Aerobus

Generated by Doxygen 1.9.4

Chapter 1

Concept Index

1.1 Concepts

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Concept Documentation

4.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

`#include <lib.h>`

4.1.1 Concept definition

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

4.1.2 Detailed Description

Concept to express R is an euclidean domain.

4.2 aerobus::IsField Concept Reference

Concept to express R is a field.

`#include <lib.h>`

4.2.1 Concept definition

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

4.2.2 Detailed Description

Concept to express R is a field.

4.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring (ordered)

`#include <lib.h>`

4.3.1 Concept definition

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

4.3.2 Detailed Description

Concept to express R is a Ring (ordered)

Chapter 5

Class Documentation

5.1 aerobus::bigint Struct Reference

Classes

- struct val
- struct val< s, a0 >

Public Types

- enum **signs** { **positive** , **negative** }
- using **zero** = val< signs::positive, 0 >
- using **one** = val< signs::positive, 1 >
- template<typename I >
  using **minus_t** = I::minus_t
- template<typename I >
  using **simplify_t** = typename simplify< I >::type
- template<typename I1 , typename I2 >
  using **add_t** = typename add< I1, I2 >::type
- template<typename I1 , typename I2 >
  using **sub_t** = typename sub< I1, I2 >::type

Static Public Member Functions

- static constexpr signs **opposite** (const signs &s)

Static Public Attributes

- template<typename I1 , typename I2 >
  static constexpr bool **eq_v** = eq<I1, I2>::value
- template<typename I >
  static constexpr bool **pos_v** = I::sign == signs::positive && !I::is_zero_v
- template<typename I1 , typename I2 >
  static constexpr bool **gt_v** = gt_helper<I1, I2>::value

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

- src/lib.h

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

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

- src/lib.h

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

Public Types

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

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

- src/lib.h

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

Public Types

- using **type** = aN

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

- src/lib.h

5.5 aerobus::ContinuedFraction< values > Struct Template Reference

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

`#include <lib.h>`

5.5.1 Detailed Description

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

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

Template Parameters

...values	

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

- src/lib.h

5.6 aerobus::ContinuedFraction< a0 > Struct Template Reference

Public Types

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

Static Public Attributes

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

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

- src/lib.h

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

Public Types

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

Static Public Attributes

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

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

- src/lib.h

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

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

- src/lib.h

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

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

- src/lib.h

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

Static Public Attributes

- static constexpr uint32_t **value** = 0

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

- src/lib.h

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

Static Public Attributes

- static constexpr uint32_t **value** = a0

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

- src/lib.h

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

Static Public Attributes

- static constexpr uint32_t **value** = 0

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

- src/lib.h

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

Static Public Attributes

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

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

- src/lib.h

5.14 aerobus::i32 Struct Reference

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

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

Classes

- struct val

  values in i32

Public Types

- using **inner_type** = int32_t
- using **zero** = val< 0 >

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

  constant one
- template<auto x>
  using **inject_constant_t** = val< static_cast< int32_t >(x)>
- template<typename v >
  using **inject_ring_t** = v
- template<typename v1 , typename v2 >
  using **add_t** = typename add< v1, v2 >::type

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

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

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

  multiplication operator

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

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

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

  strict less operator (v1 < v2)
- template<typename v1 , typename v2 >
  using **gcd_t** = gcd_t< i32, v1, v2 >

  greatest common divisor

Static Public Attributes

- static constexpr bool **is_field** = false

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

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

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

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

  positivity (v1 > 0)

5.14.1 Detailed Description

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

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

- src/lib.h

5.15 aerobus::i64 Struct Reference

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

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

Classes

- struct val

  values in i64

Public Types

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

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

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

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

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

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

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

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

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

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

  greatest common divisor

Static Public Attributes

- static constexpr bool **is_field** = false

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

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

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

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

  is v posititive

5.15.1 Detailed Description

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

5.15.2 Member Data Documentation

5.15.2.1 pos_v

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

is v posititive

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

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

- src/lib.h

5.16 aerobus::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

5.17 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

5.18 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

5.18.1 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

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

`#include <lib.h>`

Classes

- struct val
- struct val< coeffN >

Public Types

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

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

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

generator

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

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

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

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

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

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

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

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

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

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

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

Static Public Attributes

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

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

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

  checks for positivity (an > 0)

5.19.1 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

5.19.2 Member Typedef Documentation

5.19.2.1 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	

5.19.2.2 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	

5.19.2.3 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	

5.19.2.4 gcd_t

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

greatest common divisor of two polynomials

Template Parameters

v1	
v2	

5.19.2.5 lt_t

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

strict less operator

Template Parameters

v1	
v2	

5.19.2.6 mod_t

```
template<typename Ring , char variable_name = 'x'>
template<typename v1 , typename v2 >
```

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

modulo operator

Template Parameters

v1	
v2	

5.19.2.7 monomial_t

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

monomial : coeff X^deg

Template Parameters

coeff	
deg	

5.19.2.8 mul_t

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

multiplication of two polynomials

Template Parameters

v1	
v2	

5.19.2.9 simplify_t

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

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

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

Template Parameters

P	

5.19.2.10 sub_t

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

substraction of two polynomials

Template Parameters

v1	
v2	

5.19.3 Member Data Documentation

5.19.3.1 eq_v

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

equality operator

Template Parameters

v1	
v2	

5.19.3.2 gt_v

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

strict greater operator

Template Parameters

v1	
v2	

5.19.3.3 pos_v

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

checks for positivity (an > 0)

Template Parameters

v	

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

- src/lib.h

5.20 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

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

Classes

- struct val

Public Types

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

Static Public Attributes

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

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

- src/lib.h

5.22 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

5.23 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)

5.23.1 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

5.24 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

5.25 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... >
- using **minus_t** = val< opposite(s), an, as... >

Static Public Member Functions

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

Static Public Attributes

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

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

- src/lib.h

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

values in i32

`#include <lib.h>`

Static Public Member Functions

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

Static Public Attributes

- static constexpr int32_t **v** = x
- static constexpr bool **is_zero_v** = x == 0
  is value zero

5.26.1 Detailed Description

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

values in i32

Template Parameters

x	an actual integer

5.26.2 Member Function Documentation

5.26.2.1 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

5.26.2.2 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

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

values in i64

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

Static Public Member Functions

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

Static Public Attributes

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

5.27.1 Detailed Description

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

values in i64

Template Parameters

x	an actual integer

5.27.2 Member Function Documentation

5.27.2.1 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)

5.27.2.2 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

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

Public Types

- using **aN** = coeffN

heavy weight coefficient (non zero)

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

  remove largest coefficient

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

  coefficient at index

Static Public Member Functions

- static std::string to_string ()

  get a string representation of polynomial

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

  evaluates polynomial seen as a function operating on ValueRing

Static Public Attributes

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

  degree of the polynomial

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

  true if polynomial is constant zero

5.28.1 Member Typedef Documentation

5.28.1.1 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	

5.28.2 Member Function Documentation

5.28.2.1 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)

5.28.2.2 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

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

Public Types

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

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

- src/lib.h

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

Static Public Member Functions

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

Static Public Attributes

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

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

- src/lib.h

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

Classes

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

Public Types

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

Static Public Member Functions

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

Static Public Attributes

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

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

- src/lib.h

5.32 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 >
- using **minus_t** = val< opposite(s), a0 >

Static Public Member Functions

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

Static Public Attributes

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

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

- src/lib.h

5.33 aerobus::zpz< p > Struct Template Reference

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

Classes

- struct val

Public Types

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

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

Static Public Attributes

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

5.33.1 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          typename R::template minus_t<typename R::one>;
75          R::template eq_v<typename R::one, typename R::one> == true;
76      };
77
79      template <typename R>
80      concept IsEuclideanDomain = IsRing<R> && requires {
81          typename R::template div_t<typename R::one, typename R::one>;
82          typename R::template mod_t<typename R::one, typename R::one>;
83          typename R::template gcd_t<typename R::one, typename R::one>;
84
85          R::template pos_v<typename R::one> == true;
86          R::template gt_v<typename R::one, typename R::zero> == true;
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_v && Ring::template pos_v<A»>
202             {
203                 using type = A;
204             };
205 
206             // B = 0, A < 0
207             template<typename A, typename B>
208             struct gcd_helper<A, B, std::enable_if_t<
209                 B::is_zero_v && !Ring::template pos_v<A»>
210             {
211                 using type = typename Ring::template minus_t<A>;
212             };
213 
214             // B != 0
215             template<typename A, typename B>
216             struct gcd_helper<A, B, std::enable_if_t<
217                 (!B::is_zero_v)
218                 » {
219             private:
220                 // A / B
221                 using k = typename Ring::template div_t<A, B>;
222                 // A - (A/B)*B = A % B
223                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
224             public:
225                 using type = typename gcd_helper<B, m>::type;
226             };
227 
228             template<typename A, typename B>
229             using type = typename gcd_helper<A, B>::type;
230         };
231     }
232 
235     template<typename T, typename A, typename B>
236     using gcd_t = typename internal::gcd<T>::template type<A, B>;
237 }
238 
239 // quotient ring by the principal ideal generated by X
240 namespace aerobus {
241     template<typename Ring, typename X>
242     requires IsRing<Ring>
243     struct Quotient {
244         template <typename V>
245         struct val {
246         private:
247             using tmp = typename Ring::template mod_t<V, X>;
248         public:
249             using type = std::conditional_t<
250                 Ring::template pos_v<tmp>,
251                 tmp,
252                 typename Ring::template minus_t<tmp>
253             >;
254         };
255 
256         using zero = val<typename Ring::zero>;
257         using one = val<typename Ring::one>;
258 
259         template<typename v1, typename v2>
260         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
261         template<typename v1, typename v2>
262         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
263         template<typename v1, typename v2>
264         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
265         template<typename v1, typename v2>
266         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
267 
268         template<typename v1, typename v2>
269         static constexpr bool eq_v = Ring::template eq_v<typename v1::type, typename v2::type>;
270 
271         template<typename v>
272         static constexpr bool pos_v = true;
273 
274         static constexpr bool is_euclidean_domain = true;
275 
276         template<auto x>
277         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
278 
279         template<typename v>
280         using inject_ring_t = val<v>;
281     };
282 }
283 
284 // type_list
285 namespace aerobus
286 {
288     template <typename...  Ts>
```

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

