1324        template<typename v1, typename v2>
1325        static constexpr bool gt_v = gt_helper<v1, v2>::type::value;
1326
1330        template<typename v1, typename v2>
1331        using div_t = typename div<v1, v2>::q_type;
1332
1336        template<typename v1, typename v2>
1337        using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
1338
1342        template<typename coeff, size_t deg>
1343        using monomial_t = typename monomial<coeff, deg>::type;
1344
```

```
1347         template<typename v>
1348         using derive_t = typename derive_helper<v>::type;
1349 
1352         template<typename v>
1353         static constexpr bool pos_v = Ring::template pos_v<typename v::aN>;
1354 
1358         template<typename v1, typename v2>
1359         using gcd_t = std::conditional_t<
1360             Ring::is_euclidean_domain,
1361             typename make_unit<gcd_t<polynomial<Ring, variable_name>, v1, v2»::type,
1362             void>;
1363 
1367         template<auto x>
1368         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
1369 
1373         template<typename v>
1374         using inject_ring_t = val<v>;
1375     };
1376 }
1377 
1378 // big integers
1379 namespace aerobus {
1380     struct bigint {
1381         enum signs {
1382             positive,
1383             negative
1384         };
1385 
1386         template<signs s, uint32_t an, uint32_t...  as>
1387         struct val;
1388 
1389         template<uint32_t an, uint32_t...  as>
1390         struct to_hex_helper {
1391             static std::string func() {
1392                 return std::format("0X{:X}", an) + to_hex_helper<as...>::func();
1393             }
1394         };
1395 
1396         template<uint32_t x>
1397         struct to_hex_helper<x> {
1398             static std::string func() {
1399                 return std::format("{:X}", x);
1400             }
1401         };
1402 
1403     private:
1404 
1405         template<signs s>
1406         struct opposite {
1407             static constexpr signs value = s == signs::positive ?  signs::negative :  signs::positive;
1408         };
1409 
1410         template<signs s>
1411         static constexpr signs opposite_v = opposite<s>::value;
1412 
1413         static std::string to_string(const signs& s) {
1414             switch (s) {
1415                 case signs::negative:
1416                     return "-";
1417                 case signs::positive:
1418                 default:
1419                     return "+";
1420             }
1421         }
1422 
1423         template<signs s1, signs s2>
1424         static constexpr signs mul_sign() {
1425             if constexpr (s1 == signs::positive) {
1426                 return s2;
1427             }
1428 
1429             return opposite_v<s2>;
1430         }
1431 
1432         template<size_t ss, signs s, uint32_t aN, uint32_t...  as>
1433         struct shift_left_helper {
1434             using type = typename shift_left_helper<ss-1, s, aN, as..., 0>::type;
1435         };
1436 
1437         template<signs s, uint32_t aN, uint32_t...  as>
1438         struct shift_left_helper<0, s, aN, as...>
1439         {
1440             using type = val<s, aN, as...>;
1441         };
1442 
1443     public:
1444 
```

```
1445         template<signs s, uint32_t an, uint32_t...  as>
1446         struct val {
1447             template<size_t ss>
1448             using shift_left = typename shift_left_helper<ss, s, an, as...>::type;
1449             static constexpr signs sign = s;
1450 
1451             template<size_t index, typename E = void>
1452             struct digit_at {};
1453 
1454             template<size_t index>
1455             struct digit_at<index, std::enable_if_t<(index <= sizeof...(as))» {
1456                 static constexpr uint32_t value = internal::value_at<(sizeof...(as) - index), an,
     as...>::value;
1457             };
1458 
1459             template<size_t index>
1460             struct digit_at<index, std::enable_if_t<(index > sizeof...(as))» {
1461                 static constexpr uint32_t value = 0;
1462             };
1463 
1464             using strip = val<s, as...>;
1465             static constexpr uint32_t aN = an;
1466             static constexpr size_t digits = sizeof...(as) + 1;
1467 
1468             static std::string to_string() {
1469                 return bigint::to_string(s) + std::to_string(aN) + "B^" + std::to_string(digits-1) + "
     + " + strip::to_string();
1470             }
1471 
1472             static std::string to_hex() {
1473                 return bigint::to_string(s) + to_hex_helper<an, as...>::func();
1474             }
1475 
1476             static constexpr bool is_zero_v = sizeof...(as) == 0 && an == 0;
1477 
1478             using minus_t = val<opposite_v<s>, an, as...>;
1479         };
1480 
1481         template<signs s, uint32_t a0>
1482         struct val<s, a0> {
1483             template<size_t ss>
1484             using shift_left = typename shift_left_helper<ss, s, a0>::type;
1485             static constexpr signs sign = s;
1486             static constexpr uint32_t aN = a0;
1487             static constexpr size_t digits = 1;
1488             template<size_t index, typename E = void>
1489             struct digit_at {};
1490             template<size_t index>
1491             struct digit_at<index, std::enable_if_t<index == 0» {
1492                 static constexpr uint32_t value = a0;
1493             };
1494 
1495             template<size_t index>
1496             struct digit_at<index, std::enable_if_t<index != 0» {
1497                 static constexpr uint32_t value = 0;
1498             };
1499 
1500             static std::string to_string() {
1501                 return bigint::to_string(s) + std::to_string(a0);
1502             }
1503 
1504             static std::string to_hex() {
1505                 return bigint::to_string(s) + std::format("0X{:X}", a0);
1506             }
1507 
1508             static constexpr bool is_zero_v = a0 == 0;
1509 
1510             using minus_t = val<opposite_v<s>, a0>;
1511 
1512         };
1513 
1514         using zero = val<signs::positive, 0>;
1515         using one = val<signs::positive, 1>;
1516 
1517     private:
1518 
1519         template<typename I, typename E = void>
1520         struct simplify {};
1521 
1522         template<typename I>
1523         struct simplify<I, std::enable_if_t<I::digits == 1 && I::aN != 0» {
1524             using type = I;
1525         };
1526 
1527         template<typename I>
1528         struct simplify<I, std::enable_if_t<I::digits == 1 && I::aN == 0» {
1529             using type = zero;
```

```
1530         };
1531
1532         template<typename I>
1533         struct simplify<I, std::enable_if_t<I::digits != 1 && I::aN == 0» {
1534             using type = typename simplify<typename I::strip>::type;
1535         };
1536
1537         template<typename I>
1538         struct simplify<I, std::enable_if_t<I::digits != 1 && I::aN != 0» {
1539             using type = I;
1540         };
1541
1542         template<uint32_t x, uint32_t y, uint8_t carry_in = 0>
1543         struct add_digit_helper {
1544         private:
1545             static constexpr uint64_t raw = ((uint64_t) x + (uint64_t) y + (uint64_t) carry_in);
1546         public:
1547             static constexpr uint32_t value = (uint32_t)(raw & 0xFFFF'FFFF);
1548             static constexpr uint8_t carry_out = (uint32_t) (raw » 32);
1549         };
1550
1551         template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1552         struct add_at_helper {
1553         private:
1554             static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1555             static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1556         public:
1557             static constexpr uint32_t value = add_digit_helper<d1, d2, carry_in>::value;
1558             static constexpr uint8_t carry_out = add_digit_helper<d1, d2, carry_in>::carry_out;
1559         };
1560
1561         template<uint32_t x, uint32_t y, uint8_t carry_in, typename E = void>
1562         struct sub_digit_helper {};
1563
1564         // x - y
1565         template<uint32_t x, uint32_t y, uint8_t carry_in>
1566         struct sub_digit_helper<x, y, carry_in, std::enable_if_t<
1567                 (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in) > x)
1568         » {
1569
1570             static constexpr uint32_t value = static_cast<uint32_t>(
1571                 static_cast<uint32_t>(x) + 0x1'0000'0000UL - (static_cast<uint64_t>(y) +
     static_cast<uint64_t>(carry_in))
1572             );
1573             static constexpr uint8_t carry_out = 1;
1574         };
1575
1576         template<uint32_t x, uint32_t y, uint8_t carry_in>
1577         struct sub_digit_helper<x, y, carry_in, std::enable_if_t<
1578                 (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in) <= x)
1579         » {
1580
1581             static constexpr uint32_t value = static_cast<uint32_t>(
1582                 static_cast<uint64_t>(x) - (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in))
1583             );
1584             static constexpr uint8_t carry_out = 0;
1585         };
1586
1587         template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1588         struct sub_at_helper {
1589         private:
1590             static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1591             static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1592             using tmp = sub_digit_helper<d1, d2, carry_in>;
1593         public:
1594             static constexpr uint32_t value = tmp::value;
1595             static constexpr uint8_t carry_out = tmp::carry_out;
1596         };
1597
1598         template<uint32_t x, uint32_t y, uint32_t carry_in>
1599         struct mul_digit_helper {
1600         private:
1601             static constexpr uint64_t tmp = static_cast<uint64_t>(x) * static_cast<uint64_t>(y) +
     static_cast<uint64_t>(carry_in);
1602         public:
1603             static constexpr uint32_t value = static_cast<uint32_t>(tmp & 0xFFFF'FFFFU);
1604             static constexpr uint32_t carry_out = static_cast<uint32_t>(tmp » 32);
1605         };
1606
1607         template<typename I1, uint32_t d2, size_t index, uint32_t carry_in = 0>
1608         struct mul_at_helper {
1609         private:
1610             static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1611             using tmp = mul_digit_helper<d1, d2, carry_in>;
1612         public:
1613             static constexpr uint32_t value = tmp::value;
1614             static constexpr uint32_t carry_out = tmp::carry_out;
```

