Aerobus

Generated by Doxygen 1.9.4

Chapter 1

Concept Index

1.1 Concepts

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Concept Documentation

4.1 aerobus::IsEuclideanDomain Concept Reference

Concept to express R is an euclidean domain.

`#include <lib.h>`

4.1.1 Concept definition

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

4.1.2 Detailed Description

Concept to express R is an euclidean domain.

4.2 aerobus::IsField Concept Reference

Concept to express R is a field.

`#include <lib.h>`

4.2.1 Concept definition

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

4.2.2 Detailed Description

Concept to express R is a field.

4.3 aerobus::IsRing Concept Reference

Concept to express R is a Ring (ordered)

`#include <lib.h>`

4.3.1 Concept definition

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

4.3.2 Detailed Description

Concept to express R is a Ring (ordered)

Chapter 5

Class Documentation

5.1 aerobus::bigint Struct Reference

Classes

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

Public Types

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

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

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

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

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

  shift left operator (add zeros to the end)

- template<typename I , size_t s>
  using **shift_right_t** = typename shift_right_helper< I, s >::type
  shift right operator (get highest digits)
- template<typename I1 , typename I2 >
  using **mul_t** = typename mul< I1, I2 >::type
  *multiplication operator (I1 * I2)*
- template<typename... Is>
  using **vadd_t** = typename vadd< Is... >::type
  addition of multiple values
- template<typename I >
  using **div_2_t** = typename div_2< I >::type
  division by 2
- template<typename I1 , typename I2 >
  using **div_t** = typename div_helper< I1, I2 >::Q
  division operator (I1/I2)
- template<typename I1 , typename I2 >
  using **mod_t** = typename div_helper< I1, I2 >::R
  modulo (remainder) operator (I1 % I2)
- template<typename I1 , typename I2 >
  using **gcd_t** = gcd_t< bigint, I1, I2 >
  gcd operator
- template<typename I1 , typename I2 , typename I3 >
  using **fma_t** = add_t< mul_t< I1, I2 >, I3 >
  *fma operator (I1 * I2 + I3)*

Static Public Attributes

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

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

- src/lib.h

5.2 aerobus::polynomial< Ring >::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 >::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 >::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 **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>::value

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

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

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

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

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

  is v posititive

5.15.1 Detailed Description

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

5.15.2 Member Data Documentation

5.15.2.1 pos_v

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

is v posititive

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

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

- src/lib.h

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

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

- src/lib.h

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

Static Public Member Functions

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

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

- src/lib.h

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

Public Types

- using **type** = lowerbound

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

- src/lib.h

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

Public Types

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

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

- src/lib.h

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

Public Types

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

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

- src/lib.h

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

Static Public Member Functions

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

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

- src/lib.h

5.22 aerobus::is_prime< n > Struct Template Reference

checks if n is prime

`#include <lib.h>`

Static Public Attributes

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

5.22.1 Detailed Description

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

checks if n is prime

Template Parameters

n	

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

- src/lib.h

5.23 aerobus::polynomial< Ring > 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 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 >, 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 lt_v = lt_helper<v1, v2>::value

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

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

  checks for positivity (an > 0)

5.23.1 Detailed Description

template<typename Ring>
requires IsEuclideanDomain<Ring>
struct aerobus::polynomial< Ring >

polynomial with coefficients in Ring Ring must be an integral domain

5.23.2 Member Typedef Documentation

5.23.2.1 add_t

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

adds two polynomials

Template Parameters

v1	
v2	

5.23.2.2 derive_t

```
template<typename Ring >
```

```
template<typename v >
using aerobus::polynomial< Ring >::derive_t = typename derive_helper<v>::type
```

derivation operator

Template Parameters

v	

5.23.2.3 div_t

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

division operator

Template Parameters

v1	
v2	

5.23.2.4 gcd_t

```
template<typename Ring >
template<typename v1 , typename v2 >
using aerobus::polynomial< Ring >::gcd_t = std::conditional_t< Ring::is_euclidean_domain,
typename make_unit<gcd_t<polynomial<Ring>, v1, v2> >::type, void>
```

greatest common divisor of two polynomials

Template Parameters

v1	
v2	

5.23.2.5 mod_t

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

modulo operator

Template Parameters

v1	
v2	

5.23.2.6 monomial_t

```
template<typename Ring >
template<typename coeff , size_t deg>
using aerobus::polynomial< Ring >::monomial_t = typename monomial<coeff, deg>::type
```

monomial : coeff X^deg

Template Parameters

coeff	
deg	

5.23.2.7 mul_t

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

multiplication of two polynomials

Template Parameters

v1	
v2	

5.23.2.8 simplify_t

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

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

Template Parameters

P	

5.23.2.9 sub_t

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

substraction of two polynomials

Template Parameters

v1	
v2	

5.23.3 Member Data Documentation

5.23.3.1 eq_v

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

equality operator

Template Parameters

v1	
v2	

5.23.3.2 gt_v

```
template<typename Ring >
template<typename v1 , typename v2 >
constexpr bool aerobus::polynomial< Ring >::gt_v = gt_helper<v1, v2>::value  [static], [constexpr]
```

strict greater operator

Template Parameters

v1	
v2	

5.23.3.3 lt_v

```
template<typename Ring >
template<typename v1 , typename v2 >
constexpr bool aerobus::polynomial< Ring >::lt_v = lt_helper<v1, v2>::value  [static], [constexpr]
```

strict less operator

Template Parameters

v1	
v2	

5.23.3.4 pos_v

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

checks for positivity (an > 0)

Template Parameters

v	

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

- src/lib.h

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

Public Types

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

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

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

- src/lib.h

5.25 aerobus::QuadraticExtension< Field, d > Struct Template Reference

Quadratic extension of Field.

`#include <lib.h>`

Classes

- struct val

Public Types

- using **zero** = val< typename Field::template inject_constant_t< 0 >, typename Field::template inject_constant_t< 0 > >
- using **one** = val< typename Field::template inject_constant_t< 1 >, typename Field::template inject_constant_t< 0 > >
- template<auto x>
  using **inject_constant_t** = val< typename Field::template inject_constant_t< x >, typename Field::zero >
- template<auto x, auto y>
  using **inject_values_t** = val< typename Field::template inject_constant_t< x >, typename Field::template inject_constant_t< y > >
- 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 v >
  using **minus_t** = sub_t< zero, v >
- template<typename v1 , typename v2 >
  using **mod_t** = zero
- template<typename v1 , typename v2 >
  using **gcd_t** = v1

Static Public Attributes

- template<typename v >
  static constexpr bool **is_in_field_v** = (v::y == 0)
- static constexpr bool **is_field** = true
- static constexpr bool **is_euclidean_domain** = true
- template<typename v1 , typename v2 >
  static constexpr bool eq_v
- template<typename v1 , typename v2 >
  static constexpr bool gt_v
- template<typename v >
  static constexpr bool **pos_v** = gt_v<v, zero>

5.25.1 Detailed Description

template<typename Field, int64_t d>
requires IsField<Field>
struct aerobus::QuadraticExtension< Field, d >

Quadratic extension of Field.

Template Parameters

Field	can be any version of Q (q32, q64, qbintint)
d	

5.25.2 Member Data Documentation

5.25.2.1 eq_v

```
template<typename Field , int64_t d>
template<typename v1 , typename v2 >
constexpr bool aerobus::QuadraticExtension< Field, d >::eq_v  [static], [constexpr]
```

Initial value:

```
=
            (Field::template eq_v<typename v1::x, typename v2::x>) &&
            (Field::template eq_v<typename v1::y, typename v2::y>)
```

5.25.2.2 gt_v

```
template<typename Field , int64_t d>
template<typename v1 , typename v2 >
constexpr bool aerobus::QuadraticExtension< Field, d >::gt_v  [static], [constexpr]
```

Initial value:

```
=
            (Field::template gt_v<v1::x, v2::x>) ||
            ((Field::template eq_v<v1::x, v2::x>) && (Field::template gt_v<v1::y, v2::y>))
```

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

- src/lib.h

5.26 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

5.27 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

5.28 aerobus::string_literal< N > Struct Template Reference

Public Member Functions

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

Public Attributes

- char **value** [N]

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

- src/lib.h

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

Static Public Member Functions

- static std::string **func** (const bool prefix=false)

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

- src/lib.h

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

Static Public Member Functions

- static std::string **func** (const bool prefix=false)

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

- src/lib.h

5.31 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)

5.31.1 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

5.32 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

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

Classes

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

Public Types

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

Static Public Member Functions

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

Static Public Attributes

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

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

- src/lib.h

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

values in i32

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

Static Public Member Functions

- template<typename valueType >
  DEVICE static INLINED constexpr valueType get ()
  cast x into valueType
- static std::string **to_string** ()
  string representation of value
- template<typename valueRing >
  DEVICE static INLINED 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

5.34.1 Detailed Description

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

values in i32

Template Parameters

x	an actual integer

5.34.2 Member Function Documentation

5.34.2.1 eval()

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

cast x into valueRing

Template Parameters

valueRing	double for example

5.34.2.2 get()

```
template<int32_t x>
template<typename valueType >
DEVICE static INLINED 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

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

values in i64

`#include <lib.h>`

Static Public Member Functions

- template<typename valueType >
  DEVICE static INLINED constexpr valueType get ()
  cast value in valueType
- static std::string **to_string** ()
  string representation
- template<typename valueRing >
  DEVICE static INLINED 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

5.35.1 Detailed Description

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

values in i64

Template Parameters

x	an actual integer

5.35.2 Member Function Documentation

5.35.2.1 eval()

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

cast value in valueRing

Template Parameters

valueRing	(double for example)

5.35.2.2 get()

```
template<int64_t x>
template<typename valueType >
DEVICE static INLINED 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

5.36 aerobus::polynomial< Ring >::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 >
  DEVICE static INLINED 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

5.36.1 Member Typedef Documentation

5.36.1.1 coeff_at_t

```
template<typename Ring >
template<typename coeffN , typename...  coeffs>
template<size_t index>
using aerobus::polynomial< Ring >::val< coeffN, coeffs >::coeff_at_t = typename coeff_↩
at<index>::type
```

coefficient at index

Template Parameters

index	

5.36.2 Member Function Documentation

5.36.2.1 eval()

```
template<typename Ring >
template<typename coeffN , typename...  coeffs>
template<typename valueRing >
DEVICE static INLINED constexpr valueRing aerobus::polynomial< Ring >::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)

5.36.2.2 to_string()

```
template<typename Ring >
template<typename coeffN , typename...  coeffs>
static std::string aerobus::polynomial< Ring >::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

5.37 aerobus::QuadraticExtension< Field, d >::val< v1, v2 > Struct Template Reference

Public Types

- using **x** = v1
- using **y** = v2

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

- src/lib.h

5.38 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

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

Static Public Member Functions

- template<typename valueType >
  DEVICE static INLINED constexpr valueType **get** ()
- static std::string **to_string** ()
- template<typename valueRing >
  DEVICE static INLINED 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

5.40 aerobus::polynomial< Ring >::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 >
  DEVICE static INLINED 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

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

Classes

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

Public Types

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

Static Public Member Functions

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

Static Public Attributes

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

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

- src/lib.h

5.42 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 **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>::value
- template<typename v1 , typename v2 >
  static constexpr bool **lt_v** = lt<v1, v2>::value
- template<typename v1 , typename v2 >
  static constexpr bool **eq_v** = eq<v1, v2>::value
- template<typename v >
  static constexpr bool **pos_v** = pos<v>::value

