# 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

a constexpr "string" utility

`#include <lib.h>`

### 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]

### 5.28.1 Detailed Description

**template<size_t N>**
**struct aerobus::string_literal< N >**

a constexpr "string" utility

**Template Parameters**

| | |
|---|---|
| *N* | sizeof string |

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 {
32     namespace memory {
39         template<typename T>
40         T* aligned_malloc(size_t count, size_t alignment) {
41 #ifdef _MSC_VER
42             return static_cast<T*>(_aligned_malloc(count * sizeof(T), alignment));
43 #else
44             return static_cast<T*>(aligned_alloc(alignment, count * sizeof(T)));
45 #endif
46         }
47
48         template<typename T>
49         void aligned_free(T* memblock) {
50 #ifdef _MSC_VER
51             _aligned_free((void*)memblock);
52 #else
53             free((void*)memblock);
54 #endif
55         }
56     }
57
58     namespace internal {
59         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 } };
60      }
61
70      template<typename T, size_t N>
71      constexpr bool contains(const std::array<T, N>& arr, const T& v) {
72          for (const auto& vv :  arr) {
73              if (v == vv) {
74                  return true;
75              }
76          }
77
78          return false;
79      }
80
83      template <size_t N>
84      struct string_literal {
85          constexpr string_literal(const char(&str)[N]) {
86              std::reverse_copy(str, str + N, value);
87          }
88
89          template<size_t i>
90          constexpr char char_at()const {
91              // first char is always \0
92              if constexpr (i + 1 < N) {
93                  return this->value[i + 1];
94              }
95              return '\0';
96          }
97
98          constexpr size_t len()const { return N; }
99
100          char value[N];
101      };
102 }
103
104 // concepts
```

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

```
193
194
195         template<int32_t n, int32_t i>
196         struct _is_prime<n, i, std::enable_if_t<
197             (n >= 7920) &&
198             (i >= 5 && i * i <= n) &&
199             (n% i != 0 && n % (i + 2) != 0)» {
200             static constexpr bool value = _is_prime<n, i + 6>::value;
201         };
202
203         template<int32_t n, int32_t i>
204         struct _is_prime<n, i, std::enable_if_t<
205             (n >= 7920) &&
206             (i >= 5 && i * i > n)» :  std::true_type {};
207     }
208
211     template<int32_t n>
212     struct is_prime {
214         static constexpr bool value = internal::_is_prime<n, 5>::value;
215     };
216
217     namespace internal {
218         template <std::size_t...  Is>
219         constexpr auto index_sequence_reverse(std::index_sequence<Is...> const&)
220             -> decltype(std::index_sequence<sizeof...(Is) - 1U - Is...>{});
221
222         template <std::size_t N>
223         using make_index_sequence_reverse
224             = decltype(index_sequence_reverse(std::make_index_sequence<N>{}));
225
231         template<typename Ring, typename E = void>
232         struct gcd;
233
234         template<typename Ring>
235         struct gcd<Ring, std::enable_if_t<Ring::is_euclidean_domain» {
236             template<typename A, typename B, typename E = void>
237             struct gcd_helper {};
238
239             // B = 0, A > 0
240             template<typename A, typename B>
241             struct gcd_helper<A, B, std::enable_if_t<
242                 B::is_zero_v&& Ring::template pos_v<A»>
243             {
244                 using type = A;
245             };
246
247             // B = 0, A < 0
248             template<typename A, typename B>
249             struct gcd_helper<A, B, std::enable_if_t<
250                 B::is_zero_v && !Ring::template pos_v<A»>
251             {
252                 using type = typename Ring::template minus_t<A>;
253             };
254
255             // B != 0
256             template<typename A, typename B>
257             struct gcd_helper<A, B, std::enable_if_t<
258                 (!B::is_zero_v)
259                 » {
260             private:
261                 // A / B
262                 using k = typename Ring::template div_t<A, B>;
263                 // A - (A/B)*B = A % B
264                 using m = typename Ring::template sub_t<A, typename Ring::template mul_t<k, B»;
265             public:
266                 using type = typename gcd_helper<B, m>::type;
267             };
268
269             template<typename A, typename B>
270             using type = typename gcd_helper<A, B>::type;
271         };
272     }
273
276     template<typename T, typename A, typename B>
277     using gcd_t = typename internal::gcd<T>::template type<A, B>;
278 }
279
280 // quotient ring by the principal ideal generated by X
281 namespace aerobus {
282     template<typename Ring, typename X>
283         requires IsRing<Ring>
284     struct Quotient {
285         template <typename V>
286         struct val {
287         private:
288             using tmp = typename Ring::template mod_t<V, X>;
289         public:
```

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