```
1615        };
1616
1617        template<typename I1, typename I2, size_t index>
1618        struct add_low_helper {
1619            private:
1620            using helper = add_at_helper<I1, I2, index, add_low_helper<I1, I2, index-1>::carry_out>;
1621            public:
1622            static constexpr uint32_t digit = helper::value;
1623            static constexpr uint8_t carry_out = helper::carry_out;
1624        };
1625
1626        template<typename I1, typename I2>
1627        struct add_low_helper<I1, I2, 0> {
1628            static constexpr uint32_t digit = add_at_helper<I1, I2, 0, 0>::value;
1629            static constexpr uint32_t carry_out = add_at_helper<I1, I2, 0, 0>::carry_out;
1630        };
1631
1632        template<typename I1, typename I2, size_t index>
1633        struct sub_low_helper {
1634            private:
1635            using helper = sub_at_helper<I1, I2, index, sub_low_helper<I1, I2, index-1>::carry_out>;
1636            public:
1637            static constexpr uint32_t digit = helper::value;
1638            static constexpr uint8_t carry_out = helper::carry_out;
1639        };
1640
1641        template<typename I1, typename I2>
1642        struct sub_low_helper<I1, I2, 0> {
1643            static constexpr uint32_t digit = sub_at_helper<I1, I2, 0, 0>::value;
1644            static constexpr uint32_t carry_out = sub_at_helper<I1, I2, 0, 0>::carry_out;
1645        };
1646
1647        template<typename I1, uint32_t d2, size_t index>
1648        struct mul_low_helper {
1649            private:
1650            using helper = mul_at_helper<I1, d2, index, mul_low_helper<I1, d2, index-1>::carry_out>;
1651            public:
1652            static constexpr uint32_t digit = helper::value;
1653            static constexpr uint32_t carry_out = helper::carry_out;
1654        };
1655
1656        template<typename I1, uint32_t d2>
1657        struct mul_low_helper<I1, d2, 0> {
1658            static constexpr uint32_t digit = mul_at_helper<I1, d2, 0, 0>::value;
1659            static constexpr uint32_t carry_out = mul_at_helper<I1, d2, 0, 0>::carry_out;
1660        };
1661
1662        template<typename I1, uint32_t d2, typename I>
1663        struct mul_low {};
1664
1665        template<typename I1, uint32_t d2, std::size_t...  I>
1666        struct mul_low<I1, d2, std::index_sequence<I...>> {
1667            using type = val<signs::positive, mul_low_helper<I1, d2, I>::digit...>;
1668        };
1669
1670        template<typename I1, uint32_t d2, size_t shift>
1671        struct mul_row_helper {
1672            using type = typename simplify<
1673                typename mul_low<
1674                        I1,
1675                        d2,
1676                        typename internal::make_index_sequence_reverse<I1::digits + 1>
1677                    >::type>::type::template shift_left<shift>;
1678        };
1679
1680        template<typename I1, typename I2, size_t index>
1681        struct mul_row {
1682        private:
1683            static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1684        public:
1685            using type = typename mul_row_helper<I1, d2, index>::type;
1686        };
1687
1688        template<typename I1, typename...  Is>
1689        struct vadd;
1690
1691        template<typename I1, typename I2, typename E = void>
1692        struct eq;
1693
1694        template<typename I1, typename I2, typename I>
1695        struct mul_helper {};
1696
1697        template<typename I1, typename I2, std::size_t...  I>
1698        struct mul_helper<I1, I2, std::index_sequence<I...>> {
1699            using type = typename vadd<typename mul_row<I1, I2, I>::type...>::type;
1700        };
1701
```

```
1702        template<typename I, size_t index>
1703        struct div_2_digit {
1704            static constexpr uint32_t value = ((I::template digit_at<index + 1>::value & 1) « 31) +
    (I::template digit_at<index>::value » 1);
1705        };
1706
1707        template<typename X, typename I>
1708        struct div_2_helper {};
1709
1710        template<typename X, std::size_t...  I>
1711        struct div_2_helper<X, std::index_sequence<I...» {
1712            using type = val<signs::positive, div_2_digit<X, I>::value...>;
1713        };
1714
1715        template<typename X>
1716        struct div_2 {
1717            using type = typename simplify<typename div_2_helper<X,
    internal::make_index_sequence_reverse<X::digits»::type>::type;
1718        };
1719
1720        template<typename I1, typename I2, typename E = void>
1721        struct mul {};
1722
1723        template<typename I1, typename I2>
1724        struct mul<I1, I2, std::enable_if_t<
1725            I1::is_zero_v || I2::is_zero_v
1726        » {
1727            using type = zero;
1728        };
1729
1730        template<typename I1, typename I2>
1731        struct mul<I1, I2, std::enable_if_t<
1732            !I1::is_zero_v && !I2::is_zero_v && eq<I1, one>::value
1733        » {
1734            using type = I2;
1735        };
1736
1737        template<typename I1, typename I2>
1738        struct mul<I1, I2, std::enable_if_t<
1739            !I1::is_zero_v && !I2::is_zero_v && !eq<I1, one>::value && eq<I2, one>::value
1740        » {
1741            using type = I1;
1742        };
1743
1744        template<typename I1, typename I2>
1745        struct mul<I1, I2, std::enable_if_t<
1746            !I1::is_zero_v && !I2::is_zero_v && !eq<I1, one>::value && !eq<I2, one>::value
1747        » {
1748        private:
1749            static constexpr signs sign = mul_sign<I1::sign, I2::sign>();
1750            using tmp =
1751                typename simplify<
1752                    typename mul_helper<I1, I2, internal::make_index_sequence_reverse<I1::digits *
    I2::digits + 1»::type
1753                >::type;
1754        public:
1755            using type = std::conditional_t<sign == signs::positive, tmp, typename tmp::minus_t>;
1756        };
1757
1758        template<typename I1, typename I2, typename I>
1759        struct add_low {};
1760
1761        template<typename I1, typename I2, std::size_t...  I>
1762        struct add_low<I1, I2, std::index_sequence<I...» {
1763            using type = val<signs::positive, add_low_helper<I1, I2, I>::digit...>;
1764        };
1765
1766        template<typename I1, typename I2, typename I>
1767        struct sub_low {};
1768
1769        template<typename I1, typename I2, std::size_t...  I>
1770        struct sub_low<I1, I2, std::index_sequence<I...» {
1771            using type = val<signs::positive, sub_low_helper<I1, I2, I>::digit...>;
1772        };
1773
1774        template<typename I1, typename I2, typename E>
1775        struct eq {};
1776
1777        template<typename I1, typename I2>
1778        struct eq<I1, I2, std::enable_if_t<I1::digits != I2::digits» {
1779            static constexpr bool value = false;
1780        };
1781
1782        template<typename I1, typename I2>
1783        struct eq<I1, I2, std::enable_if_t<I1::digits == I2::digits && I1::digits == 1» {
1784            static constexpr bool value = (I1::is_zero_v && I2::is_zero_v) || (I1::sign == I2::sign &&
    I1::aN == I2::aN);
```