```
376
377         template <uint64_t index>
378         struct split
379         {
380         private:
381             using inner = internal::split_h<index, type_list<>, type_list<Ts...>>;
382
383         public:
384             using head = typename inner::head;
385             using tail = typename inner::tail;
386         };
387
388         template <uint64_t index, typename T>
389         using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
390
391         template <uint64_t index>
392         using remove = typename internal::remove_h<index, type_list<Ts...>>::type;
393     };
394
395     template <>
396     struct type_list<>
397     {
398         static constexpr size_t length = 0;
399
400         template <typename T>
401         using push_front = type_list<T>;
402
403         template <typename T>
404         using push_back = type_list<T>;
405
406         template <typename U>
407         using concat = U;
408
409         // TODO: assert index == 0
410         template <uint64_t index, typename T>
411         using insert = type_list<T>;
412     };
413 }
414
415 // i32
416 namespace aerobus {
418     struct i32 {
419         using inner_type = int32_t;
422         template<int32_t x>
423         struct val {
424             static constexpr int32_t v = x;
425
428             template<typename valueType>
429             static constexpr valueType get() { return static_cast<valueType>(x); }
430
432             static constexpr bool is_zero_v = x == 0;
433
435             static std::string to_string() {
436                 return std::to_string(x);
437             }
438
441             template<typename valueRing>
442             static constexpr valueRing eval(const valueRing& v) {
443                 return static_cast<valueRing>(x);
444             }
445         };
446
448         using zero = val<0>;
450         using one = val<1>;
452         static constexpr bool is_field = false;
454         static constexpr bool is_euclidean_domain = true;
458         template<auto x>
459         using inject_constant_t = val<static_cast<int32_t>(x)>;
460
461         template<typename v>
462         using inject_ring_t = v;
463
464     private:
465         template<typename v1, typename v2>
466         struct add {
467             using type = val<v1::v + v2::v>;
468         };
469
470         template<typename v1, typename v2>
471         struct sub {
472             using type = val<v1::v - v2::v>;
473         };
474
475         template<typename v1, typename v2>
476         struct mul {
477             using type = val<v1::v* v2::v>;
478         };
```

```
479
480         template<typename v1, typename v2>
481         struct div {
482             using type = val<v1::v / v2::v>;
483         };
484
485         template<typename v1, typename v2>
486         struct remainder {
487             using type = val<v1::v % v2::v>;
488         };
489
490         template<typename v1, typename v2>
491         struct gt {
492             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
493         };
494
495         template<typename v1, typename v2>
496         struct lt {
497             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
498         };
499
500         template<typename v1, typename v2>
501         struct eq {
502             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
503         };
504
505     public:
507         template<typename v1, typename v2>
508         using add_t = typename add<v1, v2>::type;
509
511         template<typename v1>
512         using minus_t = val<-v1::v>;
513
515         template<typename v1, typename v2>
516         using sub_t = typename sub<v1, v2>::type;
517
519         template<typename v1, typename v2>
520         using mul_t = typename mul<v1, v2>::type;
521
523         template<typename v1, typename v2>
524         using div_t = typename div<v1, v2>::type;
525
527         template<typename v1, typename v2>
528         using mod_t = typename remainder<v1, v2>::type;
529
531         template<typename v1, typename v2>
532         static constexpr bool gt_v = gt<v1, v2>::type::value;
533
535         template<typename v1, typename v2>
536         using lt_t = typename lt<v1, v2>::type;
537
539         template<typename v1, typename v2>
540         static constexpr bool eq_v = eq<v1, v2>::type::value;
541
543         template<typename v1>
544         static constexpr bool pos_v = (v1::v > 0);
545
547         template<typename v1, typename v2>
548         using gcd_t = gcd_t<i32, v1, v2>;
549     };
550 }
551
552 // i64
553 namespace aerobus {
555     struct i64 {
556         using inner_type = int64_t;
559         template<int64_t x>
560         struct val {
561             static constexpr int64_t v = x;
562
565             template<typename valueType>
566             static constexpr valueType get() { return static_cast<valueType>(x); }
567
569             static constexpr bool is_zero_v = x == 0;
570
572             static std::string to_string() {
573                 return std::to_string(x);
574             }
575
578             template<typename valueRing>
579             static constexpr valueRing eval(const valueRing& v) {
580                 return static_cast<valueRing>(x);
581             }
582         };
583
587         template<auto x>
588         using inject_constant_t = val<static_cast<int64_t>(x)>;
```

```
589
590         template<typename v>
591         using inject_ring_t = v;
592
594         using zero = val<0>;
596         using one = val<1>;
598         static constexpr bool is_field = false;
600         static constexpr bool is_euclidean_domain = true;
601
602     private:
603         template<typename v1, typename v2>
604         struct add {
605             using type = val<v1::v + v2::v>;
606         };
607
608         template<typename v1, typename v2>
609         struct sub {
610             using type = val<v1::v - v2::v>;
611         };
612
613         template<typename v1, typename v2>
614         struct mul {
615             using type = val<v1::v* v2::v>;
616         };
617
618         template<typename v1, typename v2>
619         struct div {
620             using type = val<v1::v / v2::v>;
621         };
622
623         template<typename v1, typename v2>
624         struct remainder {
625             using type = val<v1::v% v2::v>;
626         };
627
628         template<typename v1, typename v2>
629         struct gt {
630             using type = std::conditional_t<(v1::v > v2::v), std::true_type, std::false_type>;
631         };
632
633         template<typename v1, typename v2>
634         struct lt {
635             using type = std::conditional_t<(v1::v < v2::v), std::true_type, std::false_type>;
636         };
637
638         template<typename v1, typename v2>
639         struct eq {
640             using type = std::conditional_t<(v1::v == v2::v), std::true_type, std::false_type>;
641         };
642
643     public:
645         template<typename v1, typename v2>
646         using add_t = typename add<v1, v2>::type;
647
649         template<typename v1>
650         using minus_t = val<-v1::v>;
651
653         template<typename v1, typename v2>
654         using sub_t = typename sub<v1, v2>::type;
655
657         template<typename v1, typename v2>
658         using mul_t = typename mul<v1, v2>::type;
659
661         template<typename v1, typename v2>
662         using div_t = typename div<v1, v2>::type;
663
665         template<typename v1, typename v2>
666         using mod_t = typename remainder<v1, v2>::type;
667
669         template<typename v1, typename v2>
670         static constexpr bool gt_v = gt<v1, v2>::type::value;
671
673         template<typename v1, typename v2>
674         using lt_t = typename lt<v1, v2>::type;
675
677         template<typename v1, typename v2>
678         static constexpr bool eq_v = eq<v1, v2>::type::value;
679
682         template<typename v1>
683         static constexpr bool pos_v = (v1::v > 0);
684
686         template<typename v1, typename v2>
687         using gcd_t = gcd_t<i64, v1, v2>;
688     };
689 }
690
691 // z/pz
```

```
692 namespace aerobus {
697     template<int32_t p>
698     struct zpz {
699         using inner_type = int32_t;
700         template<int32_t x>
701         struct val {
702             static constexpr int32_t v = x % p;
703 
704             template<typename valueType>
705             static constexpr valueType get() { return static_cast<valueType>(x % p); }
706 
707             static constexpr bool is_zero_v = v == 0;
708             static std::string to_string() {
709                 return std::to_string(x % p);
710             }
711 
712             template<typename valueRing>
713             static constexpr valueRing eval(const valueRing& v) {
714                 return static_cast<valueRing>(x % p);
715             }
716         };
717 
718         template<auto x>
719         using inject_constant_t = val<static_cast<int32_t>(x)>;
720 
721         using zero = val<0>;
722         using one = val<1>;
723         static constexpr bool is_field = is_prime<p>::value;
724         static constexpr bool is_euclidean_domain = true;
725 
726     private:
727         template<typename v1, typename v2>
728         struct add {
729             using type = val<(v1::v + v2::v) % p>;
730         };
731 
732         template<typename v1, typename v2>
733         struct sub {
734             using type = val<(v1::v - v2::v) % p>;
735         };
736 
737         template<typename v1, typename v2>
738         struct mul {
739             using type = val<(v1::v* v2::v) % p>;
740         };
741 
742         template<typename v1, typename v2>
743         struct div {
744             using type = val<(v1::v% p) / (v2::v % p)>;
745         };
746 
747         template<typename v1, typename v2>
748         struct remainder {
749             using type = val<(v1::v% v2::v) % p>;
750         };
751 
752         template<typename v1, typename v2>
753         struct gt {
754             using type = std::conditional_t<(v1::v% p > v2::v% p), std::true_type, std::false_type>;
755         };
756 
757         template<typename v1, typename v2>
758         struct lt {
759             using type = std::conditional_t<(v1::v% p < v2::v% p), std::true_type, std::false_type>;
760         };
761 
762         template<typename v1, typename v2>
763         struct eq {
764             using type = std::conditional_t<(v1::v% p == v2::v % p), std::true_type, std::false_type>;
765         };
766 
767         template<typename v1>
768         struct pos {
769             using type = std::bool_constant<(v1::v > 0)>;
770         };
771 
772     public:
774         template<typename v1>
775         using minus_t = val<-v1::v>;
776 
777         template<typename v1, typename v2>
778         using add_t = typename add<v1, v2>::type;
779 
780         template<typename v1, typename v2>
781         using sub_t = typename sub<v1, v2>::type;
782 
783         template<typename v1, typename v2>
```

