Aerobus

Generated by Doxygen 1.9.4

1 Concept Index 1

1.1 Concepts . . . 1

2 Class Index 3

2.1 Class List . . . 3

3 File Index 5

3.1 File List . . . 5

4 Concept Documentation 7

4.1 aerobus::IsEuclideanDomain Concept Reference . . . 7

4.1.1 Concept definition . . . 7

4.1.2 Detailed Description . . . 7

4.2 aerobus::IsField Concept Reference . . . 7

4.2.1 Concept definition . . . 7

4.2.2 Detailed Description . . . 8

4.3 aerobus::IsRing Concept Reference . . . 8

4.3.1 Concept definition . . . 8

4.3.2 Detailed Description . . . 8

5 Class Documentation 9

5.1 aerobus::bigint::add< I1, I2 > Struct Template Reference . . . 9

5.2 aerobus::bigint::add_low< I1, I2, I > Struct Template Reference . . . 9

5.3 aerobus::bigint::add_low< I1, I2, std::index_sequence< I... > > Struct Template Reference . . . 9

5.4 aerobus::bigint::add_low_helper< I1, I2, index > Struct Template Reference . . . 10

5.5 aerobus::bigint::add_low_helper< I1, I2, 0 > Struct Template Reference . . . 10

5.6 aerobus::bigint Struct Reference . . . 10

5.7 aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, E > Struct Template Reference . . . 11

5.8 aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0||index > 0)> > Struct Template Reference . . . 11

5.9 aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > Struct Template Reference . . . 11

5.10 aerobus::ContinuedFraction< values > Struct Template Reference . . . 12

5.10.1 Detailed Description . . . 12

5.11 aerobus::ContinuedFraction< a0 > Struct Template Reference . . . 12

5.12 aerobus::ContinuedFraction< a0, rest... > Struct Template Reference . . . 12

5.13 aerobus::bigint::val< s, an, as >::digit_at< index, E > Struct Template Reference . . . 13

5.14 aerobus::bigint::val< s, a0 >::digit_at< index, E > Struct Template Reference . . . 13

5.15 aerobus::bigint::val< s, a0 >::digit_at< index, std::enable_if_t< index !=0 > > Struct Template Reference . . . 13

5.16 aerobus::bigint::val< s, a0 >::digit_at< index, std::enable_if_t< index==0 > > Struct Template Reference . . . 13

5.17 aerobus::bigint::val< s, an, as >::digit_at< index, std::enable_if_t<(index > sizeof...(as))> > Struct Template Reference . . . 14

5.18 aerobus::bigint::val< s, an, as >::digit_at< index, std::enable_if_t<(index<=sizeof...(as))> > Struct Template Reference . . . 14
5.19 aerobus::i32 Struct Reference . . . 14
5.19.1 Detailed Description . . . 16
5.20 aerobus::i64 Struct Reference . . . 16
5.20.1 Detailed Description . . . 17
5.21 aerobus::polynomial< Ring, variable_name >::eval_helper< valueRing, P >::inner< index, stop > Struct Template Reference . . . 17
5.22 aerobus::polynomial< Ring, variable_name >::eval_helper< valueRing, P >::inner< stop, stop > Struct Template Reference . . . 18
5.23 aerobus::is_prime< n > Struct Template Reference . . . 18
5.23.1 Detailed Description . . . 18
5.24 aerobus::bigint::is_zero< I > Struct Template Reference . . . 18
5.25 aerobus::polynomial< Ring, variable_name > Struct Template Reference . . . 19
5.25.1 Detailed Description . . . 20
5.25.2 Member Typedef Documentation . . . 20
5.25.2.1 add_t . . . 20
5.25.2.2 derive_t . . . 21
5.25.2.3 div_t . . . 21
5.25.2.4 eq_t . . . 21
5.25.2.5 gcd_t . . . 21
5.25.2.6 gt_t . . . 22
5.25.2.7 lt_t . . . 22
5.25.2.8 mod_t . . . 22
5.25.2.9 monomial_t . . . 23
5.25.2.10 mul_t . . . 23
5.25.2.11 pos_t . . . 23
5.25.2.12 simplify_t . . . 24
5.25.2.13 sub_t . . . 24
5.26 aerobus::type_list< Ts >::pop_front Struct Reference . . . 25
5.27 aerobus::Quotient< Ring, X > Struct Template Reference . . . 25
5.28 aerobus::type_list< Ts >::split< index > Struct Template Reference . . . 26
5.29 aerobus::type_list< Ts > Struct Template Reference . . . 26
5.29.1 Detailed Description . . . 26
5.30 aerobus::type_list<> Struct Reference . . . 27
5.31 aerobus::bigint::val< s, an, as > Struct Template Reference . . . 27
5.32 aerobus::i32::val< x > Struct Template Reference . . . 28
5.32.1 Detailed Description . . . 28
5.32.2 Member Function Documentation . . . 28
5.32.2.1 eval() . . . 28
5.32.2.2 get() . . . 29
5.33 aerobus::i64::val< x > Struct Template Reference . . . 29
5.33.1 Detailed Description . . . 30

5.33.2 Member Function Documentation . . . 30
5.33.2.1 eval() . . . 30
5.33.2.2 get() . . . 30
5.34 aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs > Struct Template Reference . . . 31
5.34.1 Member Typedef Documentation . . . 31
5.34.1.1 coeff_at_t . . . 31
5.34.2 Member Function Documentation . . . 32
5.34.2.1 eval() . . . 32
5.34.2.2 to_string() . . . 32
5.35 aerobus::Quotient< Ring, X >::val< V > Struct Template Reference . . . 33
5.36 aerobus::zpz< p >::val< x > Struct Template Reference . . . 33
5.37 aerobus::polynomial< Ring, variable_name >::val< coeffN > Struct Template Reference . . . 33
5.38 aerobus::bigint::val< s, a0 > Struct Template Reference . . . 34
5.39 aerobus::zpz< p > Struct Template Reference . . . 35
5.39.1 Detailed Description . . . 35

6 File Documentation **37**
6.1 lib.h . . . 37

7 Example Documentation **65**
7.1 i32::template . . . 65
7.2 i64::template . . . 65
7.3 polynomial . . . 65
7.4 PI_fraction::val . . . 66
7.5 E_fraction::val . . . 66

Index **67**

Chapter 1

Concept Index

1.1 Concepts

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

aerobus::IsEuclideanDomain
: Concept to express R is an euclidean domain 7

aerobus::IsField
: Concept to express R is a field 7

aerobus::IsRing
: Concept to express R is a Ring (ordered) 8

Chapter 2

Class Index

2.1 Class List

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

aerobus::bigint::add< I1, I2 > . . . 9
aerobus::bigint::add_low< I1, I2, I > . . . 9
aerobus::bigint::add_low< I1, I2, std::index_sequence< I... > > . . . 9
aerobus::bigint::add_low_helper< I1, I2, index > . . . 10
aerobus::bigint::add_low_helper< I1, I2, 0 > . . . 10
aerobus::bigint . . . 10
aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, E > . . . 11
aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0||index > 0)> > 11
aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> > 11
aerobus::ContinuedFraction< values >
Continued fraction a0 + 1/(a1 + 1/(...)) . . . 12
aerobus::ContinuedFraction< a0 > . . . 12
aerobus::ContinuedFraction< a0, rest... > . . . 12
aerobus::bigint::val< s, an, as >::digit_at< index, E > . . . 13
aerobus::bigint::val< s, a0 >::digit_at< index, E > . . . 13
aerobus::bigint::val< s, a0 >::digit_at< index, std::enable_if_t< index !=0 > > . . . 13
aerobus::bigint::val< s, a0 >::digit_at< index, std::enable_if_t< index==0 > > . . . 13
aerobus::bigint::val< s, an, as >::digit_at< index, std::enable_if_t<(index > sizeof...(as))> > . . . 14
aerobus::bigint::val< s, an, as >::digit_at< index, std::enable_if_t<(index<=sizeof...(as))> > . . . 14
aerobus::i32
32 bits signed integers, seen as a algebraic ring with related operations . . . 14
aerobus::i64
64 bits signed integers, seen as a algebraic ring with related operations . . . 16
aerobus::polynomial< Ring, variable_name >::eval_helper< valueRing, P >::inner< index, stop > . . . 17
aerobus::polynomial< Ring, variable_name >::eval_helper< valueRing, P >::inner< stop, stop > . . . 18
aerobus::is_prime< n >
Checks if n is prime . . . 18
aerobus::bigint::is_zero< I > . . . 18
aerobus::polynomial< Ring, variable_name > . . . 19
aerobus::type_list< Ts >::pop_front . . . 25
aerobus::Quotient< Ring, X > . . . 25
aerobus::type_list< Ts >::split< index > . . . 26
aerobus::type_list< Ts >
Empty pure template struct to handle type list . . . 26

aerobus::type_list<> 27
aerobus::bigint::val< s, an, as > 27
aerobus::i32::val< x >
Values in i32 28
aerobus::i64::val< x >
Values in i64 29
aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs > 31
aerobus::Quotient< Ring, X >::val< V > 33
aerobus::zpz< p >::val< x > 33
aerobus::polynomial< Ring, variable_name >::val< coeffN > 33
aerobus::bigint::val< s, a0 > 34
aerobus::zpz< p > 35