5.42.1 Detailed Description

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

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

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

- src/lib.h

Chapter 6

File Documentation

6.1 lib.h

```
1 // -*- lsst-c++ -*-
2
3 #include <cstdint> // NOLINT(clang-diagnostic-pragma-pack)
4 #include <cstddef>
5 #include <cstring>
6 #include <type_traits>
7 #include <utility>
8 #include <algorithm>
9 #include <functional>
10 #include <string>
11 #include <concepts>
12 #include <array>
13 #include <format>
14
15 #ifdef __CUDACC__
16 #define DEVICE __host__ __device__
17 #else
18 #define DEVICE
19 #endif
20
21
22 #ifdef _MSC_VER
23 #define ALIGNED(x) __declspec(align(x))
24 #define INLINED __forceinline
25 #else
26 #define ALIGNED(x) __attribute__((aligned(x)))
27 #define INLINED __attribute__((always_inline)) inline
28 #endif
29
30 // aligned allocation
31 namespace aerobus {
38     template<typename T>
39     T* aligned_malloc(size_t count, size_t alignment) {
40 #ifdef _MSC_VER
41         return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
42 #else
43         return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
44 #endif
45     }
46
47     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 } };
48
57      template<typename T, size_t N>
58      constexpr bool contains(const std::array<T, N>& arr, const T& v) {
59          for (const auto& vv :  arr) {
60              if (v == vv) {
61                  return true;
62              }
63          }
64
65          return false;
66      }
67
68      template <size_t N>
69      struct string_literal {
70          constexpr string_literal(const char(&str)[N]) {
71              std::reverse_copy(str, str + N, value);
72          }
73
74          template<size_t i>
75          constexpr char char_at()const {
76              if constexpr (i < N) {
77                  return this->value[i];
78              }
79              return 0;
80          }
81
82          constexpr size_t len()const { return N; }
83
84          char value[N];
85      };
86  }
87
88  // concepts
89  namespace aerobus
90  {
92      template <typename R>
93      concept IsRing = requires {
94          typename R::one;
95          typename R::zero;
96          typename R::template add_t<typename R::one, typename R::one>;
97          typename R::template sub_t<typename R::one, typename R::one>;
98          typename R::template mul_t<typename R::one, typename R::one>;
99          typename R::template minus_t<typename R::one>;
100          R::template eq_v<typename R::one, typename R::one> == true;
101      };
102
104      template <typename R>
```

```
105     concept IsEuclideanDomain = IsRing<R> && requires {
106         typename R::template div_t<typename R::one, typename R::one>;
107         typename R::template mod_t<typename R::one, typename R::one>;
108         typename R::template gcd_t<typename R::one, typename R::one>;
109 
110         R::template pos_v<typename R::one> == true;
111         R::template gt_v<typename R::one, typename R::zero> == true;
112         R::is_euclidean_domain == true;
113     };
114 
115     template<typename R>
116     concept IsField = IsEuclideanDomain<R> && requires {
117         R::is_field == true;
118     };
119 }
120 
121 // utilities
122 namespace aerobus {
123     namespace internal
124     {
125         template<template<typename...> typename TT, typename T>
126         struct is_instantiation_of :  std::false_type { };
127 
128         template<template<typename...> typename TT, typename...  Ts>
129         struct is_instantiation_of<TT, TT<Ts...>> :  std::true_type { };
130 
131         template<template<typename...> typename TT, typename T>
132         inline constexpr bool is_instantiation_of_v = is_instantiation_of<TT, T>::value;
133 
134         template <size_t i, typename T, typename...  Ts>
135         struct type_at
136         {
137             static_assert(i < sizeof...(Ts) + 1, "index out of range");
138             using type = typename type_at<i - 1, Ts...>::type;
139         };
140 
141         template <typename T, typename...  Ts> struct type_at<0, T, Ts...> {
142             using type = T;
143         };
144 
145         template <size_t i, typename...  Ts>
146         using type_at_t = typename type_at<i, Ts...>::type;
147 
148         template<size_t i, auto x, auto...  xs>
149         struct value_at {
150             static_assert(i < sizeof...(xs) + 1, "index out of range");
151             static constexpr auto value = value_at<i - 1, xs...>::value;
152         };
153 
154         template<auto x, auto...  xs>
155         struct value_at<0, x, xs...> {
156             static constexpr auto value = x;
157         };
158 
159 
160         template<int32_t n, int32_t i, typename E = void>
161         struct _is_prime {};
162 
163         // first 1000 primes are precomputed and stored in a table
164         template<int32_t n, int32_t i>
165         struct _is_prime<n, i, std::enable_if_t<(n < 7920) && (contains<int32_t, 1000>(primes, n))> :
    std::true_type {};
166 
167         // first 1000 primes are precomputed and stored in a table
168         template<int32_t n, int32_t i>
169         struct _is_prime<n, i, std::enable_if_t<(n < 7920) && (!contains<int32_t, 1000>(primes, n))> :
    std::false_type {};
170 
171         template<int32_t n, int32_t i>
172         struct _is_prime<n, i, std::enable_if_t<
173             (n >= 7920) &&
174             (i >= 5 && i * i <= n) &&
175             (n% i == 0 || n % (i + 2) == 0)> :  std::false_type {};
176 
177 
178         template<int32_t n, int32_t i>
179         struct _is_prime<n, i, std::enable_if_t<
180             (n >= 7920) &&
181             (i >= 5 && i * i <= n) &&
182             (n% i != 0 && n % (i + 2) != 0)> {
183             static constexpr bool value = _is_prime<n, i + 6>::value;
184         };
185 
186         template<int32_t n, int32_t i>
187         struct _is_prime<n, i, std::enable_if_t<
188             (n >= 7920) &&
189             (i >= 5 && i * i > n)> :  std::true_type {};
190 
```

```
191     }
192
195     template<int32_t n>
196     struct is_prime {
198         static constexpr bool value = internal::_is_prime<n, 5>::value;
199     };
200
201     namespace internal {
202         template <std::size_t...  Is>
203         constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
204             -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
205
206         template <std::size_t N>
207         using make_index_sequence_reverse
208             = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
209
215         template<typename Ring, typename E = void>
216         struct gcd;
217
218         template<typename Ring>
219         struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
220             template<typename A, typename B, typename E = void>
221             struct gcd_helper {};
222
223             // B = 0, A > 0
224             template<typename A, typename B>
225             struct gcd_helper<A, B, std::enable_if_t<
226                 B::is_zero_v&& Ring::template pos_v<A»>
227             {
228                 using type = A;
229             };
230
231             // B = 0, A < 0
232             template<typename A, typename B>
233             struct gcd_helper<A, B, std::enable_if_t<
234                 B::is_zero_v && !Ring::template pos_v<A»>
235             {
236                 using type = typename Ring::template minus_t<A>;
237             };
238
239             // B != 0
240             template<typename A, typename B>
241             struct gcd_helper<A, B, std::enable_if_t<
242                 (!B::is_zero_v)
243                 » {
244             private:
245                 // A / B
246                 using k = typename Ring::template div_t<A, B>;
247                 // A - (A/B)*B = A % B
248                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
249             public:
250                 using type = typename gcd_helper<B, m>::type;
251             };
252
253             template<typename A, typename B>
254             using type = typename gcd_helper<A, B>::type;
255         };
256     }
257
260     template<typename T, typename A, typename B>
261     using gcd_t = typename internal::gcd<T>::template type<A, B>;
262 }
263
264 // quotient ring by the principal ideal generated by X
265 namespace aerobus {
266     template<typename Ring, typename X>
267         requires IsRing<Ring>
268     struct Quotient {
269         template <typename V>
270         struct val {
271         private:
272             using tmp = typename Ring::template mod_t<V, X>;
273         public:
274             using type = std::conditional_t<
275                 Ring::template pos_v<tmp>,
276                 tmp,
277                 typename Ring::template minus_t<tmp>
278             >;
279         };
280
281         using zero = val<typename Ring::zero>;
282         using one = val<typename Ring::one>;
283
284         template<typename v1, typename v2>
285         using add_t = val<typename Ring::template add_t<typename v1::type, typename v2::type>>;
286         template<typename v1, typename v2>
287         using mul_t = val<typename Ring::template mul_t<typename v1::type, typename v2::type>>;
```

```
288         template<typename v1, typename v2>
289         using div_t = val<typename Ring::template div_t<typename v1::type, typename v2::type>>;
290         template<typename v1, typename v2>
291         using mod_t = val<typename Ring::template mod_t<typename v1::type, typename v2::type>>;
292 
293         template<typename v1, typename v2>
294         static constexpr bool eq_v = Ring::template eq_v<typename v1::type, typename v2::type>;
295 
296         template<typename v>
297         static constexpr bool pos_v = true;
298 
299         static constexpr bool is_euclidean_domain = true;
300 
301         template<auto x>
302         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
303 
304         template<typename v>
305         using inject_ring_t = val<v>;
306     };
307 }
308 
309 // type_list
310 namespace aerobus
311 {
313     template <typename...  Ts>
314     struct type_list;
315 
316     namespace internal
317     {
318         template <typename T, typename...  Us>
319         struct pop_front_h
320         {
321             using tail = type_list<Us...>;
322             using head = T;
323         };
324 
325         template <uint64_t index, typename L1, typename L2>
326         struct split_h
327         {
328         private:
329             static_assert(index <= L2::length, "index ouf of bounds");
330             using a = typename L2::pop_front::type;
331             using b = typename L2::pop_front::tail;
332             using c = typename L1::template push_back<a>;
333 
334         public:
335             using head = typename split_h<index - 1, c, b>::head;
336             using tail = typename split_h<index - 1, c, b>::tail;
337         };
338 
339         template <typename L1, typename L2>
340         struct split_h<0, L1, L2>
341         {
342             using head = L1;
343             using tail = L2;
344         };
345 
346         template <uint64_t index, typename L, typename T>
347         struct insert_h
348         {
349             static_assert(index <= L::length, "index ouf of bounds");
350             using s = typename L::template split<index>;
351             using left = typename s::head;
352             using right = typename s::tail;
353             using ll = typename left::template push_back<T>;
354             using type = typename ll::template concat<right>;
355         };
356 
357         template <uint64_t index, typename L>
358         struct remove_h
359         {
360             using s = typename L::template split<index>;
361             using left = typename s::head;
362             using right = typename s::tail;
363             using rr = typename right::pop_front::tail;
364             using type = typename left::template concat<rr>;
365         };
366     }
367 
368     template <typename...  Ts>
369     struct type_list
370     {
371     private:
372         template <typename T>
373         struct concat_h;
374 
375         template <typename...  Us>
```

```
376         struct concat_h<type_list<Us...»
377         {
378             using type = type_list<Ts..., Us...>;
379         };
380 
381     public:
382         static constexpr size_t length = sizeof...(Ts);
383 
384         template <typename T>
385         using push_front = type_list<T, Ts...>;
386 
387         template <uint64_t index>
388         using at = internal::type_at_t<index, Ts...>;
389 
390         struct pop_front
391         {
392             using type = typename internal::pop_front_h<Ts...>::head;
393             using tail = typename internal::pop_front_h<Ts...>::tail;
394         };
395 
396         template <typename T>
397         using push_back = type_list<Ts..., T>;
398 
399         template <typename U>
400         using concat = typename concat_h<U>::type;
401 
402         template <uint64_t index>
403         struct split
404         {
405         private:
406             using inner = internal::split_h<index, type_list<>, type_list<Ts...»;
407 
408         public:
409             using head = typename inner::head;
410             using tail = typename inner::tail;
411         };
412 
413         template <uint64_t index, typename T>
414         using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
415 
416         template <uint64_t index>
417         using remove = typename internal::remove_h<index, type_list<Ts...»::type;
418     };
419 
420     template <>
421     struct type_list<>
422     {
423         static constexpr size_t length = 0;
424 
425         template <typename T>
426         using push_front = type_list<T>;
427 
428         template <typename T>
429         using push_back = type_list<T>;
430 
431         template <typename U>
432         using concat = U;
433 
434         // TODO: assert index == 0
435         template <uint64_t index, typename T>
436         using insert = type_list<T>;
437     };
438 }
439 
440 // i32
441 namespace aerobus {
443     struct i32 {
444         using inner_type = int32_t;
447         template<int32_t x>
448         struct val {
449             static constexpr int32_t v = x;
450 
453             template<typename valueType>
454             DEVICE INLINED static constexpr valueType get() { return static_cast<valueType>(x); }
455 
457             static constexpr bool is_zero_v = x == 0;
458 
460             static std::string to_string() {
461                 return std::to_string(x);
462             }
463 
466             template<typename valueRing>
467             DEVICE INLINED static constexpr valueRing eval(const valueRing& v) {
468                 return static_cast<valueRing>(x);
469             }
470         };
471 
```

```
473         using zero = val<0>;
475         using one = val<1>;
477         static constexpr bool is_field = false;
479         static constexpr bool is_euclidean_domain = true;
483         template<auto x>
484         using inject_constant_t = val<static_cast<int32_t>(x)>;
485
486         template<typename v>
487         using inject_ring_t = v;
488
489     private:
490         template<typename v1, typename v2>
491         struct add {
492             using type = val<v1::v + v2::v>;
493         };
494
495         template<typename v1, typename v2>
496         struct sub {
497             using type = val<v1::v - v2::v>;
498         };
499
500         template<typename v1, typename v2>
501         struct mul {
502             using type = val<v1::v* v2::v>;
503         };
504
505         template<typename v1, typename v2>
506         struct div {
507             using type = val<v1::v / v2::v>;
508         };
509
510         template<typename v1, typename v2>
511         struct remainder {
512             using type = val<v1::v% v2::v>;
513         };
514
515         template<typename v1, typename v2>
516         struct gt {
517             static constexpr bool value = (v1::v > v2::v);
518         };
519
520         template<typename v1, typename v2>
521         struct lt {
522             static constexpr bool value = (v1::v < v2::v);
523         };
524
525         template<typename v1, typename v2>
526         struct eq {
527             static constexpr bool value = (v1::v == v2::v);
528         };
529
530     public:
532         template<typename v1, typename v2>
533         using add_t = typename add<v1, v2>::type;
534
536         template<typename v1>
537         using minus_t = val<-v1::v>;
538
540         template<typename v1, typename v2>
541         using sub_t = typename sub<v1, v2>::type;
542
544         template<typename v1, typename v2>
545         using mul_t = typename mul<v1, v2>::type;
546
548         template<typename v1, typename v2>
549         using div_t = typename div<v1, v2>::type;
550
552         template<typename v1, typename v2>
553         using mod_t = typename remainder<v1, v2>::type;
554
556         template<typename v1, typename v2>
557         static constexpr bool gt_v = gt<v1, v2>::value;
558
560         template<typename v1, typename v2>
561         static constexpr bool lt_v = lt<v1, v2>::value;
562
564         template<typename v1, typename v2>
565         static constexpr bool eq_v = eq<v1, v2>::value;
566
568         template<typename v1>
569         static constexpr bool pos_v = (v1::v > 0);
570
572         template<typename v1, typename v2>
573         using gcd_t = gcd_t<i32, v1, v2>;
574     };
575 }
576
```

```
577 // i64
578 namespace aerobus {
580     struct i64 {
581         using inner_type = int64_t;
584         template<int64_t x>
585         struct val {
586             static constexpr int64_t v = x;
587 
590             template<typename valueType>
591             DEVICE INLINED static constexpr valueType get() { return static_cast<valueType>(x); }
592 
594             static constexpr bool is_zero_v = x == 0;
595 
597             static std::string to_string() {
598                 return std::to_string(x);
599             }
600 
603             template<typename valueRing>
604             DEVICE INLINED static constexpr valueRing eval(const valueRing& v) {
605                 return static_cast<valueRing>(x);