```
784         using mul_t = typename mul<v1, v2>::type;
785
786         template<typename v1, typename v2>
787         using div_t = typename div<v1, v2>::type;
788
789         template<typename v1, typename v2>
790         using mod_t = typename remainder<v1, v2>::type;
791
792         template<typename v1, typename v2>
793         static constexpr bool gt_v = gt<v1, v2>::type::value;
794
795         template<typename v1, typename v2>
796         using lt_t = typename lt<v1, v2>::type;
797
798         template<typename v1, typename v2>
799         static constexpr bool eq_v = eq<v1, v2>::type::value;
800
801         template<typename v1, typename v2>
802         using gcd_t = gcd_t<i32, v1, v2>;
803
804         template<typename v>
805         static constexpr bool pos_v = pos<v>::type::value;
806     };
807 }
808
809 // polynomial
810 namespace aerobus {
811     // coeffN x^N + ...
816     template<typename Ring, char variable_name = 'x'>
817     requires IsEuclideanDomain<Ring>
818     struct polynomial {
819         static constexpr bool is_field = false;
820         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
821
822         template<typename coeffN, typename...  coeffs>
823         struct val {
825             static constexpr size_t degree = sizeof...(coeffs);
827             using aN = coeffN;
829             using strip = val<coeffs...>;
831             static constexpr bool is_zero_v = degree == 0 && aN::is_zero_v;
832
833             private:
834             template<size_t index, typename E = void>
835             struct coeff_at {};
836
837             template<size_t index>
838             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))>> {
839                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
840             };
841
842             template<size_t index>
843             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))>> {
844                 using type = typename Ring::zero;
845             };
846
847             public:
850             template<size_t index>
851             using coeff_at_t = typename coeff_at<index>::type;
852
855             static std::string to_string() {
856                 return string_helper<coeffN, coeffs...>::func();
857             }
858
863             template<typename valueRing>
864             static constexpr valueRing eval(const valueRing& x) {
865                 return eval_helper<valueRing, val>::template inner<0, degree +
    1>::func(static_cast<valueRing>(0), x);
866             }
867         };
868
869         // specialization for constants
870         template<typename coeffN>
871         struct val<coeffN> {
872             static constexpr size_t degree = 0;
873             using aN = coeffN;
874             using strip = val<coeffN>;
875             static constexpr bool is_zero_v = coeffN::is_zero_v;
876
877             template<size_t index, typename E = void>
878             struct coeff_at {};
879
880             template<size_t index>
881             struct coeff_at<index, std::enable_if_t<(index == 0)>> {
882                 using type = aN;
883             };
884
885             template<size_t index>
```

```
886             struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
887                 using type = typename Ring::zero;
888             };
889 
890             template<size_t index>
891             using coeff_at_t = typename coeff_at<index>::type;
892 
893             static std::string to_string() {
894                 return string_helper<coeffN>::func();
895             }
896 
897             template<typename valueRing>
898             static constexpr valueRing eval(const valueRing& x) {
899                 return static_cast<valueRing>(aN::template get<valueRing>());
900             }
901         };
902 
904         using zero = val<typename Ring::zero>;
906         using one = val<typename Ring::one>;
908         using X = val<typename Ring::one, typename Ring::zero>;
909 
910     private:
911         template<typename P, typename E = void>
912         struct simplify;
913 
914         template <typename P1, typename P2, typename I>
915         struct add_low;
916 
917         template<typename P1, typename P2>
918         struct add {
919             using type = typename simplify<typename add_low<
920             P1,
921             P2,
922             internal::make_index_sequence_reverse<
923             std::max(P1::degree, P2::degree) + 1
924             »::type>::type;
925         };
926 
927         template <typename P1, typename P2, typename I>
928         struct sub_low;
929 
930         template <typename P1, typename P2, typename I>
931         struct mul_low;
932 
933         template<typename v1, typename v2>
934         struct mul {
935                 using type = typename mul_low<
936                     v1,
937                     v2,
938                     internal::make_index_sequence_reverse<
939                     v1::degree + v2::degree + 1
940                     »::type;
941         };
942 
943         template<typename coeff, size_t deg>
944         struct monomial;
945 
946         template<typename v, typename E = void>
947         struct derive_helper {};
948 
949         template<typename v>
950         struct derive_helper<v, std::enable_if_t<v::degree == 0» {
951             using type = zero;
952         };
953 
954         template<typename v>
955         struct derive_helper<v, std::enable_if_t<v::degree != 0» {
956             using type = typename add<
957                 typename derive_helper<typename simplify<typename v::strip>::type>::type,
958                 typename monomial<
959                     typename Ring::template mul_t<
960                         typename v::aN,
961                         typename Ring::template inject_constant_t<(v::degree)>
962                     >,
963                     v::degree - 1
964                 >::type
965             >::type;
966         };
967 
968         template<typename v1, typename v2, typename E = void>
969         struct eq_helper {};
970 
971         template<typename v1, typename v2>
972         struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
973             static constexpr bool value = false;
974         };
975 
```

```
976
977         template<typename v1, typename v2>
978         struct eq_helper<v1, v2, std::enable_if_t<
979             v1::degree == v2::degree &&
980             (v1::degree != 0 || v2::degree != 0) &&
981             (!Ring::template eq_v<typename v1::aN, typename v2::aN>)
982         » {
983             static constexpr bool value = false;
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             (Ring::template eq_v<typename v1::aN, typename v2::aN>)
991         » {
992             static constexpr bool value = eq_helper<typename v1::strip, typename v2::strip>::value;
993         };
994
995         template<typename v1, typename v2>
996         struct eq_helper<v1, v2, std::enable_if_t<
997             v1::degree == v2::degree &&
998             (v1::degree == 0)
999         » {
1000             static constexpr bool value = Ring::template eq_v<typename v1::aN, typename v2::aN>;
1001         };
1002
1003         template<typename v1, typename v2, typename E = void>
1004         struct lt_helper {};
1005
1006         template<typename v1, typename v2>
1007         struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1008             using type = std::true_type;
1009         };
1010
1011         template<typename v1, typename v2>
1012         struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1013             using type = typename Ring::template lt_t<typename v1::aN, typename v2::aN>;
1014         };
1015
1016         template<typename v1, typename v2>
1017         struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1018             using type = std::false_type;
1019         };
1020
1021         template<typename v1, typename v2, typename E = void>
1022         struct gt_helper {};
1023
1024         template<typename v1, typename v2>
1025         struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1026             using type = std::true_type;
1027         };
1028
1029         template<typename v1, typename v2>
1030         struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1031             using type = std::false_type;
1032         };
1033
1034         template<typename v1, typename v2>
1035         struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1036             using type = std::false_type;
1037         };
1038
1039         // when high power is zero :  strip
1040         template<typename P>
1041         struct simplify<P, std::enable_if_t<
1042             std::is_same<
1043             typename Ring::zero,
1044             typename P::aN
1045             >::value && (P::degree > 0)
1046         »
1047         {
1048             using type = typename simplify<typename P::strip>::type;
1049         };
1050
1051         // otherwise :  do nothing
1052         template<typename P>
1053         struct simplify<P, std::enable_if_t<
1054             !std::is_same<
1055             typename Ring::zero,
1056             typename P::aN
1057             >::value && (P::degree > 0)
1058         »
1059         {
1060             using type = P;
1061         };
1062
```

```
1063         // do not simplify constants
1064         template<typename P>
1065         struct simplify<P, std::enable_if_t<P::degree == 0>> {
1066             using type = P;
1067         };
1068 
1069         // addition at
1070         template<typename P1, typename P2, size_t index>
1071         struct add_at {
1072             using type =
1073                 typename Ring::template add_t<typename P1::template coeff_at_t<index>, typename
      P2::template coeff_at_t<index>>;
1074         };
1075 
1076         template<typename P1, typename P2, size_t index>
1077         using add_at_t = typename add_at<P1, P2, index>::type;
1078 
1079         template<typename P1, typename P2, std::size_t...  I>
1080         struct add_low<P1, P2, std::index_sequence<I...>> {
1081             using type = val<add_at_t<P1, P2, I>...>;
1082         };
1083 
1084         // substraction at
1085         template<typename P1, typename P2, size_t index>
1086         struct sub_at {
1087             using type =
1088                 typename Ring::template sub_t<typename P1::template coeff_at_t<index>, typename
      P2::template coeff_at_t<index>>;
1089         };
1090 
1091         template<typename P1, typename P2, size_t index>
1092         using sub_at_t = typename sub_at<P1, P2, index>::type;
1093 
1094         template<typename P1, typename P2, std::size_t...  I>
1095         struct sub_low<P1, P2, std::index_sequence<I...>> {
1096             using type = val<sub_at_t<P1, P2, I>...>;
1097         };
1098 
1099         template<typename P1, typename P2>
1100         struct sub {
1101             using type = typename simplify<typename sub_low<
1102             P1,
1103             P2,
1104             internal::make_index_sequence_reverse<
1105             std::max(P1::degree, P2::degree) + 1
1106             >::type>::type;
1107         };
1108 
1109         // multiplication at
1110         template<typename v1, typename v2, size_t k, size_t index, size_t stop>
1111         struct mul_at_loop_helper {
1112             using type = typename Ring::template add_t<
1113                 typename Ring::template mul_t<
1114                 typename v1::template coeff_at_t<index>,
1115                 typename v2::template coeff_at_t<k - index>
1116                 >,
1117                 typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
1118             >;
1119         };
1120 
1121         template<typename v1, typename v2, size_t k, size_t stop>
1122         struct mul_at_loop_helper<v1, v2, k, stop, stop> {
1123             using type = typename Ring::template mul_t<typename v1::template coeff_at_t<stop>, typename
      v2::template coeff_at_t<0>>;
1124         };
1125 
1126         template <typename v1, typename v2, size_t k, typename E = void>
1127         struct mul_at {};
1128 
1129         template<typename v1, typename v2, size_t k>
1130         struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)>> {
1131             using type = typename Ring::zero;
1132         };
1133 
1134         template<typename v1, typename v2, size_t k>
1135         struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)>> {
1136             using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
1137         };
1138 
1139         template<typename P1, typename P2, size_t index>
1140         using mul_at_t = typename mul_at<P1, P2, index>::type;
1141 
1142         template<typename P1, typename P2, std::size_t...  I>
1143         struct mul_low<P1, P2, std::index_sequence<I...>> {
1144             using type = val<mul_at_t<P1, P2, I>...>;
1145         };
1146 
```

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