Chapter 3

File Index

3.1 File List

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

src/lib.h 37

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>;
        typename R::template eq_t<typename R::one, typename R::one>;
        typename R::template pos_t<typename R::one>;
        R::template pos_v<typename R::one> == 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>;
    }
```

4.3.2 Detailed Description

Concept to express R is a Ring (ordered)

Chapter 5

Class Documentation

5.1 aerobus::bigint::add< I1, I2 > Struct Template Reference

Public Types

- using **type** = simplify_t< typename add_low< I1, I2, typename internal::make_index_sequence_reverse< std::max(I1::digits, I2::digits)+1 > >::type >

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

- src/lib.h

5.2 aerobus::bigint::add_low< I1, I2, I > Struct Template Reference

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

- src/lib.h

5.3 aerobus::bigint::add_low< I1, I2, std::index_sequence< I... > > Struct Template Reference

Public Types

- using **type** = val< sign::positive, add_low_helper< I1, I2, I >::digit... >

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

- src/lib.h

5.4 aerobus::bigint::add_low_helper< I1, I2, index > Struct Template Reference

Static Public Attributes

- static constexpr uint32_t **digit** = helper::value
- static constexpr uint8_t **carry_out** = helper::carry_out

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

- src/lib.h

5.5 aerobus::bigint::add_low_helper< I1, I2, 0 > Struct Template Reference

Static Public Attributes

- static constexpr uint32_t **digit** = add_at_helper<I1, I2, 0, 0>::value
- static constexpr uint32_t **carry_out** = add_at_helper<I1, I2, 0, 0>::carry_out

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

- src/lib.h

5.6 aerobus::bigint Struct Reference

Classes

- struct add
- struct add_low
- struct add_low< I1, I2, std::index_sequence< I... > >
- struct add_low_helper
- struct add_low_helper< I1, I2, 0 >
- struct is_zero
- struct val
- struct val< s, a0 >

Public Types

- enum **sign** { **positive** , **negative** }
- using **zero** = val< sign::positive, 0 >
- using **one** = val< sign::positive, 1 >
- template<typename I >
  using **simplify_t** = typename simplify< I >::type

Static Public Attributes

- template<typename I >
  static constexpr bool **is_zero_v** = is_zero<I>::value
- template<typename I1 , typename I2 , size_t index, uint8_t carry_in = 0>
  static constexpr uint32_t **add_at_digit** = add_at_helper<I1, I2, index, carry_in>::value
- template<typename I1 , typename I2 , size_t index, uint8_t carry_in = 0>
  static constexpr uint8_t **add_at_carry** = add_at_helper<I1, I2, index, carry_in>::carry_out

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

- src/lib.h

5.7 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

5.8 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

5.9 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

5.10 aerobus::ContinuedFraction< values > Struct Template Reference

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

`#include <lib.h>`

5.10.1 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

5.11 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

5.12 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

5.13 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

5.14 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

5.15 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

5.16 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

5.17 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

5.18 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

5.19 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 , 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 **gt_t** = typename gt< v1, v2 >::type

  strictly greater operator (v1 > v2)
- template<typename v1 , typename v2 >
  using **lt_t** = typename lt< v1, v2 >::type

  strict less operator (v1 < v2)
- template<typename v1 , typename v2 >
  using **eq_t** = typename eq< v1, v2 >::type

  equality operator
- template<typename v1 , typename v2 >
  using **gcd_t** = gcd_t< i32, v1, v2 >

  greatest common divisor
- template<typename v >
  using **pos_t** = typename pos< v >::type

  positivity (v > 0)

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 v >
  static constexpr bool **pos_v** = pos_t<v>::value

5.19.1 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

5.20 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 , 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 **gt_t** = typename gt< v1, v2 >::type
  strictly greater operator (v1 > v2)

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

  strict less operator (v1 < v2)
- template<typename v1 , typename v2 >
  using **eq_t** = typename eq< v1, v2 >::type

  equality operator
- template<typename v1 , typename v2 >
  using **gcd_t** = gcd_t< i64, v1, v2 >

  greatest common divisor
- template<typename v >
  using **pos_t** = typename pos< v >::type

  is v positive

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 v >
  static constexpr bool **pos_v** = pos_t<v>::value

5.20.1 Detailed Description

64 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

5.21 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

5.22 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

5.23 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

5.23.1 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

5.24 aerobus::bigint::is_zero< I > Struct Template Reference

Static Public Attributes

- static constexpr bool **value** = I::digits == 1 && I::aN == 0

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

- src/lib.h

5.25 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 , typename v2 >
  using mul_t = typename mul< v1, v2 >::type
  multiplication of two polynomials
- template<typename v1 , typename v2 >
  using eq_t = typename eq_helper< v1, v2 >::type
  equality operator
- template<typename v1 , typename v2 >
  using lt_t = typename lt_helper< v1, v2 >::type
  strict less operator
- template<typename v1 , typename v2 >
  using gt_t = typename gt_helper< v1, v2 >::type
  strict greater 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 v >
  using pos_t = typename Ring::template pos_t< typename v::aN >

  checks for positivity (an > 0)
- 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 v >
  static constexpr bool **pos_v** = pos_t<v>::value

5.25.1 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

5.25.2 Member Typedef Documentation

5.25.2.1 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	

5.25.2.2 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	

5.25.2.3 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	

5.25.2.4 eq_t

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

equality operator

Template Parameters

v1	
v2	

5.25.2.5 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	

5.25.2.6 gt_t

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

strict greater operator

Template Parameters

v1	
v2	

5.25.2.7 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	

5.25.2.8 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	

5.25.2.9 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	

5.25.2.10 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	

5.25.2.11 pos_t

```
template<typename Ring , char variable_name = 'x'>
```

```
template<typename v >
using aerobus::polynomial< Ring, variable_name >::pos_t = typename Ring::template pos_t<typename
v::aN>
```

checks for positivity (an $>$ 0)

Template Parameters

v	

5.25.2.12 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	

5.25.2.13 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	

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

- src/lib.h

5.26 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

5.27 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<typename v1 , typename v2 >
  using **eq_t** = typename Ring::template eq_t< typename v1::type, typename v2::type >
- template<typename v1 >
  using **pos_t** = std::true_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 v >
  static constexpr bool **pos_v** = pos_t<v>::value
- static constexpr bool **is_euclidean_domain** = true

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

- src/lib.h

5.28 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

5.29 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)

5.29.1 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

5.30 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

5.31 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

- using **strip** = val< s, as... >

Static Public Member Functions

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

Static Public Attributes

- static constexpr bool **is_positive** = s != sign::negative
- static constexpr uint32_t **aN** = an
- static constexpr size_t **digits** = sizeof...(as) + 1

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

- src/lib.h

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

values in i32

`#include <lib.h>`

Public Types

- using **is_zero_t** = std::bool_constant< x==0 >
  is value zero

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

5.32.1 Detailed Description

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

values in i32

Template Parameters

x	an actual integer

5.32.2 Member Function Documentation

5.32.2.1 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

5.32.2.2 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

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

values in i64

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

Public Types

- using **is_zero_t** = std::bool_constant< x==0 >
  is value zero

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

5.33.1 Detailed Description

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

values in i64

Template Parameters

x	an actual integer

5.33.2 Member Function Documentation

5.33.2.1 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)

5.33.2.2 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

5.34 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
- using **is_zero_t** = std::bool_constant<(degree==0) &&(aN::is_zero_t::value)>
  true if polynomial is constant zero
- 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

5.34.1 Member Typedef Documentation

5.34.1.1 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	

5.34.2 Member Function Documentation

5.34.2.1 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)

5.34.2.2 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

5.35 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 sub_t< typename Ring::zero, tmp > >

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

- src/lib.h

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

Public Types

- using **is_zero_t** = std::bool_constant< x% p==0 >

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

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

- src/lib.h

5.37 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 >
- using **is_zero_t** = std::bool_constant< aN::is_zero_t::value >
- 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

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

- src/lib.h

5.38 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

- using **strip** = val< s, a0 >

Static Public Member Functions

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

Static Public Attributes

- static constexpr bool **is_positive** = s != sign::negative
- static constexpr uint32_t **aN** = a0
- static constexpr size_t **digits** = 1

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

- src/lib.h

5.39 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 , 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 **gt_t** = typename gt< v1, v2 >::type
- template<typename v1 , typename v2 >
  using **lt_t** = typename lt< v1, v2 >::type
- template<typename v1 , typename v2 >
  using **eq_t** = typename eq< v1, v2 >::type
- template<typename v1 , typename v2 >
  using **gcd_t** = gcd_t< i32, v1, v2 >
- template<typename v1 >
  using **pos_t** = typename pos< v1 >::type

Static Public Attributes

- static constexpr bool **is_field** = is_prime<p>::value
- static constexpr bool **is_euclidean_domain** = true
- template<typename v >
  static constexpr bool **pos_v** = pos_t<v>::value

5.39.1 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
14
15 #ifdef _MSC_VER
16 #define ALIGNED(x) __declspec(align(x))
17 #define INLINED __forceinline
18 #else
19 #define ALIGNED(x) __attribute__((aligned(x)))
20 #define INLINED __attribute__((always_inline)) inline
21 #endif
22
23 // aligned allocation
24 namespace aerobus {
31     template<typename T>
32     T* aligned_malloc(size_t count, size_t alignment) {
33 #ifdef _MSC_VER
34         return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
35 #else
36         return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
37 #endif
38     }
39
40     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 } };
41
50      template<typename T, size_t N>
51      constexpr bool contains(const std::array<T, N>& arr, const T& v) {
52          for (const auto& vv :  arr) {
53              if (v == vv) {
54                  return true;
55              }
56          }
57
58          return false;
59      }
60
61  }
62
63  // concepts
64  namespace aerobus
65  {
67      template <typename R>
68      concept IsRing = requires {
69          typename R::one;
70          typename R::zero;
71          typename R::template add_t<typename R::one, typename R::one>;
72          typename R::template sub_t<typename R::one, typename R::one>;
73          typename R::template mul_t<typename R::one, typename R::one>;
74      };
75
77      template <typename R>
78      concept IsEuclideanDomain = IsRing<R> && requires {
79          typename R::template div_t<typename R::one, typename R::one>;
80          typename R::template mod_t<typename R::one, typename R::one>;
81          typename R::template gcd_t<typename R::one, typename R::one>;
82          typename R::template eq_t<typename R::one, typename R::one>;
83          typename R::template pos_t<typename R::one>;
84
85          R::template pos_v<typename R::one> == true;
86          //typename R::template gt_t<typename R::one, typename R::zero>;
87          R::is_euclidean_domain == true;
88      };
89
91      template<typename R>
92      concept IsField = IsEuclideanDomain<R> && requires {
93          R::is_field == true;
94      };
95  }
96
97  // utilities
98  namespace aerobus {
99      namespace internal
100      {
101          template<template<typename...> typename TT, typename T>
102          struct is_instantiation_of :  std::false_type { };
103
104          template<template<typename...> typename TT, typename...  Ts>
105          struct is_instantiation_of<TT, TT<Ts...» :  std::true_type { };
```

```
106
107         template<template<typename...> typename TT, typename T>
108         inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
109
110         template <size_t i, typename T, typename...  Ts>
111         struct type_at
112         {
113             static_assert(i < sizeof...(Ts) + 1, "index out of range");
114             using type = typename type_at<i - 1, Ts...>::type;
115         };
116
117         template <typename T, typename...  Ts> struct type_at<0, T, Ts...> {
118             using type = T;
119         };
120
121         template <size_t i, typename...  Ts>
122         using type_at_t = typename type_at<i, Ts...>::type;
123
124         template<size_t i, auto x, auto...  xs>
125         struct value_at {
126             static_assert(i < sizeof...(xs) + 1, "index out of range");
127             static constexpr auto value = value_at<i-1, xs...>::value;
128         };
129
130         template<auto x, auto...  xs>
131         struct value_at<0, x, xs...> {
132             static constexpr auto value = x;
133         };
134
135
136         template<int32_t n, int32_t i, typename E = void>
137         struct _is_prime {};
138
139         // first 1000 primes are precomputed and stored in a table
140         template<int32_t n, int32_t i>
141         struct _is_prime<n, i, std::enable_if_t<(n < 7920) && (contains<int32_t, 1000>(primes, n))» :
    std::true_type {};
142
143         // first 1000 primes are precomputed and stored in a table
144         template<int32_t n, int32_t i>
145         struct _is_prime<n, i, std::enable_if_t<(n < 7920) && (!contains<int32_t, 1000>(primes, n))» :
    std::false_type {};
146
147         template<int32_t n, int32_t i>
148         struct _is_prime<n, i, std::enable_if_t<
149             (n >= 7920) &&
150             (i >= 5 && i * i <= n) &&
151             (n % i == 0 || n % (i + 2) == 0)» :  std::false_type {};
152
153
154         template<int32_t n, int32_t i>
155         struct _is_prime<n, i, std::enable_if_t<
156             (n >= 7920) &&
157             (i >= 5 && i * i <= n) &&
158             (n % i != 0 && n % (i + 2) != 0)» {
159             static constexpr bool value = _is_prime<n, i + 6>::value;
160         };
161
162         template<int32_t n, int32_t i>
163         struct _is_prime<n, i, std::enable_if_t<
164             (n >= 7920) &&
165             (i >= 5 && i * i > n)» :  std::true_type {};
166     }
167
170     template<int32_t n>
171     struct is_prime {
173         static constexpr bool value = internal::_is_prime<n, 5>::value;
174     };
175
176     namespace internal {
177         template <std::size_t...  Is>
178         constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
179             -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
180
181         template <std::size_t N>
182         using make_index_sequence_reverse
183             = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
184
190         template<typename Ring, typename E = void>
191         struct gcd;
192
193         template<typename Ring>
194         struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
195             template<typename A, typename B, typename E = void>
196             struct gcd_helper {};
197
198             // B = 0, A > 0
```

```
199             template<typename A, typename B>
200             struct gcd_helper<A, B, std::enable_if_t<
201                 ((B::is_zero_t::value) &&
202                     (Ring::template gt_t<A, typename Ring::zero>::value))»
203             {
204                 using type = A;
205             };
206 
207             // B = 0, A < 0
208             template<typename A, typename B>
209             struct gcd_helper<A, B, std::enable_if_t<
210                 ((B::is_zero_t::value) &&
211                     !(Ring::template gt_t<A, typename Ring::zero>::value))»
212             {
213                 using type = typename Ring::template sub_t<typename Ring::zero, A>;
214             };
215 
216             // B != 0
217             template<typename A, typename B>
218             struct gcd_helper<A, B, std::enable_if_t<
219                 (!B::is_zero_t::value)
220                 » {
221             private:
222                 // A / B
223                 using k = typename Ring::template div_t<A, B>;
224                 // A - (A/B)*B = A % B
225                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
226             public:
227                 using type = typename gcd_helper<B, m>::type;
228             };
229 
230             template<typename A, typename B>
231             using type = typename gcd_helper<A, B>::type;
232         };
233     }
234 
237     template<typename T, typename A, typename B>
238     using gcd_t = typename internal::gcd<T>::template type<A, B>;
239 }
240 
241 // quotient ring by the principal ideal generated by X
242 namespace aerobus {
243     template<typename Ring, typename X>
244     requires IsRing<Ring>
245     struct Quotient {
246         template <typename V>
247         struct val {
248         private:
249             using tmp = typename Ring::template mod_t<V, X>;
250         public:
251             using type = std::conditional_t<
252                 Ring::template pos_v<tmp>,
253                 tmp,
254                 typename Ring::template sub_t<typename Ring::zero, tmp>
255             >;
256         };
257 
258         using zero = val<typename Ring::zero>;
259         using one = val<typename Ring::one>;
260 
261         template<typename v1, typename v2>
262         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
263         template<typename v1, typename v2>
264         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
265         template<typename v1, typename v2>
266         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
267         template<typename v1, typename v2>
268         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
269         template<typename v1, typename v2>
270         using eq_t = typename Ring::template eq_t<typename v1::type, typename v2::type>;
271         template<typename v1>
272         using pos_t = std::true_type;
273 
274         template<typename v>
275         static constexpr bool pos_v = pos_t<v>::value;
276 
277         static constexpr bool is_euclidean_domain = true;
278 
279         template<auto x>
280         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
281 
282         template<typename v>
283         using inject_ring_t = val<v>;
284     };
285 }
286 
287 // type_list
```