```
1785         };
1786
1787         template<typename I1, typename I2>
1788         struct eq<I1, I2, std::enable_if_t<I1::digits == I2::digits && I1::digits != 1» {
1789             static constexpr bool value =
1790                 I1::sign == I2::sign &&
1791                 I1::aN == I2::aN &&
1792                 eq<typename I1::strip, typename I2::strip>::value;
1793         };
1794
1795         template<typename I1, typename I2, typename E = void>
1796         struct gt_helper {};
1797
1798         template<typename I1, typename I2>
1799         struct gt_helper<I1, I2, std::enable_if_t<eq<I1, I2>::value» {
1800             static constexpr bool value = false;
1801         };
1802
1803         template<typename I1, typename I2>
1804         struct gt_helper<I1, I2, std::enable_if_t<!eq<I1, I2>::value && I1::sign != I2::sign» {
1805             static constexpr bool value = I1::sign == signs::positive;
1806         };
1807
1808         template<typename I1, typename I2>
1809         struct gt_helper<I1, I2,
1810             std::enable_if_t<
1811                 !eq<I1, I2>::value &&
1812                 I1::sign == I2::sign &&
1813                 I1::sign == signs::negative
1814             » {
1815             static constexpr bool value = gt_helper<typename I2::minus_t, typename I1::minus_t>::value;
1816         };
1817
1818         template<typename I1, typename I2>
1819         struct gt_helper<I1, I2,
1820             std::enable_if_t<
1821                 !eq<I1, I2>::value &&
1822                 I1::sign == I2::sign &&
1823                 I1::sign == signs::positive &&
1824                 (I1::digits > I2::digits)
1825             » {
1826             static constexpr bool value = true;
1827         };
1828
1829         template<typename I1, typename I2>
1830         struct gt_helper<I1, I2,
1831             std::enable_if_t<
1832                 !eq<I1, I2>::value &&
1833                 I1::sign == I2::sign &&
1834                 I1::sign == signs::positive &&
1835                 (I1::digits < I2::digits)
1836             » {
1837             static constexpr bool value = false;
1838         };
1839
1840         template<typename I1, typename I2>
1841         struct gt_helper<I1, I2,
1842             std::enable_if_t<
1843                 !eq<I1, I2>::value &&
1844                 I1::sign == I2::sign &&
1845                 I1::sign == signs::positive &&
1846                 (I1::digits == I2::digits) && I1::digits == 1
1847             » {
1848             static constexpr bool value = I1::aN > I2::aN;
1849         };
1850
1851         template<typename I1, typename I2>
1852         struct gt_helper<I1, I2,
1853             std::enable_if_t<
1854                 !eq<I1, I2>::value &&
1855                 I1::sign == I2::sign &&
1856                 I1::sign == signs::positive &&
1857                 (I1::digits == I2::digits) && I1::digits != 1 && (I1::aN > I2::aN)
1858             » {
1859             static constexpr bool value = true;
1860         };
1861
1862         template<typename I1, typename I2>
1863         struct gt_helper<I1, I2,
1864             std::enable_if_t<
1865                 !eq<I1, I2>::value &&
1866                 I1::sign == I2::sign &&
1867                 I1::sign == signs::positive &&
1868                 (I1::digits == I2::digits) && I1::digits != 1 && (I1::aN < I2::aN)
1869             » {
1870             static constexpr bool value = false;
1871         };
```

```
1872
1873        template<typename I1, typename I2>
1874        struct gt_helper<I1, I2,
1875            std::enable_if_t<
1876                !eq<I1, I2>::value &&
1877                I1::sign == I2::sign &&
1878                I1::sign == signs::positive &&
1879                (I1::digits == I2::digits) && I1::digits != 1 && I1::aN == I2::aN
1880            » {
1881            static constexpr bool value = gt_helper<typename I1::strip, typename I2::strip>::value;
1882        };
1883
1884
1885
1886        template<typename I1, typename I2, typename E = void>
1887        struct add {};
1888
1889        template<typename I1, typename I2, typename E = void>
1890        struct sub {};
1891
1892        // +x + +y -> x + y
1893        template<typename I1, typename I2>
1894        struct add<I1, I2, std::enable_if_t<
1895            gt_helper<I1, zero>::value &&
1896            gt_helper<I2, zero>::value
1897        » {
1898            using type = typename simplify<
1899                typename add_low<
1900                        I1,
1901                        I2,
1902                        typename internal::make_index_sequence_reverse<std::max(I1::digits, I2::digits)
    + 1>
1903                    >::type>::type;
1904        };
1905
1906        // -x + -y -> -(x+y)
1907        template<typename I1, typename I2>
1908        struct add<I1, I2, std::enable_if_t<
1909            gt_helper<zero, I1>::value &&
1910            gt_helper<zero, I2>::value
1911        » {
1912            using type = typename add<typename I1::minus_t, typename I2::minus_t>::type::minus_t;
1913        };
1914
1915        // 0 + x -> x
1916        template<typename I1, typename I2>
1917        struct add<I1, I2, std::enable_if_t<
1918            I1::is_zero_v
1919        » {
1920            using type = I2;
1921        };
1922
1923        // x + 0 -> x
1924        template<typename I1, typename I2>
1925        struct add<I1, I2, std::enable_if_t<
1926            I2::is_zero_v
1927        » {
1928            using type = I1;
1929        };
1930
1931        // x + (-y) -> x - y
1932        template<typename I1, typename I2>
1933        struct add<I1, I2, std::enable_if_t<
1934            !I1::is_zero_v && !I2::is_zero_v &&
1935            gt_helper<I1, zero>::value &&
1936            gt_helper<zero, I2>::value
1937        » {
1938            using type = typename sub<I1, typename I2::minus_t>::type;
1939        };
1940
1941        // -x + y -> y - x
1942        template<typename I1, typename I2>
1943        struct add<I1, I2, std::enable_if_t<
1944            !I1::is_zero_v && !I2::is_zero_v &&
1945            gt_helper<zero, I1>::value &&
1946            gt_helper<I2, zero>::value
1947        » {
1948            using type = typename sub<I2, typename I1::minus_t>::type;
1949        };
1950
1951        // I1 == I2
1952        template<typename I1, typename I2>
1953        struct sub<I1, I2, std::enable_if_t<
1954            eq<I1, I2>::value
1955        » {
1956            using type = zero;
1957        };
```

```
1958
1959        // I1 != I2, I2 == 0
1960        template<typename I1, typename I2>
1961        struct sub<I1, I2, std::enable_if_t<
1962            !eq<I1, I2>::value &&
1963            eq<I2, zero>::value
1964        » {
1965            using type = I1;
1966        };
1967
1968        // I1 != I2, I1 == 0
1969        template<typename I1, typename I2>
1970        struct sub<I1, I2, std::enable_if_t<
1971            !eq<I1, I2>::value &&
1972            eq<I1, zero>::value
1973        » {
1974            using type = typename I2::minus_t;
1975        };
1976
1977        // 0 < I2 < I1
1978        template<typename I1, typename I2>
1979        struct sub<I1, I2, std::enable_if_t<
1980            gt_helper<I2, zero>::value &&
1981            gt_helper<I1, I2>::value
1982        » {
1983            using type = typename simplify<
1984                typename sub_low<
1985                        I1,
1986                        I2,
1987                        typename internal::make_index_sequence_reverse<std::max(I1::digits, I2::digits)
     + 1>
1988                    >::type>::type;
1989        };
1990
1991        // 0 < I1 < I2
1992        template<typename I1, typename I2>
1993        struct sub<I1, I2, std::enable_if_t<
1994            gt_helper<I1, zero>::value &&
1995            gt_helper<I2, I1>::value
1996        » {
1997            using type = typename sub<I2, I1>::type::minus_t;
1998        };
1999
2000        // I2 < I1 < 0
2001        template<typename I1, typename I2>
2002        struct sub<I1, I2, std::enable_if_t<
2003            gt_helper<zero, I1>::value &&
2004            gt_helper<I1, I2>::value
2005        » {
2006            using type = typename sub<typename I2::minus_t, typename I1::minus_t>::type;
2007        };
2008
2009        // I1 < I2 < 0
2010        template<typename I1, typename I2>
2011        struct sub<I1, I2, std::enable_if_t<
2012            gt_helper<zero, I2>::value &&
2013            gt_helper<I2, I1>::value
2014        » {
2015            using type = typename sub<typename I1::minus_t, typename I2::minus_t>::type::minus_t;
2016        };
2017
2018        // I2 < 0 < I1
2019        template<typename I1, typename I2>
2020        struct sub<I1, I2, std::enable_if_t<
2021            gt_helper<zero, I2>::value &&
2022            gt_helper<I1, zero>::value
2023        » {
2024            using type = typename add<I1, typename I2::minus_t>::type;
2025        };
2026
2027        // I1 < 0 < I2
2028        template<typename I1, typename I2>
2029        struct sub<I1, I2, std::enable_if_t<
2030            gt_helper<zero, I1>::value &&
2031            gt_helper<I2, zero>::value
2032        » {
2033            using type = typename add<I2, typename I1::minus_t>::type::minus_t;
2034        };
2035
2036        // useful for multiplication
2037        template<typename I1, typename...  Is>
2038        struct vadd {
2039            using type = typename add<I1, typename vadd<Is...>::type>::type;
2040        };
2041
2042        template<typename I1, typename I2>
2043        struct vadd<I1, I2> {
```