606             }
607         };
608 
612         template<auto x>
613         using inject_constant_t = val<static_cast<int64_t>(x)>;
614 
615         template<typename v>
616         using inject_ring_t = v;
617 
619         using zero = val<0>;
621         using one = val<1>;
623         static constexpr bool is_field = false;
625         static constexpr bool is_euclidean_domain = true;
626 
627     private:
628         template<typename v1, typename v2>
629         struct add {
630             using type = val<v1::v + v2::v>;
631         };
632 
633         template<typename v1, typename v2>
634         struct sub {
635             using type = val<v1::v - v2::v>;
636         };
637 
638         template<typename v1, typename v2>
639         struct mul {
640             using type = val<v1::v* v2::v>;
641         };
642 
643         template<typename v1, typename v2>
644         struct div {
645             using type = val<v1::v / v2::v>;
646         };
647 
648         template<typename v1, typename v2>
649         struct remainder {
650             using type = val<v1::v% v2::v>;
651         };
652 
653         template<typename v1, typename v2>
654         struct gt {
655             static constexpr bool value = (v1::v > v2::v);
656         };
657 
658         template<typename v1, typename v2>
659         struct lt {
660             static constexpr bool value = (v1::v < v2::v);
661         };
662 
663         template<typename v1, typename v2>
664         struct eq {
665             static constexpr bool value = (v1::v == v2::v);
666         };
667 
668     public:
670         template<typename v1, typename v2>
671         using add_t = typename add<v1, v2>::type;
672 
674         template<typename v1>
675         using minus_t = val<-v1::v>;
676 
678         template<typename v1, typename v2>
679         using sub_t = typename sub<v1, v2>::type;
680 
682         template<typename v1, typename v2>
683         using mul_t = typename mul<v1, v2>::type;
```

```
684
686         template<typename v1, typename v2>
687         using div_t = typename div<v1, v2>::type;
688
690         template<typename v1, typename v2>
691         using mod_t = typename remainder<v1, v2>::type;
692
694         template<typename v1, typename v2>
695         static constexpr bool gt_v = gt<v1, v2>::value;
696
698         template<typename v1, typename v2>
699         static constexpr bool lt_v = lt<v1, v2>::value;
700
702         template<typename v1, typename v2>
703         static constexpr bool eq_v = eq<v1, v2>::value;
704
707         template<typename v1>
708         static constexpr bool pos_v = (v1::v > 0);
709
711         template<typename v1, typename v2>
712         using gcd_t = gcd_t<i64, v1, v2>;
713     };
714 }
715
716 // z/pz
717 namespace aerobus {
722     template<int32_t p>
723     struct zpz {
724         using inner_type = int32_t;
725         template<int32_t x>
726         struct val {
727             static constexpr int32_t v = x % p;
728
729             template<typename valueType>
730             DEVICE INLINED static constexpr valueType get() { return static_cast<valueType>(x % p); }
731
732             static constexpr bool is_zero_v = v == 0;
733             static std::string to_string() {
734                 return std::to_string(x % p);
735             }
736
737             template<typename valueRing>
738             DEVICE INLINED static constexpr valueRing eval(const valueRing& v) {
739                 return static_cast<valueRing>(x % p);
740             }
741         };
742
743         template<auto x>
744         using inject_constant_t = val<static_cast<int32_t>(x)>;
745
746         using zero = val<0>;
747         using one = val<1>;
748         static constexpr bool is_field = is_prime<p>::value;
749         static constexpr bool is_euclidean_domain = true;
750
751     private:
752         template<typename v1, typename v2>
753         struct add {
754             using type = val<(v1::v + v2::v) % p>;
755         };
756
757         template<typename v1, typename v2>
758         struct sub {
759             using type = val<(v1::v - v2::v) % p>;
760         };
761
762         template<typename v1, typename v2>
763         struct mul {
764             using type = val<(v1::v* v2::v) % p>;
765         };
766
767         template<typename v1, typename v2>
768         struct div {
769             using type = val<(v1::v% p) / (v2::v % p)>;
770         };
771
772         template<typename v1, typename v2>
773         struct remainder {
774             using type = val<(v1::v% v2::v) % p>;
775         };
776
777         template<typename v1, typename v2>
778         struct gt {
779             static constexpr bool value = (v1::v % p > v2::v % p);
780         };
781
782         template<typename v1, typename v2>
```

```
783         struct lt {
784             static constexpr bool value = (v1::v % p < v2::v % p);
785         };
786 
787         template<typename v1, typename v2>
788         struct eq {
789             static constexpr bool value = (v1::v % p == v2::v % p);
790         };
791 
792         template<typename v1>
793         struct pos {
794             static constexpr bool value = v1::v % p > 0;
795         };
796 
797     public:
799         template<typename v1>
800         using minus_t = val<-v1::v>;
801 
802         template<typename v1, typename v2>
803         using add_t = typename add<v1, v2>::type;
804 
805         template<typename v1, typename v2>
806         using sub_t = typename sub<v1, v2>::type;
807 
808         template<typename v1, typename v2>
809         using mul_t = typename mul<v1, v2>::type;
810 
811         template<typename v1, typename v2>
812         using div_t = typename div<v1, v2>::type;
813 
814         template<typename v1, typename v2>
815         using mod_t = typename remainder<v1, v2>::type;
816 
817         template<typename v1, typename v2>
818         static constexpr bool gt_v = gt<v1, v2>::value;
819 
820         template<typename v1, typename v2>
821         static constexpr bool lt_v = lt<v1, v2>::value;
822 
823         template<typename v1, typename v2>
824         static constexpr bool eq_v = eq<v1, v2>::value;
825 
826         template<typename v1, typename v2>
827         using gcd_t = gcd_t<i32, v1, v2>;
828 
829         template<typename v>
830         static constexpr bool pos_v = pos<v>::value;
831     };
832 }
833 
834 
835 // K[sqrt(x)]
836 namespace aerobus {
840     template<typename Field, int64_t d>
841     requires IsField<Field>
842     struct QuadraticExtension {
843         // v1 + sqrt(x) v2
844         template<typename v1, typename v2>
845         struct val {
846             using x = v1;
847             using y = v2;
848         };
849 
850         using zero = val<typename Field::template inject_constant_t<0>, typename Field::template
    inject_constant_t<0>>;
851         using one = val<typename Field::template inject_constant_t<1>, typename Field::template
    inject_constant_t<0>>;
852 
853         template<typename v>
854         static constexpr bool is_in_field_v = (v::y == 0);
855 
856     private:
857         template<typename v1, typename v2>
858         struct add {
859             using type = val<typename Field::template add_t<typename v1::x, typename v2::x>, typename
    Field::template add_t<typename v1::y, typename v2::y»;
860         };
861 
862         template<typename v1, typename v2>
863         struct sub {
864             using type = val<typename Field::template sub_t<typename v1::x, typename v2::x>, typename
    Field::template sub_t<typename v1::y, typename v2::y»;
865         };
866 
867         template<typename v1, typename v2>
868         struct mul {
869             using type = val<
```

```
870                 typename Field::template add_t<
871                 typename Field::template mul_t<typename v1::x, typename v2::x>,
872                 typename Field::template mul_t<
873                 typename Field::template inject_constant_t<d>,
874                 typename Field::template mul_t<typename v1::y, typename v2::y>
875                 >
876                 >,
877                 typename Field::template add_t<
878                 typename Field::template mul_t<typename v1::x, typename v2::y>,
879                 typename Field::template mul_t<typename v1::y, typename v2::x>
880                 >
881             >;
882         };
883 
884         template<typename v1, typename v2>
885         struct div {
886         private:
887             using inner = typename Field::template div_t<
888                 typename Field::one, typename Field::template sub_t<
889                 typename Field::template mul_t<typename v2::x, typename v2::x>,
890                 typename Field::template mul_t<
891                 typename Field::template inject_constant_t<d>,
892                 typename Field::template mul_t<typename v2::y, typename v2::y>
893                 >
894                 >
895             >;
896             using inv_v2 = val<
897                 typename Field::template mul_t<typename v2::x, inner>,
898                 typename Field::template mul_t<typename Field::template minus_t<typename v2::y>, inner>
899             >;
900         public:
901             using type = typename mul<v1, inv_v2>::type;
902         };
903 
904     public:
905         static constexpr bool is_field = true;
906         static constexpr bool is_euclidean_domain = true;
907 
908         template<auto x>
909         using inject_constant_t = val<typename Field::template inject_constant_t<x>, typename
    Field::zero>;
910 
911         template<auto x, auto y>
912         using inject_values_t = val<typename Field::template inject_constant_t<x>, typename
    Field::template inject_constant_t<y»;
913 
914         template<typename v1, typename v2>
915         using add_t = typename add<v1, v2>::type;
916 
917         template<typename v1, typename v2>
918         using sub_t = typename sub<v1, v2>::type;
919 
920         template<typename v1, typename v2>
921         using mul_t = typename mul<v1, v2>::type;
922 
923         template<typename v1, typename v2>
924         using div_t = typename div<v1, v2>::type;
925 
926         template<typename v>
927         using minus_t = sub_t<zero, v>;
928 
929         template<typename v1, typename v2>
930         static constexpr bool eq_v =
931             (Field::template eq_v<typename v1::x, typename v2::x>) &&
932             (Field::template eq_v<typename v1::y, typename v2::y>);
933 
934         template<typename v1, typename v2>
935         static constexpr bool gt_v =
936             (Field::template gt_v<v1::x, v2::x>) ||
937             ((Field::template eq_v<v1::x, v2::x>) && (Field::template gt_v<v1::y, v2::y>));
938 
939         template<typename v>
940         static constexpr bool pos_v = gt_v<v, zero>;
941 
942         template<typename v1, typename v2>
943         using mod_t = zero;
944 
945         template<typename v1, typename v2>
946         using gcd_t = v1;
947     };
948 }
949 
950 // polynomial
951 namespace aerobus {
952     // coeffN x^N + ...
957     template<typename Ring>
958         requires IsEuclideanDomain<Ring>
```

```
959     struct polynomial {
960         static constexpr bool is_field = false;
961         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
962 
963         template<typename coeffN, typename...  coeffs>
964         struct val {
966             static constexpr size_t degree = sizeof...(coeffs);
968             using aN = coeffN;
970             using strip = val<coeffs...>;
972             static constexpr bool is_zero_v = degree == 0 && aN::is_zero_v;
973 
974         private:
975             template<size_t index, typename E = void>
976             struct coeff_at {};
977 
978             template<size_t index>
979             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))» {
980                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
981             };
982 
983             template<size_t index>
984             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))» {
985                 using type = typename Ring::zero;
986             };
987 
988         public:
991             template<size_t index>
992             using coeff_at_t = typename coeff_at<index>::type;
993 
996             static std::string to_string() {
997                 return string_helper<coeffN, coeffs...>::func();
998             }
999 
1004             template<typename valueRing>
1005             DEVICE INLINED static constexpr valueRing eval(const valueRing& x) {
1006                 return eval_helper<valueRing, val>::template inner<0, degree +
     1>::func(static_cast<valueRing>(0), x);
1007             }
1008         };
1009 
1010         // specialization for constants
1011         template<typename coeffN>
1012         struct val<coeffN> {
1013             static constexpr size_t degree = 0;
1014             using aN = coeffN;
1015             using strip = val<coeffN>;
1016             static constexpr bool is_zero_v = coeffN::is_zero_v;
1017 
1018             template<size_t index, typename E = void>
1019             struct coeff_at {};
1020 
1021             template<size_t index>
1022             struct coeff_at<index, std::enable_if_t<(index == 0)» {
1023                 using type = aN;
1024             };
1025 
1026             template<size_t index>
1027             struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)» {
1028                 using type = typename Ring::zero;
1029             };
1030 
1031             template<size_t index>
1032             using coeff_at_t = typename coeff_at<index>::type;
1033 
1034             static std::string to_string() {
1035                 return string_helper<coeffN>::func();
1036             }
1037 
1038             template<typename valueRing>
1039             DEVICE INLINED static constexpr valueRing eval(const valueRing& x) {
1040                 return static_cast<valueRing>(aN::template get<valueRing>());
1041             }
1042         };
1043 
1045         using zero = val<typename Ring::zero>;
1047         using one = val<typename Ring::one>;
1049         using X = val<typename Ring::one, typename Ring::zero>;
1050 
1051     private:
1052         template<typename P, typename E = void>
1053         struct simplify;
1054 
1055         template <typename P1, typename P2, typename I>
1056         struct add_low;
1057 
1058         template<typename P1, typename P2>
1059         struct add {
```

```
1060            using type = typename simplify<typename add_low<
1061                P1,
1062                P2,
1063                internal::make_index_sequence_reverse<
1064                std::max(P1::degree, P2::degree) + 1
1065                »::type>::type;
1066        };
1067
1068        template <typename P1, typename P2, typename I>
1069        struct sub_low;
1070
1071        template <typename P1, typename P2, typename I>
1072        struct mul_low;
1073
1074        template<typename v1, typename v2>
1075        struct mul {
1076            using type = typename mul_low<
1077                v1,
1078                v2,
1079                internal::make_index_sequence_reverse<
1080                v1::degree + v2::degree + 1
1081                »::type;
1082        };
1083
1084        template<typename coeff, size_t deg>
1085        struct monomial;
1086
1087        template<typename v, typename E = void>
1088        struct derive_helper {};
1089
1090        template<typename v>
1091        struct derive_helper<v, std::enable_if_t<v::degree == 0» {
1092            using type = zero;
1093        };
1094
1095        template<typename v>
1096        struct derive_helper<v, std::enable_if_t<v::degree != 0» {
1097            using type = typename add<
1098                typename derive_helper<typename simplify<typename v::strip>::type>::type,
1099                typename monomial<
1100                typename Ring::template mul_t<
1101                typename v::aN,
1102                typename Ring::template inject_constant_t<(v::degree)>
1103                >,
1104                v::degree - 1
1105                >::type
1106            >::type;
1107        };
1108
1109        template<typename v1, typename v2, typename E = void>
1110        struct eq_helper {};
1111
1112        template<typename v1, typename v2>
1113        struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
1114            static constexpr bool value = false;
1115        };
1116
1117
1118        template<typename v1, typename v2>
1119        struct eq_helper<v1, v2, std::enable_if_t<
1120            v1::degree == v2::degree &&
1121            (v1::degree != 0 || v2::degree != 0) &&
1122            (!Ring::template eq_v<typename v1::aN, typename v2::aN>)
1123            » {
1124            static constexpr bool value = false;
1125        };
1126
1127        template<typename v1, typename v2>
1128        struct eq_helper<v1, v2, std::enable_if_t<
1129            v1::degree == v2::degree &&
1130            (v1::degree != 0 || v2::degree != 0) &&
1131            (Ring::template eq_v<typename v1::aN, typename v2::aN>)
1132            » {
1133            static constexpr bool value = eq_helper<typename v1::strip, typename v2::strip>::value;
1134        };
1135
1136        template<typename v1, typename v2>
1137        struct eq_helper<v1, v2, std::enable_if_t<
1138            v1::degree == v2::degree &&
1139            (v1::degree == 0)
1140            » {
1141            static constexpr bool value = Ring::template eq_v<typename v1::aN, typename v2::aN>;
1142        };
1143
1144        template<typename v1, typename v2, typename E = void>
1145        struct lt_helper {};
1146
```

```
1147        template<typename v1, typename v2>
1148        struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1149            static constexpr bool value = true;
1150        };
1151
1152        template<typename v1, typename v2>
1153        struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1154            static constexpr bool value = Ring::template lt_v<typename v1::aN, typename v2::aN>;
1155        };
1156
1157        template<typename v1, typename v2>
1158        struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1159            static constexpr bool value = false;