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

```
376
377         template <typename U>
378         using concat = typename concat_h<U>::type;
379
380         template <uint64_t index>
381         struct split
382         {
383         private:
384             using inner = internal::split_h<index, type_list<>, type_list<Ts...»;
385
386         public:
387             using head = typename inner::head;
388             using tail = typename inner::tail;
389         };
390
391         template <uint64_t index, typename T>
392         using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
393
394         template <uint64_t index>
395         using remove = typename internal::remove_h<index, type_list<Ts...»::type;
396     };
397
398     template <>
399     struct type_list<>
400     {
401         static constexpr size_t length = 0;
402
403         template <typename T>
404         using push_front = type_list<T>;
405
406         template <typename T>
407         using push_back = type_list<T>;
408
409         template <typename U>
410         using concat = U;
411
412         // TODO: assert index == 0
413         template <uint64_t index, typename T>
414         using insert = type_list<T>;
415     };
416 }
417
418 // i32
419 namespace aerobus {
421     struct i32 {
422         using inner_type = int32_t;
425         template<int32_t x>
426         struct val {
427             static constexpr int32_t v = x;
428
431             template<typename valueType>
432             static constexpr valueType get() { return static_cast<valueType>(x); }
433
435             using is_zero_t = std::bool_constant<x == 0>;
436
438             static std::string to_string() {
439                 return std::to_string(x);
440             }
441
444             template<typename valueRing>
445             static constexpr valueRing eval(const valueRing& v) {
446                 return static_cast<valueRing>(x);
447             }
448         };
449
451         using zero = val<0>;
453         using one = val<1>;
455         static constexpr bool is_field = false;
457         static constexpr bool is_euclidean_domain = true;
461         template<auto x>
462         using inject_constant_t = val<static_cast<int32_t>(x)>;
463
464         template<typename v>
465         using inject_ring_t = v;
466
467     private:
468         template<typename v1, typename v2>
469         struct add {
470             using type = val<v1::v + v2::v>;
471         };
472
473         template<typename v1, typename v2>
474         struct sub {
475             using type = val<v1::v - v2::v>;
476         };
477
478         template<typename v1, typename v2>
```

```
479         struct mul {
480             using type = val<v1::v* v2::v>;
481         };
482 
483         template<typename v1, typename v2>
484         struct div {
485             using type = val<v1::v / v2::v>;
486         };
487 
488         template<typename v1, typename v2>
489         struct remainder {
490             using type = val<v1::v % v2::v>;
491         };
492 
493         template<typename v1, typename v2>
494         struct gt {
495             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
496         };
497 
498         template<typename v1, typename v2>
499         struct lt {
500             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
501         };
502 
503         template<typename v1, typename v2>
504         struct eq {
505             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
506         };
507 
508         template<typename v1>
509         struct pos {
510             using type = std::bool_constant<(v1::v > 0)>;
511         };
512 
513     public:
515         template<typename v1, typename v2>
516         using add_t = typename add<v1, v2>::type;
517 
519         template<typename v1, typename v2>
520         using sub_t = typename sub<v1, v2>::type;
521 
523         template<typename v1, typename v2>
524         using mul_t = typename mul<v1, v2>::type;
525 
527         template<typename v1, typename v2>
528         using div_t = typename div<v1, v2>::type;
529 
531         template<typename v1, typename v2>
532         using mod_t = typename remainder<v1, v2>::type;
533 
535         template<typename v1, typename v2>
536         using gt_t = typename gt<v1, v2>::type;
537 
539         template<typename v1, typename v2>
540         using lt_t = typename lt<v1, v2>::type;
541 
543         template<typename v1, typename v2>
544         using eq_t = typename eq<v1, v2>::type;
545 
547         template<typename v1, typename v2>
548         using gcd_t = gcd_t<i32, v1, v2>;
549 
551         template<typename v>
552         using pos_t = typename pos<v>::type;
553 
554         template<typename v>
555         static constexpr bool pos_v = pos_t<v>::value;
556     };
557 }
558 
559 // i64
560 namespace aerobus {
562     struct i64 {
563         using inner_type = int64_t;
566         template<int64_t x>
567         struct val {
568             static constexpr int64_t v = x;
569 
572             template<typename valueType>
573             static constexpr valueType get() { return static_cast<valueType>(x); }
574 
576             using is_zero_t = std::bool_constant<x == 0>;
577 
579             static std::string to_string() {
580                 return std::to_string(x);
581             }
582 
```

```
585             template<typename valueRing>
586             static constexpr valueRing eval(const valueRing& v) {
587                 return static_cast<valueRing>(x);
588             }
589         };
590 
594         template<auto x>
595         using inject_constant_t = val<static_cast<int64_t>(x)>;
596 
597         template<typename v>
598         using inject_ring_t = v;
599 
601         using zero = val<0>;
603         using one = val<1>;
605         static constexpr bool is_field = false;
607         static constexpr bool is_euclidean_domain = true;
608 
609     private:
610         template<typename v1, typename v2>
611         struct add {
612             using type = val<v1::v + v2::v>;
613         };
614 
615         template<typename v1, typename v2>
616         struct sub {
617             using type = val<v1::v - v2::v>;
618         };
619 
620         template<typename v1, typename v2>
621         struct mul {
622             using type = val<v1::v* v2::v>;
623         };
624 
625         template<typename v1, typename v2>
626         struct div {
627             using type = val<v1::v / v2::v>;
628         };
629 
630         template<typename v1, typename v2>
631         struct remainder {
632             using type = val<v1::v% v2::v>;
633         };
634 
635         template<typename v1, typename v2>
636         struct gt {
637             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
638         };
639 
640         template<typename v1, typename v2>
641         struct lt {
642             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
643         };
644 
645         template<typename v1, typename v2>
646         struct eq {
647             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
648         };
649 
650         template<typename v>
651         struct pos {
652             using type = std::bool_constant<(v::v > 0)>;
653         };
654 
655     public:
657         template<typename v1, typename v2>
658         using add_t = typename add<v1, v2>::type;
659 
661         template<typename v1, typename v2>
662         using sub_t = typename sub<v1, v2>::type;
663 
665         template<typename v1, typename v2>
666         using mul_t = typename mul<v1, v2>::type;
667 
669         template<typename v1, typename v2>
670         using div_t = typename div<v1, v2>::type;
671 
673         template<typename v1, typename v2>
674         using mod_t = typename remainder<v1, v2>::type;
675 
677         template<typename v1, typename v2>
678         using gt_t = typename gt<v1, v2>::type;
679 
681         template<typename v1, typename v2>
682         using lt_t = typename lt<v1, v2>::type;
683 
685         template<typename v1, typename v2>
686         using eq_t = typename eq<v1, v2>::type;
```

```
687
689         template<typename v1, typename v2>
690         using gcd_t = gcd_t<i64, v1, v2>;
691
693         template<typename v>
694         using pos_t = typename pos<v>::type;
695
696         template<typename v>
697         static constexpr bool pos_v = pos_t<v>::value;
698     };
699 }
700
701 // z/pz
702 namespace aerobus {
707     template<int32_t p>
708     struct zpz {
709         using inner_type = int32_t;
710         template<int32_t x>
711         struct val {
712             static constexpr int32_t v = x % p;
713
714             template<typename valueType>
715             static constexpr valueType get() { return static_cast<valueType>(x % p); }
716
717             using is_zero_t = std::bool_constant<x% p == 0>;
718             static std::string to_string() {
719                 return std::to_string(x % p);
720             }
721
722             template<typename valueRing>
723             static constexpr valueRing eval(const valueRing& v) {
724                 return static_cast<valueRing>(x % p);
725             }
726         };
727
728         template<auto x>
729         using inject_constant_t = val<static_cast<int32_t>(x)>;
730
731         using zero = val<0>;
732         using one = val<1>;
733         static constexpr bool is_field = is_prime<p>::value;
734         static constexpr bool is_euclidean_domain = true;
735
736     private:
737         template<typename v1, typename v2>
738         struct add {
739             using type = val<(v1::v + v2::v) % p>;
740         };
741
742         template<typename v1, typename v2>
743         struct sub {
744             using type = val<(v1::v - v2::v) % p>;
745         };
746
747         template<typename v1, typename v2>
748         struct mul {
749             using type = val<(v1::v* v2::v) % p>;
750         };
751
752         template<typename v1, typename v2>
753         struct div {
754             using type = val<(v1::v% p) / (v2::v % p)>;
755         };
756
757         template<typename v1, typename v2>
758         struct remainder {
759             using type = val<(v1::v% v2::v) % p>;
760         };
761
762         template<typename v1, typename v2>
763         struct gt {
764             using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
765         };
766
767         template<typename v1, typename v2>
768         struct lt {
769             using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
770         };
771
772         template<typename v1, typename v2>
773         struct eq {
774             using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
775         };
776
777         template<typename v1>
778         struct pos {
779             using type = std::bool_constant<(v1::v > 0)>;
```

```
780         };
781
782     public:
783         template<typename v1, typename v2>
784         using add_t = typename add<v1, v2>::type;
785
786         template<typename v1, typename v2>
787         using sub_t = typename sub<v1, v2>::type;
788
789         template<typename v1, typename v2>
790         using mul_t = typename mul<v1, v2>::type;
791
792         template<typename v1, typename v2>
793         using div_t = typename div<v1, v2>::type;
794
795         template<typename v1, typename v2>
796         using mod_t = typename remainder<v1, v2>::type;
797
798         template<typename v1, typename v2>
799         using gt_t = typename gt<v1, v2>::type;
800
801         template<typename v1, typename v2>
802         using lt_t = typename lt<v1, v2>::type;
803
804         template<typename v1, typename v2>
805         using eq_t = typename eq<v1, v2>::type;
806
807         template<typename v1, typename v2>
808         using gcd_t = gcd_t<i32, v1, v2>;
809
810         template<typename v1>
811         using pos_t = typename pos<v1>::type;
812
813         template<typename v>
814         static constexpr bool pos_v = pos_t<v>::value;
815     };
816 }
817
818 // polynomial
819 namespace aerobus {
820     // coeffN x^N + ...
825     template<typename Ring, char variable_name = 'x'>
826     requires IsEuclideanDomain<Ring>
827     struct polynomial {
828         static constexpr bool is_field = false;
829         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
830
831         template<typename coeffN, typename...  coeffs>
832         struct val {
834             static constexpr size_t degree = sizeof...(coeffs);
836             using aN = coeffN;
838             using strip = val<coeffs...>;
840             using is_zero_t = std::bool_constant<(degree == 0) && (aN::is_zero_t::value)>;
841
842             private:
843             template<size_t index, typename E = void>
844             struct coeff_at {};
845
846             template<size_t index>
847             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))>> {
848                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
849             };
850
851             template<size_t index>
852             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))>> {
853                 using type = typename Ring::zero;
854             };
855
856             public:
859             template<size_t index>
860             using coeff_at_t = typename coeff_at<index>::type;
861
864             static std::string to_string() {
865                 return string_helper<coeffN, coeffs...>::func();
866             }
867
872             template<typename valueRing>
873             static constexpr valueRing eval(const valueRing& x) {
874                 return eval_helper<valueRing, val>::template inner<0, degree +
    1>::func(static_cast<valueRing>(0), x);
875             }
876         };
877
878         // specialization for constants
879         template<typename coeffN>
880         struct val<coeffN> {
881             static constexpr size_t degree = 0;
```

```
882             using aN = coeffN;
883             using strip = val<coeffN>;
884             using is_zero_t = std::bool_constant<aN::is_zero_t::value>;
885 
886             template<size_t index, typename E = void>
887             struct coeff_at {};
888 
889             template<size_t index>
890             struct coeff_at<index, std::enable_if_t<(index == 0)» {
891                 using type = aN;
892             };
893 
894             template<size_t index>
895             struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