```
2044            using type = typename add<I1, I2>::type;
2045        };
2046
2047        template<typename I, size_t s, typename E = void>
2048        struct shift_right_helper { };
2049
2050        template<typename I, size_t s>
2051        struct shift_right_helper<I, s, std::enable_if_t<(s >= I::digits)» {
2052            using type = zero;
2053        };
2054
2055        template<typename I, size_t s>
2056        struct shift_right_helper<I, s, std::enable_if_t<(s == 0)» {
2057            using type = I;
2058        };
2059
2060        template<typename I, size_t s>
2061        struct shift_right_helper<I, s, std::enable_if_t<(s != 0) && (s < I::digits)» {
2062        private:
2063            using digit = val<I::sign, I::template digit_at<s>::value>;
2064            using tmp = typename shift_right_helper<I, s + 1>::type;
2065        public:
2066            using type = typename add<
2067                digit,
2068                typename tmp::template shift_left<1>
2069            >::type;
2070        };
2071
2072        template<typename A, typename B, typename E = void>
2073        struct floor_helper {};
2074
2075        template<typename A, typename B>
2076        struct floor_helper<A, B, std::enable_if_t<gt_helper<B, A>::value» {
2077            using type = zero;
2078        };
2079
2080        template<typename A, typename B>
2081        struct floor_helper<A, B, std::enable_if_t<eq<A, B>::value» {
2082            using type = one;
2083        };
2084
2085        template<typename A, typename B>
2086        struct floor_helper<A, B, std::enable_if_t<gt_helper<A, B>::value && (A::digits == 1 &&
    B::digits == 1)» {
2087            using type = val<signs::positive, A::aN / B::aN>;
2088        };
2089
2090        template<typename A, typename B>
2091        struct floor_helper<A, B, std::enable_if_t<gt_helper<A, B>::value && (A::digits != 1 ||
    B::digits != 1)» {
2092            static_assert(A::sign == signs::positive);
2093            static_assert(B::sign == signs::positive);
2094            static constexpr size_t N = A::digits;
2095            static constexpr size_t K = B::digits;
2096            static constexpr size_t min_shift = N >= K + 1 ?  N - K - 1 :  0;
2097
2098            using from = typename one::template shift_left<min_shift>;
2099            using to = typename one::template shift_left<N - K + 1>;
2100
2101            template<typename X, typename Y>
2102            using average_t = typename div_2<typename add<X, Y>::type>::type;
2103
2104            template<typename lowerbound, typename upperbound, typename E = void>
2105            struct inner {};
2106
2107            template<typename lowerbound, typename upperbound>
2108            struct inner<lowerbound, upperbound, std::enable_if_t<eq<
2109                    typename add<lowerbound, one>::type, upperbound>::value
2110            » {
2111                using type = lowerbound;
2112            };
2113
2114            template<typename lowerbound, typename upperbound>
2115            struct inner<lowerbound, upperbound, std::enable_if_t<
2116                gt_helper<upperbound, typename add<lowerbound, one>::type>::value &&
2117                gt_helper<typename mul<average_t<upperbound, lowerbound>, B>::type, A>::value
2118                » {
2119                using type = typename simplify<typename inner<lowerbound, average_t<upperbound,
    lowerbound»::type>::type;
2120            };
2121
2122            template<typename lowerbound, typename upperbound>
2123            struct inner<lowerbound, upperbound, std::enable_if_t<
2124                gt_helper<upperbound, typename add<lowerbound, one>::type>::value &&
2125                !gt_helper<typename mul<average_t<upperbound, lowerbound>, B>::type, A>::value
2126                » {
2127                using type = typename simplify<typename inner<average_t<upperbound, lowerbound>,
```

```
    upperbound>::type>::type;
2128            };
2129
2130            using type = typename inner<from, to>::type;
2131        };
2132
2133        template<typename N, typename M, int64_t i>
2134        struct div_helper_inner {
2135            static_assert(N::sign == signs::positive);
2136            static_assert(M::sign == signs::positive);
2137            static constexpr size_t l = M::digits;
2138            static constexpr size_t k = N::digits;
2139            using Qm1 = typename div_helper_inner<N, M, i - 1>::Q;
2140            using Rm1 = typename div_helper_inner<N, M, i - 1>::R;
2141            using D = typename add<
2142                typename Rm1::template shift_left<1>,
2143                val<signs::positive, N::template digit_at<k-(i + l)>::value>
2144            >::type;
2145            using Beta = typename floor_helper<D, M>::type;
2146            using Q = typename simplify<typename add<typename Qm1::template shift_left<1>,
    Beta>::type>::type;
2147
2148            using R = typename simplify<typename sub<D, typename mul<M, Beta>::type>::type>::type;
2149        };
2150
2151        template<typename N, typename M>
2152        struct div_helper_inner<N, M, -1> {
2153            static_assert(N::sign == signs::positive);
2154            static_assert(M::sign == signs::positive);
2155            static constexpr size_t l = M::digits;
2156            static constexpr size_t k = N::digits;
2157            using Q = zero;
2158            using R = typename shift_right_helper<N, k - l + 1>::type; // first l-1 digits of N
2159        };
2160
2161        template<typename N, typename M, typename E = void>
2162        struct div_helper {};
2163
2164        template<typename N, typename M>
2165        struct div_helper<N, M, std::enable_if_t<
2166                M::sign == signs::positive &&
2167                N::sign == signs::positive &&
2168                !M::is_zero_v
2169        » {
2170            static constexpr size_t l = M::digits;
2171            static constexpr size_t k = N::digits;
2172            using Q = typename simplify<typename div_helper_inner<N, M, k - l>::Q>::type;
2173            using R = typename simplify<typename div_helper_inner<N, M, k - l>::R>::type;
2174        };
2175
2176        template<typename N, typename M>
2177        struct div_helper<N, M, std::enable_if_t<
2178                M::sign == signs::negative &&
2179                !M::is_zero_v
2180        » {
2181            using tmp = div_helper<N, typename M::minus_t>;
2182            using Q = typename tmp::Q::minus_t;
2183            using R = typename tmp::R;
2184        };
2185
2186        template<typename N, typename M>
2187        struct div_helper<N, M, std::enable_if_t<
2188                N::sign == signs::negative &&
2189                !M::is_zero_v
2190        » {
2191            using tmp = div_helper<typename N::minus_t, M>;
2192            using R_i = typename simplify<typename tmp::R>::type;
2193            using Q_i = typename simplify<typename tmp::Q>::type;
2194            using Q = std::conditional_t<R_i::is_zero_v, typename Q_i::minus_t, typename sub<typename
    Q_i::minus_t, one>::type>;
2195            using R = std::conditional_t<R_i::is_zero_v, zero, typename sub<M, R_i>::type>;
2196        };
2197
2198        template<string_literal S>
2199        struct digit_from_string {
2200            static constexpr size_t N = S.len();
2201
2202            template<size_t i>
2203            static constexpr char char_at = (i < N) ? S.template char_at<i>() :  '0';
2204
2205            template <char c>
2206            static constexpr uint32_t from_hex = (c >= '0' && c <= '9') ?    c - '0' :  10 + c - 'A';
2207
2208            template<size_t index>
2209            static constexpr uint32_t value() {
2210                constexpr uint32_t d1 = from_hex<char_at<8*index + 1»;
2211                constexpr uint32_t d2 = from_hex<char_at<8*index + 2» « 4;
```