1160        };
1161
1162        template<typename v1, typename v2, typename E = void>
1163        struct gt_helper {};
1164
1165        template<typename v1, typename v2>
1166        struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1167            static constexpr bool value = true;
1168        };
1169
1170        template<typename v1, typename v2>
1171        struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1172            static constexpr bool value = Ring::template gt_v<typename v1::aN, typename v2::aN>;
1173        };
1174
1175        template<typename v1, typename v2>
1176        struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1177            static constexpr bool value = false;
1178        };
1179
1180        // when high power is zero :  strip
1181        template<typename P>
1182        struct simplify<P, std::enable_if_t<
1183            std::is_same<
1184            typename Ring::zero,
1185            typename P::aN
1186            >::value && (P::degree > 0)
1187            »
1188        {
1189            using type = typename simplify<typename P::strip>::type;
1190        };
1191
1192        // otherwise :  do nothing
1193        template<typename P>
1194        struct simplify<P, std::enable_if_t<
1195            !std::is_same<
1196            typename Ring::zero,
1197            typename P::aN
1198            >::value && (P::degree > 0)
1199            »
1200        {
1201            using type = P;
1202        };
1203
1204        // do not simplify constants
1205        template<typename P>
1206        struct simplify<P, std::enable_if_t<P::degree == 0» {
1207            using type = P;
1208        };
1209
1210        // addition at
1211        template<typename P1, typename P2, size_t index>
1212        struct add_at {
1213            using type =
1214                typename Ring::template add_t<typename P1::template coeff_at_t<index>, typename
     P2::template coeff_at_t<index»;
1215        };
1216
1217        template<typename P1, typename P2, size_t index>
1218        using add_at_t = typename add_at<P1, P2, index>::type;
1219
1220        template<typename P1, typename P2, std::size_t...  I>
1221        struct add_low<P1, P2, std::index_sequence<I...» {
1222            using type = val<add_at_t<P1, P2, I>...>;
1223        };
1224
1225        // substraction at
1226        template<typename P1, typename P2, size_t index>
1227        struct sub_at {
1228            using type =
1229                typename Ring::template sub_t<typename P1::template coeff_at_t<index>, typename
     P2::template coeff_at_t<index»;
1230        };
1231
```

```
1232         template<typename P1, typename P2, size_t index>
1233         using sub_at_t = typename sub_at<P1, P2, index>::type;
1234 
1235         template<typename P1, typename P2, std::size_t...  I>
1236         struct sub_low<P1, P2, std::index_sequence<I...» {
1237             using type = val<sub_at_t<P1, P2, I>...>;
1238         };
1239 
1240         template<typename P1, typename P2>
1241         struct sub {
1242             using type = typename simplify<typename sub_low<
1243                 P1,
1244                 P2,
1245                 internal::make_index_sequence_reverse<
1246                 std::max(P1::degree, P2::degree) + 1
1247                 »::type>::type;
1248         };
1249 
1250         // multiplication at
1251         template<typename v1, typename v2, size_t k, size_t index, size_t stop>
1252         struct mul_at_loop_helper {
1253             using type = typename Ring::template add_t<
1254                 typename Ring::template mul_t<
1255                 typename v1::template coeff_at_t<index>,
1256                 typename v2::template coeff_at_t<k - index>
1257                 >,
1258                 typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
1259             >;
1260         };
1261 
1262         template<typename v1, typename v2, size_t k, size_t stop>
1263         struct mul_at_loop_helper<v1, v2, k, stop, stop> {
1264             using type = typename Ring::template mul_t<typename v1::template coeff_at_t<stop>, typename
     v2::template coeff_at_t<0»;
1265         };
1266 
1267         template <typename v1, typename v2, size_t k, typename E = void>
1268         struct mul_at {};
1269 
1270         template<typename v1, typename v2, size_t k>
1271         struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)» {
1272             using type = typename Ring::zero;
1273         };
1274 
1275         template<typename v1, typename v2, size_t k>
1276         struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)» {
1277             using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
1278         };
1279 
1280         template<typename P1, typename P2, size_t index>
1281         using mul_at_t = typename mul_at<P1, P2, index>::type;
1282 
1283         template<typename P1, typename P2, std::size_t...  I>
1284         struct mul_low<P1, P2, std::index_sequence<I...» {
1285             using type = val<mul_at_t<P1, P2, I>...>;
1286         };
1287 
1288         // division helper
1289         template< typename A, typename B, typename Q, typename R, typename E = void>
1290         struct div_helper {};
1291 
1292         template<typename A, typename B, typename Q, typename R>
1293         struct div_helper<A, B, Q, R, std::enable_if_t<
1294             (R::degree < B::degree) ||
1295             (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
1296             using q_type = Q;
1297             using mod_type = R;
1298             using gcd_type = B;
1299         };
1300 
1301         template<typename A, typename B, typename Q, typename R>
1302         struct div_helper<A, B, Q, R, std::enable_if_t<
1303             (R::degree >= B::degree) &&
1304             !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
1305         private:
1306             using rN = typename R::aN;
1307             using bN = typename B::aN;
1308             using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
     B::degree>::type;
1309             using rr = typename sub<R, typename mul<pT, B>::type>::type;
1310             using qq = typename add<Q, pT>::type;
1311 
1312         public:
1313             using q_type = typename simplify<typename div_helper<A, B, qq, rr>::q_type>::type;
1314             using mod_type = typename simplify<typename div_helper<A, B, qq, rr>::mod_type>::type;
1315             using gcd_type = typename simplify<rr>::type;
1316         };
```

```
1317
1318        template<typename A, typename B>
1319        struct div {
1320            static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
1321            using q_type = typename div_helper<A, B, zero, A>::q_type;
1322            using m_type = typename div_helper<A, B, zero, A>::mod_type;
1323        };
1324
1325
1326        template<typename P>
1327        struct make_unit {
1328            using type = typename div<P, val<typename P::aN»::q_type;
1329        };
1330
1331        template<typename coeff, size_t deg>
1332        struct monomial {
1333            using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
1334        };
1335
1336        template<typename coeff>
1337        struct monomial<coeff, 0> {
1338            using type = val<coeff>;
1339        };
1340
1341        template<typename valueRing, typename P>
1342        struct eval_helper
1343        {
1344            template<size_t index, size_t stop>
1345            struct inner {
1346                DEVICE INLINED static constexpr valueRing func(const valueRing& accum, const valueRing&
     x) {
1347                    constexpr valueRing coeff = static_cast<valueRing>(P::template coeff_at_t<P::degree
     - index>::template get<valueRing>());
1348                    return eval_helper<valueRing, P>::template inner<index + 1, stop>::func(x * accum +
     coeff, x);
1349                }
1350            };
1351
1352            template<size_t stop>
1353            struct inner<stop, stop> {
1354                DEVICE INLINED static constexpr valueRing func(const valueRing& accum, const valueRing&
     x) {
1355                    return accum;
1356                }
1357            };
1358        };
1359
1360        template<typename coeff, typename...  coeffs>
1361        struct string_helper {
1362            static std::string func() {
1363                std::string tail = string_helper<coeffs...>::func();
1364                std::string result = "";
1365                if (Ring::template eq_v<coeff, typename Ring::zero>) {
1366                    return tail;
1367                }
1368                else if (Ring::template eq_v<coeff, typename Ring::one>) {
1369                    if (sizeof...(coeffs) == 1) {
1370                        result += 'X';
1371                    }
1372                    else {
1373                        result += "X^" + std::to_string(sizeof...(coeffs));
1374                    }
1375                }
1376                else {
1377                    if (sizeof...(coeffs) == 1) {
1378                        result += coeff::to_string() + " X";
1379                    }
1380                    else {
1381                        result += coeff::to_string() + " X^" + std::to_string(sizeof...(coeffs));
1382                    }
1383                }
1384
1385                if (!tail.empty()) {
1386                    result += " + " + tail;
1387                }
1388
1389                return result;
1390            }
1391        };
1392
1393        template<typename coeff>
1394        struct string_helper<coeff> {
1395            static std::string func() {
1396                if (!std::is_same<coeff, typename Ring::zero>::value) {
1397                    return coeff::to_string();
1398                }
1399                else {
```

```
1400                     return "";
1401                 }
1402             }
1403         };
1404 
1405     public:
1408         template<typename P>
1409         using simplify_t = typename simplify<P>::type;
1410 
1414         template<typename v1, typename v2>
1415         using add_t = typename add<v1, v2>::type;
1416 
1420         template<typename v1, typename v2>
1421         using sub_t = typename sub<v1, v2>::type;
1422 
1423         template<typename v1>
1424         using minus_t = sub_t<zero, v1>;
1425 
1429         template<typename v1, typename v2>
1430         using mul_t = typename mul<v1, v2>::type;
1431 
1435         template<typename v1, typename v2>
1436         static constexpr bool eq_v = eq_helper<v1, v2>::value;
1437 
1441         template<typename v1, typename v2>
1442         static constexpr bool lt_v = lt_helper<v1, v2>::value;
1443 
1447         template<typename v1, typename v2>
1448         static constexpr bool gt_v = gt_helper<v1, v2>::value;
1449 
1453         template<typename v1, typename v2>
1454         using div_t = typename div<v1, v2>::q_type;
1455 
1459         template<typename v1, typename v2>
1460         using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
1461 
1465         template<typename coeff, size_t deg>
1466         using monomial_t = typename monomial<coeff, deg>::type;
1467 
1470         template<typename v>
1471         using derive_t = typename derive_helper<v>::type;
1472 
1475         template<typename v>
1476         static constexpr bool pos_v = Ring::template pos_v<typename v::aN>;
1477 
1481         template<typename v1, typename v2>
1482         using gcd_t = std::conditional_t<
1483             Ring::is_euclidean_domain,
1484             typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
1485             void>;
1486 
1490         template<auto x>
1491         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
1492 
1496         template<typename v>
1497         using inject_ring_t = val<v>;
1498     };
1499 }
1500 
1501 // big integers
1502 namespace aerobus {
1503     struct bigint {
1504         enum signs {
1505             positive,
1506             negative
1507         };
1508 
1509         template<signs s, uint32_t an, uint32_t...  as>
1510         struct val;
1511 
1512         template<uint32_t an, uint32_t...  as>
1513         struct to_hex_helper {
1514             static std::string func(const bool prefix = false) {
1515                 std::string head = prefix ?  std::format("{:08X}", an) :  std::format("{:X}", an);
1516                 return head + to_hex_helper<as...>::func(true);
1517             }
1518         };
1519 
1520         template<uint32_t x>
1521         struct to_hex_helper<x> {
1522             static std::string func(const bool prefix = false) {
1523                 return prefix ?  std::format("{:08X}", x) :  std::format("{:X}", x);
1524             }
1525         };
1526 
1527     private:
1528 
```

```
1529         template<signs s>
1530         struct opposite {
1531             static constexpr signs value = s == signs::positive ?  signs::negative :  signs::positive;
1532         };
1533 
1534         template<signs s>
1535         static constexpr signs opposite_v = opposite<s>::value;
1536 
1537         static std::string to_string(const signs& s) {
1538             switch (s) {
1539             case signs::negative:
1540                 return "-";
1541             case signs::positive:
1542             default:
1543                 return "+";
1544             }
1545         }
1546 
1547         template<signs s1, signs s2>
1548         static constexpr signs mul_sign() {
1549             if constexpr (s1 == signs::positive) {
1550                 return s2;
1551             }
1552 
1553             return opposite_v<s2>;
1554         }
1555 
1556         template<size_t ss, signs s, uint32_t aN, uint32_t...  as>
1557         struct shift_left_helper {
1558             using type = typename shift_left_helper<ss - 1, s, aN, as..., 0>::type;
1559         };
1560 
1561         template<signs s, uint32_t aN, uint32_t...  as>
1562         struct shift_left_helper<0, s, aN, as...>
1563         {
1564             using type = val<s, aN, as...>;
1565         };
1566 
1567     public:
1568 
1569         template<signs s, uint32_t an, uint32_t...  as>
1570         struct val {
1571             template<size_t ss>
1572             using shift_left = typename shift_left_helper<ss, s, an, as...>::type;
1573             static constexpr signs sign = s;
1574 
1575             template<size_t index, typename E = void>
1576             struct digit_at {};
1577 
1578             template<size_t index>
1579             struct digit_at<index, std::enable_if_t<(index <= sizeof...(as))>> {
1580                 static constexpr uint32_t value = internal::value_at<(sizeof...(as) - index), an,
     as...>::value;
1581             };
1582 
1583             template<size_t index>
1584             struct digit_at<index, std::enable_if_t<(index > sizeof...(as))>> {
1585                 static constexpr uint32_t value = 0;
1586             };
1587 
1588             using strip = val<s, as...>;
1589             static constexpr uint32_t aN = an;
1590             static constexpr size_t digits = sizeof...(as) + 1;
1591 
1592             static std::string to_string() {
1593                 return bigint::to_string(s) + std::to_string(aN) + "B^" + std::to_string(digits - 1) +
     " + " + strip::to_string();
1594             }
1595 
1596             static std::string to_hex() {
1597                 return bigint::to_string(s) + "0X" + to_hex_helper<an, as...>::func(false);
1598             }
1599 
1600             static constexpr bool is_zero_v = sizeof...(as) == 0 && an == 0;
1601 
1602             using minus_t = val<opposite_v<s>, an, as...>;
1603         };
1604 
1605         template<signs s, uint32_t a0>
1606         struct val<s, a0> {
1607             template<size_t ss>
1608             using shift_left = typename shift_left_helper<ss, s, a0>::type;
1609             static constexpr signs sign = s;
1610             static constexpr uint32_t aN = a0;
1611             static constexpr size_t digits = 1;
1612             template<size_t index, typename E = void>
1613             struct digit_at {};
```

```
1614            template<size_t index>
1615            struct digit_at<index, std::enable_if_t<index == 0» {
1616                static constexpr uint32_t value = a0;
1617            };
1618
1619            template<size_t index>
1620            struct digit_at<index, std::enable_if_t<index != 0» {
1621                static constexpr uint32_t value = 0;
1622            };
1623
1624            static std::string to_string() {
1625                return bigint::to_string(s) + std::to_string(a0);
1626            }
1627
1628            static std::string to_hex() {
1629                return bigint::to_string(s) + std::format("0X{:X}", a0);
1630            }
1631
1632            static constexpr bool is_zero_v = a0 == 0;
1633
1634            using minus_t = val<opposite_v<s>, a0>;
1635
1636        };
1637
1638        using zero = val<signs::positive, 0>;
1639        using one = val<signs::positive, 1>;
1640
1641    private:
1642
1643        template<typename I, typename E = void>
1644        struct simplify {};
1645
1646        template<typename I>
1647        struct simplify<I, std::enable_if_t<I::digits == 1 && I::aN != 0» {
1648            using type = I;
1649        };
1650
1651        template<typename I>
1652        struct simplify<I, std::enable_if_t<I::digits == 1 && I::aN == 0» {
1653            using type = zero;
1654        };
1655
1656        template<typename I>
1657        struct simplify<I, std::enable_if_t<I::digits != 1 && I::aN == 0» {
1658            using type = typename simplify<typename I::strip>::type;
1659        };
1660
1661        template<typename I>
1662        struct simplify<I, std::enable_if_t<I::digits != 1 && I::aN != 0» {
1663            using type = I;
1664        };
1665
1666        template<uint32_t x, uint32_t y, uint8_t carry_in = 0>
1667        struct add_digit_helper {
1668        private:
1669            static constexpr uint64_t raw = ((uint64_t)x + (uint64_t)y + (uint64_t)carry_in);
1670        public:
1671            static constexpr uint32_t value = (uint32_t)(raw & 0xFFFF'FFFF);
1672            static constexpr uint8_t carry_out = (uint32_t)(raw » 32);
1673        };
1674