896                 using type = typename Ring::zero;
897             };
898 
899             template<size_t index>
900             using coeff_at_t = typename coeff_at<index>::type;
901 
902             static std::string to_string() {
903                 return string_helper<coeffN>::func();
904             }
905 
906             template<typename valueRing>
907             static constexpr valueRing eval(const valueRing& x) {
908                 return static_cast<valueRing>(aN::template get<valueRing>());
909             }
910         };
911 
913         using zero = val<typename Ring::zero>;
915         using one = val<typename Ring::one>;
917         using X = val<typename Ring::one, typename Ring::zero>;
918 
919     private:
920         template<typename P, typename E = void>
921         struct simplify;
922 
923         template <typename P1, typename P2, typename I>
924         struct add_low;
925 
926         template<typename P1, typename P2>
927         struct add {
928             using type = typename simplify<typename add_low<
929             P1,
930             P2,
931             internal::make_index_sequence_reverse<
932             std::max(P1::degree, P2::degree) + 1
933             »::type>::type;
934         };
935 
936         template <typename P1, typename P2, typename I>
937         struct sub_low;
938 
939         template <typename P1, typename P2, typename I>
940         struct mul_low;
941 
942         template<typename v1, typename v2>
943         struct mul {
944                 using type = typename mul_low<
945                     v1,
946                     v2,
947                     internal::make_index_sequence_reverse<
948                     v1::degree + v2::degree + 1
949                     »::type;
950         };
951 
952         template<typename coeff, size_t deg>
953         struct monomial;
954 
955         template<typename v, typename E = void>
956         struct derive_helper {};
957 
958         template<typename v>
959         struct derive_helper<v, std::enable_if_t<v::degree == 0» {
960             using type = zero;
961         };
962 
963         template<typename v>
964         struct derive_helper<v, std::enable_if_t<v::degree != 0» {
965             using type = typename add<
966                 typename derive_helper<typename simplify<typename v::strip>::type>::type,
967                 typename monomial<
968                     typename Ring::template mul_t<
969                         typename v::aN,
970                         typename Ring::template inject_constant_t<(v::degree)>
971                     >,
```

```
972                 v::degree - 1
973             >::type
974         >::type;
975     };
976
977     template<typename v1, typename v2, typename E = void>
978     struct eq_helper {};
979
980     template<typename v1, typename v2>
981     struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
982         using type = std::false_type;
983     };
984
985
986     template<typename v1, typename v2>
987     struct eq_helper<v1, v2, std::enable_if_t<
988         v1::degree == v2::degree &&
989         (v1::degree != 0 || v2::degree != 0) &&
990         std::is_same<
991         typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
992         std::false_type
993         >::value
994     >
995     > {
996         using type = std::false_type;
997     };
998
999     template<typename v1, typename v2>
1000     struct eq_helper<v1, v2, std::enable_if_t<
1001         v1::degree == v2::degree &&
1002         (v1::degree != 0 || v2::degree != 0) &&
1003         std::is_same<
1004         typename Ring::template eq_t<typename v1::aN, typename v2::aN>,
1005         std::true_type
1006         >::value
1007     » {
1008         using type = typename eq_helper<typename v1::strip, typename v2::strip>::type;
1009     };
1010
1011     template<typename v1, typename v2>
1012     struct eq_helper<v1, v2, std::enable_if_t<
1013         v1::degree == v2::degree &&
1014         (v1::degree == 0)
1015     » {
1016         using type = typename Ring::template eq_t<typename v1::aN, typename v2::aN>;
1017     };
1018
1019     template<typename v1, typename v2, typename E = void>
1020     struct lt_helper {};
1021
1022     template<typename v1, typename v2>
1023     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1024         using type = std::true_type;
1025     };
1026
1027     template<typename v1, typename v2>
1028     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1029         using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
1030     };
1031
1032     template<typename v1, typename v2>
1033     struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1034         using type = std::false_type;
1035     };
1036
1037     template<typename v1, typename v2, typename E = void>
1038     struct gt_helper {};
1039
1040     template<typename v1, typename v2>
1041     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1042         using type = std::true_type;
1043     };
1044
1045     template<typename v1, typename v2>
1046     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1047         using type = std::false_type;
1048     };
1049
1050     template<typename v1, typename v2>
1051     struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1052         using type = std::false_type;
1053     };
1054
1055     // when high power is zero :  strip
1056     template<typename P>
1057     struct simplify<P, std::enable_if_t<
1058         std::is_same<
```

```
1059            typename Ring::zero,
1060            typename P::aN
1061            >::value && (P::degree > 0)
1062        »
1063        {
1064            using type = typename simplify<typename P::strip>::type;
1065        };
1066
1067        // otherwise :  do nothing
1068        template<typename P>
1069        struct simplify<P, std::enable_if_t<
1070            !std::is_same<
1071            typename Ring::zero,
1072            typename P::aN
1073            >::value && (P::degree > 0)
1074        »
1075        {
1076            using type = P;
1077        };
1078
1079        // do not simplify constants
1080        template<typename P>
1081        struct simplify<P, std::enable_if_t<P::degree == 0» {
1082            using type = P;
1083        };
1084
1085        // addition at
1086        template<typename P1, typename P2, size_t index>
1087        struct add_at {
1088            using type =
1089                typename Ring::template add_t<typename P1::template coeff_at_t<index>, typename
    P2::template coeff_at_t<index»;
1090        };
1091
1092        template<typename P1, typename P2, size_t index>
1093        using add_at_t = typename add_at<P1, P2, index>::type;
1094
1095        template<typename P1, typename P2, std::size_t...  I>
1096        struct add_low<P1, P2, std::index_sequence<I...» {
1097            using type = val<add_at_t<P1, P2, I>...>;
1098        };
1099
1100        // substraction at
1101        template<typename P1, typename P2, size_t index>
1102        struct sub_at {
1103            using type =
1104                typename Ring::template sub_t<typename P1::template coeff_at_t<index>, typename
    P2::template coeff_at_t<index»;
1105        };
1106
1107        template<typename P1, typename P2, size_t index>
1108        using sub_at_t = typename sub_at<P1, P2, index>::type;
1109
1110        template<typename P1, typename P2, std::size_t...  I>
1111        struct sub_low<P1, P2, std::index_sequence<I...» {
1112            using type = val<sub_at_t<P1, P2, I>...>;
1113        };
1114
1115        template<typename P1, typename P2>
1116        struct sub {
1117            using type = typename simplify<typename sub_low<
1118            P1,
1119            P2,
1120            internal::make_index_sequence_reverse<
1121            std::max(P1::degree, P2::degree) + 1
1122            »::type>::type;
1123        };
1124
1125        // multiplication at
1126        template<typename v1, typename v2, size_t k, size_t index, size_t stop>
1127        struct mul_at_loop_helper {
1128            using type = typename Ring::template add_t<
1129                typename Ring::template mul_t<
1130                typename v1::template coeff_at_t<index>,
1131                typename v2::template coeff_at_t<k - index>
1132                >,
1133                typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
1134            >;
1135        };
1136
1137        template<typename v1, typename v2, size_t k, size_t stop>
1138        struct mul_at_loop_helper<v1, v2, k, stop, stop> {
1139            using type = typename Ring::template mul_t<typename v1::template coeff_at_t<stop>, typename
    v2::template coeff_at_t<0»;
1140        };
1141
1142        template <typename v1, typename v2, size_t k, typename E = void>
```

```
1143        struct mul_at {};
1144
1145        template<typename v1, typename v2, size_t k>
1146        struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)» {
1147            using type = typename Ring::zero;
1148        };
1149
1150        template<typename v1, typename v2, size_t k>
1151        struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)» {
1152            using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
1153        };
1154
1155        template<typename P1, typename P2, size_t index>
1156        using mul_at_t = typename mul_at<P1, P2, index>::type;
1157
1158        template<typename P1, typename P2, std::size_t...  I>
1159        struct mul_low<P1, P2, std::index_sequence<I...» {
1160            using type = val<mul_at_t<P1, P2, I>...>;
1161        };
1162
1163        // division helper
1164        template< typename A, typename B, typename Q, typename R, typename E = void>
1165        struct div_helper {};
1166
1167        template<typename A, typename B, typename Q, typename R>
1168        struct div_helper<A, B, Q, R, std::enable_if_t<
1169            (R::degree < B::degree) ||
1170            (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
1171            using q_type = Q;
1172            using mod_type = R;
1173            using gcd_type = B;
1174        };
1175
1176        template<typename A, typename B, typename Q, typename R>
1177        struct div_helper<A, B, Q, R, std::enable_if_t<
1178            (R::degree >= B::degree) &&
1179            !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
1180        private:
1181            using rN = typename R::aN;
1182            using bN = typename B::aN;
1183            using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
    B::degree>::type;
1184            using rr = typename sub<R, typename mul<pT, B>::type>::type;
1185            using qq = typename add<Q, pT>::type;
1186
1187        public:
1188            using q_type = typename div_helper<A, B, qq, rr>::q_type;
1189            using mod_type = typename div_helper<A, B, qq, rr>::mod_type;
1190            using gcd_type = rr;
1191        };
1192
1193        template<typename A, typename B>
1194        struct div {
1195            static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
1196            using q_type = typename div_helper<A, B, zero, A>::q_type;
1197            using m_type = typename div_helper<A, B, zero, A>::mod_type;
1198        };
1199
1200
1201        template<typename P>
1202        struct make_unit {
1203            using type = typename div<P, val<typename P::aN»::q_type;
1204        };
1205
1206        template<typename coeff, size_t deg>
1207        struct monomial {
1208            using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
1209        };
1210
1211        template<typename coeff>
1212        struct monomial<coeff, 0> {
1213            using type = val<coeff>;
1214        };
1215
1216        template<typename valueRing, typename P>
1217        struct eval_helper
1218        {
1219            template<size_t index, size_t stop>
1220            struct inner {
1221                static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
1222                    constexpr valueRing coeff = static_cast<valueRing>(P::template coeff_at_t<P::degree
    - index>::template get<valueRing>());
1223                    return eval_helper<valueRing, P>::template inner<index + 1, stop>::func(x * accum +
    coeff, x);
1224                }
1225            };
1226
```

```
1227            template<size_t stop>
1228            struct inner<stop, stop> {
1229                static constexpr valueRing func(const valueRing& accum, const valueRing& x) {
1230                    return accum;
1231                }
1232            };
1233        };
1234
1235        template<typename coeff, typename...  coeffs>
1236        struct string_helper {
1237            static std::string func() {
1238                std::string tail = string_helper<coeffs...>::func();
1239                std::string result = "";
1240                if (Ring::template eq_t<coeff, typename Ring::zero>::value) {
1241                    return tail;
1242                }
1243                else if (Ring::template eq_t<coeff, typename Ring::one>::value) {
1244                    if (sizeof...(coeffs) == 1) {
1245                        result += std::string(1, variable_name);
1246                    }
1247                    else {
1248                        result += std::string(1, variable_name) + "^" +
     std::to_string(sizeof...(coeffs));
1249                    }
1250                }
1251                else {
1252                    if (sizeof...(coeffs) == 1) {
1253                        result += coeff::to_string() + " " + std::string(1, variable_name);
1254                    }
1255                    else {
1256                        result += coeff::to_string() + " " + std::string(1, variable_name) + "^" +
     std::to_string(sizeof...(coeffs));
1257                    }
1258                }
1259
1260                if(!tail.empty()) {
1261                    result += " + " + tail;
1262                }
1263
1264                return result;
1265            }
1266        };
1267
1268        template<typename coeff>
1269        struct string_helper<coeff> {
1270            static std::string func() {
1271                if(!std::is_same<coeff, typename Ring::zero>::value) {
1272                    return coeff::to_string();
1273                } else {
1274                    return "";
1275                }
1276            }
1277        };
1278
1279    public:
1282        template<typename P>