```
1231                     else {
1232                         result += std::string(1, variable_name) + "^" +
     std::to_string(sizeof...(coeffs));
1233                     }
1234                 }
1235                 else {
1236                     if (sizeof...(coeffs) == 1) {
1237                         result += coeff::to_string() + " " + std::string(1, variable_name);
1238                     }
1239                     else {
1240                         result += coeff::to_string() + " " + std::string(1, variable_name) + "^" +
     std::to_string(sizeof...(coeffs));
1241                     }
1242                 }
1243 
1244                 if(!tail.empty()) {
1245                     result += " + " + tail;
1246                 }
1247 
1248                 return result;
1249             }
1250         };
1251 
1252         template<typename coeff>
1253         struct string_helper<coeff> {
1254             static std::string func() {
1255                 if(!std::is_same<coeff, typename Ring::zero>::value) {
1256                     return coeff::to_string();
1257                 } else {
1258                     return "";
1259                 }
1260             }
1261         };
1262 
1263     public:
1266         template<typename P>
1267         using simplify_t = typename simplify<P>::type;
1268 
1272         template<typename v1, typename v2>
1273         using add_t = typename add<v1, v2>::type;
1274 
1278         template<typename v1, typename v2>
1279         using sub_t = typename sub<v1, v2>::type;
1280 
1281         template<typename v1>
1282         using minus_t = sub_t<zero, v1>;
1283 
1287         template<typename v1, typename v2>
1288         using mul_t = typename mul<v1, v2>::type;
1289 
1293         template<typename v1, typename v2>
1294         static constexpr bool eq_v = eq_helper<v1, v2>::value;
1295 
1299         template<typename v1, typename v2>
1300         using lt_t = typename lt_helper<v1, v2>::type;
1301 
1305         template<typename v1, typename v2>
1306         static constexpr bool gt_v = gt_helper<v1, v2>::type::value;
1307 
1311         template<typename v1, typename v2>
1312         using div_t = typename div<v1, v2>::q_type;
1313 
1317         template<typename v1, typename v2>
1318         using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
1319 
1323         template<typename coeff, size_t deg>
1324         using monomial_t = typename monomial<coeff, deg>::type;
1325 
1328         template<typename v>
1329         using derive_t = typename derive_helper<v>::type;
1330 
1333         template<typename v>
1334         static constexpr bool pos_v = Ring::template pos_v<typename v::aN>;
1335 
1339         template<typename v1, typename v2>
1340         using gcd_t = std::conditional_t<
1341             Ring::is_euclidean_domain,
1342             typename make_unit<gcd_t<polynomial<Ring, variable_name>, v1, v2»::type,
1343             void>;
1344 
1348         template<auto x>
1349         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
1350 
1354         template<typename v>
1355         using inject_ring_t = val<v>;
1356     };
1357 }
```

```
1358
1359 // big integers
1360 namespace aerobus {
1361     struct bigint {
1362         enum signs {
1363             positive,
1364             negative
1365         };
1366
1367         static constexpr signs opposite(const signs& s) {
1368             return s == signs::positive ?  signs::negative :  signs::positive;
1369         }
1370
1371         template<signs s, uint32_t an, uint32_t...  as>
1372         struct val {
1373             static constexpr signs sign = s;
1374
1375             template<size_t index, typename E = void>
1376             struct digit_at {};
1377
1378             template<size_t index>
1379             struct digit_at<index, std::enable_if_t<(index <= sizeof...(as))» {
1380                 static constexpr uint32_t value = internal::value_at<(sizeof...(as) - index), an,
    as...>::value;
1381             };
1382
1383             template<size_t index>
1384             struct digit_at<index, std::enable_if_t<(index > sizeof...(as))» {
1385                 static constexpr uint32_t value = 0;
1386             };
1387
1388             using strip = val<s, as...>;
1389             static constexpr uint32_t aN = an;
1390             static constexpr size_t digits = sizeof...(as) + 1;
1391
1392             static std::string to_string() {
1393                 return std::to_string(aN) + "B^" + std::to_string(digits-1) + " + " +
    strip::to_string();
1394             }
1395
1396             static constexpr bool is_zero_v = sizeof...(as) == 0 && an == 0;
1397
1398             using minus_t = val<opposite(s), an, as...>;
1399         };
1400
1401         template<signs s, uint32_t a0>
1402         struct val<s, a0> {
1403             static constexpr signs sign = s;
1404             using strip = val<s, a0>;
1405             static constexpr uint32_t aN = a0;
1406             static constexpr size_t digits = 1;
1407             template<size_t index, typename E = void>
1408             struct digit_at {};
1409             template<size_t index>
1410             struct digit_at<index, std::enable_if_t<index == 0» {
1411                 static constexpr uint32_t value = a0;
1412             };
1413
1414             template<size_t index>
1415             struct digit_at<index, std::enable_if_t<index != 0» {
1416                 static constexpr uint32_t value = 0;
1417             };
1418
1419             static std::string to_string() {
1420                 return std::to_string(a0);
1421             }
1422
1423             static constexpr bool is_zero_v = a0 == 0;
1424
1425             using minus_t = val<opposite(s), a0>;
1426         };
1427
1428         using zero = val<signs::positive, 0>;
1429         using one = val<signs::positive, 1>;
1430
1431     private:
1432         template<typename I, typename E = void>
1433         struct simplify {};
1434
1435         template<typename I>
1436         struct simplify<I, std::enable_if_t<I::digits == 1 && I::aN != 0» {
1437             using type = I;
1438         };
1439
1440         template<typename I>
1441         struct simplify<I, std::enable_if_t<I::digits == 1 && I::aN == 0» {
1442             using type = zero;
```

```
1443         };
1444
1445         template<typename I>
1446         struct simplify<I, std::enable_if_t<I::digits != 1 && I::aN == 0» {
1447             using type = typename simplify<typename I::strip>::type;
1448         };
1449
1450         template<typename I>
1451         struct simplify<I, std::enable_if_t<I::digits != 1 && I::aN != 0» {
1452             using type = I;
1453         };
1454
1455         template<uint32_t x, uint32_t y, uint8_t carry_in = 0>
1456         struct add_digit_helper {
1457         private:
1458             static constexpr uint64_t raw = ((uint64_t) x + (uint64_t) y + (uint64_t) carry_in);
1459         public:
1460             static constexpr uint32_t value = (uint32_t)(raw & 0xFFFF'FFFF);
1461             static constexpr uint8_t carry_out = (uint32_t) (raw » 32);
1462         };
1463
1464         template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1465         struct add_at_helper {
1466         private:
1467             static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1468             static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1469         public:
1470             static constexpr uint32_t value = add_digit_helper<d1, d2, carry_in>::value;
1471             static constexpr uint8_t carry_out = add_digit_helper<d1, d2, carry_in>::carry_out;
1472         };
1473
1474         template<uint32_t x, uint32_t y, uint8_t carry_in, typename E = void>
1475         struct sub_digit_helper {};
1476
1477         // x - y
1478         template<uint32_t x, uint32_t y, uint8_t carry_in>
1479         struct sub_digit_helper<x, y, carry_in, std::enable_if_t<
1480                 (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in) > x)
1481         » {
1482
1483             static constexpr uint32_t value = static_cast<uint32_t>(
1484                 static_cast<uint32_t>(x) + 0x1'0000'0000UL - (static_cast<uint64_t>(y) +
     static_cast<uint64_t>(carry_in))
1485             );
1486             static constexpr uint8_t carry_out = 1;
1487         };
1488
1489         template<uint32_t x, uint32_t y, uint8_t carry_in>
1490         struct sub_digit_helper<x, y, carry_in, std::enable_if_t<
1491                 (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in) <= x)
1492         » {
1493
1494             static constexpr uint32_t value = static_cast<uint32_t>(
1495                 static_cast<uint64_t>(x) - (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in))
1496             );
1497             static constexpr uint8_t carry_out = 0;
1498         };
1499
1500         template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1501         struct sub_at_helper {
1502         private:
1503             static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1504             static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1505             using tmp = sub_digit_helper<d1, d2, carry_in>;
1506         public:
1507             static constexpr uint32_t value = tmp::value;
1508             static constexpr uint8_t carry_out = tmp::carry_out;
1509         };
1510
1511         template<typename I1, typename I2, size_t index>
1512         struct add_low_helper {
1513             private:
1514             using helper = add_at_helper<I1, I2, index, add_low_helper<I1, I2, index-1>::carry_out>;
1515             public:
1516             static constexpr uint32_t digit = helper::value;
1517             static constexpr uint8_t carry_out = helper::carry_out;
1518         };
1519
1520         template<typename I1, typename I2>
1521         struct add_low_helper<I1, I2, 0> {
1522             static constexpr uint32_t digit = add_at_helper<I1, I2, 0, 0>::value;
1523             static constexpr uint32_t carry_out = add_at_helper<I1, I2, 0, 0>::carry_out;
1524         };
1525
1526         template<typename I1, typename I2, size_t index>
1527         struct sub_low_helper {
1528             private:
```