1675        template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1676        struct add_at_helper {
1677        private:
1678            static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1679            static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1680        public:
1681            static constexpr uint32_t value = add_digit_helper<d1, d2, carry_in>::value;
1682            static constexpr uint8_t carry_out = add_digit_helper<d1, d2, carry_in>::carry_out;
1683        };
1684
1685        template<uint32_t x, uint32_t y, uint8_t carry_in, typename E = void>
1686        struct sub_digit_helper {};
1687
1688        // x - y
1689        template<uint32_t x, uint32_t y, uint8_t carry_in>
1690        struct sub_digit_helper<x, y, carry_in, std::enable_if_t<
1691            (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in) > x)
1692            » {
1693
1694            static constexpr uint32_t value = static_cast<uint32_t>(
1695                static_cast<uint32_t>(x) + 0x1'0000'0000UL - (static_cast<uint64_t>(y) +
    static_cast<uint64_t>(carry_in))
1696                );
1697            static constexpr uint8_t carry_out = 1;
1698        };
1699
```

```
1700         template<uint32_t x, uint32_t y, uint8_t carry_in>
1701         struct sub_digit_helper<x, y, carry_in, std::enable_if_t<
1702             (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in) <= x)
1703             » {
1704 
1705             static constexpr uint32_t value = static_cast<uint32_t>(
1706                 static_cast<uint64_t>(x) - (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in))
1707                 );
1708             static constexpr uint8_t carry_out = 0;
1709         };
1710 
1711         template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1712         struct sub_at_helper {
1713         private:
1714             static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1715             static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1716             using tmp = sub_digit_helper<d1, d2, carry_in>;
1717         public:
1718             static constexpr uint32_t value = tmp::value;
1719             static constexpr uint8_t carry_out = tmp::carry_out;
1720         };
1721 
1722         template<uint32_t x, uint32_t y, uint32_t carry_in>
1723         struct mul_digit_helper {
1724         private:
1725             static constexpr uint64_t tmp = static_cast<uint64_t>(x) * static_cast<uint64_t>(y) +
     static_cast<uint64_t>(carry_in);
1726         public:
1727             static constexpr uint32_t value = static_cast<uint32_t>(tmp & 0xFFFF'FFFFU);
1728             static constexpr uint32_t carry_out = static_cast<uint32_t>(tmp » 32);
1729         };
1730 
1731         template<typename I1, uint32_t d2, size_t index, uint32_t carry_in = 0>
1732         struct mul_at_helper {
1733         private:
1734             static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1735             using tmp = mul_digit_helper<d1, d2, carry_in>;
1736         public:
1737             static constexpr uint32_t value = tmp::value;
1738             static constexpr uint32_t carry_out = tmp::carry_out;
1739         };
1740 
1741         template<typename I1, typename I2, size_t index>
1742         struct add_low_helper {
1743         private:
1744             using helper = add_at_helper<I1, I2, index, add_low_helper<I1, I2, index - 1>::carry_out>;
1745         public:
1746             static constexpr uint32_t digit = helper::value;
1747             static constexpr uint8_t carry_out = helper::carry_out;
1748         };
1749 
1750         template<typename I1, typename I2>
1751         struct add_low_helper<I1, I2, 0> {
1752             static constexpr uint32_t digit = add_at_helper<I1, I2, 0, 0>::value;
1753             static constexpr uint32_t carry_out = add_at_helper<I1, I2, 0, 0>::carry_out;
1754         };
1755 
1756         template<typename I1, typename I2, size_t index>
1757         struct sub_low_helper {
1758         private:
1759             using helper = sub_at_helper<I1, I2, index, sub_low_helper<I1, I2, index - 1>::carry_out>;
1760         public:
1761             static constexpr uint32_t digit = helper::value;
1762             static constexpr uint8_t carry_out = helper::carry_out;
1763         };
1764 
1765         template<typename I1, typename I2>
1766         struct sub_low_helper<I1, I2, 0> {
1767             static constexpr uint32_t digit = sub_at_helper<I1, I2, 0, 0>::value;
1768             static constexpr uint32_t carry_out = sub_at_helper<I1, I2, 0, 0>::carry_out;
1769         };
1770 
1771         template<typename I1, uint32_t d2, size_t index>
1772         struct mul_low_helper {
1773         private:
1774             using helper = mul_at_helper<I1, d2, index, mul_low_helper<I1, d2, index - 1>::carry_out>;
1775         public:
1776             static constexpr uint32_t digit = helper::value;
1777             static constexpr uint32_t carry_out = helper::carry_out;
1778         };
1779 
1780         template<typename I1, uint32_t d2>
1781         struct mul_low_helper<I1, d2, 0> {
1782             static constexpr uint32_t digit = mul_at_helper<I1, d2, 0, 0>::value;
1783             static constexpr uint32_t carry_out = mul_at_helper<I1, d2, 0, 0>::carry_out;
1784         };
1785 
```

```
1786         template<typename I1, uint32_t d2, typename I>
1787         struct mul_low {};
1788 
1789         template<typename I1, uint32_t d2, std::size_t...  I>
1790         struct mul_low<I1, d2, std::index_sequence<I...» {
1791             using type = val<signs::positive, mul_low_helper<I1, d2, I>::digit...>;
1792         };
1793 
1794         template<typename I1, uint32_t d2, size_t shift>
1795         struct mul_row_helper {
1796             using type = typename simplify<
1797                 typename mul_low<
1798                 I1,
1799                 d2,
1800                 typename internal::make_index_sequence_reverse<I1::digits + 1>
1801                 >::type>::type::template shift_left<shift>;
1802         };
1803 
1804         template<typename I1, typename I2, size_t index>
1805         struct mul_row {
1806         private:
1807             static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1808         public:
1809             using type = typename mul_row_helper<I1, d2, index>::type;
1810         };
1811 
1812         template<typename I1, typename...  Is>
1813         struct vadd;
1814 
1815         template<typename I1, typename I2, typename E = void>
1816         struct eq;
1817 
1818         template<typename I1, typename I2, typename I>
1819         struct mul_helper {};
1820 
1821         template<typename I1, typename I2, std::size_t...  I>
1822         struct mul_helper<I1, I2, std::index_sequence<I...» {
1823             using type = typename vadd<typename mul_row<I1, I2, I>::type...>::type;
1824         };
1825 
1826         template<typename I, size_t index>
1827         struct div_2_digit {
1828             static constexpr uint32_t value = ((I::template digit_at<index + 1>::value & 1) « 31) |
     (I::template digit_at<index>::value » 1);
1829         };
1830 
1831         template<typename X, typename I>
1832         struct div_2_helper {};
1833 
1834         template<typename X, std::size_t...  I>
1835         struct div_2_helper<X, std::index_sequence<I...» {
1836             using type = val<signs::positive, div_2_digit<X, I>::value...>;
1837         };
1838 
1839         template<typename X, typename E = void>
1840         struct div_2 {};
1841 
1842         template<typename X>
1843         struct div_2<X, std::enable_if_t<X::digits == 1» {
1844             using type = val<X::sign, (X::aN » 1)>;
1845         };
1846 
1847         template<typename X>
1848         struct div_2<X, std::enable_if_t<X::digits != 1» {
1849             using type = typename simplify<typename div_2_helper<X,
     internal::make_index_sequence_reverse<X::digits»::type>::type;
1850         };
1851 
1852         template<typename I1, typename I2, typename E = void>
1853         struct mul {};
1854 
1855         template<typename I1, typename I2>
1856         struct mul<I1, I2, std::enable_if_t<
1857             I1::is_zero_v || I2::is_zero_v
1858             » {
1859             using type = zero;
1860         };
1861 
1862         template<typename I1, typename I2>
1863         struct mul<I1, I2, std::enable_if_t<
1864             !I1::is_zero_v && !I2::is_zero_v&& eq<I1, one>::value
1865             » {
1866             using type = I2;
1867         };
1868 
1869         template<typename I1, typename I2>
1870         struct mul<I1, I2, std::enable_if_t<
```

```
1871            !I1::is_zero_v && !I2::is_zero_v && !eq<I1, one>::value&& eq<I2, one>::value
1872            » {
1873            using type = I1;
1874        };
1875
1876        template<typename I1, typename I2>
1877        struct mul<I1, I2, std::enable_if_t<
1878            !I1::is_zero_v && !I2::is_zero_v && !eq<I1, one>::value && !eq<I2, one>::value
1879            » {
1880        private:
1881            static constexpr signs sign = mul_sign<I1::sign, I2::sign>();
1882            using tmp =
1883                typename simplify<
1884                typename mul_helper<I1, I2, internal::make_index_sequence_reverse<I1::digits*
    I2::digits + 1»::type
1885                >::type;
1886        public:
1887            using type = std::conditional_t<sign == signs::positive, tmp, typename tmp::minus_t>;
1888        };
1889
1890        template<typename I1, typename I2, typename I>
1891        struct add_low {};
1892
1893        template<typename I1, typename I2, std::size_t...  I>
1894        struct add_low<I1, I2, std::index_sequence<I...» {
1895            using type = val<signs::positive, add_low_helper<I1, I2, I>::digit...>;
1896        };
1897
1898        template<typename I1, typename I2, typename I>
1899        struct sub_low {};
1900
1901        template<typename I1, typename I2, std::size_t...  I>
1902        struct sub_low<I1, I2, std::index_sequence<I...» {
1903            using type = val<signs::positive, sub_low_helper<I1, I2, I>::digit...>;
1904        };
1905
1906        template<typename I1, typename I2, typename E>
1907        struct eq {};
1908
1909        template<typename I1, typename I2>
1910        struct eq<I1, I2, std::enable_if_t<I1::digits != I2::digits» {
1911            static constexpr bool value = false;
1912        };
1913
1914        template<typename I1, typename I2>
1915        struct eq<I1, I2, std::enable_if_t<I1::digits == I2::digits && I1::digits == 1» {
1916            static constexpr bool value = (I1::is_zero_v && I2::is_zero_v) || (I1::sign == I2::sign &&
    I1::aN == I2::aN);
1917        };
1918
1919        template<typename I1, typename I2>
1920        struct eq<I1, I2, std::enable_if_t<I1::digits == I2::digits && I1::digits != 1» {
1921            static constexpr bool value =
1922                I1::sign == I2::sign &&
1923                I1::aN == I2::aN &&
1924                eq<typename I1::strip, typename I2::strip>::value;
1925        };
1926
1927        template<typename I1, typename I2, typename E = void>
1928        struct gt_helper {};
1929
1930        template<typename I1, typename I2>
1931        struct gt_helper<I1, I2, std::enable_if_t<eq<I1, I2>::value» {
1932            static constexpr bool value = false;
1933        };
1934
1935        template<typename I1, typename I2>
1936        struct gt_helper<I1, I2, std::enable_if_t<!eq<I1, I2>::value&& I1::sign != I2::sign» {
1937            static constexpr bool value = I1::sign == signs::positive;
1938        };
1939
1940        template<typename I1, typename I2>
1941        struct gt_helper<I1, I2,
1942            std::enable_if_t<
1943            !eq<I1, I2>::value&&
1944            I1::sign == I2::sign &&
1945            I1::sign == signs::negative
1946            » {
1947            static constexpr bool value = gt_helper<typename I2::minus_t, typename I1::minus_t>::value;
1948        };
1949
1950        template<typename I1, typename I2>
1951        struct gt_helper<I1, I2,
1952            std::enable_if_t<
1953            !eq<I1, I2>::value&&
1954            I1::sign == I2::sign &&
1955            I1::sign == signs::positive &&
```

```
1956            (I1::digits > I2::digits)
1957            » {
1958            static constexpr bool value = true;
1959        };
1960
1961        template<typename I1, typename I2>
1962        struct gt_helper<I1, I2,
1963            std::enable_if_t<
1964            !eq<I1, I2>::value&&
1965            I1::sign == I2::sign &&
1966            I1::sign == signs::positive &&
1967            (I1::digits < I2::digits)
1968            » {
1969            static constexpr bool value = false;
1970        };
1971
1972        template<typename I1, typename I2>
1973        struct gt_helper<I1, I2,
1974            std::enable_if_t<
1975            !eq<I1, I2>::value&&
1976            I1::sign == I2::sign &&
1977            I1::sign == signs::positive &&
1978            (I1::digits == I2::digits) && I1::digits == 1
1979            » {
1980            static constexpr bool value = I1::aN > I2::aN;
1981        };
1982
1983        template<typename I1, typename I2>
1984        struct gt_helper<I1, I2,
1985            std::enable_if_t<
1986            !eq<I1, I2>::value&&
1987            I1::sign == I2::sign &&
1988            I1::sign == signs::positive &&
1989            (I1::digits == I2::digits) && I1::digits != 1 && (I1::aN > I2::aN)
1990            » {
1991            static constexpr bool value = true;
1992        };
1993
1994        template<typename I1, typename I2>
1995        struct gt_helper<I1, I2,
1996            std::enable_if_t<
1997            !eq<I1, I2>::value&&
1998            I1::sign == I2::sign &&
1999            I1::sign == signs::positive &&
2000            (I1::digits == I2::digits) && I1::digits != 1 && (I1::aN < I2::aN)
2001            » {
2002            static constexpr bool value = false;
2003        };
2004
2005        template<typename I1, typename I2>
2006        struct gt_helper<I1, I2,
2007            std::enable_if_t<
2008            !eq<I1, I2>::value&&
2009            I1::sign == I2::sign &&
2010            I1::sign == signs::positive &&
2011            (I1::digits == I2::digits) && I1::digits != 1 && I1::aN == I2::aN
2012            » {
2013            static constexpr bool value = gt_helper<typename I1::strip, typename I2::strip>::value;
2014        };
2015
2016
2017
2018        template<typename I1, typename I2, typename E = void>
2019        struct add {};
2020
2021        template<typename I1, typename I2, typename E = void>
2022        struct sub {};
2023
2024
2025
2026        // 0 + x -> x
2027        template<typename I1, typename I2>
2028        struct add<I1, I2, std::enable_if_t<
2029            I1::is_zero_v && !I2::is_zero_v
2030            » {
2031            using type = I2;
2032        };
2033
2034        // x + 0 -> x
2035        template<typename I1, typename I2>
2036        struct add<I1, I2, std::enable_if_t<
2037            I2::is_zero_v && !I1::is_zero_v
2038            » {
2039            using type = I1;
2040        };
2041
2042        // 0 + 0 -> x
```

```
2043        template<typename I1, typename I2>
2044        struct add<I1, I2, std::enable_if_t<
2045            I2::is_zero_v && I1::is_zero_v
2046            » {
2047            using type = zero;
2048        };
2049
2050        // +x + +y -> x + y
2051        template<typename I1, typename I2>
2052        struct add<I1, I2, std::enable_if_t<
2053            !I2::is_zero_v && !I1::is_zero_v &&
2054            gt_helper<I1, zero>::value &&
2055            gt_helper<I2, zero>::value
2056            » {
2057            using type = typename simplify<
2058                typename add_low<
2059                I1,
2060                I2,
2061                typename internal::make_index_sequence_reverse<std::max(I1::digits, I2::digits) + 1>
2062                >::type>::type;
2063        };
2064
2065        // -x + -y -> -(x+y)
2066        template<typename I1, typename I2>
2067        struct add<I1, I2, std::enable_if_t<
2068            !I2::is_zero_v && !I1::is_zero_v &&
2069            gt_helper<zero, I1>::value &&
2070            gt_helper<zero, I2>::value
2071            » {
2072            using type = typename add<typename I1::minus_t, typename I2::minus_t>::type::minus_t;
2073        };
2074
2075        // x + (-y) -> x - y
2076        template<typename I1, typename I2>
2077        struct add<I1, I2, std::enable_if_t<
2078            !I1::is_zero_v && !I2::is_zero_v &&
2079            gt_helper<I1, zero>::value&&
2080            gt_helper<zero, I2>::value
2081            » {
2082            using type = typename sub<I1, typename I2::minus_t>::type;
2083        };
2084
2085        // -x + y -> y - x
2086        template<typename I1, typename I2>
2087        struct add<I1, I2, std::enable_if_t<
2088            !I1::is_zero_v && !I2::is_zero_v &&
2089            gt_helper<zero, I1>::value&&
2090            gt_helper<I2, zero>::value
2091            » {
2092            using type = typename sub<I2, typename I1::minus_t>::type;
2093        };
2094
2095        // I1 == I2
2096        template<typename I1, typename I2>
2097        struct sub<I1, I2, std::enable_if_t<
2098            eq<I1, I2>::value
2099            » {
2100            using type = zero;
2101        };
2102
2103        // I1 != I2, I2 == 0
2104        template<typename I1, typename I2>
2105        struct sub<I1, I2, std::enable_if_t<
2106            !eq<I1, I2>::value&&
2107            eq<I2, zero>::value
2108            » {
2109            using type = I1;
2110        };
2111
2112        // I1 != I2, I1 == 0
2113        template<typename I1, typename I2>
2114        struct sub<I1, I2, std::enable_if_t<
2115            !eq<I1, I2>::value&&
2116            eq<I1, zero>::value
2117            » {
2118            using type = typename I2::minus_t;
2119        };
2120
2121        // 0 < I2 < I1
2122        template<typename I1, typename I2>
2123        struct sub<I1, I2, std::enable_if_t<
2124            gt_helper<I2, zero>::value&&
2125            gt_helper<I1, I2>::value
2126            » {
2127            using type = typename simplify<
2128                typename sub_low<
2129                I1,
```

```
2130                I2,