1283        using simplify_t = typename simplify<P>::type;
1284
1288        template<typename v1, typename v2>
1289        using add_t = typename add<v1, v2>::type;
1290
1294        template<typename v1, typename v2>
1295        using sub_t = typename sub<v1, v2>::type;
1296
1300        template<typename v1, typename v2>
1301        using mul_t = typename mul<v1, v2>::type;
1302
1306        template<typename v1, typename v2>
1307        using eq_t = typename eq_helper<v1, v2>::type;
1308
1312        template<typename v1, typename v2>
1313        using lt_t = typename lt_helper<v1, v2>::type;
1314
1318        template<typename v1, typename v2>
1319        using gt_t = typename gt_helper<v1, v2>::type;
1320
1324        template<typename v1, typename v2>
1325        using div_t = typename div<v1, v2>::q_type;
1326
1330        template<typename v1, typename v2>
1331        using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
1332
1336        template<typename coeff, size_t deg>
1337        using monomial_t = typename monomial<coeff, deg>::type;
1338
1341        template<typename v>
1342        using derive_t = typename derive_helper<v>::type;
```

```
1343
1346         template<typename v>
1347         using pos_t = typename Ring::template pos_t<typename v::aN>;
1348
1349         template<typename v>
1350         static constexpr bool pos_v = pos_t<v>::value;
1351
1355         template<typename v1, typename v2>
1356         using gcd_t = std::conditional_t<
1357             Ring::is_euclidean_domain,
1358             typename make_unit<gcd_t<polynomial<Ring, variable_name>, v1, v2»::type,
1359             void>;
1360
1364         template<auto x>
1365         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
1366
1370         template<typename v>
1371         using inject_ring_t = val<v>;
1372     };
1373 }
1374
1375 // big integers
1376 namespace aerobus {
1377     struct bigint {
1378         enum sign {
1379             positive,
1380             negative
1381         };
1382
1383         template<sign s, uint32_t an, uint32_t...  as>
1384         struct val {
1385             template<size_t index, typename E = void>
1386             struct digit_at {};
1387
1388             template<size_t index>
1389             struct digit_at<index, std::enable_if_t<(index <= sizeof...(as))» {
1390                 static constexpr uint32_t value = internal::value_at<(sizeof...(as) - index), an,
     as...>::value;
1391             };
1392
1393             template<size_t index>
1394             struct digit_at<index, std::enable_if_t<(index > sizeof...(as))» {
1395                 static constexpr uint32_t value = 0;
1396             };
1397
1398             static constexpr bool is_positive = s != sign::negative;
1399
1400             using strip = val<s, as...>;
1401             static constexpr uint32_t aN = an;
1402             static constexpr size_t digits = sizeof...(as) + 1;
1403
1404             static std::string to_string() {
1405                 return std::to_string(aN) + "B^" + std::to_string(digits-1) + " + " +
     strip::to_string();
1406             }
1407         };
1408
1409         template<typename I>
1410         struct is_zero {
1411             static constexpr bool value = I::digits == 1 && I::aN == 0;
1412         };
1413
1414         template<typename I>
1415         static constexpr bool is_zero_v = is_zero<I>::value;
1416
1417         template<sign s, uint32_t a0>
1418         struct val<s, a0> {
1419             using strip = val<s, a0>;
1420             static constexpr bool is_positive = s != sign::negative;
1421             static constexpr uint32_t aN = a0;
1422             static constexpr size_t digits = 1;
1423             template<size_t index, typename E = void>
1424             struct digit_at {};
1425             template<size_t index>
1426             struct digit_at<index, std::enable_if_t<index == 0» {
1427                 static constexpr uint32_t value = a0;
1428             };
1429
1430             template<size_t index>
1431             struct digit_at<index, std::enable_if_t<index != 0» {
1432                 static constexpr uint32_t value = 0;
1433             };
1434
1435             static std::string to_string() {
1436                 return std::to_string(a0);
1437             }
1438         };
```

```
1439
1440        using zero = val<sign::positive, 0>;
1441        using one = val<sign::positive, 1>;
1442
1443    private:
1444        template<uint32_t x, uint32_t y, uint8_t carry_in = 0>
1445        struct add_digit_helper {
1446        private:
1447            static constexpr uint64_t raw = ((uint64_t) x + (uint64_t) y + (uint64_t) carry_in);
1448        public:
1449            static constexpr uint32_t value = (uint32_t)(raw & 0xFFFF'FFFF);
1450            static constexpr uint8_t carry_out = (uint32_t) (raw » 32);
1451        };
1452
1453        template<typename I1, typename I2, size_t index, uint16_t carry_in = 0>
1454        struct add_at_helper {
1455            static_assert(I1::is_positive, "always add positive values");
1456            static_assert(I2::is_positive, "always add positive values");
1457        private:
1458            static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1459            static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1460        public:
1461            static constexpr uint32_t value = add_digit_helper<d1, d2, carry_in>::value;
1462            static constexpr uint8_t carry_out = add_digit_helper<d1, d2, carry_in>::carry_out;
1463        };
1464
1465        template<typename I, typename E = void>
1466        struct simplify {};
1467
1468        template<typename I>
1469        struct simplify<I, std::enable_if_t<I::aN == 0» {
1470            using type = typename I::strip;
1471        };
1472
1473        template<typename I>
1474        struct simplify<I, std::enable_if_t<I::aN != 0» {
1475            using type = I;
1476        };
1477
1478    public:
1479
1480        template<typename I>
1481        using simplify_t = typename simplify<I>::type;
1482
1483        // exposed for testing -- DO NOT USE
1484        template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1485        static constexpr uint32_t add_at_digit = add_at_helper<I1, I2, index, carry_in>::value;
1486        template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1487        static constexpr uint8_t add_at_carry = add_at_helper<I1, I2, index, carry_in>::carry_out;
1488
1489        // exposed for testing -- DO NOT USE
1490        template<typename I1, typename I2, size_t index>
1491        struct add_low_helper {
1492            private:
1493            using helper = add_at_helper<I1, I2, index, add_low_helper<I1, I2, index-1>::carry_out>;
1494            public:
1495            static constexpr uint32_t digit = helper::value;
1496            static constexpr uint8_t carry_out = helper::carry_out;
1497        };
1498
1499        // exposed for testing -- DO NOT USE
1500        template<typename I1, typename I2>
1501        struct add_low_helper<I1, I2, 0> {
1502            static constexpr uint32_t digit = add_at_helper<I1, I2, 0, 0>::value;
1503            static constexpr uint32_t carry_out = add_at_helper<I1, I2, 0, 0>::carry_out;
1504        };
1505
1506        template<typename I1, typename I2, typename I>
1507        struct add_low {};
1508
1509        template<typename I1, typename I2, std::size_t...  I>
1510        struct add_low<I1, I2, std::index_sequence<I...» {
1511            static_assert(I1::is_positive, "add works on positive values");
1512            static_assert(I2::is_positive, "add works on positive values");
1513            using type = val<sign::positive, add_low_helper<I1, I2, I>::digit...>;
1514        };
1515
1516        template<typename I1, typename I2>
1517        struct add {
1518            static_assert(I1::is_positive, "add works on positive values");
1519            static_assert(I2::is_positive, "add works on positive values");
1520            using type = simplify_t<
1521                typename add_low<
1522                        I1,
1523                        I2,
1524                        typename internal::make_index_sequence_reverse<std::max(I1::digits, I2::digits)
      + 1>
```

```
1525                     >::type>;
1526         };
1527     };
1528 }
1529
1530 // fraction field
1531 namespace aerobus {
1532     namespace internal {
1533         template<typename Ring, typename E = void>
1534         requires IsEuclideanDomain<Ring>
1535         struct _FractionField {};
1536
1537         template<typename Ring>
1538         requires IsEuclideanDomain<Ring>
1539         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain»
1540         {
1542             static constexpr bool is_field = true;
1543             static constexpr bool is_euclidean_domain = true;
1544
1545             private:
1546             template<typename val1, typename val2, typename E = void>
1547             struct to_string_helper {};
1548
1549             template<typename val1, typename val2>
1550             struct to_string_helper <val1, val2,
1551                 std::enable_if_t<
1552                 Ring::template eq_t<
1553                 val2, typename Ring::one
1554                 >::value
1555                 >
1556             > {
1557                 static std::string func() {
1558                     return val1::to_string();
1559                 }
1560             };
1561
1562             template<typename val1, typename val2>
1563             struct to_string_helper<val1, val2,
1564                 std::enable_if_t<
1565                 !Ring::template eq_t<
1566                 val2,
1567                 typename Ring::one
1568                 >::value
1569                 >
1570             > {
1571                 static std::string func() {
1572                     return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
1573                 }
1574             };
1575
1576             public:
1580             template<typename val1, typename val2>
1581             struct val {
1582                 using x = val1;
1583                 using y = val2;
1585                 using is_zero_t = typename val1::is_zero_t;
1586                 using ring_type = Ring;
1587                 using field_type = _FractionField<Ring>;
1588
1590                 static constexpr bool is_integer = std::is_same<val2, typename Ring::one>::value;
1591
1595                 template<typename valueType>
1596                 static constexpr valueType get() { return static_cast<valueType>(x::v) /
     static_cast<valueType>(y::v); }
1597
1600                 static std::string to_string() {
1601                     return to_string_helper<val1, val2>::func();
1602                 }
1603
1608                 template<typename valueRing>
1609                 static constexpr valueRing eval(const valueRing& v) {
1610                     return x::eval(v) / y::eval(v);
1611                 }
1612             };
1613
1615             using zero = val<typename Ring::zero, typename Ring::one>;
1617             using one = val<typename Ring::one, typename Ring::one>;
1618
1621             template<typename v>
1622             using inject_t = val<v, typename Ring::one>;
1623
1626             template<auto x>
1627             using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
     Ring::one>;
1628
1631             template<typename v>
1632             using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
```

```
1633
1634            using ring_type = Ring;
1635
1636        private:
1637            template<typename v, typename E = void>
1638            struct simplify {};
1639
1640            // x = 0
1641            template<typename v>
1642            struct simplify<v, std::enable_if_t<v::x::is_zero_t::value» {
1643                using type = typename _FractionField<Ring>::zero;
1644            };
1645
1646            // x != 0
1647            template<typename v>
1648            struct simplify<v, std::enable_if_t<!v::x::is_zero_t::value» {
1649
1650            private:
1651                using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
1652                using newx = typename Ring::template div_t<typename v::x, _gcd>;
1653                using newy = typename Ring::template div_t<typename v::y, _gcd>;
1654
1655                using posx = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
    sub_t<typename Ring::zero, newx>, newx>;
1656                using posy = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
    sub_t<typename Ring::zero, newy>, newy>;
1657            public:
1658                using type = typename _FractionField<Ring>::template val<posx, posy>;
1659            };
1660
1661        public:
1664            template<typename v>
1665            using simplify_t = typename simplify<v>::type;
1666
1667        private:
1668
1669            template<typename v1, typename v2>
1670            struct add {
1671            private:
1672                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
1673                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
1674                using dividend = typename Ring::template add_t<a, b>;
1675                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
1676                using g = typename Ring::template gcd_t<dividend, diviser>;
1677
1678            public:
1679                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
1680            };
1681
1682            template<typename v>
1683            struct pos {
1684                using type = std::conditional_t<
1685                    (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
1686                    (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
1687                    std::true_type,
1688                    std::false_type>;
1689
1690            };
1691
1692            template<typename v1, typename v2>
1693            struct sub {
1694            private:
1695                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
1696                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
1697                using dividend = typename Ring::template sub_t<a, b>;
1698                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
1699                using g = typename Ring::template gcd_t<dividend, diviser>;
1700
1701            public:
1702                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
1703            };