```
2212                 constexpr uint32_t d3 = from_hex<char_at<8*index + 3» « 8;
2213                 constexpr uint32_t d4 = from_hex<char_at<8*index + 4» « 12;
2214                 constexpr uint32_t d5 = from_hex<char_at<8*index + 5» « 16;
2215                 constexpr uint32_t d6 = from_hex<char_at<8*index + 6» « 20;
2216                 constexpr uint32_t d7 = from_hex<char_at<8*index + 7» « 24;
2217                 constexpr uint32_t d8 = from_hex<char_at<8*index + 8» « 28;
2218                 return d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8;
2219             }
2220         };
2221 
2222         template<string_literal S, typename I>
2223         struct from_hex_helper {};
2224 
2225         template<string_literal S, std::size_t...  I>
2226         struct from_hex_helper<S, std::index_sequence<I...» {
2227             using type = typename simplify<val<signs::positive, digit_from_string<S>::template
     value<I>()...»::type;
2228         };
2229 
2230     public:
2231         static constexpr bool is_euclidean_domain = true;
2232 
2235         template<string_literal S>
2236         using from_hex_t = typename from_hex_helper<S,
     internal::make_index_sequence_reverse<(S.len()-1) / 8 + 1»::type;
2237 
2239         template<typename I>
2240         using minus_t = typename I::minus_t;
2241 
2243         template<typename I1, typename I2>
2244         static constexpr bool eq_v = eq<I1, I2>::value;
2245 
2247         template<typename I>
2248         static constexpr bool pos_v = I::sign == signs::positive && !I::is_zero_v;
2249 
2251         template<typename I1, typename I2>
2252         static constexpr bool gt_v = gt_helper<I1, I2>::value;
2253 
2255         template<typename I1, typename I2>
2256         static constexpr bool ge_v = eq_v<I1, I2> || gt_v<I1, I2>;
2257 
2259         template<typename I>
2260         using simplify_t = typename simplify<I>::type;
2261 
2263         template<typename I1, typename I2>
2264         using add_t = typename add<I1, I2>::type;
2265 
2267         template<typename I1, typename I2>
2268         using sub_t = typename sub<I1, I2>::type;
2269 
2271         template<typename I, size_t s>
2272         using shift_left_t = typename I::template shift_left<s>;
2273 
2275         template<typename I, size_t s>
2276         using shift_right_t = typename shift_right_helper<I, s>::type;
2277 
2279         template<typename I1, typename I2>
2280         using mul_t = typename mul<I1, I2>::type;
2281 
2283         template<typename...  Is>
2284         using vadd_t = typename vadd<Is...>::type;
2285 
2287         template<typename I>
2288         using div_2_t = typename div_2<I>::type;
2289 
2291         template<typename I1, typename I2>
2292         using floor_t = typename floor_helper<I1, I2>::type;
2293 
2295         template<typename I1, typename I2>
2296         using div_t = typename div_helper<I1, I2>::Q;
2297 
2299         template<typename I1, typename I2>
2300         using mod_t = typename div_helper<I1, I2>::R;
2301 
2303         template<typename I1, typename I2>
2304         using gcd_t = gcd_t<bigint, I1, I2>;
2305 
2307         template<typename I1, typename I2, typename I3>
2308         using fma_t = add_t<mul_t<I1, I2>, I3>;
2309 
2310 
2311     };
2312 }
2313 
2314 // fraction field
2315 namespace aerobus {
2316     namespace internal {
```

```
2317         template<typename Ring, typename E = void>
2318         requires IsEuclideanDomain<Ring>
2319         struct _FractionField {};
2320 
2321         template<typename Ring>
2322         requires IsEuclideanDomain<Ring>
2323         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain»
2324         {
2326             static constexpr bool is_field = true;
2327             static constexpr bool is_euclidean_domain = true;
2328 
2329             private:
2330             template<typename val1, typename val2, typename E = void>
2331             struct to_string_helper {};
2332 
2333             template<typename val1, typename val2>
2334             struct to_string_helper <val1, val2,
2335                 std::enable_if_t<
2336                 Ring::template eq_v<val2, typename Ring::one>
2337                 » {
2338                 static std::string func() {
2339                     return val1::to_string();
2340                 }
2341             };
2342 
2343             template<typename val1, typename val2>
2344             struct to_string_helper<val1, val2,
2345                 std::enable_if_t<
2346                 !Ring::template eq_v<val2,typename Ring::one>
2347                 » {
2348                 static std::string func() {
2349                     return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
2350                 }
2351             };
2352 
2353             public:
2357             template<typename val1, typename val2>
2358             struct val {
2359                 using x = val1;
2360                 using y = val2;
2361 
2363                 static constexpr bool is_zero_v = val1::is_zero_v;
2364                 using ring_type = Ring;
2365                 using field_type = _FractionField<Ring>;
2366 
2368                 static constexpr bool is_integer = std::is_same<val2, typename Ring::one>::value;
2369 
2373                 template<typename valueType>
2374                 static constexpr valueType get() { return static_cast<valueType>(x::v) /
     static_cast<valueType>(y::v); }
2375 
2378                 static std::string to_string() {
2379                     return to_string_helper<val1, val2>::func();
2380                 }
2381 
2386                 template<typename valueRing>
2387                 static constexpr valueRing eval(const valueRing& v) {
2388                     return x::eval(v) / y::eval(v);
2389                 }
2390             };
2391 
2393             using zero = val<typename Ring::zero, typename Ring::one>;
2395             using one = val<typename Ring::one, typename Ring::one>;
2396 
2399             template<typename v>
2400             using inject_t = val<v, typename Ring::one>;
2401 
2404             template<auto x>
2405             using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
     Ring::one>;
2406 
2409             template<typename v>
2410             using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
2411 
2412             using ring_type = Ring;
2413 
2414         private:
2415             template<typename v, typename E = void>
2416             struct simplify {};
2417 
2418             // x = 0
2419             template<typename v>
2420             struct simplify<v, std::enable_if_t<v::x::is_zero_v» {
2421                 using type = typename _FractionField<Ring>::zero;
2422             };
2423 
2424             // x != 0
```

```
2425            template<typename v>
2426            struct simplify<v, std::enable_if_t<!v::x::is_zero_v» {
2427
2428            private:
2429                using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
2430                using newx = typename Ring::template div_t<typename v::x, _gcd>;
2431                using newy = typename Ring::template div_t<typename v::y, _gcd>;
2432
2433                using posx = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
    minus_t<newx>, newx>;
2434                using posy = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
    minus_t<newy>, newy>;
2435            public:
2436                using type = typename _FractionField<Ring>::template val<posx, posy>;
2437            };
2438
2439        public:
2442            template<typename v>
2443            using simplify_t = typename simplify<v>::type;
2444
2445        private:
2446
2447            template<typename v1, typename v2>
2448            struct add {
2449            private:
2450                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2451                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2452                using dividend = typename Ring::template add_t<a, b>;
2453                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2454                using g = typename Ring::template gcd_t<dividend, diviser>;
2455
2456            public:
2457                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
2458            };
2459
2460            template<typename v>
2461            struct pos {
2462                using type = std::conditional_t<
2463                    (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
2464                    (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
2465                    std::true_type,
2466                    std::false_type>;
2467
2468            };
2469
2470            template<typename v1, typename v2>
2471            struct sub {
2472            private:
2473                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2474                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2475                using dividend = typename Ring::template sub_t<a, b>;
2476                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2477                using g = typename Ring::template gcd_t<dividend, diviser>;
2478
2479            public:
2480                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
2481            };
2482
2483            template<typename v1, typename v2>
2484            struct mul {
2485            private:
2486                using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
2487                using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2488
2489            public:
2490                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
2491            };
2492
2493            template<typename v1, typename v2, typename E = void>
2494            struct div {};
2495
2496            template<typename v1, typename v2>
2497            struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
    _FractionField<Ring>::zero>::value»  {
2498            private:
2499                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2500                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2501
2502            public:
2503                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
2504            };
2505
2506            template<typename v1, typename v2>
2507            struct div<v1, v2, std::enable_if_t<
2508                std::is_same<zero, v1>::value && std::is_same<v2, zero>::value»  {
2509                using type = one;
2510            };
```