```
1529            using helper = sub_at_helper<I1, I2, index, sub_low_helper<I1, I2, index-1>::carry_out>;
1530            public:
1531            static constexpr uint32_t digit = helper::value;
1532            static constexpr uint8_t carry_out = helper::carry_out;
1533        };
1534
1535        template<typename I1, typename I2>
1536        struct sub_low_helper<I1, I2, 0> {
1537            static constexpr uint32_t digit = sub_at_helper<I1, I2, 0, 0>::value;
1538            static constexpr uint32_t carry_out = sub_at_helper<I1, I2, 0, 0>::carry_out;
1539        };
1540
1541        template<typename I1, typename I2, typename I>
1542        struct add_low {};
1543
1544        template<typename I1, typename I2, std::size_t...  I>
1545        struct add_low<I1, I2, std::index_sequence<I...» {
1546            using type = val<signs::positive, add_low_helper<I1, I2, I>::digit...>;
1547        };
1548
1549        template<typename I1, typename I2, typename I>
1550        struct sub_low {};
1551
1552        template<typename I1, typename I2, std::size_t...  I>
1553        struct sub_low<I1, I2, std::index_sequence<I...» {
1554            using type = val<signs::positive, sub_low_helper<I1, I2, I>::digit...>;
1555        };
1556
1557        template<typename I1, typename I2, typename E = void>
1558        struct eq {};
1559
1560        template<typename I1, typename I2>
1561        struct eq<I1, I2, std::enable_if_t<I1::digits != I2::digits» {
1562            static constexpr bool value = false;
1563        };
1564
1565        template<typename I1, typename I2>
1566        struct eq<I1, I2, std::enable_if_t<I1::digits == I2::digits && I1::digits == 1» {
1567            static constexpr bool value = (I1::is_zero_v && I2::is_zero_v) || (I1::sign == I2::sign &&
    I1::aN == I2::aN);
1568        };
1569
1570        template<typename I1, typename I2>
1571        struct eq<I1, I2, std::enable_if_t<I1::digits == I2::digits && I1::digits != 1» {
1572            static constexpr bool value =
1573                I1::sign == I2::sign &&
1574                I1::aN == I2::aN &&
1575                eq<typename I1::strip, typename I2::strip>::value;
1576        };
1577
1578        template<typename I1, typename I2, typename E = void>
1579        struct gt_helper {};
1580
1581        template<typename I1, typename I2>
1582        struct gt_helper<I1, I2, std::enable_if_t<eq<I1, I2>::value» {
1583            static constexpr bool value = false;
1584        };
1585
1586        template<typename I1, typename I2>
1587        struct gt_helper<I1, I2, std::enable_if_t<!eq<I1, I2>::value && I1::sign != I2::sign» {
1588            static constexpr bool value = I1::sign == signs::positive;
1589        };
1590
1591        template<typename I1, typename I2>
1592        struct gt_helper<I1, I2,
1593            std::enable_if_t<
1594                !eq<I1, I2>::value &&
1595                I1::sign == I2::sign &&
1596                I1::sign == signs::negative
1597            » {
1598            static constexpr bool value = gt_helper<typename I2::minus_t, typename I1::minus_t>::value;
1599        };
1600
1601        template<typename I1, typename I2>
1602        struct gt_helper<I1, I2,
1603            std::enable_if_t<
1604                !eq<I1, I2>::value &&
1605                I1::sign == I2::sign &&
1606                I1::sign == signs::positive &&
1607                (I1::digits > I2::digits)
1608            » {
1609            static constexpr bool value = true;
1610        };
1611
1612        template<typename I1, typename I2>
1613        struct gt_helper<I1, I2,
1614            std::enable_if_t<
```

```
1615                !eq<I1, I2>::value &&
1616                I1::sign == I2::sign &&
1617                I1::sign == signs::positive &&
1618                (I1::digits < I2::digits)
1619            » {
1620            static constexpr bool value = false;
1621        };
1622
1623        template<typename I1, typename I2>
1624        struct gt_helper<I1, I2,
1625            std::enable_if_t<
1626                !eq<I1, I2>::value &&
1627                I1::sign == I2::sign &&
1628                I1::sign == signs::positive &&
1629                (I1::digits == I2::digits) && I1::digits == 1
1630            » {
1631            static constexpr bool value = I1::aN > I2::aN;
1632        };
1633
1634        template<typename I1, typename I2>
1635        struct gt_helper<I1, I2,
1636            std::enable_if_t<
1637                !eq<I1, I2>::value &&
1638                I1::sign == I2::sign &&
1639                I1::sign == signs::positive &&
1640                (I1::digits == I2::digits) && I1::digits != 1 && (I1::aN > I2::aN)
1641            » {
1642            static constexpr bool value = true;
1643        };
1644
1645        template<typename I1, typename I2>
1646        struct gt_helper<I1, I2,
1647            std::enable_if_t<
1648                !eq<I1, I2>::value &&
1649                I1::sign == I2::sign &&
1650                I1::sign == signs::positive &&
1651                (I1::digits == I2::digits) && I1::digits != 1 && (I1::aN < I2::aN)
1652            » {
1653            static constexpr bool value = false;
1654        };
1655
1656        template<typename I1, typename I2>
1657        struct gt_helper<I1, I2,
1658            std::enable_if_t<
1659                !eq<I1, I2>::value &&
1660                I1::sign == I2::sign &&
1661                I1::sign == signs::positive &&
1662                (I1::digits == I2::digits) && I1::digits != 1 && I1::aN == I2::aN
1663            » {
1664            static constexpr bool value = gt_helper<typename I1::strip, typename I2::strip>::value;
1665        };
1666
1667
1668
1669        template<typename I1, typename I2, typename E = void>
1670        struct add {};
1671
1672        template<typename I1, typename I2, typename E = void>
1673        struct sub {};
1674
1675        // +x + +y -> x + y
1676        template<typename I1, typename I2>
1677        struct add<I1, I2, std::enable_if_t<
1678            gt_helper<I1, zero>::value &&
1679            gt_helper<I2, zero>::value
1680        » {
1681            using type = typename simplify<
1682                typename add_low<
1683                        I1,
1684                        I2,
1685                        typename internal::make_index_sequence_reverse<std::max(I1::digits, I2::digits)
     + 1>
1686                    >::type>::type;
1687        };
1688
1689        // -x + -y -> -(x+y)
1690        template<typename I1, typename I2>
1691        struct add<I1, I2, std::enable_if_t<
1692            gt_helper<zero, I1>::value &&
1693            gt_helper<zero, I2>::value
1694        » {
1695            using type = typename add<typename I1::minus_t, typename I2::minus_t>::type::minus_t;
1696        };
1697
1698        // 0 + x -> x
1699        template<typename I1, typename I2>
1700        struct add<I1, I2, std::enable_if_t<
```

```
1701            I1::is_zero_v
1702        » {
1703            using type = I2;
1704        };
1705
1706        // x + 0 -> x
1707        template<typename I1, typename I2>
1708        struct add<I1, I2, std::enable_if_t<
1709            I2::is_zero_v
1710        » {
1711            using type = I1;
1712        };
1713
1714        // x + (-y) -> x - y
1715        template<typename I1, typename I2>
1716        struct add<I1, I2, std::enable_if_t<
1717            !I1::is_zero_v && !I2::is_zero_v &&
1718            gt_helper<I1, zero>::value &&
1719            gt_helper<zero, I2>::value
1720        » {
1721            using type = typename sub<I1, typename I2::minus_t>::type;
1722        };
1723
1724        // -x + y -> y - x
1725        template<typename I1, typename I2>
1726        struct add<I1, I2, std::enable_if_t<
1727            !I1::is_zero_v && !I2::is_zero_v &&
1728            gt_helper<zero, I1>::value &&
1729            gt_helper<I2, zero>::value
1730        » {
1731            using type = typename sub<I2, typename I1::minus_t>::type;
1732        };
1733
1734        // I1 == I2
1735        template<typename I1, typename I2>
1736        struct sub<I1, I2, std::enable_if_t<
1737            eq<I1, I2>::value
1738        » {
1739            using type = zero;
1740        };
1741
1742        // I1 != I2, I2 == 0
1743        template<typename I1, typename I2>
1744        struct sub<I1, I2, std::enable_if_t<
1745            !eq<I1, I2>::value &&
1746            eq<I2, zero>::value
1747        » {
1748            using type = I1;
1749        };
1750
1751        // I1 != I2, I1 == 0
1752        template<typename I1, typename I2>
1753        struct sub<I1, I2, std::enable_if_t<
1754            !eq<I1, I2>::value &&
1755            eq<I1, zero>::value
1756        » {
1757            using type = typename I2::minus_t;
1758        };
1759
1760        // 0 < I2 < I1
1761        template<typename I1, typename I2>
1762        struct sub<I1, I2, std::enable_if_t<
1763            gt_helper<I2, zero>::value &&
1764            gt_helper<I1, I2>::value
1765        » {
1766            using type = typename simplify<
1767                typename sub_low<
1768                        I1,
1769                        I2,
1770                        typename internal::make_index_sequence_reverse<std::max(I1::digits, I2::digits)
     + 1>
1771                    >::type>::type;
1772        };
1773
1774        // 0 < I1 < I2
1775        template<typename I1, typename I2>
1776        struct sub<I1, I2, std::enable_if_t<
1777            gt_helper<I1, zero>::value &&
1778            gt_helper<I2, I1>::value
1779        » {
1780            using type = typename sub<I2, I1>::type::minus_t;
1781        };
1782
1783        // I2 < I1 < 0
1784        template<typename I1, typename I2>
1785        struct sub<I1, I2, std::enable_if_t<
1786            gt_helper<zero, I1>::value &&
```