1704
1705            template<typename v1, typename v2>
1706            struct mul {
1707            private:
1708                using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
1709                using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
1710
1711            public:
1712                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
1713            };
1714
1715            template<typename v1, typename v2, typename E = void>
1716            struct div {};
1717
1718            template<typename v1, typename v2>
1719            struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
```

```
    _FractionField<Ring>::zero>::value»  {
1720            private:
1721                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
1722                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
1723
1724            public:
1725                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
1726            };
1727
1728            template<typename v1, typename v2>
1729            struct div<v1, v2, std::enable_if_t<
1730                std::is_same<zero, v1>::value && std::is_same<v2, zero>::value»  {
1731                using type = one;
1732            };
1733
1734            template<typename v1, typename v2>
1735            struct eq {
1736                using type = std::conditional_t<
1737                        std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
1738                        std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
1739                    std::true_type,
1740                    std::false_type>;
1741            };
1742
1743            template<typename TL, typename E = void>
1744            struct vadd {};
1745
1746            template<typename TL>
1747            struct vadd<TL, std::enable_if_t<(TL::length > 1)» {
1748                using head = typename TL::pop_front::type;
1749                using tail = typename TL::pop_front::tail;
1750                using type = typename add<head, typename vadd<tail>::type>::type;
1751            };
1752
1753            template<typename TL>
1754            struct vadd<TL, std::enable_if_t<(TL::length == 1)» {
1755                using type = typename TL::template at<0>;
1756            };
1757
1758            template<typename...  vals>
1759            struct vmul {};
1760
1761            template<typename v1, typename...  vals>
1762            struct vmul<v1, vals...> {
1763                using type = typename mul<v1, typename vmul<vals...>::type>::type;
1764            };
1765
1766            template<typename v1>
1767            struct vmul<v1> {
1768                using type = v1;
1769            };
1770
1771
1772            template<typename v1, typename v2, typename E = void>
1773            struct gt;
1774
1775            template<typename v1, typename v2>
1776            struct gt<v1, v2, std::enable_if_t<
1777                (eq<v1, v2>::type::value)
1778                » {
1779                using type = std::false_type;
1780            };
1781
1782            template<typename v1, typename v2>
1783            struct gt<v1, v2, std::enable_if_t<
1784                (!eq<v1, v2>::type::value) &&
1785                (!pos<v1>::type::value) && (!pos<v2>::type::value)
1786                » {
1787                using type = typename gt<
1788                    typename sub<zero, v1>::type, typename sub<zero, v2>::type
1789                >::type;
1790            };
1791
1792            template<typename v1, typename v2>
1793            struct gt<v1, v2, std::enable_if_t<
1794                (!eq<v1, v2>::type::value) &&
1795                (pos<v1>::type::value) && (!pos<v2>::type::value)
1796                » {
1797                using type = std::true_type;
1798            };
1799
1800            template<typename v1, typename v2>
1801            struct gt<v1, v2, std::enable_if_t<
1802                (!eq<v1, v2>::type::value) &&
1803                (!pos<v1>::type::value) && (pos<v2>::type::value)
1804                » {
1805                using type = std::false_type;
```

```
1806             };
1807
1808             template<typename v1, typename v2>
1809             struct gt<v1, v2, std::enable_if_t<
1810                 (!eq<v1, v2>::type::value) &&
1811                 (pos<v1>::type::value) && (pos<v2>::type::value)
1812                 » {
1813                 using type = typename Ring::template gt_t<
1814                     typename Ring::template mul_t<v1::x, v2::y>,
1815                     typename Ring::template mul_t<v2::y, v2::x>
1816                 >;
1817             };
1818
1819
1820         public:
1821             template<typename v1, typename v2>
1822             using add_t = typename add<v1, v2>::type;
1823             template<typename v1, typename v2>
1824             using mod_t = zero;
1825             template<typename v1, typename v2>
1826             using gcd_t = v1;
1827             template<typename...  vs>
1828             using vadd_t = typename vadd<vs...>::type;
1829             template<typename...  vs>
1830             using vmul_t = typename vmul<vs...>::type;
1831             template<typename v1, typename v2>
1832             using sub_t = typename sub<v1, v2>::type;
1833             template<typename v1, typename v2>
1834             using mul_t = typename mul<v1, v2>::type;
1835             template<typename v1, typename v2>
1836             using div_t = typename div<v1, v2>::type;
1837             template<typename v1, typename v2>
1838             using eq_t = typename eq<v1, v2>::type;
1839             template<typename v1, typename v2>
1840             using gt_t = typename gt<v1, v2>::type;
1841             template<typename v1>
1842             using pos_t = typename pos<v1>::type;
1843
1844             template<typename v>
1845             static constexpr bool pos_v = pos_t<v>::value;
1846         };
1847
1848         template<typename Ring, typename E = void>
1849         requires IsEuclideanDomain<Ring>
1850         struct FractionFieldImpl {};
1851
1852         // fraction field of a field is the field itself
1853         template<typename Field>
1854         requires IsEuclideanDomain<Field>
1855         struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {
1856             using type = Field;
1857             template<typename v>
1858             using inject_t = v;
1859         };
1860
1861         // fraction field of a ring is the actual fraction field
1862         template<typename Ring>
1863         requires IsEuclideanDomain<Ring>
1864         struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {
1865             using type = _FractionField<Ring>;
1866         };
1867     }
1868
1869     template<typename Ring>
1870     requires IsEuclideanDomain<Ring>
1871     using FractionField = typename internal::FractionFieldImpl<Ring>::type;
1872 }
1873
1874 // short names for common types
1875 namespace aerobus {
1876     using q32 = FractionField<i32>;
1877     using fpq32 = FractionField<polynomial<q32»;
1878     using q64 = FractionField<i64>;
1879     using pi64 = polynomial<i64>;
1880     using fpq64 = FractionField<polynomial<q64»;
1881     template<typename Ring, typename v1, typename v2>
1882     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
1883
1884     template<typename Ring, typename v1, typename v2>
1885     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
1886     template<typename Ring, typename v1, typename v2>
1887     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
1888 }
1889
1890 // taylor series and common integers (factorial, bernouilli...)  appearing in taylor coefficients
1891 namespace aerobus {
1892     namespace internal {
```

```
1917        template<typename T, size_t x, typename E = void>
1918        struct factorial {};
1919
1920        template<typename T, size_t x>
1921        struct factorial<T, x, std::enable_if_t<(x > 0)» {
1922        private:
1923            template<typename, size_t, typename>
1924            friend struct factorial;
1925        public:
1926            using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
    x - 1>::type>;
1927            static constexpr typename T::inner_type value = type::template get<typename
    T::inner_type>();
1928        };
1929
1930        template<typename T>
1931        struct factorial<T, 0> {
1932        public:
1933            using type = typename T::one;
1934            static constexpr typename T::inner_type value = type::template get<typename
    T::inner_type>();
1935        };
1936    }
1937
1941    template<typename T, size_t i>
1942    using factorial_t = typename internal::factorial<T, i>::type;
1943
1944    template<typename T, size_t i>
1945    inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
1946
1947    namespace internal {
1948        template<typename T, size_t k, size_t n, typename E = void>
1949        struct combination_helper {};
1950
1951        template<typename T, size_t k, size_t n>
1952        struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
1953            using type = typename FractionField<T>::template mul_t<
1954                typename combination_helper<T, k - 1, n - 1>::type,
1955                makefraction_t<T, typename T::template val<n>, typename T::template val<k»>;
1956        };
1957
1958        template<typename T, size_t k, size_t n>
1959        struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
1960            using type = typename combination_helper<T, n - k, n>::type;
1961        };
1962
1963        template<typename T, size_t n>
1964        struct combination_helper<T, 0, n> {
1965            using type = typename FractionField<T>::one;
1966        };
1967
1968        template<typename T, size_t k, size_t n>
1969        struct combination {
1970            using type = typename internal::combination_helper<T, k, n>::type::x;
1971            static constexpr typename T::inner_type value = internal::combination_helper<T, k,
    n>::type::template get<typename T::inner_type>();
1972        };
1973    }
1974
1977    template<typename T, size_t k, size_t n>
1978    using combination_t = typename internal::combination<T, k, n>::type;
1979
1980    template<typename T, size_t k, size_t n>
1981    inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
1982
1983    namespace internal {
1984        template<typename T, size_t m>
1985        struct bernouilli;
1986
1987        template<typename T, typename accum, size_t k, size_t m>
1988        struct bernouilli_helper {
1989            using type = typename bernouilli_helper<
1990                T,
1991                addfractions_t<T,
1992                    accum,
1993                    mulfractions_t<T,
1994                        makefraction_t<T,
1995                            combination_t<T, k, m + 1>,
1996                            typename T::one>,
1997                        typename bernouilli<T, k>::type
1998                    >
1999                >,
2000                k + 1,
2001                m>::type;
2002        };
2003
2004        template<typename T, typename accum, size_t m>
```

```
2005         struct bernouilli_helper<T, accum, m, m>
2006         {
2007             using type = accum;
2008         };
2009 
2010 
2011 
2012         template<typename T, size_t m>
2013         struct bernouilli {
2014             using type = typename FractionField<T>::template mul_t<
2015                 typename internal::bernouilli_helper<T, typename FractionField<T>::zero, 0, m>::type,
2016                 makefraction_t<T,
2017                 typename T::template val<static_cast<typename T::inner_type>(-1)>,
2018                 typename T::template val<static_cast<typename T::inner_type>(m + 1)>
2019                 >
2020             >;
2021 
2022             template<typename floatType>
2023             static constexpr floatType value = type::template get<floatType>();
2024         };
2025 
2026         template<typename T>
2027         struct bernouilli<T, 0> {
2028             using type = typename FractionField<T>::one;
2029 
2030             template<typename floatType>
2031             static constexpr floatType value = type::template get<floatType>();
2032         };
2033     }
2034 
2038     template<typename T, size_t n>
2039     using bernouilli_t = typename internal::bernouilli<T, n>::type;
2040 
2041     template<typename FloatType, typename T, size_t n >
2042     inline constexpr FloatType bernouilli_v = internal::bernouilli<T, n>::template value<FloatType>;
2043 
2044     namespace internal {
2045         template<typename T, int k, typename E = void>
2046         struct alternate {};
2047 
2048         template<typename T, int k>
2049         struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
2050             using type = typename T::one;
2051             static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
2052         };
2053 
2054         template<typename T, int k>
2055         struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
2056             using type = typename T::template sub_t<typename T::zero, typename T::one>;
2057             static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
2058         };
2059     }
2060 
2063     template<typename T, int k>
2064     using alternate_t = typename internal::alternate<T, k>::type;
2065 
2066     template<typename T, size_t k>
2067     inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
2068 
2069     // pow
2070     namespace internal {
2071         template<typename T, auto p, auto n>
2072         struct pow {
2073             using type = typename T::template mul_t<typename T::template val<p>, typename pow<T, p, n -
     1>::type>;
2074         };
2075 
2076         template<typename T, auto p>
2077         struct pow<T, p, 0> { using type = typename T::one; };
2078     }
2079 
2080     template<typename T, auto p, auto n>
2081     using pow_t = typename internal::pow<T, p, n>::type;
2082 
2083     namespace internal {
2084         template<typename, template<typename, size_t> typename, class>
2085         struct make_taylor_impl;
2086 
2087         template<typename T, template<typename, size_t> typename coeff_at, size_t...  Is>
2088         struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {
2089             using type = typename polynomial<FractionField<T»::template val<typename coeff_at<T,
     Is>::type...>;
2090         };
2091     }
2092 
```

```