```
2511
2512            template<typename v1, typename v2>
2513            struct eq {
2514                using type = std::conditional_t<
2515                        std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
2516                        std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
2517                    std::true_type,
2518                    std::false_type>;
2519            };
2520
2521            template<typename TL, typename E = void>
2522            struct vadd {};
2523
2524            template<typename TL>
2525            struct vadd<TL, std::enable_if_t<(TL::length > 1)» {
2526                using head = typename TL::pop_front::type;
2527                using tail = typename TL::pop_front::tail;
2528                using type = typename add<head, typename vadd<tail>::type>::type;
2529            };
2530
2531            template<typename TL>
2532            struct vadd<TL, std::enable_if_t<(TL::length == 1)» {
2533                using type = typename TL::template at<0>;
2534            };
2535
2536            template<typename...  vals>
2537            struct vmul {};
2538
2539            template<typename v1, typename...  vals>
2540            struct vmul<v1, vals...> {
2541                using type = typename mul<v1, typename vmul<vals...>::type>::type;
2542            };
2543
2544            template<typename v1>
2545            struct vmul<v1> {
2546                using type = v1;
2547            };
2548
2549
2550            template<typename v1, typename v2, typename E = void>
2551            struct gt;
2552
2553            template<typename v1, typename v2>
2554            struct gt<v1, v2, std::enable_if_t<
2555                (eq<v1, v2>::type::value)
2556                » {
2557                using type = std::false_type;
2558            };
2559
2560            template<typename v1, typename v2>
2561            struct gt<v1, v2, std::enable_if_t<
2562                (!eq<v1, v2>::type::value) &&
2563                (!pos<v1>::type::value) && (!pos<v2>::type::value)
2564                » {
2565                using type = typename gt<
2566                    typename sub<zero, v1>::type, typename sub<zero, v2>::type
2567                >::type;
2568            };
2569
2570            template<typename v1, typename v2>
2571            struct gt<v1, v2, std::enable_if_t<
2572                (!eq<v1, v2>::type::value) &&
2573                (pos<v1>::type::value) && (!pos<v2>::type::value)
2574                » {
2575                using type = std::true_type;
2576            };
2577
2578            template<typename v1, typename v2>
2579            struct gt<v1, v2, std::enable_if_t<
2580                (!eq<v1, v2>::type::value) &&
2581                (!pos<v1>::type::value) && (pos<v2>::type::value)
2582                » {
2583                using type = std::false_type;
2584            };
2585
2586            template<typename v1, typename v2>
2587            struct gt<v1, v2, std::enable_if_t<
2588                (!eq<v1, v2>::type::value) &&
2589                (pos<v1>::type::value) && (pos<v2>::type::value)
2590                » {
2591                using type = std::bool_constant<Ring::template gt_v<
2592                    typename Ring::template mul_t<v1::x, v2::y>,
2593                    typename Ring::template mul_t<v2::y, v2::x>
2594                »;
2595            };
2596
2597        public:
```

```
2598
2600             template<typename v1, typename v2>
2601             using add_t = typename add<v1, v2>::type;
2602
2604             template<typename v1, typename v2>
2605             using mod_t = zero;
2606
2610             template<typename v1, typename v2>
2611             using gcd_t = v1;
2612
2615             template<typename...  vs>
2616             using vadd_t = typename vadd<vs...>::type;
2617
2620             template<typename...  vs>
2621             using vmul_t = typename vmul<vs...>::type;
2622
2624             template<typename v1, typename v2>
2625             using sub_t = typename sub<v1, v2>::type;
2626
2627             template<typename v>
2628             using minus_t = sub_t<zero, v>;
2629
2631             template<typename v1, typename v2>
2632             using mul_t = typename mul<v1, v2>::type;
2633
2635             template<typename v1, typename v2>
2636             using div_t = typename div<v1, v2>::type;
2637
2639             template<typename v1, typename v2>
2640             static constexpr bool eq_v = eq<v1, v2>::type::value;
2641
2643             template<typename v1, typename v2>
2644             static constexpr bool gt_v = gt<v1, v2>::type::value;
2645
2647             template<typename v>
2648             static constexpr bool pos_v = pos<v>::type::value;
2649         };
2650
2651         template<typename Ring, typename E = void>
2652         requires IsEuclideanDomain<Ring>
2653         struct FractionFieldImpl {};
2654
2655         // fraction field of a field is the field itself
2656         template<typename Field>
2657         requires IsEuclideanDomain<Field>
2658         struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {
2659             using type = Field;
2660             template<typename v>
2661             using inject_t = v;
2662         };
2663
2664         // fraction field of a ring is the actual fraction field
2665         template<typename Ring>
2666         requires IsEuclideanDomain<Ring>
2667         struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {
2668             using type = _FractionField<Ring>;
2669         };
2670     }
2671
2672     template<typename Ring>
2673     requires IsEuclideanDomain<Ring>
2674     using FractionField = typename internal::FractionFieldImpl<Ring>::type;
2675 }
2676
2677 // short names for common types
2678 namespace aerobus {
2680     using q32 = FractionField<i32>;
2682     using fpq32 = FractionField<polynomial<q32»;
2684     using q64 = FractionField<i64>;
2686     using pi64 = polynomial<i64>;
2688     using fpq64 = FractionField<polynomial<q64»;
2689
2692     template<uint32_t...  digits>
2693     using bigint_pos = bigint::template val<bigint::signs::positive, digits...>;
2696     template<uint32_t...  digits>
2697     using bigint_neg = bigint::template val<bigint::signs::negative, digits...>;
2698
2703     template<typename Ring, typename v1, typename v2>
2704     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
2705
2706     template<typename Ring, typename v1, typename v2>
2707     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
2708     template<typename Ring, typename v1, typename v2>
2709     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
2710 }
2711
2712 // taylor series and common integers (factorial, bernouilli...)  appearing in taylor coefficients
```

```
2713 namespace aerobus {
2714     namespace internal {
2715         template<typename T, size_t x, typename E = void>
2716         struct factorial {};
2717 
2718         template<typename T, size_t x>
2719         struct factorial<T, x, std::enable_if_t<(x > 0)» {
2720         private:
2721             template<typename, size_t, typename>
2722             friend struct factorial;
2723         public:
2724             using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
      x - 1>::type>;
2725             static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
2726         };
2727 
2728         template<typename T>
2729         struct factorial<T, 0> {
2730         public:
2731             using type = typename T::one;
2732             static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
2733         };
2734     }
2735 
2739     template<typename T, size_t i>
2740     using factorial_t = typename internal::factorial<T, i>::type;
2741 
2742     template<typename T, size_t i>
2743     inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
2744 
2745     namespace internal {
2746         template<typename T, size_t k, size_t n, typename E = void>
2747         struct combination_helper {};
2748 
2749         template<typename T, size_t k, size_t n>
2750         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
2751             using type = typename FractionField<T>::template mul_t<
2752                 typename combination_helper<T, k - 1, n - 1>::type,
2753                 makefraction_t<T, typename T::template val<n>, typename T::template val<k»>;
2754         };
2755 
2756         template<typename T, size_t k, size_t n>
2757         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
2758             using type = typename combination_helper<T, n - k, n>::type;
2759         };
2760 
2761         template<typename T, size_t n>
2762         struct combination_helper<T, 0, n> {
2763             using type = typename FractionField<T>::one;
2764         };
2765 
2766         template<typename T, size_t k, size_t n>
2767         struct combination {
2768             using type = typename internal::combination_helper<T, k, n>::type::x;
2769             static constexpr typename T::inner_type value = internal::combination_helper<T, k,
      n>::type::template get<typename T::inner_type>();
2770         };
2771     }
2772 
2775     template<typename T, size_t k, size_t n>
2776     using combination_t = typename internal::combination<T, k, n>::type;
2777 
2778     template<typename T, size_t k, size_t n>
2779     inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
2780 
2781     namespace internal {
2782         template<typename T, size_t m>
2783         struct bernouilli;
2784 
2785         template<typename T, typename accum, size_t k, size_t m>
2786         struct bernouilli_helper {
2787             using type = typename bernouilli_helper<
2788                 T,
2789                 addfractions_t<T,
2790                     accum,
2791                     mulfractions_t<T,
2792                         makefraction_t<T,
2793                             combination_t<T, k, m + 1>,
2794                             typename T::one>,
2795                         typename bernouilli<T, k>::type
2796                     >
2797                 >,
2798                 k + 1,
2799                 m>::type;
2800         };
```