2131                typename internal::make_index_sequence_reverse<std::max(I1::digits, I2::digits) + 1>
2132                >::type>::type;
2133        };
2134
2135        // 0 < I1 < I2
2136        template<typename I1, typename I2>
2137        struct sub<I1, I2, std::enable_if_t<
2138            gt_helper<I1, zero>::value&&
2139            gt_helper<I2, I1>::value
2140            » {
2141            using type = typename sub<I2, I1>::type::minus_t;
2142        };
2143
2144        // I2 < I1 < 0
2145        template<typename I1, typename I2>
2146        struct sub<I1, I2, std::enable_if_t<
2147            gt_helper<zero, I1>::value&&
2148            gt_helper<I1, I2>::value
2149            » {
2150            using type = typename sub<typename I2::minus_t, typename I1::minus_t>::type;
2151        };
2152
2153        // I1 < I2 < 0
2154        template<typename I1, typename I2>
2155        struct sub<I1, I2, std::enable_if_t<
2156            gt_helper<zero, I2>::value&&
2157            gt_helper<I2, I1>::value
2158            » {
2159            using type = typename sub<typename I1::minus_t, typename I2::minus_t>::type::minus_t;
2160        };
2161
2162        // I2 < 0 < I1
2163        template<typename I1, typename I2>
2164        struct sub<I1, I2, std::enable_if_t<
2165            gt_helper<zero, I2>::value&&
2166            gt_helper<I1, zero>::value
2167            » {
2168            using type = typename add<I1, typename I2::minus_t>::type;
2169        };
2170
2171        // I1 < 0 < I2
2172        template<typename I1, typename I2>
2173        struct sub<I1, I2, std::enable_if_t<
2174            gt_helper<zero, I1>::value&&
2175            gt_helper<I2, zero>::value
2176            » {
2177            using type = typename add<I2, typename I1::minus_t>::type::minus_t;
2178        };
2179
2180        // useful for multiplication
2181        template<typename I1, typename...  Is>
2182        struct vadd {
2183            using type = typename add<I1, typename vadd<Is...>::type>::type;
2184        };
2185
2186        template<typename I1, typename I2>
2187        struct vadd<I1, I2> {
2188            using type = typename add<I1, I2>::type;
2189        };
2190
2191        template<typename I, size_t s, typename E = void>
2192        struct shift_right_helper { };
2193
2194        template<typename I, size_t s>
2195        struct shift_right_helper<I, s, std::enable_if_t<(s >= I::digits)» {
2196            using type = zero;
2197        };
2198
2199        template<typename I, size_t s>
2200        struct shift_right_helper<I, s, std::enable_if_t<(s == 0)» {
2201            using type = I;
2202        };
2203
2204        template<typename I, size_t s>
2205        struct shift_right_helper<I, s, std::enable_if_t<(s != 0) && (s < I::digits)» {
2206        private:
2207            using digit = val<I::sign, I::template digit_at<s>::value>;
2208            using tmp = typename shift_right_helper<I, s + 1>::type;
2209        public:
2210            using type = typename add<
2211                digit,
2212                typename tmp::template shift_left<1>
2213            >::type;
2214        };
2215
2216        template<typename A, typename B, typename E = void>
```

```
2217        struct floor_helper {};
2218
2219        template<typename A, typename B>
2220        struct floor_helper<A, B, std::enable_if_t<gt_helper<B, A>::value» {
2221            using type = zero;
2222        };
2223
2224        template<typename A, typename B>
2225        struct floor_helper<A, B, std::enable_if_t<eq<A, B>::value» {
2226            using type = one;
2227        };
2228
2229        template<typename A, typename B>
2230        struct floor_helper<A, B, std::enable_if_t<gt_helper<A, B>::value && (A::digits == 1 &&
    B::digits == 1)» {
2231            using type = val<signs::positive, A::aN / B::aN>;
2232        };
2233
2234        template<typename A, typename B>
2235        struct floor_helper<A, B, std::enable_if_t<gt_helper<A, B>::value && (A::digits != 1 ||
    B::digits != 1)» {
2236            template<typename X, typename Y>
2237            using average_t = typename div_2<typename add<X, Y>::type>::type;
2238
2239            template<typename lowerbound, typename upperbound, typename E = void>
2240            struct inner {};
2241
2242            template<typename lowerbound, typename upperbound>
2243            struct inner<lowerbound, upperbound, std::enable_if_t<eq<
2244                typename add<lowerbound, one>::type, upperbound>::value
2245                » {
2246                using type = lowerbound;
2247            };
2248
2249            template<typename lowerbound, typename upperbound>
2250            struct inner<lowerbound, upperbound, std::enable_if_t<
2251                gt_helper<upperbound, typename add<lowerbound, one>::type>::value&&
2252                gt_helper<typename mul<average_t<upperbound, lowerbound>, B>::type, A>::value
2253                » {
2254                using type = typename simplify<typename inner<lowerbound, average_t<upperbound,
    lowerbound»::type>::type;
2255            };
2256
2257            template<typename lowerbound, typename upperbound>
2258            struct inner<lowerbound, upperbound, std::enable_if_t<
2259                gt_helper<upperbound, typename add<lowerbound, one>::type>::value &&
2260                !gt_helper<typename mul<average_t<upperbound, lowerbound>, B>::type, A>::value
2261                » {
2262                using type = typename simplify<typename inner<average_t<upperbound, lowerbound>,
    upperbound>::type>::type;
2263            };
2264
2265            // this type is ONLY used for division where we know this bound
2266            using type = typename inner<zero, val<signs::positive, 1, 1»::type;
2267        };
2268
2269        template<typename N, typename M, int64_t i>
2270        struct div_helper_inner {
2271            static_assert(N::sign == signs::positive);
2272            static_assert(M::sign == signs::positive);
2273            static constexpr size_t l = M::digits;
2274            static constexpr size_t k = N::digits;
2275            using Qm1 = typename simplify<typename div_helper_inner<N, M, i - 1>::Q>::type;
2276            using Rm1 = typename simplify<typename div_helper_inner<N, M, i - 1>::R>::type;
2277            using D = typename simplify<typename add<
2278                typename Rm1::template shift_left<1>,
2279                val<signs::positive, N::template digit_at<k - (i + 1)>::value>
2280            >::type>::type;
2281            using Beta = typename simplify<typename floor_helper<D, M>::type>::type;
2282            using Q = typename simplify<typename add<typename Qm1::template shift_left<1>,
    Beta>::type>::type;
2283
2284            using R = typename simplify<typename sub<D, typename mul<M, Beta>::type>::type>::type;
2285        };
2286
2287        template<typename N, typename M>
2288        struct div_helper_inner<N, M, -1> {
2289            static_assert(N::sign == signs::positive);
2290            static_assert(M::sign == signs::positive);
2291            static constexpr size_t l = M::digits;
2292            static constexpr size_t k = N::digits;
2293            using Q = zero;
2294            using R = typename shift_right_helper<N, k - l + 1>::type; // first l-1 digits of N
2295        };
2296
2297        template<typename N, typename M, typename E = void>
2298        struct div_helper {};
```

```
2299
2300        template<typename N, typename M>
2301        struct div_helper<N, M, std::enable_if_t<
2302            M::sign == signs::positive &&
2303            N::sign == signs::positive &&
2304            !M::is_zero_v
2305            » {
2306            static constexpr size_t l = M::digits;
2307            static constexpr size_t k = N::digits;
2308            using Q = typename simplify<typename div_helper_inner<N, M, k - l>::Q>::type;
2309            using R = typename simplify<typename div_helper_inner<N, M, k - l>::R>::type;
2310        };
2311
2312        template<typename N, typename M>
2313        struct div_helper<N, M, std::enable_if_t<
2314            M::sign == signs::negative &&
2315            !M::is_zero_v
2316            » {
2317            using tmp = div_helper<N, typename M::minus_t>;
2318            using Q = typename tmp::Q::minus_t;
2319            using R = typename tmp::R;
2320        };
2321
2322        template<typename N, typename M>
2323        struct div_helper<N, M, std::enable_if_t<
2324            N::sign == signs::negative &&
2325            !M::is_zero_v
2326            » {
2327            using tmp = div_helper<typename N::minus_t, M>;
2328            using R_i = typename simplify<typename tmp::R>::type;
2329            using Q_i = typename simplify<typename tmp::Q>::type;
2330            using Q = std::conditional_t<R_i::is_zero_v, typename Q_i::minus_t, typename sub<typename
      Q_i::minus_t, one>::type>;
2331            using R = std::conditional_t<R_i::is_zero_v, zero, typename sub<M, R_i>::type>;
2332        };
2333
2334        template<string_literal S>
2335        struct digit_from_string {
2336            static constexpr size_t N = S.len();
2337
2338            template<size_t i>
2339            static constexpr char char_at = (i < N) ? S.template char_at<i>() :  '0';
2340
2341            template <char c>
2342            static constexpr uint32_t from_hex = (c >= '0' && c <= '9') ?  c - '0' :  10 + c - 'A';
2343
2344            template<size_t index>
2345            static constexpr uint32_t value() {
2346                constexpr uint32_t d1 = from_hex<char_at<8 * index + 1»;
2347                constexpr uint32_t d2 = from_hex<char_at<8 * index + 2» « 4;
2348                constexpr uint32_t d3 = from_hex<char_at<8 * index + 3» « 8;
2349                constexpr uint32_t d4 = from_hex<char_at<8 * index + 4» « 12;
2350                constexpr uint32_t d5 = from_hex<char_at<8 * index + 5» « 16;
2351                constexpr uint32_t d6 = from_hex<char_at<8 * index + 6» « 20;
2352                constexpr uint32_t d7 = from_hex<char_at<8 * index + 7» « 24;
2353                constexpr uint32_t d8 = from_hex<char_at<8 * index + 8» « 28;
2354                return d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8;
2355            }
2356        };
2357
2358        template<string_literal S, typename I>
2359        struct from_hex_helper {};
2360
2361        template<string_literal S, std::size_t...  I>
2362        struct from_hex_helper<S, std::index_sequence<I...» {
2363            using type = typename simplify<val<signs::positive, digit_from_string<S>::template
      value<I>()...»::type;
2364        };
2365
2366    public:
2367        static constexpr bool is_euclidean_domain = true;
2368        static constexpr bool is_field = false;
2369
2370        template<typename v>
2371        using inject_ring_t = v;
2372
2373        template<auto v>
2374        using inject_constant_t = val<(v < 0) ?  bigint::signs::negative :  bigint::signs::positive, (v
      >= 0 ?  v :  -v)>;
2375
2378        template<string_literal S>
2379        using from_hex_t = typename from_hex_helper<S, internal::make_index_sequence_reverse<(S.len() -
      1) / 8 + 1»::type;
2380
2382        template<typename I>
2383        using minus_t = typename I::minus_t;
2384
```

```
2386         template<typename I1, typename I2>
2387         static constexpr bool eq_v = eq<I1, I2>::value;
2388
2390         template<typename I>
2391         static constexpr bool pos_v = I::sign == signs::positive && !I::is_zero_v;
2392
2394         template<typename I1, typename I2>
2395         static constexpr bool gt_v = gt_helper<I1, I2>::value;
2396
2398         template<typename I1, typename I2>
2399         static constexpr bool ge_v = eq_v<I1, I2> || gt_v<I1, I2>;
2400
2402         template<typename I>
2403         using simplify_t = typename simplify<I>::type;
2404
2406         template<typename I1, typename I2>
2407         using add_t = typename add<I1, I2>::type;
2408
2410         template<typename I1, typename I2>
2411         using sub_t = typename sub<I1, I2>::type;
2412
2414         template<typename I, size_t s>
2415         using shift_left_t = typename I::template shift_left<s>;
2416
2418         template<typename I, size_t s>
2419         using shift_right_t = typename shift_right_helper<I, s>::type;
2420
2422         template<typename I1, typename I2>
2423         using mul_t = typename mul<I1, I2>::type;
2424
2426         template<typename...  Is>
2427         using vadd_t = typename vadd<Is...>::type;
2428
2430         template<typename I>
2431         using div_2_t = typename div_2<I>::type;
2432
2434         template<typename I1, typename I2>
2435         using div_t = typename div_helper<I1, I2>::Q;
2436
2438         template<typename I1, typename I2>
2439         using mod_t = typename div_helper<I1, I2>::R;
2440
2442         template<typename I1, typename I2>
2443         using gcd_t = gcd_t<bigint, I1, I2>;
2444
2446         template<typename I1, typename I2, typename I3>
2447         using fma_t = add_t<mul_t<I1, I2>, I3>;
2448
2449
2450     };
2451 }
2452
2453 // fraction field
2454 namespace aerobus {
2455     namespace internal {
2456         template<typename Ring, typename E = void>
2457             requires IsEuclideanDomain<Ring>
2458         struct _FractionField {};
2459
2460         template<typename Ring>
2461             requires IsEuclideanDomain<Ring>
2462         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain>>
2463         {
2465             static constexpr bool is_field = true;
2466             static constexpr bool is_euclidean_domain = true;
2467
2468         private:
2469             template<typename val1, typename val2, typename E = void>
2470             struct to_string_helper {};
2471
2472             template<typename val1, typename val2>
2473             struct to_string_helper <val1, val2,
2474                 std::enable_if_t<
2475                 Ring::template eq_v<val2, typename Ring::one>
2476                 >> {
2477                 static std::string func() {
2478                     return val1::to_string();
2479                 }
2480             };
2481
2482             template<typename val1, typename val2>
2483             struct to_string_helper<val1, val2,
2484                 std::enable_if_t<
2485                 !Ring::template eq_v<val2, typename Ring::one>
2486                 >> {
2487                 static std::string func() {
2488                     return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
```

```
2489                }
2490            };
2491
2492        public:
2496            template<typename val1, typename val2>
2497            struct val {
2498                using x = val1;
2499                using y = val2;
2500
2502                static constexpr bool is_zero_v = val1::is_zero_v;
2503                using ring_type = Ring;
2504                using field_type = _FractionField<Ring>;
2505
2507                static constexpr bool is_integer = std::is_same<val2, typename Ring::one>::value;
2508
2512                template<typename valueType>
2513                DEVICE INLINED static constexpr valueType get() { return static_cast<valueType>(x::v) /
    static_cast<valueType>(y::v); }
2514
2517                static std::string to_string() {
2518                    return to_string_helper<val1, val2>::func();
2519                }
2520
2525                template<typename valueRing>
2526                static constexpr valueRing eval(const valueRing& v) {
2527                    return x::eval(v) / y::eval(v);
2528                }
2529            };
2530
2532            using zero = val<typename Ring::zero, typename Ring::one>;
2534            using one = val<typename Ring::one, typename Ring::one>;
2535
2538            template<typename v>
2539            using inject_t = val<v, typename Ring::one>;
2540
2543            template<auto x>
2544            using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
    Ring::one>;
2545
2548            template<typename v>
2549            using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
2550
2551            using ring_type = Ring;
2552
2553        private:
2554            template<typename v, typename E = void>
2555            struct simplify {};
2556
2557            // x = 0
2558            template<typename v>
2559            struct simplify<v, std::enable_if_t<v::x::is_zero_v>> {
2560                using type = typename _FractionField<Ring>::zero;
2561            };
2562
2563            // x != 0
2564            template<typename v>
2565            struct simplify<v, std::enable_if_t<!v::x::is_zero_v>> {
2566
2567            private:
2568                using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
2569                using newx = typename Ring::template div_t<typename v::x, _gcd>;
2570                using newy = typename Ring::template div_t<typename v::y, _gcd>;
2571
2572                using posx = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
    minus_t<newx>, newx>;
2573                using posy = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
    minus_t<newy>, newy>;
2574            public:
2575                using type = typename _FractionField<Ring>::template val<posx, posy>;
2576            };
2577
2578        public:
2581            template<typename v>
2582            using simplify_t = typename simplify<v>::type;
2583
2584        private:
2585
2586            template<typename v1, typename v2>
2587            struct add {
2588            private:
2589                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2590                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2591                using dividend = typename Ring::template add_t<a, b>;
2592                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2593                using g = typename Ring::template gcd_t<dividend, diviser>;
2594
2595            public:
```

```
2596                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
2597            };
2598
2599            template<typename v>
2600            struct pos {
2601                static constexpr bool value =
2602                    (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