```
1787             gt_helper<I1, I2>::value
1788         » {
1789             using type = typename sub<typename I2::minus_t, typename I1::minus_t>::type;
1790         };
1791
1792         // I1 < I2 < 0
1793         template<typename I1, typename I2>
1794         struct sub<I1, I2, std::enable_if_t<
1795             gt_helper<zero, I2>::value &&
1796             gt_helper<I2, I1>::value
1797         » {
1798             using type = typename sub<typename I1::minus_t, typename I2::minus_t>::type::minus_t;
1799         };
1800
1801         // I2 < 0 < I1
1802         template<typename I1, typename I2>
1803         struct sub<I1, I2, std::enable_if_t<
1804             gt_helper<zero, I2>::value &&
1805             gt_helper<I1, zero>::value
1806         » {
1807             using type = typename add<I1, typename I2::minus_t>::type;
1808         };
1809
1810         // I1 < 0 < I2
1811         template<typename I1, typename I2>
1812         struct sub<I1, I2, std::enable_if_t<
1813             gt_helper<zero, I1>::value &&
1814             gt_helper<I2, zero>::value
1815         » {
1816             using type = typename add<I2, typename I1::minus_t>::type::minus_t;
1817         };
1818
1819     public:
1820         template<typename I>
1821         using minus_t = I::minus_t;
1822
1823         template<typename I1, typename I2>
1824         static constexpr bool eq_v = eq<I1, I2>::value;
1825
1826         template<typename I>
1827         static constexpr bool pos_v = I::sign == signs::positive && !I::is_zero_v;
1828
1829         template<typename I1, typename I2>
1830         static constexpr bool gt_v = gt_helper<I1, I2>::value;
1831
1832         template<typename I>
1833         using simplify_t = typename simplify<I>::type;
1834
1835         template<typename I1, typename I2>
1836         using add_t = typename add<I1, I2>::type;
1837
1838         template<typename I1, typename I2>
1839         using sub_t = typename sub<I1, I2>::type;
1840     };
1841 }
1842
1843 // fraction field
1844 namespace aerobus {
1845     namespace internal {
1846         template<typename Ring, typename E = void>
1847         requires IsEuclideanDomain<Ring>
1848         struct _FractionField {};
1849
1850         template<typename Ring>
1851         requires IsEuclideanDomain<Ring>
1852         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain»
1853         {
1855             static constexpr bool is_field = true;
1856             static constexpr bool is_euclidean_domain = true;
1857
1858             private:
1859             template<typename val1, typename val2, typename E = void>
1860             struct to_string_helper {};
1861
1862             template<typename val1, typename val2>
1863             struct to_string_helper <val1, val2,
1864                 std::enable_if_t<
1865                 Ring::template eq_v<val2, typename Ring::one>
1866                 » {
1867                 static std::string func() {
1868                     return val1::to_string();
1869                 }
1870             };
1871
1872             template<typename val1, typename val2>
1873             struct to_string_helper<val1, val2,
1874                 std::enable_if_t<
```

```
1875                !Ring::template eq_v<val2,typename Ring::one>
1876                » {
1877                static std::string func() {
1878                    return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
1879                }
1880            };
1881
1882            public:
1886            template<typename val1, typename val2>
1887            struct val {
1888                using x = val1;
1889                using y = val2;
1890
1892                static constexpr bool is_zero_v = val1::is_zero_v;
1893                using ring_type = Ring;
1894                using field_type = _FractionField<Ring>;
1895
1897                static constexpr bool is_integer = std::is_same<val2, typename Ring::one>::value;
1898
1902                template<typename valueType>
1903                static constexpr valueType get() { return static_cast<valueType>(x::v) /
     static_cast<valueType>(y::v); }
1904
1907                static std::string to_string() {
1908                    return to_string_helper<val1, val2>::func();
1909                }
1910
1915                template<typename valueRing>
1916                static constexpr valueRing eval(const valueRing& v) {
1917                    return x::eval(v) / y::eval(v);
1918                }
1919            };
1920
1922            using zero = val<typename Ring::zero, typename Ring::one>;
1924            using one = val<typename Ring::one, typename Ring::one>;
1925
1928            template<typename v>
1929            using inject_t = val<v, typename Ring::one>;
1930
1933            template<auto x>
1934            using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
     Ring::one>;
1935
1938            template<typename v>
1939            using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
1940
1941            using ring_type = Ring;
1942
1943        private:
1944            template<typename v, typename E = void>
1945            struct simplify {};
1946
1947            // x = 0
1948            template<typename v>
1949            struct simplify<v, std::enable_if_t<v::x::is_zero_v» {
1950                using type = typename _FractionField<Ring>::zero;
1951            };
1952
1953            // x != 0
1954            template<typename v>
1955            struct simplify<v, std::enable_if_t<!v::x::is_zero_v» {
1956
1957            private:
1958                using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
1959                using newx = typename Ring::template div_t<typename v::x, _gcd>;
1960                using newy = typename Ring::template div_t<typename v::y, _gcd>;
1961
1962                using posx = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
     minus_t<newx>, newx>;
1963                using posy = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
     minus_t<newy>, newy>;
1964            public:
1965                using type = typename _FractionField<Ring>::template val<posx, posy>;
1966            };
1967
1968        public:
1971            template<typename v>
1972            using simplify_t = typename simplify<v>::type;
1973
1974        private:
1975
1976            template<typename v1, typename v2>
1977            struct add {
1978            private:
1979                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
1980                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
1981                using dividend = typename Ring::template add_t<a, b>;
```

```
1982                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
1983                using g = typename Ring::template gcd_t<dividend, diviser>;
1984
1985            public:
1986                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
1987            };
1988
1989            template<typename v>
1990            struct pos {
1991                using type = std::conditional_t<
1992                    (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
1993                    (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>),
1994                    std::true_type,
1995                    std::false_type>;
1996
1997            };
1998
1999            template<typename v1, typename v2>
2000            struct sub {
2001            private:
2002                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2003                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2004                using dividend = typename Ring::template sub_t<a, b>;
2005                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2006                using g = typename Ring::template gcd_t<dividend, diviser>;
2007
2008            public:
2009                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
2010            };
2011
2012            template<typename v1, typename v2>
2013            struct mul {
2014            private:
2015                using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
2016                using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2017
2018            public:
2019                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
2020            };
2021
2022            template<typename v1, typename v2, typename E = void>
2023            struct div {};
2024
2025            template<typename v1, typename v2>
2026            struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
    _FractionField<Ring>::zero>::value»  {
2027            private:
2028                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2029                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2030
2031            public:
2032                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
2033            };
2034
2035            template<typename v1, typename v2>
2036            struct div<v1, v2, std::enable_if_t<
2037                std::is_same<zero, v1>::value && std::is_same<v2, zero>::value»  {
2038                using type = one;
2039            };
2040
2041            template<typename v1, typename v2>
2042            struct eq {
2043                using type = std::conditional_t<
2044                        std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value &&
2045                        std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value,
2046                    std::true_type,
2047                    std::false_type>;
2048            };
2049
2050            template<typename TL, typename E = void>
2051            struct vadd {};
2052
2053            template<typename TL>
2054            struct vadd<TL, std::enable_if_t<(TL::length > 1)» {
2055                using head = typename TL::pop_front::type;
2056                using tail = typename TL::pop_front::tail;
2057                using type = typename add<head, typename vadd<tail>::type>::type;
2058            };
2059
2060            template<typename TL>
2061            struct vadd<TL, std::enable_if_t<(TL::length == 1)» {
2062                using type = typename TL::template at<0>;
2063            };
2064
2065            template<typename...  vals>
2066            struct vmul {};
2067
```

```
2068            template<typename v1, typename...  vals>
2069            struct vmul<v1, vals...> {
2070                using type = typename mul<v1, typename vmul<vals...>::type>::type;
2071            };
2072
2073            template<typename v1>
2074            struct vmul<v1> {
2075                using type = v1;
2076            };
2077
2078
2079            template<typename v1, typename v2, typename E = void>
2080            struct gt;
2081
2082            template<typename v1, typename v2>
2083            struct gt<v1, v2, std::enable_if_t<
2084                (eq<v1, v2>::type::value)
2085                » {
2086                using type = std::false_type;
2087            };
2088
2089            template<typename v1, typename v2>
2090            struct gt<v1, v2, std::enable_if_t<
2091                (!eq<v1, v2>::type::value) &&
2092                (!pos<v1>::type::value) && (!pos<v2>::type::value)
2093                » {
2094                using type = typename gt<
2095                    typename sub<zero, v1>::type, typename sub<zero, v2>::type
2096                >::type;
2097            };
2098
2099            template<typename v1, typename v2>
2100            struct gt<v1, v2, std::enable_if_t<
2101                (!eq<v1, v2>::type::value) &&
2102                (pos<v1>::type::value) && (!pos<v2>::type::value)
2103                » {
2104                using type = std::true_type;
2105            };
2106
2107            template<typename v1, typename v2>
2108            struct gt<v1, v2, std::enable_if_t<
2109                (!eq<v1, v2>::type::value) &&
2110                (!pos<v1>::type::value) && (pos<v2>::type::value)
2111                » {
2112                using type = std::false_type;
2113            };
2114
2115            template<typename v1, typename v2>
2116            struct gt<v1, v2, std::enable_if_t<
2117                (!eq<v1, v2>::type::value) &&
2118                (pos<v1>::type::value) && (pos<v2>::type::value)
2119                » {
2120                using type = std::bool_constant<Ring::template gt_v<
2121                    typename Ring::template mul_t<v1::x, v2::y>,
2122                    typename Ring::template mul_t<v2::y, v2::x>
2123                »;
2124            };
2125
2126
2127        public:
2128
2130            template<typename v1, typename v2>
2131            using add_t = typename add<v1, v2>::type;
2132
2134            template<typename v1, typename v2>
2135            using mod_t = zero;
2136
2140            template<typename v1, typename v2>
2141            using gcd_t = v1;
2142
2145            template<typename...  vs>
2146            using vadd_t = typename vadd<vs...>::type;
2147
2150            template<typename...  vs>
2151            using vmul_t = typename vmul<vs...>::type;
2152
2154            template<typename v1, typename v2>
2155            using sub_t = typename sub<v1, v2>::type;
2156
2157            template<typename v>
2158            using minus_t = sub_t<zero, v>;
2159
2161            template<typename v1, typename v2>
2162            using mul_t = typename mul<v1, v2>::type;
2163
2165            template<typename v1, typename v2>
2166            using div_t = typename div<v1, v2>::type;
```