```
2801
2802        template<typename T, typename accum, size_t m>
2803        struct bernouilli_helper<T, accum, m, m>
2804        {
2805            using type = accum;
2806        };
2807
2808
2809
2810        template<typename T, size_t m>
2811        struct bernouilli {
2812            using type = typename FractionField<T>::template mul_t<
2813                typename internal::bernouilli_helper<T, typename FractionField<T>::zero, 0, m>::type,
2814                makefraction_t<T,
2815                typename T::template val<static_cast<typename T::inner_type>(-1)>,
2816                typename T::template val<static_cast<typename T::inner_type>(m + 1)>
2817                >
2818            >;
2819
2820            template<typename floatType>
2821            static constexpr floatType value = type::template get<floatType>();
2822        };
2823
2824        template<typename T>
2825        struct bernouilli<T, 0> {
2826            using type = typename FractionField<T>::one;
2827
2828            template<typename floatType>
2829            static constexpr floatType value = type::template get<floatType>();
2830        };
2831    }
2832
2836    template<typename T, size_t n>
2837    using bernouilli_t = typename internal::bernouilli<T, n>::type;
2838
2839    template<typename FloatType, typename T, size_t n >
2840    inline constexpr FloatType bernouilli_v = internal::bernouilli<T, n>::template value<FloatType>;
2841
2842    namespace internal {
2843        template<typename T, int k, typename E = void>
2844        struct alternate {};
2845
2846        template<typename T, int k>
2847        struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
2848            using type = typename T::one;
2849            static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
2850        };
2851
2852        template<typename T, int k>
2853        struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
2854            using type = typename T::template minus_t<typename T::one>;
2855            static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
2856        };
2857    }
2858
2861    template<typename T, int k>
2862    using alternate_t = typename internal::alternate<T, k>::type;
2863
2864    template<typename T, size_t k>
2865    inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
2866
2867    // pow
2868    namespace internal {
2869        template<typename T, auto p, auto n>
2870        struct pow {
2871            using type = typename T::template mul_t<typename T::template val<p>, typename pow<T, p, n -
     1>::type>;
2872        };
2873
2874        template<typename T, auto p>
2875        struct pow<T, p, 0> { using type = typename T::one; };
2876    }
2877
2878    template<typename T, auto p, auto n>
2879    using pow_t = typename internal::pow<T, p, n>::type;
2880
2881    namespace internal {
2882        template<typename, template<typename, size_t> typename, class>
2883        struct make_taylor_impl;
2884
2885        template<typename T, template<typename, size_t> typename coeff_at, size_t...  Is>
2886        struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {
2887            using type = typename polynomial<FractionField<T»::template val<typename coeff_at<T,
     Is>::type...>;
2888        };
```

```
2889     }
2890
2891     // generic taylor serie, depending on coefficients
2892     template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
2893     using taylor = typename internal::make_taylor_impl<T, coeff_at,
       internal::make_index_sequence_reverse<deg + 1>::type;
2894
2895     namespace internal {
2896         template<typename T, size_t i>
2897         struct exp_coeff {
2898             using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
2899         };
2900
2901         template<typename T, size_t i, typename E = void>
2902         struct sin_coeff_helper {};
2903
2904         template<typename T, size_t i>
2905         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
2906             using type = typename FractionField<T>::zero;
2907         };
2908
2909         template<typename T, size_t i>
2910         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
2911             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>;
2912         };
2913
2914         template<typename T, size_t i>
2915         struct sin_coeff {
2916             using type = typename sin_coeff_helper<T, i>::type;
2917         };
2918
2919         template<typename T, size_t i, typename E = void>
2920         struct sh_coeff_helper {};
2921
2922         template<typename T, size_t i>
2923         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
2924             using type = typename FractionField<T>::zero;
2925         };
2926
2927         template<typename T, size_t i>
2928         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
2929             using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
2930         };
2931
2932         template<typename T, size_t i>
2933         struct sh_coeff {
2934             using type = typename sh_coeff_helper<T, i>::type;
2935         };
2936
2937         template<typename T, size_t i, typename E = void>
2938         struct cos_coeff_helper {};
2939
2940         template<typename T, size_t i>
2941         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
2942             using type = typename FractionField<T>::zero;
2943         };
2944
2945         template<typename T, size_t i>
2946         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
2947             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>;
2948         };
2949
2950         template<typename T, size_t i>
2951         struct cos_coeff {
2952             using type = typename cos_coeff_helper<T, i>::type;
2953         };
2954
2955         template<typename T, size_t i, typename E = void>
2956         struct cosh_coeff_helper {};
2957
2958         template<typename T, size_t i>
2959         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
2960             using type = typename FractionField<T>::zero;
2961         };
2962
2963         template<typename T, size_t i>
2964         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
2965             using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
2966         };
2967
2968         template<typename T, size_t i>
2969         struct cosh_coeff {
2970             using type = typename cosh_coeff_helper<T, i>::type;
2971         };
2972
2973         template<typename T, size_t i>
2974         struct geom_coeff { using type = typename FractionField<T>::one; };
```

```
2975
2976
2977        template<typename T, size_t i, typename E = void>
2978        struct atan_coeff_helper;
2979
2980        template<typename T, size_t i>
2981        struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
2982            using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i»;
2983        };
2984
2985        template<typename T, size_t i>
2986        struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
2987            using type = typename FractionField<T>::zero;
2988        };
2989
2990        template<typename T, size_t i>
2991        struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
2992
2993        template<typename T, size_t i, typename E = void>
2994        struct asin_coeff_helper;
2995
2996        template<typename T, size_t i>
2997        struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
2998        {
2999            using type = makefraction_t<T,
3000                factorial_t<T, i - 1>,
3001                typename T::template mul_t<
3002                    typename T::template val<i>,
3003                    T::template mul_t<
3004                        pow_t<T, 4, i / 2>,
3005                        pow<T, factorial<T, i / 2>::value, 2
3006                    >
3007                >
3008                »;
3009        };
3010
3011        template<typename T, size_t i>
3012        struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
3013        {
3014            using type = typename FractionField<T>::zero;
3015        };
3016
3017        template<typename T, size_t i>
3018        struct asin_coeff {
3019            using type = typename asin_coeff_helper<T, i>::type;
3020        };
3021
3022        template<typename T, size_t i>
3023        struct lnp1_coeff {
3024            using type = makefraction_t<T,
3025                alternate_t<T, i + 1>,
3026                typename T::template val<i»;
3027        };
3028
3029        template<typename T>
3030        struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
3031
3032        template<typename T, size_t i, typename E = void>
3033        struct asinh_coeff_helper;
3034
3035        template<typename T, size_t i>
3036        struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
3037        {
3038            using type = makefraction_t<T,
3039                typename T::template mul_t<
3040                    alternate_t<T, i / 2>,
3041                    factorial_t<T, i - 1>
3042                >,
3043                typename T::template mul_t<
3044                    T::template mul_t<
3045                        typename T::template val<i>,
3046                        pow_t<T, (factorial<T, i / 2>::value), 2>
3047                    >,
3048                    pow_t<T, 4, i / 2>
3049                >
3050            >;
3051        };
3052
3053        template<typename T, size_t i>
3054        struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
3055        {
3056            using type = typename FractionField<T>::zero;
3057        };
3058
3059        template<typename T, size_t i>
3060        struct asinh_coeff {
3061            using type = typename asinh_coeff_helper<T, i>::type;
```