2603                    (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>);
2604
2605            };
2606
2607            template<typename v1, typename v2>
2608            struct sub {
2609            private:
2610                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2611                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2612                using dividend = typename Ring::template sub_t<a, b>;
2613                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2614                using g = typename Ring::template gcd_t<dividend, diviser>;
2615
2616            public:
2617                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
2618            };
2619
2620            template<typename v1, typename v2>
2621            struct mul {
2622            private:
2623                using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
2624                using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2625
2626            public:
2627                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
2628            };
2629
2630            template<typename v1, typename v2, typename E = void>
2631            struct div {};
2632
2633            template<typename v1, typename v2>
2634            struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
    _FractionField<Ring>::zero>::value» {
2635            private:
2636                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2637                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2638
2639            public:
2640                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
2641            };
2642
2643            template<typename v1, typename v2>
2644            struct div<v1, v2, std::enable_if_t<
2645                std::is_same<zero, v1>::value&& std::is_same<v2, zero>::value»  {
2646                using type = one;
2647            };
2648
2649            template<typename v1, typename v2>
2650            struct eq {
2651                static constexpr bool value =
2652                    std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value&&
2653                    std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value;
2654            };
2655
2656            template<typename TL, typename E = void>
2657            struct vadd {};
2658
2659            template<typename TL>
2660            struct vadd<TL, std::enable_if_t<(TL::length > 1)» {
2661                using head = typename TL::pop_front::type;
2662                using tail = typename TL::pop_front::tail;
2663                using type = typename add<head, typename vadd<tail>::type>::type;
2664            };
2665
2666            template<typename TL>
2667            struct vadd<TL, std::enable_if_t<(TL::length == 1)» {
2668                using type = typename TL::template at<0>;
2669            };
2670
2671            template<typename...  vals>
2672            struct vmul {};
2673
2674            template<typename v1, typename...  vals>
2675            struct vmul<v1, vals...> {
2676                using type = typename mul<v1, typename vmul<vals...>::type>::type;
2677            };
2678
2679            template<typename v1>
2680            struct vmul<v1> {
2681                using type = v1;
```

```
2682            };
2683
2684
2685            template<typename v1, typename v2, typename E = void>
2686            struct gt;
2687
2688            template<typename v1, typename v2>
2689            struct gt<v1, v2, std::enable_if_t<
2690                (eq<v1, v2>::value)
2691                » {
2692                static constexpr bool value = false;
2693            };
2694
2695            template<typename v1, typename v2>
2696            struct gt<v1, v2, std::enable_if_t<
2697                (!eq<v1, v2>::value) &&
2698                (!pos<v1>::value) && (!pos<v2>::value)
2699                » {
2700                static constexpr bool value = gt<
2701                    typename sub<zero, v1>::type, typename sub<zero, v2>::type
2702                >::value;
2703            };
2704
2705            template<typename v1, typename v2>
2706            struct gt<v1, v2, std::enable_if_t<
2707                (!eq<v1, v2>::value) &&
2708                (pos<v1>::value) && (!pos<v2>::value)
2709                » {
2710                static constexpr bool value = true;
2711            };
2712
2713            template<typename v1, typename v2>
2714            struct gt<v1, v2, std::enable_if_t<
2715                (!eq<v1, v2>::value) &&
2716                (!pos<v1>::value) && (pos<v2>::value)
2717                » {
2718                static constexpr bool value = false;
2719            };
2720
2721            template<typename v1, typename v2>
2722            struct gt<v1, v2, std::enable_if_t<
2723                (!eq<v1, v2>::value) &&
2724                (pos<v1>::value) && (pos<v2>::value)
2725                » {
2726                static constexpr bool value = Ring::template gt_v<
2727                    typename Ring::template mul_t<v1::x, v2::y>,
2728                    typename Ring::template mul_t<v2::y, v2::x>
2729                >;
2730            };
2731
2732        public:
2733
2735            template<typename v1, typename v2>
2736            using add_t = typename add<v1, v2>::type;
2737
2739            template<typename v1, typename v2>
2740            using mod_t = zero;
2741
2745            template<typename v1, typename v2>
2746            using gcd_t = v1;
2747
2750            template<typename...  vs>
2751            using vadd_t = typename vadd<vs...>::type;
2752
2755            template<typename...  vs>
2756            using vmul_t = typename vmul<vs...>::type;
2757
2759            template<typename v1, typename v2>
2760            using sub_t = typename sub<v1, v2>::type;
2761
2762            template<typename v>
2763            using minus_t = sub_t<zero, v>;
2764
2766            template<typename v1, typename v2>
2767            using mul_t = typename mul<v1, v2>::type;
2768
2770            template<typename v1, typename v2>
2771            using div_t = typename div<v1, v2>::type;
2772
2774            template<typename v1, typename v2>
2775            static constexpr bool eq_v = eq<v1, v2>::value;
2776
2778            template<typename v1, typename v2>
2779            static constexpr bool gt_v = gt<v1, v2>::value;
2780
2782            template<typename v>
2783            static constexpr bool pos_v = pos<v>::value;
```

```
2784         };
2785
2786         template<typename Ring, typename E = void>
2787             requires IsEuclideanDomain<Ring>
2788         struct FractionFieldImpl {};
2789
2790         // fraction field of a field is the field itself
2791         template<typename Field>
2792             requires IsEuclideanDomain<Field>
2793         struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {
2794             using type = Field;
2795             template<typename v>
2796             using inject_t = v;
2797         };
2798
2799         // fraction field of a ring is the actual fraction field
2800         template<typename Ring>
2801             requires IsEuclideanDomain<Ring>
2802         struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {
2803             using type = _FractionField<Ring>;
2804         };
2805     }
2806
2807     template<typename Ring>
2808         requires IsEuclideanDomain<Ring>
2809     using FractionField = typename internal::FractionFieldImpl<Ring>::type;
2810 }
2811
2812 // short names for common types
2813 namespace aerobus {
2815     using q32 = FractionField<i32>;
2817     using fpq32 = FractionField<polynomial<q32»;
2819     using q64 = FractionField<i64>;
2821     using pi64 = polynomial<i64>;
2823     using pq64 = polynomial<q64>;
2825     using fpq64 = FractionField<polynomial<q64»;
2826
2829     template<uint32_t...  digits>
2830     using bigint_pos = bigint::template val<bigint::signs::positive, digits...>;
2833     template<uint32_t...  digits>
2834     using bigint_neg = bigint::template val<bigint::signs::negative, digits...>;
2835
2840     template<typename Ring, typename v1, typename v2>
2841     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
2842
2843     template<typename Ring, typename v1, typename v2>
2844     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
2845     template<typename Ring, typename v1, typename v2>
2846     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
2847 }
2848
2849 // taylor series and common integers (factorial, bernouilli...)  appearing in taylor coefficients
2850 namespace aerobus {
2851     namespace internal {
2852         template<typename T, size_t x, typename E = void>
2853         struct factorial {};
2854
2855         template<typename T, size_t x>
2856         struct factorial<T, x, std::enable_if_t<(x > 0)» {
2857         private:
2858             template<typename, size_t, typename>
2859             friend struct factorial;
2860         public:
2861             using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
      x - 1>::type>;
2862             static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
2863         };
2864
2865         template<typename T>
2866         struct factorial<T, 0> {
2867         public:
2868             using type = typename T::one;
2869             static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
2870         };
2871     }
2872
2876     template<typename T, size_t i>
2877     using factorial_t = typename internal::factorial<T, i>::type;
2878
2879     template<typename T, size_t i>
2880     inline constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
2881
2882     namespace internal {
2883         template<typename T, size_t k, size_t n, typename E = void>
2884         struct combination_helper {};
```

```
2885
2886        template<typename T, size_t k, size_t n>
2887        struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
2888            using type = typename FractionField<T>::template mul_t<
2889                typename combination_helper<T, k - 1, n - 1>::type,
2890                makefraction_t<T, typename T::template val<n>, typename T::template val<k»>;
2891        };
2892
2893        template<typename T, size_t k, size_t n>
2894        struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
2895            using type = typename combination_helper<T, n - k, n>::type;
2896        };
2897
2898        template<typename T, size_t n>
2899        struct combination_helper<T, 0, n> {
2900            using type = typename FractionField<T>::one;
2901        };
2902
2903        template<typename T, size_t k, size_t n>
2904        struct combination {
2905            using type = typename internal::combination_helper<T, k, n>::type::x;
2906            static constexpr typename T::inner_type value = internal::combination_helper<T, k,
     n>::type::template get<typename T::inner_type>();
2907        };
2908    }
2909
2912    template<typename T, size_t k, size_t n>
2913    using combination_t = typename internal::combination<T, k, n>::type;
2914
2915    template<typename T, size_t k, size_t n>
2916    inline constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
2917
2918    namespace internal {
2919        template<typename T, size_t m>
2920        struct bernouilli;
2921
2922        template<typename T, typename accum, size_t k, size_t m>
2923        struct bernouilli_helper {
2924            using type = typename bernouilli_helper<
2925                T,
2926                addfractions_t<T,
2927                accum,
2928                mulfractions_t<T,
2929                makefraction_t<T,
2930                combination_t<T, k, m + 1>,
2931                typename T::one>,
2932                typename bernouilli<T, k>::type
2933                >
2934                >,
2935                k + 1,
2936                m>::type;
2937        };
2938
2939        template<typename T, typename accum, size_t m>
2940        struct bernouilli_helper<T, accum, m, m>
2941        {
2942            using type = accum;
2943        };
2944
2945
2946
2947        template<typename T, size_t m>
2948        struct bernouilli {
2949            using type = typename FractionField<T>::template mul_t<
2950                typename internal::bernouilli_helper<T, typename FractionField<T>::zero, 0, m>::type,
2951                makefraction_t<T,
2952                typename T::template val<static_cast<typename T::inner_type>(-1)>,
2953                typename T::template val<static_cast<typename T::inner_type>(m + 1)>
2954                >
2955            >;
2956
2957            template<typename floatType>
2958            static constexpr floatType value = type::template get<floatType>();
2959        };
2960
2961        template<typename T>
2962        struct bernouilli<T, 0> {
2963            using type = typename FractionField<T>::one;
2964
2965            template<typename floatType>
2966            static constexpr floatType value = type::template get<floatType>();
2967        };
2968    }
2969
2973    template<typename T, size_t n>
2974    using bernouilli_t = typename internal::bernouilli<T, n>::type;
2975
```

```
2976    template<typename FloatType, typename T, size_t n >
2977    inline constexpr FloatType bernouilli_v = internal::bernouilli<T, n>::template value<FloatType>;
2978
2979    namespace internal {
2980        template<typename T, int k, typename E = void>
2981        struct alternate {};
2982
2983        template<typename T, int k>
2984        struct alternate<T, k, std::enable_if_t<k % 2 == 0>> {
2985            using type = typename T::one;
2986            static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
2987        };
2988
2989        template<typename T, int k>
2990        struct alternate<T, k, std::enable_if_t<k % 2 != 0>> {
2991            using type = typename T::template minus_t<typename T::one>;
2992            static constexpr typename T::inner_type value = type::template get<typename
      T::inner_type>();
2993        };
2994    }
2995
2998    template<typename T, int k>
2999    using alternate_t = typename internal::alternate<T, k>::type;
3000
3001    template<typename T, size_t k>
3002    inline constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
3003
3004    // pow
3005    namespace internal {
3006        template<typename T, auto p, auto n>
3007        struct pow {
3008            using type = typename T::template mul_t<typename T::template val<p>, typename pow<T, p, n -
      1>::type>;
3009        };
3010
3011        template<typename T, auto p>
3012        struct pow<T, p, 0> { using type = typename T::one; };
3013    }
3014
3015    template<typename T, auto p, auto n>
3016    using pow_t = typename internal::pow<T, p, n>::type;
3017
3018    namespace internal {
3019        template<typename, template<typename, size_t> typename, class>
3020        struct make_taylor_impl;
3021
3022        template<typename T, template<typename, size_t> typename coeff_at, size_t...  Is>
3023        struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...>> {
3024            using type = typename polynomial<FractionField<T>>::template val<typename coeff_at<T,
      Is>::type...>;
3025        };
3026    }
3027
3028    // generic taylor serie, depending on coefficients
3029    template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
3030    using taylor = typename internal::make_taylor_impl<T, coeff_at,
      internal::make_index_sequence_reverse<deg + 1>>::type;
3031
3032    namespace internal {
3033        template<typename T, size_t i>
3034        struct exp_coeff {
3035            using type = makefraction_t<T, typename T::one, factorial_t<T, i>>;
3036        };
3037
3038        template<typename T, size_t i, typename E = void>
3039        struct sin_coeff_helper {};
3040
3041        template<typename T, size_t i>
3042        struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0>> {
3043            using type = typename FractionField<T>::zero;
3044        };
3045
3046        template<typename T, size_t i>
3047        struct sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1>> {
3048            using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i>>;
3049        };
3050
3051        template<typename T, size_t i>
3052        struct sin_coeff {
3053            using type = typename sin_coeff_helper<T, i>::type;
3054        };
3055
3056        template<typename T, size_t i, typename E = void>
3057        struct sh_coeff_helper {};
3058
3059        template<typename T, size_t i>
```

```
3060         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
3061             using type = typename FractionField<T>::zero;
3062         };
3063
3064         template<typename T, size_t i>
3065         struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
3066             using type = makefraction_t<T, typename T::one, factorial_t<T, i»;
3067         };
3068
3069         template<typename T, size_t i>
3070         struct sh_coeff {
3071             using type = typename sh_coeff_helper<T, i>::type;
3072         };
3073
3074         template<typename T, size_t i, typename E = void>
3075         struct cos_coeff_helper {};
3076
3077         template<typename T, size_t i>
3078         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
3079             using type = typename FractionField<T>::zero;
3080         };
3081
3082         template<typename T, size_t i>
3083         struct cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
3084             using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i»;
3085         };
3086
3087         template<typename T, size_t i>
3088         struct cos_coeff {
3089             using type = typename cos_coeff_helper<T, i>::type;
3090         };
3091
3092         template<typename T, size_t i, typename E = void>
3093         struct cosh_coeff_helper {};
3094
3095         template<typename T, size_t i>
3096         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
3097             using type = typename FractionField<T>::zero;
3098         };
3099
3100         template<typename T, size_t i>
3101         struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
3102             using type = makefraction_t<T, typename T::one, factorial_t<T, i»;
3103         };
3104
3105         template<typename T, size_t i>
3106         struct cosh_coeff {