```
378             using right = typename s::tail;
379             using rr = typename right::pop_front::tail;
380             using type = typename left::template concat<rr>;
381         };
382     }
383 
384     template <typename...  Ts>
385     struct type_list
386     {
387     private:
388         template <typename T>
389         struct concat_h;
390 
391         template <typename...  Us>
392         struct concat_h<type_list<Us...»
393         {
394             using type = type_list<Ts..., Us...>;
395         };
396 
397     public:
398         static constexpr size_t length = sizeof...(Ts);
399 
400         template <typename T>
401         using push_front = type_list<T, Ts...>;
402 
403         template <uint64_t index>
404         using at = internal::type_at_t<index, Ts...>;
405 
406         struct pop_front
407         {
408             using type = typename internal::pop_front_h<Ts...>::head;
409             using tail = typename internal::pop_front_h<Ts...>::tail;
410         };
411 
412         template <typename T>
413         using push_back = type_list<Ts..., T>;
414 
415         template <typename U>
416         using concat = typename concat_h<U>::type;
417 
418         template <uint64_t index>
419         struct split
420         {
421         private:
422             using inner = internal::split_h<index, type_list<>, type_list<Ts...»;
423 
424         public:
425             using head = typename inner::head;
426             using tail = typename inner::tail;
427         };
428 
429         template <uint64_t index, typename T>
430         using insert = typename internal::insert_h<index, type_list<Ts...>, T>::type;
431 
432         template <uint64_t index>
433         using remove = typename internal::remove_h<index, type_list<Ts...»::type;
434     };
435 
436     template <>
437     struct type_list<>
438     {
439         static constexpr size_t length = 0;
440 
441         template <typename T>
442         using push_front = type_list<T>;
443 
444         template <typename T>
445         using push_back = type_list<T>;
446 
447         template <typename U>
448         using concat = U;
449 
450         // TODO: assert index == 0
451         template <uint64_t index, typename T>
452         using insert = type_list<T>;
453     };
454 }
455 
456 // i32
457 namespace aerobus {
459     struct i32 {
460         using inner_type = int32_t;
463         template<int32_t x>
464         struct val {
465             static constexpr int32_t v = x;
466 
469             template<typename valueType>
```

```
470             DEVICE INLINED static constexpr valueType get() { return static_cast<valueType>(x); }
471
473             static constexpr bool is_zero_v = x == 0;
474
476             static std::string to_string() {
477                 return std::to_string(x);
478             }
479
482             template<typename valueRing>
483             DEVICE INLINED static constexpr valueRing eval(const valueRing& v) {
484                 return static_cast<valueRing>(x);
485             }
486         };
487
489         using zero = val<0>;
491         using one = val<1>;
493         static constexpr bool is_field = false;
495         static constexpr bool is_euclidean_domain = true;
499         template<auto x>
500         using inject_constant_t = val<static_cast<int32_t>(x)>;
501
502         template<typename v>
503         using inject_ring_t = v;
504
505     private:
506         template<typename v1, typename v2>
507         struct add {
508             using type = val<v1::v + v2::v>;
509         };
510
511         template<typename v1, typename v2>
512         struct sub {
513             using type = val<v1::v - v2::v>;
514         };
515
516         template<typename v1, typename v2>
517         struct mul {
518             using type = val<v1::v * v2::v>;
519         };
520
521         template<typename v1, typename v2>
522         struct div {
523             using type = val<v1::v / v2::v>;
524         };
525
526         template<typename v1, typename v2>
527         struct remainder {
528             using type = val<v1::v% v2::v>;
529         };
530
531         template<typename v1, typename v2>
532         struct gt {
533             static constexpr bool value = (v1::v > v2::v);
534         };
535
536         template<typename v1, typename v2>
537         struct lt {
538             static constexpr bool value = (v1::v < v2::v);
539         };
540
541         template<typename v1, typename v2>
542         struct eq {
543             static constexpr bool value = (v1::v == v2::v);
544         };
545
546     public:
548         template<typename v1, typename v2>
549         using add_t = typename add<v1, v2>::type;
550
552         template<typename v1>
553         using minus_t = val<-v1::v>;
554
556         template<typename v1, typename v2>
557         using sub_t = typename sub<v1, v2>::type;
558
560         template<typename v1, typename v2>
561         using mul_t = typename mul<v1, v2>::type;
562
564         template<typename v1, typename v2>
565         using div_t = typename div<v1, v2>::type;
566
568         template<typename v1, typename v2>
569         using mod_t = typename remainder<v1, v2>::type;
570
572         template<typename v1, typename v2>
573         static constexpr bool gt_v = gt<v1, v2>::value;
574
```