2093     // generic taylor serie, depending on coefficients
2094     template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
2095     using taylor = typename internal::make_taylor_impl<T, coeff_at,
     internal::make_index_sequence_reverse<deg + 1>::type;
2096
2097     namespace internal {
2098         template<typename T, size_t i>
2099         struct exp_coeff {
2100             using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
2101         };
2102
2103         template<typename T, size_t i, typename E = void>
2104         struct sin_coeff_helper {};
2105
2106         template<typename T, size_t i>
2107         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
2108             using type = typename FractionField<T>::zero;
2109         };
2110
2111         template<typename T, size_t i>
2112         struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
2113             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>;
2114         };
2115
2116         template<typename T, size_t i>
2117         struct sin_coeff {
2118             using type = typename sin_coeff_helper<T, i>::type;
2119         };
2120
2121         template<typename T, size_t i, typename E = void>
2122         struct sh_coeff_helper {};
2123
2124         template<typename T, size_t i>
2125         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
2126             using type = typename FractionField<T>::zero;
2127         };
2128
2129         template<typename T, size_t i>
2130         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
2131             using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
2132         };
2133
2134         template<typename T, size_t i>
2135         struct sh_coeff {
2136             using type = typename sh_coeff_helper<T, i>::type;
2137         };
2138
2139         template<typename T, size_t i, typename E = void>
2140         struct cos_coeff_helper {};
2141
2142         template<typename T, size_t i>
2143         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
2144             using type = typename FractionField<T>::zero;
2145         };
2146
2147         template<typename T, size_t i>
2148         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
2149             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>;
2150         };
2151
2152         template<typename T, size_t i>
2153         struct cos_coeff {
2154             using type = typename cos_coeff_helper<T, i>::type;
2155         };
2156
2157         template<typename T, size_t i, typename E = void>
2158         struct cosh_coeff_helper {};
2159
2160         template<typename T, size_t i>
2161         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
2162             using type = typename FractionField<T>::zero;
2163         };
2164
2165         template<typename T, size_t i>
2166         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
2167             using type = makefraction_t<T, typename T::one, factorial_t<T, i>;
2168         };
2169
2170         template<typename T, size_t i>
2171         struct cosh_coeff {
2172             using type = typename cosh_coeff_helper<T, i>::type;
2173         };
2174
2175         template<typename T, size_t i>
2176         struct geom_coeff { using type = typename FractionField<T>::one; };
2177
2178
```

```
2179        template<typename T, size_t i, typename E = void>
2180        struct atan_coeff_helper;
2181
2182        template<typename T, size_t i>
2183        struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
2184            using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i»;
2185        };
2186
2187        template<typename T, size_t i>
2188        struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
2189            using type = typename FractionField<T>::zero;
2190        };
2191
2192        template<typename T, size_t i>
2193        struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
2194
2195        template<typename T, size_t i, typename E = void>
2196        struct asin_coeff_helper;
2197
2198        template<typename T, size_t i>
2199        struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
2200        {
2201            using type = makefraction_t<T,
2202                factorial_t<T, i - 1>,
2203                typename T::template mul_t<
2204                    typename T::template val<i>,
2205                    T::template mul_t<
2206                        pow_t<T, 4, i / 2>,
2207                        pow<T, factorial<T, i / 2>::value, 2
2208                    >
2209                >
2210                »;
2211        };
2212
2213        template<typename T, size_t i>
2214        struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
2215        {
2216            using type = typename FractionField<T>::zero;
2217        };
2218
2219        template<typename T, size_t i>
2220        struct asin_coeff {
2221            using type = typename asin_coeff_helper<T, i>::type;
2222        };
2223
2224        template<typename T, size_t i>
2225        struct lnp1_coeff {
2226            using type = makefraction_t<T,
2227                alternate_t<T, i + 1>,
2228                typename T::template val<i»;
2229        };
2230
2231        template<typename T>
2232        struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
2233
2234        template<typename T, size_t i, typename E = void>
2235        struct asinh_coeff_helper;
2236
2237        template<typename T, size_t i>
2238        struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
2239        {
2240            using type = makefraction_t<T,
2241                typename T::template mul_t<
2242                    alternate_t<T, i / 2>,
2243                    factorial_t<T, i - 1>
2244                >,
2245                typename T::template mul_t<
2246                    T::template mul_t<
2247                        typename T::template val<i>,
2248                        pow_t<T, (factorial<T, i / 2>::value), 2>
2249                    >,
2250                    pow_t<T, 4, i / 2>
2251                >
2252            >;
2253        };
2254
2255        template<typename T, size_t i>
2256        struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
2257        {
2258            using type = typename FractionField<T>::zero;
2259        };
2260
2261        template<typename T, size_t i>
2262        struct asinh_coeff {
2263            using type = typename asinh_coeff_helper<T, i>::type;
2264        };
2265
```

```
2266        template<typename T, size_t i, typename E = void>
2267        struct atanh_coeff_helper;
2268
2269        template<typename T, size_t i>
2270        struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
2271        {
2272            // 1/i
2273            using type = typename FractionField<T>::  template val<
2274                typename T::one,
2275                typename T::template val<static_cast<typename T::inner_type>(i)»;
2276        };
2277
2278        template<typename T, size_t i>
2279        struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
2280        {
2281            using type = typename FractionField<T>::zero;
2282        };
2283
2284        template<typename T, size_t i>
2285        struct atanh_coeff {
2286            using type = typename asinh_coeff_helper<T, i>::type;
2287        };
2288
2289        template<typename T, size_t i, typename E = void>
2290        struct tan_coeff_helper;
2291
2292        template<typename T, size_t i>
2293        struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
2294            using type = typename FractionField<T>::zero;
2295        };
2296
2297        template<typename T, size_t i>
2298        struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
2299        private:
2300            // 4^((i+1)/2)
2301            using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
2302            // 4^((i+1)/2) - 1
2303            using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
    FractionField<T>::one>;
2304            // (-1)^((i-1)/2)
2305            using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
2306            using dividend = typename FractionField<T>::template mul_t<
2307                altp,
2308                FractionField<T>::template mul_t<
2309                _4p,
2310                FractionField<T>::template mul_t<
2311                _4pm1,
2312                bernouilli_t<T, (i + 1)>
2313                >
2314                >
2315            >;
2316        public:
2317            using type = typename FractionField<T>::template div_t<dividend,
2318                typename FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
2319        };
2320
2321        template<typename T, size_t i>
2322        struct tan_coeff {
2323            using type = typename tan_coeff_helper<T, i>::type;
2324        };
2325
2326        template<typename T, size_t i, typename E = void>
2327        struct tanh_coeff_helper;
2328
2329        template<typename T, size_t i>
2330        struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
2331            using type = typename FractionField<T>::zero;
2332        };
2333
2334        template<typename T, size_t i>
2335        struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
2336        private:
2337            using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
2338            using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
    FractionField<T>::one>;
2339            using dividend =
2340                typename FractionField<T>::template mul_t<
2341                _4p,
2342                typename FractionField<T>::template mul_t<
2343                _4pm1,
2344                bernouilli_t<T, (i + 1)>
2345                >
2346                >::type;
2347        public:
2348            using type = typename FractionField<T>::template div_t<dividend,
2349                FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
2350        };
```

```
2351
2352         template<typename T, size_t i>
2353         struct tanh_coeff {
2354             using type = typename tanh_coeff_helper<T, i>::type;
2355         };
2356     }
2357
2361     template<typename T, size_t deg>
2362     using exp = taylor<T, internal::exp_coeff, deg>;
2363
2367     template<typename T, size_t deg>
2368     using expm1 = typename polynomial<FractionField<T>>::template sub_t<
2369         exp<T, deg>,
2370         typename polynomial<FractionField<T>>::one>;
2371
2375     template<typename T, size_t deg>
2376     using lnp1 = taylor<T, internal::lnp1_coeff, deg>;
2377
2381     template<typename T, size_t deg>
2382     using atan = taylor<T, internal::atan_coeff, deg>;
2383
2387     template<typename T, size_t deg>
2388     using sin = taylor<T, internal::sin_coeff, deg>;
2389
2393     template<typename T, size_t deg>
2394     using sinh = taylor<T, internal::sh_coeff, deg>;
2395
2399     template<typename T, size_t deg>
2400     using cosh = taylor<T, internal::cosh_coeff, deg>;
2401
2405     template<typename T, size_t deg>
2406     using cos = taylor<T, internal::cos_coeff, deg>;
2407
2411     template<typename T, size_t deg>
2412     using geometric_sum = taylor<T, internal::geom_coeff, deg>;
2413
2417     template<typename T, size_t deg>
2418     using asin = taylor<T, internal::asin_coeff, deg>;
2419
2423     template<typename T, size_t deg>
2424     using asinh = taylor<T, internal::asinh_coeff, deg>;
2425
2429     template<typename T, size_t deg>
2430     using atanh = taylor<T, internal::atanh_coeff, deg>;
2431
2435     template<typename T, size_t deg>
2436     using tan = taylor<T, internal::tan_coeff, deg>;
2437
2441     template<typename T, size_t deg>
2442     using tanh = taylor<T, internal::tanh_coeff, deg>;
2443 }
2444
2445 // continued fractions
2446 namespace aerobus {
2449     template<int64_t...  values>
2450     struct ContinuedFraction {};
2451
2452     template<int64_t a0>
2453     struct ContinuedFraction<a0> {
2454         using type = typename q64::template inject_constant_t<a0>;
2455         static constexpr double val = type::template get<double>();
2456     };
2457
2458     template<int64_t a0, int64_t...  rest>
2459     struct ContinuedFraction<a0, rest...> {
2460         using type = q64::template add_t<
2461                 typename q64::template inject_constant_t<a0>,
2462                 typename q64::template div_t<
2463                     typename q64::one,
2464                     typename ContinuedFraction<rest...>::type
2465                 »;
2466         static constexpr double val = type::template get<double>();
2467     };
2468
2473     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>;
2476     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>;
2478     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>;
2480     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>;
2481 }
2482
2483 // known polynomials
2484 namespace aerobus {
2485     namespace internal {
```

```
2486         template<int kind, int deg>
2487         struct chebyshev_helper {
2488             using type = typename pi64::template sub_t<
2489                 typename pi64::template mul_t<
2490                     typename pi64::template mul_t<
2491                         pi64::inject_constant_t<2>,
2492                         typename pi64::X
2493                     >,
2494                     typename chebyshev_helper<kind, deg-1>::type
2495                 >,
2496                 typename chebyshev_helper<kind, deg-2>::type
2497             >;
2498         };
2499 
2500         template<>
2501         struct chebyshev_helper<1, 0> {
2502             using type = typename pi64::one;
2503         };
2504 
2505         template<>
2506         struct chebyshev_helper<1, 1> {
2507             using type = typename pi64::X;
2508         };
2509 
2510         template<>
2511         struct chebyshev_helper<2, 0> {
2512             using type = typename pi64::one;
2513         };
2514 
2515         template<>
2516         struct chebyshev_helper<2, 1> {
2517             using type = typename pi64::template mul_t<
2518                             typename pi64::inject_constant_t<2>,
2519                             typename pi64::X>;
2520         };
2521     }
2522 
2525     template<size_t deg>
2526     using chebyshev_T = typename internal::chebyshev_helper<1, deg>::type;
2527 
2530     template<size_t deg>
2531     using chebyshev_U = typename internal::chebyshev_helper<2, deg>::type;
2532 }
```