```
2167
2169            template<typename v1, typename v2>
2170            static constexpr bool eq_v = eq<v1, v2>::type::value;
2171
2173            template<typename v1, typename v2>
2174            static constexpr bool gt_v = gt<v1, v2>::type::value;
2175
2177            template<typename v>
2178            static constexpr bool pos_v = pos<v>::type::value;
2179        };
2180
2181        template<typename Ring, typename E = void>
2182        requires IsEuclideanDomain<Ring>
2183        struct FractionFieldImpl {};
2184
2185        // fraction field of a field is the field itself
2186        template<typename Field>
2187        requires IsEuclideanDomain<Field>
2188        struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {
2189            using type = Field;
2190            template<typename v>
2191            using inject_t = v;
2192        };
2193
2194        // fraction field of a ring is the actual fraction field
2195        template<typename Ring>
2196        requires IsEuclideanDomain<Ring>
2197        struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {
2198            using type = _FractionField<Ring>;
2199        };
2200    }
2201
2202    template<typename Ring>
2203    requires IsEuclideanDomain<Ring>
2204    using FractionField = typename internal::FractionFieldImpl<Ring>::type;
2205 }
2206
2207 // short names for common types
2208 namespace aerobus {
2210    using q32 = FractionField<i32>;
2212    using fpq32 = FractionField<polynomial<q32»;
2214    using q64 = FractionField<i64>;
2216    using pi64 = polynomial<i64>;
2218    using fpq64 = FractionField<polynomial<q64»;
2219
2222    template<uint32_t...  digits>
2223    using bigint_pos = bigint::template val<bigint::signs::positive, digits...>;
2226    template<uint32_t...  digits>
2227    using bigint_neg = bigint::template val<bigint::signs::negative, digits...>;
2228
2233    template<typename Ring, typename v1, typename v2>
2234    using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
2235
2236    template<typename Ring, typename v1, typename v2>
2237    using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
2238    template<typename Ring, typename v1, typename v2>
2239    using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
2240 }
2241
2242 // taylor series and common integers (factorial, bernouilli...)  appearing in taylor coefficients
2243 namespace aerobus {
2244    namespace internal {
2245        template<typename T, size_t x, typename E = void>
2246        struct factorial {};
2247
2248        template<typename T, size_t x>
2249        struct factorial<T, x, std::enable_if_t<(x > 0)» {
2250        private:
2251            template<typename, size_t, typename>
2252            friend struct factorial;
2253        public:
2254            using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
     x - 1>::type>;
2255            static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
2256        };
2257
2258        template<typename T>
2259        struct factorial<T, 0> {
2260        public:
2261            using type = typename T::one;
2262            static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
2263        };
2264    }
2265
2269    template<typename T, size_t i>
```

```
2270     using factorial_t = typename internal::factorial<T, i>::type;
2271
2272     template<typename T, size_t i>
2273     inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
2274
2275     namespace internal {
2276         template<typename T, size_t k, size_t n, typename E = void>
2277         struct combination_helper {};
2278
2279         template<typename T, size_t k, size_t n>
2280         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
2281             using type = typename FractionField<T>::template mul_t<
2282                 typename combination_helper<T, k - 1, n - 1>::type,
2283                 makefraction_t<T, typename T::template val<n>, typename T::template val<k»>;
2284         };
2285
2286         template<typename T, size_t k, size_t n>
2287         struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
2288             using type = typename combination_helper<T, n - k, n>::type;
2289         };
2290
2291         template<typename T, size_t n>
2292         struct combination_helper<T, 0, n> {
2293             using type = typename FractionField<T>::one;
2294         };
2295
2296         template<typename T, size_t k, size_t n>
2297         struct combination {
2298             using type = typename internal::combination_helper<T, k, n>::type::x;
2299             static constexpr typename T::inner_type value = internal::combination_helper<T, k,
     n>::type::template get<typename T::inner_type>();
2300         };
2301     }
2302
2305     template<typename T, size_t k, size_t n>
2306     using combination_t = typename internal::combination<T, k, n>::type;
2307
2308     template<typename T, size_t k, size_t n>
2309     inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
2310
2311     namespace internal {
2312         template<typename T, size_t m>
2313         struct bernouilli;
2314
2315         template<typename T, typename accum, size_t k, size_t m>
2316         struct bernouilli_helper {
2317             using type = typename bernouilli_helper<
2318                 T,
2319                 addfractions_t<T,
2320                     accum,
2321                     mulfractions_t<T,
2322                         makefraction_t<T,
2323                             combination_t<T, k, m + 1>,
2324                             typename T::one>,
2325                         typename bernouilli<T, k>::type
2326                     >
2327                 >,
2328                 k + 1,
2329                 m>::type;
2330         };
2331
2332         template<typename T, typename accum, size_t m>
2333         struct bernouilli_helper<T, accum, m, m>
2334         {
2335             using type = accum;
2336         };
2337
2338
2339
2340         template<typename T, size_t m>
2341         struct bernouilli {
2342             using type = typename FractionField<T>::template mul_t<
2343                 typename internal::bernouilli_helper<T, typename FractionField<T>::zero, 0, m>::type,
2344                 makefraction_t<T,
2345                 typename T::template val<static_cast<typename T::inner_type>(-1)>,
2346                 typename T::template val<static_cast<typename T::inner_type>(m + 1)>
2347                 >
2348             >;
2349
2350             template<typename floatType>
2351             static constexpr floatType value = type::template get<floatType>();
2352         };
2353
2354         template<typename T>
2355         struct bernouilli<T, 0> {
2356             using type = typename FractionField<T>::one;
2357
```

```
2358            template<typename floatType>
2359            static constexpr floatType value = type::template get<floatType>();
2360        };
2361    }
2362
2366    template<typename T, size_t n>
2367    using bernouilli_t = typename internal::bernouilli<T, n>::type;
2368
2369    template<typename FloatType, typename T, size_t n >
2370    inline constexpr FloatType bernouilli_v = internal::bernouilli<T, n>::template value<FloatType>;
2371
2372    namespace internal {
2373        template<typename T, int k, typename E = void>
2374        struct alternate {};
2375
2376        template<typename T, int k>
2377        struct alternate<T, k, std::enable_if_t<k % 2 == 0>> {
2378            using type = typename T::one;
2379            static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
2380        };
2381
2382        template<typename T, int k>
2383        struct alternate<T, k, std::enable_if_t<k % 2 != 0>> {
2384            using type = typename T::template minus_t<typename T::one>;
2385            static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
2386        };
2387    }
2388
2391    template<typename T, int k>
2392    using alternate_t = typename internal::alternate<T, k>::type;
2393
2394    template<typename T, size_t k>
2395    inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
2396
2397    // pow
2398    namespace internal {
2399        template<typename T, auto p, auto n>
2400        struct pow {
2401            using type = typename T::template mul_t<typename T::template val<p>, typename pow<T, p, n -
     1>::type>;
2402        };
2403
2404        template<typename T, auto p>
2405        struct pow<T, p, 0> { using type = typename T::one; };
2406    }
2407
2408    template<typename T, auto p, auto n>
2409    using pow_t = typename internal::pow<T, p, n>::type;
2410
2411    namespace internal {
2412        template<typename, template<typename, size_t> typename, class>
2413        struct make_taylor_impl;
2414
2415        template<typename T, template<typename, size_t> typename coeff_at, size_t...  Is>
2416        struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...>> {
2417            using type = typename polynomial<FractionField<T>::template val<typename coeff_at<T,
     Is>::type...>;
2418        };
2419    }
2420
2421    // generic taylor serie, depending on coefficients
2422    template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
2423    using taylor = typename internal::make_taylor_impl<T, coeff_at,
     internal::make_index_sequence_reverse<deg + 1>>::type;
2424
2425    namespace internal {
2426        template<typename T, size_t i>
2427        struct exp_coeff {
2428            using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
2429        };
2430
2431        template<typename T, size_t i, typename E = void>
2432        struct sin_coeff_helper {};
2433
2434        template<typename T, size_t i>
2435        struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
2436            using type = typename FractionField<T>::zero;
2437        };
2438
2439        template<typename T, size_t i>
2440        struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
2441            using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
2442        };
2443
2444        template<typename T, size_t i>
```