```
576         template<typename v1, typename v2>
577         static constexpr bool lt_v = lt<v1, v2>::value;
578
580         template<typename v1, typename v2>
581         static constexpr bool eq_v = eq<v1, v2>::value;
582
584         template<typename v1>
585         static constexpr bool pos_v = (v1::v > 0);
586
588         template<typename v1, typename v2>
589         using gcd_t = gcd_t<i32, v1, v2>;
590     };
591 }
592
593 // i64
594 namespace aerobus {
596     struct i64 {
597         using inner_type = int64_t;
600         template<int64_t x>
601         struct val {
602             static constexpr int64_t v = x;
603
606             template<typename valueType>
607             DEVICE INLINED static constexpr valueType get() { return static_cast<valueType>(x); }
608
610             static constexpr bool is_zero_v = x == 0;
611
613             static std::string to_string() {
614                 return std::to_string(x);
615             }
616
619             template<typename valueRing>
620             DEVICE INLINED static constexpr valueRing eval(const valueRing& v) {
621                 return static_cast<valueRing>(x);
622             }
623         };
624
628         template<auto x>
629         using inject_constant_t = val<static_cast<int64_t>(x)>;
630
631         template<typename v>
632         using inject_ring_t = v;
633
635         using zero = val<0>;
637         using one = val<1>;
639         static constexpr bool is_field = false;
641         static constexpr bool is_euclidean_domain = true;
642
643     private:
644         template<typename v1, typename v2>
645         struct add {
646             using type = val<v1::v + v2::v>;
647         };
648
649         template<typename v1, typename v2>
650         struct sub {
651             using type = val<v1::v - v2::v>;
652         };
653
654         template<typename v1, typename v2>
655         struct mul {
656             using type = val<v1::v* v2::v>;
657         };
658
659         template<typename v1, typename v2>
660         struct div {
661             using type = val<v1::v / v2::v>;
662         };
663
664         template<typename v1, typename v2>
665         struct remainder {
666             using type = val<v1::v% v2::v>;
667         };
668
669         template<typename v1, typename v2>
670         struct gt {
671             static constexpr bool value = (v1::v > v2::v);
672         };
673
674         template<typename v1, typename v2>
675         struct lt {
676             static constexpr bool value = (v1::v < v2::v);
677         };
678
679         template<typename v1, typename v2>
680         struct eq {
681             static constexpr bool value = (v1::v == v2::v);
```