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 PI_fraction::val

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

7.5 E_fraction::val

approximation of e -> 2.718...

approximation of e -> 2.718...

Index

add_t
    aerobus::polynomial< Ring, variable_name >, 20
aerobus::bigint, 10
aerobus::bigint::add< I1, I2 >, 9
aerobus::bigint::add_low< I1, I2, I >, 9
aerobus::bigint::add_low< I1, I2, std::index_sequence< I... > >, 9
aerobus::bigint::add_low_helper< I1, I2, 0 >, 10
aerobus::bigint::add_low_helper< I1, I2, index >, 10
aerobus::bigint::is_zero< I >, 18
aerobus::bigint::val< s, a0 >, 34
aerobus::bigint::val< s, a0 >::digit_at< index, E >, 13
aerobus::bigint::val< s, a0 >::digit_at< index, std::enable_if_t< index !=0 > >, 13
aerobus::bigint::val< s, a0 >::digit_at< index, std::enable_if_t< index==0 > >, 13
aerobus::bigint::val< s, an, as >, 27
aerobus::bigint::val< s, an, as >::digit_at< index, E >, 13
aerobus::bigint::val< s, an, as >::digit_at< index, std::enable_if_t<(index > sizeof...(as))> >, 14
aerobus::bigint::val< s, an, as >::digit_at< index, std::enable_if_t<(index<=sizeof...(as))> >, 14
aerobus::ContinuedFraction< a0 >, 12
aerobus::ContinuedFraction< a0, rest... >, 12
aerobus::ContinuedFraction< values >, 12
aerobus::i32, 14
aerobus::i32::val< x >, 28
    eval, 28
    get, 29
aerobus::i64, 16
aerobus::i64::val< x >, 29
    eval, 30
    get, 30
aerobus::is_prime< n >, 18
aerobus::IsEuclideanDomain, 7
aerobus::IsField, 7
aerobus::IsRing, 8
aerobus::polynomial< Ring, variable_name >, 19
    add_t, 20
    derive_t, 20
    div_t, 21
    eq_t, 21
    gcd_t, 21
    gt_t, 22
    lt_t, 22
    mod_t, 22
    monomial_t, 23
    mul_t, 23
    pos_t, 23
    simplify_t, 24
    sub_t, 24
aerobus::polynomial< Ring, variable_name >::eval_helper< valueRing, P >::inner< index, stop >, 17
aerobus::polynomial< Ring, variable_name >::eval_helper< valueRing, P >::inner< stop, stop >, 18
aerobus::polynomial< Ring, variable_name >::val< coeffN >, 33
aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, E >, 11
aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, std::enable_if_t<(index< 0 || index > 0)> >, 11
aerobus::polynomial< Ring, variable_name >::val< coeffN >::coeff_at< index, std::enable_if_t<(index==0)> >, 11
aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs >, 31
    coeff_at_t, 31
    eval, 32
    to_string, 32
aerobus::Quotient< Ring, X >, 25
aerobus::Quotient< Ring, X >::val< V >, 33
aerobus::type_list< Ts >, 26
aerobus::type_list< Ts >::pop_front, 25
aerobus::type_list< Ts >::split< index >, 26
aerobus::type_list<>, 27
aerobus::zpz< p >, 35
aerobus::zpz< p >::val< x >, 33

coeff_at_t
    aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs >, 31

derive_t
    aerobus::polynomial< Ring, variable_name >, 20
div_t
    aerobus::polynomial< Ring, variable_name >, 21

eq_t
    aerobus::polynomial< Ring, variable_name >, 21
eval
    aerobus::i32::val< x >, 28
    aerobus::i64::val< x >, 30
    aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs >, 32

gcd_t

aerobus::polynomial< Ring, variable_name >, 21
get
aerobus::i32::val< x >, 29
aerobus::i64::val< x >, 30
gt_t
aerobus::polynomial< Ring, variable_name >, 22

lt_t
aerobus::polynomial< Ring, variable_name >, 22

mod_t
aerobus::polynomial< Ring, variable_name >, 22
monomial_t
aerobus::polynomial< Ring, variable_name >, 23
mul_t
aerobus::polynomial< Ring, variable_name >, 23

pos_t
aerobus::polynomial< Ring, variable_name >, 23

simplify_t
aerobus::polynomial< Ring, variable_name >, 24
src/lib.h, 37
sub_t
aerobus::polynomial< Ring, variable_name >, 24

to_string
aerobus::polynomial< Ring, variable_name >::val< coeffN, coeffs >, 32