3107             using type = typename cosh_coeff_helper<T, i>::type;
3108         };
3109
3110         template<typename T, size_t i>
3111         struct geom_coeff { using type = typename FractionField<T>::one; };
3112
3113
3114         template<typename T, size_t i, typename E = void>
3115         struct atan_coeff_helper;
3116
3117         template<typename T, size_t i>
3118         struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
3119             using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i»;
3120         };
3121
3122         template<typename T, size_t i>
3123         struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
3124             using type = typename FractionField<T>::zero;
3125         };
3126
3127         template<typename T, size_t i>
3128         struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
3129
3130         template<typename T, size_t i, typename E = void>
3131         struct asin_coeff_helper;
3132
3133         template<typename T, size_t i>
3134         struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
3135         {
3136             using type = makefraction_t<T,
3137                 factorial_t<T, i - 1>,
3138                 typename T::template mul_t<
3139                 typename T::template val<i>,
3140                 T::template mul_t<
3141                 pow_t<T, 4, i / 2>,
3142                 pow<T, factorial<T, i / 2>::value, 2
3143                 >
3144                 >
3145                 »;
3146         };
```

```
3147
3148        template<typename T, size_t i>
3149        struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
3150        {
3151            using type = typename FractionField<T>::zero;
3152        };
3153
3154        template<typename T, size_t i>
3155        struct asin_coeff {
3156            using type = typename asin_coeff_helper<T, i>::type;
3157        };
3158
3159        template<typename T, size_t i>
3160        struct lnp1_coeff {
3161            using type = makefraction_t<T,
3162                alternate_t<T, i + 1>,
3163                typename T::template val<i»;
3164        };
3165
3166        template<typename T>
3167        struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
3168
3169        template<typename T, size_t i, typename E = void>
3170        struct asinh_coeff_helper;
3171
3172        template<typename T, size_t i>
3173        struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
3174        {
3175            using type = makefraction_t<T,
3176                typename T::template mul_t<
3177                alternate_t<T, i / 2>,
3178                factorial_t<T, i - 1>
3179                >,
3180                typename T::template mul_t<
3181                T::template mul_t<
3182                typename T::template val<i>,
3183                pow_t<T, (factorial<T, i / 2>::value), 2>
3184                >,
3185                pow_t<T, 4, i / 2>
3186                >
3187            >;
3188        };
3189
3190        template<typename T, size_t i>
3191        struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
3192        {
3193            using type = typename FractionField<T>::zero;
3194        };
3195
3196        template<typename T, size_t i>
3197        struct asinh_coeff {
3198            using type = typename asinh_coeff_helper<T, i>::type;
3199        };
3200
3201        template<typename T, size_t i, typename E = void>
3202        struct atanh_coeff_helper;
3203
3204        template<typename T, size_t i>
3205        struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
3206        {
3207            // 1/i
3208            using type = typename FractionField<T>::  template val<
3209                typename T::one,
3210                typename T::template val<static_cast<typename T::inner_type>(i)»;
3211        };
3212
3213        template<typename T, size_t i>
3214        struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
3215        {
3216            using type = typename FractionField<T>::zero;
3217        };
3218
3219        template<typename T, size_t i>
3220        struct atanh_coeff {
3221            using type = typename asinh_coeff_helper<T, i>::type;
3222        };
3223
3224        template<typename T, size_t i, typename E = void>
3225        struct tan_coeff_helper;
3226
3227        template<typename T, size_t i>
3228        struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
3229            using type = typename FractionField<T>::zero;
3230        };
3231
3232        template<typename T, size_t i>
3233        struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
```

```
3234         private:
3235             // 4^((i+1)/2)
3236             using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
3237             // 4^((i+1)/2) - 1
3238             using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
     FractionField<T>::one>;
3239             // (-1)^((i-1)/2)
3240             using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
3241             using dividend = typename FractionField<T>::template mul_t<
3242                 altp,
3243                 FractionField<T>::template mul_t<
3244                 _4p,
3245                 FractionField<T>::template mul_t<
3246                 _4pm1,
3247                 bernouilli_t<T, (i + 1)>
3248                 >
3249                 >
3250             >;
3251         public:
3252             using type = typename FractionField<T>::template div_t<dividend,
3253                 typename FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
3254         };
3255
3256         template<typename T, size_t i>
3257         struct tan_coeff {
3258             using type = typename tan_coeff_helper<T, i>::type;
3259         };
3260
3261         template<typename T, size_t i, typename E = void>
3262         struct tanh_coeff_helper;
3263
3264         template<typename T, size_t i>
3265         struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
3266             using type = typename FractionField<T>::zero;
3267         };
3268
3269         template<typename T, size_t i>
3270         struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
3271         private:
3272             using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
3273             using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
     FractionField<T>::one>;
3274             using dividend =
3275                 typename FractionField<T>::template mul_t<
3276                 _4p,
3277                 typename FractionField<T>::template mul_t<
3278                 _4pm1,
3279                 bernouilli_t<T, (i + 1)>
3280                 >
3281                 >::type;
3282         public:
3283             using type = typename FractionField<T>::template div_t<dividend,
3284                 FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
3285         };
3286
3287         template<typename T, size_t i>
3288         struct tanh_coeff {
3289             using type = typename tanh_coeff_helper<T, i>::type;
3290         };
3291     }
3292
3296     template<typename T, size_t deg>
3297     using exp = taylor<T, internal::exp_coeff, deg>;
3298
3302     template<typename T, size_t deg>
3303     using expm1 = typename polynomial<FractionField<T»::template sub_t<
3304         exp<T, deg>,
3305         typename polynomial<FractionField<T»::one>;
3306
3310     template<typename T, size_t deg>
3311     using lnp1 = taylor<T, internal::lnp1_coeff, deg>;
3312
3316     template<typename T, size_t deg>
3317     using atan = taylor<T, internal::atan_coeff, deg>;
3318
3322     template<typename T, size_t deg>
3323     using sin = taylor<T, internal::sin_coeff, deg>;
3324
3328     template<typename T, size_t deg>
3329     using sinh = taylor<T, internal::sh_coeff, deg>;
3330
3334     template<typename T, size_t deg>
3335     using cosh = taylor<T, internal::cosh_coeff, deg>;
3336
3340     template<typename T, size_t deg>
3341     using cos = taylor<T, internal::cos_coeff, deg>;
3342
```

```
3346     template<typename T, size_t deg>
3347     using geometric_sum = taylor<T, internal::geom_coeff, deg>;
3348
3352     template<typename T, size_t deg>
3353     using asin = taylor<T, internal::asin_coeff, deg>;
3354
3358     template<typename T, size_t deg>
3359     using asinh = taylor<T, internal::asinh_coeff, deg>;
3360
3364     template<typename T, size_t deg>
3365     using atanh = taylor<T, internal::atanh_coeff, deg>;
3366
3370     template<typename T, size_t deg>
3371     using tan = taylor<T, internal::tan_coeff, deg>;
3372
3376     template<typename T, size_t deg>
3377     using tanh = taylor<T, internal::tanh_coeff, deg>;
3378 }
3379
3380 // continued fractions
3381 namespace aerobus {
3384     template<int64_t...  values>
3385     struct ContinuedFraction {};
3386
3387     template<int64_t a0>
3388     struct ContinuedFraction<a0> {
3389         using type = typename q64::template inject_constant_t<a0>;
3390         static constexpr double val = type::template get<double>();
3391     };
3392
3393     template<int64_t a0, int64_t...  rest>
3394     struct ContinuedFraction<a0, rest...> {
3395         using type = q64::template add_t<
3396             typename q64::template inject_constant_t<a0>,
3397             typename q64::template div_t<
3398             typename q64::one,
3399             typename ContinuedFraction<rest...>::type
3400             »;
3401         static constexpr double val = type::template get<double>();
3402     };
3403
3408     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>;
3411     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>;
3413     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>;
3415     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, 1, 2>;
3416 }
3417
3418 // known polynomials
3419 namespace aerobus {
3420     // CChebyshev
3421     namespace internal {
3422         template<int kind, int deg>
3423         struct chebyshev_helper {
3424             using type = typename pi64::template sub_t<
3425                 typename pi64::template mul_t<
3426                 typename pi64::template mul_t<
3427                 pi64::inject_constant_t<2>,
3428                 typename pi64::X
3429                 >,
3430                 typename chebyshev_helper<kind, deg - 1>::type
3431                 >,
3432                 typename chebyshev_helper<kind, deg - 2>::type
3433             >;
3434         };
3435
3436         template<>
3437         struct chebyshev_helper<1, 0> {
3438             using type = typename pi64::one;
3439         };
3440
3441         template<>
3442         struct chebyshev_helper<1, 1> {
3443             using type = typename pi64::X;
3444         };
3445
3446         template<>
3447         struct chebyshev_helper<2, 0> {
3448             using type = typename pi64::one;
3449         };
3450
3451         template<>
3452         struct chebyshev_helper<2, 1> {
3453             using type = typename pi64::template mul_t<
```

```
3454                 typename pi64::inject_constant_t<2>,
3455                 typename pi64::X>;
3456         };
3457     }
3458
3459     // Laguerre
3460     namespace internal {
3461         template<size_t deg>
3462         struct laguerre_helper {
3463         private:
3464             // Lk = (1 / k) * ((2 * k - 1 - x) * lkm1 - (k - 2)Lkm2)
3465             using lnm2 = typename laguerre_helper<deg - 2>::type;
3466             using lnm1 = typename laguerre_helper<deg - 1>::type;
3467             // -x + 2k-1
3468             using p = typename pq64::template val<
3469                 typename q64::template inject_constant_t<-1>,
3470                 typename q64::template inject_constant_t<2 * deg - 1»;
3471             // 1/n
3472             using factor = typename pq64::template inject_ring_t<
3473                 q64::val<typename i64::one, typename i64::template inject_constant_t<deg»>;
3474
3475         public:
3476             using type = typename pq64::template mul_t <
3477                 factor,
3478                 typename pq64::template sub_t<
3479                     typename pq64::template mul_t<
3480                         p,
3481                         lnm1
3482                     >,
3483                     typename pq64::template mul_t<
3484                         typename pq64::template inject_constant_t<deg-1>,
3485                         lnm2
3486                     >
3487                 >
3488             >;
3489
3490         };
3491
3492         template<>
3493         struct laguerre_helper<0> {
3494             using type = typename pq64::one;
3495         };
3496
3497         template<>
3498         struct laguerre_helper<1> {
3499             using type = typename pq64::template sub_t<typename pq64::one, typename pq64::X>;
3500         };
3501     };
3502
3504     enum hermite_kind {
3505         probabilist,
3506         physicist
3507     };
3508
3509     namespace internal {
3510         template<size_t deg, hermite_kind kind>
3511         struct hermite_helper {};
3512
3513         template<size_t deg>
3514         struct hermite_helper<deg, hermite_kind::probabilist> {
3515         private:
3516             using hnm1 = typename hermite_helper<deg - 1, hermite_kind::probabilist>::type;
3517             using hnm2 = typename hermite_helper<deg - 2, hermite_kind::probabilist>::type;
3518
3519         public:
3520             using type = typename pi64::template sub_t<
3521                 typename pi64::template mul_t<typename pi64::X, hnm1>,
3522                 typename pi64::template mul_t<
3523                     typename pi64::template inject_constant_t<deg - 1>,
3524                     hnm2
3525                 >
3526             >;
3527         };
3528
3529         template<size_t deg>
3530         struct hermite_helper<deg, hermite_kind::physicist> {
3531         private:
3532             using hnm1 = typename hermite_helper<deg - 1, hermite_kind::physicist>::type;
3533             using hnm2 = typename hermite_helper<deg - 2, hermite_kind::physicist>::type;
3534
3535         public:
3536             using type = typename pi64::template sub_t<
3537                 // 2X Hn-1
3538                 typename pi64::template mul_t<typename pi64::val<typename i64::template
      inject_constant_t<2>, typename i64::zero>, hnm1>,
3539
3540                 typename pi64::template mul_t<
```

```
3541                     typename pi64::template inject_constant_t<2*(deg - 1)>,
3542                     hnm2
3543                 >
3544             >;
3545         };
3546 
3547         template<>
3548         struct hermite_helper<0, hermite_kind::probabilist> {
3549             using type = typename pi64::one;
3550         };
3551 
3552         template<>
3553         struct hermite_helper<1, hermite_kind::probabilist> {
3554             using type = typename pi64::X;
3555         };
3556 
3557         template<>
3558         struct hermite_helper<0, hermite_kind::physicist> {
3559             using type = typename pi64::one;
3560         };
3561 
3562         template<>
3563         struct hermite_helper<1, hermite_kind::physicist> {
3564             // 2X
3565             using type = typename pi64::template val<typename i64::template inject_constant_t<2>,
    typename i64::zero>;
3566         };
3567     }
3568 
3569     // Legendre
3570     namespace internal {
3571         template<size_t deg>
3572         struct legendre_helper {
3573         private:
3574             using lnm1 = typename legendre_helper<deg - 1>::type;
3575             using lnm2 = typename legendre_helper<deg - 2>::type;
3576             using factor = typename pq64::template val<
3577                 typename q64::template val<
3578                     typename i64::template inject_constant_t<2*deg-1>,
3579                     typename i64::template inject_constant_t<deg>
3580                 >,
3581                 typename q64::zero
3582             >;
3583         public:
3584             using type = typename pq64::template sub_t<
3585                 typename pq64::template mul_t<factor, lnm1>,
3586                 typename pq64::template mul_t<
3587                     typename pq64::template inject_ring_t<
3588                         typename q64::template val<
3589                             typename i64::template inject_constant_t<deg-1>,
3590                             typename i64::template inject_constant_t<deg>
3591                         >
3592                     >,
3593                     lnm2
3594                 >
3595             >;
3596         };
3597 
3598         template<>
3599         struct legendre_helper<0> {
3600             using type = typename pq64::one;
3601         };
3602 
3603         template<>
3604         struct legendre_helper<1> {
3605             using type = typename pq64::X;
3606         };
3607     }
3608 
3611     template<size_t deg>
3612     using chebyshev_T = typename internal::chebyshev_helper<1, deg>::type;
3613 
3616     template<size_t deg>
3617     using chebyshev_U = typename internal::chebyshev_helper<2, deg>::type;
3618 
3621     template<size_t deg>
3622     using laguerre = typename internal::laguerre_helper<deg>::type;
3623 
3626     template<size_t deg>
3627     using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist>::type;
3628 
3631     template<size_t deg>
3632     using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist>::type;
3633 
3636     template<size_t deg>
3637     using legendre = typename internal::legendre_helper<deg>::type;
3638 }
```

Chapter 7

Example Documentation

7.1 i32::template

inject a native constant

inject a native constant

Template Parameters

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

7.2 i64::template

injects constant as an i64 value

injects constant as an i64 value

Template Parameters

x	inject_constant_t<2>

7.3 polynomial

makes the constant (native type) polynomial a_0

makes the constant (native type) polynomial a_0

Template Parameters

x	<i32>::template inject_constant_t<2>

7.4 bigint::from_hex_t

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

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

7.5 PI_fraction::val

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

7.6 E_fraction::val

approximation of e -> 2.718...

approximation of e -> 2.718...

Index