```
2445        struct sin_coeff {
2446            using type = typename sin_coeff_helper<T, i>::type;
2447        };
2448
2449        template<typename T, size_t i, typename E = void>
2450        struct sh_coeff_helper {};
2451
2452        template<typename T, size_t i>
2453        struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
2454            using type = typename FractionField<T>::zero;
2455        };
2456
2457        template<typename T, size_t i>
2458        struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
2459            using type = makefraction_t<T, typename T::one, factorial_t<T, i»;
2460        };
2461
2462        template<typename T, size_t i>
2463        struct sh_coeff {
2464            using type = typename sh_coeff_helper<T, i>::type;
2465        };
2466
2467        template<typename T, size_t i, typename E = void>
2468        struct cos_coeff_helper {};
2469
2470        template<typename T, size_t i>
2471        struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
2472            using type = typename FractionField<T>::zero;
2473        };
2474
2475        template<typename T, size_t i>
2476        struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
2477            using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i»;
2478        };
2479
2480        template<typename T, size_t i>
2481        struct cos_coeff {
2482            using type = typename cos_coeff_helper<T, i>::type;
2483        };
2484
2485        template<typename T, size_t i, typename E = void>
2486        struct cosh_coeff_helper {};
2487
2488        template<typename T, size_t i>
2489        struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
2490            using type = typename FractionField<T>::zero;
2491        };
2492
2493        template<typename T, size_t i>
2494        struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
2495            using type = makefraction_t<T, typename T::one, factorial_t<T, i»;
2496        };
2497
2498        template<typename T, size_t i>
2499        struct cosh_coeff {
2500            using type = typename cosh_coeff_helper<T, i>::type;
2501        };
2502
2503        template<typename T, size_t i>
2504        struct geom_coeff { using type = typename FractionField<T>::one; };
2505
2506
2507        template<typename T, size_t i, typename E = void>
2508        struct atan_coeff_helper;
2509
2510        template<typename T, size_t i>
2511        struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
2512            using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i»;
2513        };
2514
2515        template<typename T, size_t i>
2516        struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
2517            using type = typename FractionField<T>::zero;
2518        };
2519
2520        template<typename T, size_t i>
2521        struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
2522
2523        template<typename T, size_t i, typename E = void>
2524        struct asin_coeff_helper;
2525
2526        template<typename T, size_t i>
2527        struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
2528        {
2529            using type = makefraction_t<T,
2530                factorial_t<T, i - 1>,
2531                typename T::template mul_t<
```

```
2532                     typename T::template val<i>,
2533                     T::template mul_t<
2534                         pow_t<T, 4, i / 2>,
2535                         pow<T, factorial<T, i / 2>::value, 2
2536                     >
2537                 >
2538                 »;
2539         };
2540
2541         template<typename T, size_t i>
2542         struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
2543         {
2544             using type = typename FractionField<T>::zero;
2545         };
2546
2547         template<typename T, size_t i>
2548         struct asin_coeff {
2549             using type = typename asin_coeff_helper<T, i>::type;
2550         };
2551
2552         template<typename T, size_t i>
2553         struct lnp1_coeff {
2554             using type = makefraction_t<T,
2555                 alternate_t<T, i + 1>,
2556                 typename T::template val<i»;
2557         };
2558
2559         template<typename T>
2560         struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
2561
2562         template<typename T, size_t i, typename E = void>
2563         struct asinh_coeff_helper;
2564
2565         template<typename T, size_t i>
2566         struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
2567         {
2568             using type = makefraction_t<T,
2569                 typename T::template mul_t<
2570                     alternate_t<T, i / 2>,
2571                     factorial_t<T, i - 1>
2572                 >,
2573                 typename T::template mul_t<
2574                     T::template mul_t<
2575                         typename T::template val<i>,
2576                         pow_t<T, (factorial<T, i / 2>::value), 2>
2577                     >,
2578                     pow_t<T, 4, i / 2>
2579                 >
2580             >;
2581         };
2582
2583         template<typename T, size_t i>
2584         struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
2585         {
2586             using type = typename FractionField<T>::zero;
2587         };
2588
2589         template<typename T, size_t i>
2590         struct asinh_coeff {
2591             using type = typename asinh_coeff_helper<T, i>::type;
2592         };
2593
2594         template<typename T, size_t i, typename E = void>
2595         struct atanh_coeff_helper;
2596
2597         template<typename T, size_t i>
2598         struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
2599         {
2600             // 1/i
2601             using type = typename FractionField<T>::  template val<
2602                 typename T::one,
2603                 typename T::template val<static_cast<typename T::inner_type>(i)»;
2604         };
2605
2606         template<typename T, size_t i>
2607         struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
2608         {
2609             using type = typename FractionField<T>::zero;
2610         };
2611
2612         template<typename T, size_t i>
2613         struct atanh_coeff {
2614             using type = typename asinh_coeff_helper<T, i>::type;
2615         };
2616
2617         template<typename T, size_t i, typename E = void>
2618         struct tan_coeff_helper;
```

```
2619
2620        template<typename T, size_t i>
2621        struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
2622            using type = typename FractionField<T>::zero;
2623        };
2624
2625        template<typename T, size_t i>
2626        struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
2627        private:
2628            // 4^((i+1)/2)
2629            using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
2630            // 4^((i+1)/2) - 1
2631            using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
     FractionField<T>::one>;
2632            // (-1)^((i-1)/2)
2633            using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
2634            using dividend = typename FractionField<T>::template mul_t<
2635                altp,
2636                FractionField<T>::template mul_t<
2637                _4p,
2638                FractionField<T>::template mul_t<
2639                _4pm1,
2640                bernouilli_t<T, (i + 1)>
2641                >
2642                >
2643            >;
2644        public:
2645            using type = typename FractionField<T>::template div_t<dividend,
2646                typename FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
2647        };
2648
2649        template<typename T, size_t i>
2650        struct tan_coeff {
2651            using type = typename tan_coeff_helper<T, i>::type;
2652        };
2653
2654        template<typename T, size_t i, typename E = void>
2655        struct tanh_coeff_helper;
2656
2657        template<typename T, size_t i>
2658        struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
2659            using type = typename FractionField<T>::zero;
2660        };
2661
2662        template<typename T, size_t i>
2663        struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
2664        private:
2665            using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
2666            using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
     FractionField<T>::one>;
2667            using dividend =
2668                typename FractionField<T>::template mul_t<
2669                _4p,
2670                typename FractionField<T>::template mul_t<
2671                _4pm1,
2672                bernouilli_t<T, (i + 1)>
2673                >
2674                >::type;
2675        public:
2676            using type = typename FractionField<T>::template div_t<dividend,
2677                FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
2678        };
2679
2680        template<typename T, size_t i>
2681        struct tanh_coeff {
2682            using type = typename tanh_coeff_helper<T, i>::type;
2683        };
2684    }
2685
2689    template<typename T, size_t deg>
2690    using exp = taylor<T, internal::exp_coeff, deg>;
2691
2695    template<typename T, size_t deg>
2696    using expm1 = typename polynomial<FractionField<T»::template sub_t<
2697        exp<T, deg>,
2698        typename polynomial<FractionField<T»::one>;
2699
2703    template<typename T, size_t deg>
2704    using lnp1 = taylor<T, internal::lnp1_coeff, deg>;
2705
2709    template<typename T, size_t deg>
2710    using atan = taylor<T, internal::atan_coeff, deg>;
2711
2715    template<typename T, size_t deg>
2716    using sin = taylor<T, internal::sin_coeff, deg>;
2717
2721    template<typename T, size_t deg>
```

```
2722     using sinh = taylor<T, internal::sh_coeff, deg>;
2723
2727     template<typename T, size_t deg>
2728     using cosh = taylor<T, internal::cosh_coeff, deg>;
2729
2733     template<typename T, size_t deg>
2734     using cos = taylor<T, internal::cos_coeff, deg>;
2735
2739     template<typename T, size_t deg>
2740     using geometric_sum = taylor<T, internal::geom_coeff, deg>;
2741
2745     template<typename T, size_t deg>
2746     using asin = taylor<T, internal::asin_coeff, deg>;
2747
2751     template<typename T, size_t deg>
2752     using asinh = taylor<T, internal::asinh_coeff, deg>;
2753
2757     template<typename T, size_t deg>
2758     using atanh = taylor<T, internal::atanh_coeff, deg>;
2759
2763     template<typename T, size_t deg>
2764     using tan = taylor<T, internal::tan_coeff, deg>;
2765
2769     template<typename T, size_t deg>
2770     using tanh = taylor<T, internal::tanh_coeff, deg>;
2771 }
2772
2773 // continued fractions
2774 namespace aerobus {
2777     template<int64_t...  values>
2778     struct ContinuedFraction {};
2779
2780     template<int64_t a0>
2781     struct ContinuedFraction<a0> {
2782         using type = typename q64::template inject_constant_t<a0>;
2783         static constexpr double val = type::template get<double>();
2784     };
2785
2786     template<int64_t a0, int64_t...  rest>
2787     struct ContinuedFraction<a0, rest...> {
2788         using type = q64::template add_t<
2789                 typename q64::template inject_constant_t<a0>,
2790                 typename q64::template div_t<
2791                     typename q64::one,
2792                     typename ContinuedFraction<rest...>::type
2793                 »;
2794         static constexpr double val = type::template get<double>();
2795     };
2796
2801     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>;
2804     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>;
2806     using SQRT2_fraction =
     ContinuedFraction<1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2>;
2808     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>;
2809 }
2810
2811 // known polynomials
2812 namespace aerobus {
2813     namespace internal {
2814         template<int kind, int deg>
2815         struct chebyshev_helper {
2816             using type = typename pi64::template sub_t<
2817                 typename pi64::template mul_t<
2818                     typename pi64::template mul_t<
2819                         pi64::inject_constant_t<2>,
2820                         typename pi64::X
2821                     >,
2822                     typename chebyshev_helper<kind, deg-1>::type
2823                 >,
2824                 typename chebyshev_helper<kind, deg-2>::type
2825             >;
2826         };
2827
2828         template<>
2829         struct chebyshev_helper<1, 0> {
2830             using type = typename pi64::one;
2831         };
2832
2833         template<>
2834         struct chebyshev_helper<1, 1> {
2835             using type = typename pi64::X;
2836         };
2837
2838         template<>
```

```
2839         struct chebyshev_helper<2, 0> {
2840             using type = typename pi64::one;
2841         };
2842 
2843         template<>
2844         struct chebyshev_helper<2, 1> {
2845             using type = typename pi64::template mul_t<
2846                             typename pi64::inject_constant_t<2>,
2847                             typename pi64::X>;
2848         };
2849     }
2850 
2853     template<size_t deg>
2854     using chebyshev_T = typename internal::chebyshev_helper<1, deg>::type;
2855 
2858     template<size_t deg>
2859     using chebyshev_U = typename internal::chebyshev_helper<2, deg>::type;
2860 }
```

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