```
3062         };
3063
3064         template<typename T, size_t i, typename E = void>
3065         struct atanh_coeff_helper;
3066
3067         template<typename T, size_t i>
3068         struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
3069         {
3070             // 1/i
3071             using type = typename FractionField<T>::  template val<
3072                 typename T::one,
3073                 typename T::template val<static_cast<typename T::inner_type>(i)»;
3074         };
3075
3076         template<typename T, size_t i>
3077         struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
3078         {
3079             using type = typename FractionField<T>::zero;
3080         };
3081
3082         template<typename T, size_t i>
3083         struct atanh_coeff {
3084             using type = typename asinh_coeff_helper<T, i>::type;
3085         };
3086
3087         template<typename T, size_t i, typename E = void>
3088         struct tan_coeff_helper;
3089
3090         template<typename T, size_t i>
3091         struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
3092             using type = typename FractionField<T>::zero;
3093         };
3094
3095         template<typename T, size_t i>
3096         struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
3097         private:
3098             // 4^((i+1)/2)
3099             using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
3100             // 4^((i+1)/2) - 1
3101             using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
     FractionField<T>::one>;
3102             // (-1)^((i-1)/2)
3103             using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
3104             using dividend = typename FractionField<T>::template mul_t<
3105                 altp,
3106                 FractionField<T>::template mul_t<
3107                 _4p,
3108                 FractionField<T>::template mul_t<
3109                 _4pm1,
3110                 bernouilli_t<T, (i + 1)>
3111                 >
3112                 >
3113             >;
3114         public:
3115             using type = typename FractionField<T>::template div_t<dividend,
3116                 typename FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
3117         };
3118
3119         template<typename T, size_t i>
3120         struct tan_coeff {
3121             using type = typename tan_coeff_helper<T, i>::type;
3122         };
3123
3124         template<typename T, size_t i, typename E = void>
3125         struct tanh_coeff_helper;
3126
3127         template<typename T, size_t i>
3128         struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
3129             using type = typename FractionField<T>::zero;
3130         };
3131
3132         template<typename T, size_t i>
3133         struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
3134         private:
3135             using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
3136             using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
     FractionField<T>::one>;
3137             using dividend =
3138                 typename FractionField<T>::template mul_t<
3139                 _4p,
3140                 typename FractionField<T>::template mul_t<
3141                 _4pm1,
3142                 bernouilli_t<T, (i + 1)>
3143                 >
3144                 >::type;
3145         public:
3146             using type = typename FractionField<T>::template div_t<dividend,
```

```
3147                 FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
3148         };
3149
3150         template<typename T, size_t i>
3151         struct tanh_coeff {
3152             using type = typename tanh_coeff_helper<T, i>::type;
3153         };
3154     }
3155
3159     template<typename T, size_t deg>
3160     using exp = taylor<T, internal::exp_coeff, deg>;
3161
3165     template<typename T, size_t deg>
3166     using expm1 = typename polynomial<FractionField<T»::template sub_t<
3167         exp<T, deg>,
3168         typename polynomial<FractionField<T»::one>;
3169
3173     template<typename T, size_t deg>
3174     using lnp1 = taylor<T, internal::lnp1_coeff, deg>;
3175
3179     template<typename T, size_t deg>
3180     using atan = taylor<T, internal::atan_coeff, deg>;
3181
3185     template<typename T, size_t deg>
3186     using sin = taylor<T, internal::sin_coeff, deg>;
3187
3191     template<typename T, size_t deg>
3192     using sinh = taylor<T, internal::sh_coeff, deg>;
3193
3197     template<typename T, size_t deg>
3198     using cosh = taylor<T, internal::cosh_coeff, deg>;
3199
3203     template<typename T, size_t deg>
3204     using cos = taylor<T, internal::cos_coeff, deg>;
3205
3209     template<typename T, size_t deg>
3210     using geometric_sum = taylor<T, internal::geom_coeff, deg>;
3211
3215     template<typename T, size_t deg>
3216     using asin = taylor<T, internal::asin_coeff, deg>;
3217
3221     template<typename T, size_t deg>
3222     using asinh = taylor<T, internal::asinh_coeff, deg>;
3223
3227     template<typename T, size_t deg>
3228     using atanh = taylor<T, internal::atanh_coeff, deg>;
3229
3233     template<typename T, size_t deg>
3234     using tan = taylor<T, internal::tan_coeff, deg>;
3235
3239     template<typename T, size_t deg>
3240     using tanh = taylor<T, internal::tanh_coeff, deg>;
3241 }
3242
3243 // continued fractions
3244 namespace aerobus {
3247     template<int64_t...  values>
3248     struct ContinuedFraction {};
3249
3250     template<int64_t a0>
3251     struct ContinuedFraction<a0> {
3252         using type = typename q64::template inject_constant_t<a0>;
3253         static constexpr double val = type::template get<double>();
3254     };
3255
3256     template<int64_t a0, int64_t...  rest>
3257     struct ContinuedFraction<a0, rest...> {
3258         using type = q64::template add_t<
3259                 typename q64::template inject_constant_t<a0>,
3260                 typename q64::template div_t<
3261                     typename q64::one,
3262                     typename ContinuedFraction<rest...>::type
3263                 »;
3264         static constexpr double val = type::template get<double>();
3265     };
3266
3271     using PI_fraction =
     ContinuedFraction<3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1>;
3274     using E_fraction =
     ContinuedFraction<2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, 1, 1, 12, 1, 1, 14, 1, 1>;
3276     using SQRT2_fraction =
     ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2>;
3278     using SQRT3_fraction =
     ContinuedFraction<1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2>;
3279 }
3280
3281 // known polynomials
```

```
3282 namespace aerobus {
3283     namespace internal {
3284         template<int kind, int deg>
3285         struct chebyshev_helper {
3286             using type = typename pi64::template sub_t<
3287                 typename pi64::template mul_t<
3288                     typename pi64::template mul_t<
3289                         pi64::inject_constant_t<2>,
3290                         typename pi64::X
3291                     >,
3292                     typename chebyshev_helper<kind, deg-1>::type
3293                 >,
3294                 typename chebyshev_helper<kind, deg-2>::type
3295             >;
3296         };
3297 
3298         template<>
3299         struct chebyshev_helper<1, 0> {
3300             using type = typename pi64::one;
3301         };
3302 
3303         template<>
3304         struct chebyshev_helper<1, 1> {
3305             using type = typename pi64::X;
3306         };
3307 
3308         template<>
3309         struct chebyshev_helper<2, 0> {
3310             using type = typename pi64::one;
3311         };
3312 
3313         template<>
3314         struct chebyshev_helper<2, 1> {
3315             using type = typename pi64::template mul_t<
3316                             typename pi64::inject_constant_t<2>,
3317                             typename pi64::X>;
3318         };
3319     }
3320 
3323     template<size_t deg>
3324     using chebyshev_T = typename internal::chebyshev_helper<1, deg>::type;
3325 
3328     template<size_t deg>
3329     using chebyshev_U = typename internal::chebyshev_helper<2, deg>::type;
3330 }
```

Chapter 7

Example Documentation

7.1 i32::template

inject a native constant

inject a native constant

Template Parameters

x	inject_constant_2<2> -> i32::template val<2>

7.2 i64::template

injects constant as an i64 value

injects constant as an i64 value

Template Parameters

x	inject_constant_t<2>

7.3 polynomial

makes the constant (native type) polynomial a_0

makes the constant (native type) polynomial a_0

Template Parameters

x	<i32>::template inject_constant_t<2>

7.4 bigint::from_hex_t

"constructor" from constant hex string (no prefix – all caps) <"12AB456FFE0">;

"constructor" from constant hex string (no prefix – all caps) <"12AB456FFE0">;

7.5 PI_fraction::val

representation of PI as a continued fraction -> 3.14...

7.6 E_fraction::val

approximation of e -> 2.718...

approximation of e -> 2.718...

Index