```
682         };
683
684     public:
686         template<typename v1, typename v2>
687         using add_t = typename add<v1, v2>::type;
688
690         template<typename v1>
691         using minus_t = val<-v1::v>;
692
694         template<typename v1, typename v2>
695         using sub_t = typename sub<v1, v2>::type;
696
698         template<typename v1, typename v2>
699         using mul_t = typename mul<v1, v2>::type;
700
702         template<typename v1, typename v2>
703         using div_t = typename div<v1, v2>::type;
704
706         template<typename v1, typename v2>
707         using mod_t = typename remainder<v1, v2>::type;
708
710         template<typename v1, typename v2>
711         static constexpr bool gt_v = gt<v1, v2>::value;
712
714         template<typename v1, typename v2>
715         static constexpr bool lt_v = lt<v1, v2>::value;
716
718         template<typename v1, typename v2>
719         static constexpr bool eq_v = eq<v1, v2>::value;
720
723         template<typename v1>
724         static constexpr bool pos_v = (v1::v > 0);
725
727         template<typename v1, typename v2>
728         using gcd_t = gcd_t<i64, v1, v2>;
729     };
730 }
731
732 // z/pz
733 namespace aerobus {
738     template<int32_t p>
739     struct zpz {
740         using inner_type = int32_t;
741         template<int32_t x>
742         struct val {
743             static constexpr int32_t v = x % p;
744
745             template<typename valueType>
746             DEVICE INLINED static constexpr valueType get() { return static_cast<valueType>(x % p); }
747
748             static constexpr bool is_zero_v = v == 0;
749             static std::string to_string() {
750                 return std::to_string(x % p);
751             }
752
753             template<typename valueRing>
754             DEVICE INLINED static constexpr valueRing eval(const valueRing& v) {
755                 return static_cast<valueRing>(x % p);
756             }
757         };
758
759         template<auto x>
760         using inject_constant_t = val<static_cast<int32_t>(x)>;
761
762         using zero = val<0>;
763         using one = val<1>;
764         static constexpr bool is_field = is_prime<p>::value;
765         static constexpr bool is_euclidean_domain = true;
766
767     private:
768         template<typename v1, typename v2>
769         struct add {
770             using type = val<(v1::v + v2::v) % p>;
771         };
772
773         template<typename v1, typename v2>
774         struct sub {
775             using type = val<(v1::v - v2::v) % p>;
776         };
777
778         template<typename v1, typename v2>
779         struct mul {
780             using type = val<(v1::v* v2::v) % p>;
781         };
782
783         template<typename v1, typename v2>
784         struct div {
```

```
785             using type = val<(v1::v% p) / (v2::v % p)>;
786         };
787 
788         template<typename v1, typename v2>
789         struct remainder {
790             using type = val<(v1::v% v2::v) % p>;
791         };
792 
793         template<typename v1, typename v2>
794         struct gt {
795             static constexpr bool value = (v1::v % p > v2::v % p);
796         };
797 
798         template<typename v1, typename v2>
799         struct lt {
800             static constexpr bool value = (v1::v % p < v2::v % p);
801         };
802 
803         template<typename v1, typename v2>
804         struct eq {
805             static constexpr bool value = (v1::v % p == v2::v % p);
806         };
807 
808         template<typename v1>
809         struct pos {
810             static constexpr bool value = v1::v % p > 0;
811         };
812 
813     public:
815         template<typename v1>
816         using minus_t = val<-v1::v>;
817 
818         template<typename v1, typename v2>
819         using add_t = typename add<v1, v2>::type;
820 
821         template<typename v1, typename v2>
822         using sub_t = typename sub<v1, v2>::type;
823 
824         template<typename v1, typename v2>
825         using mul_t = typename mul<v1, v2>::type;
826 
827         template<typename v1, typename v2>
828         using div_t = typename div<v1, v2>::type;
829 
830         template<typename v1, typename v2>
831         using mod_t = typename remainder<v1, v2>::type;
832 
833         template<typename v1, typename v2>
834         static constexpr bool gt_v = gt<v1, v2>::value;
835 
836         template<typename v1, typename v2>
837         static constexpr bool lt_v = lt<v1, v2>::value;
838 
839         template<typename v1, typename v2>
840         static constexpr bool eq_v = eq<v1, v2>::value;
841 
842         template<typename v1, typename v2>
843         using gcd_t = gcd_t<i32, v1, v2>;
844 
845         template<typename v>
846         static constexpr bool pos_v = pos<v>::value;
847     };
848 }
849 
850 
851 // K[sqrt(x)]
852 namespace aerobus {
856     template<typename Field, int64_t d>
857     requires IsField<Field>
858     struct QuadraticExtension {
859         // v1 + sqrt(x) v2
860         template<typename v1, typename v2>
861         struct val {
862             using x = v1;
863             using y = v2;
864         };
865 
866         using zero = val<typename Field::template inject_constant_t<0>, typename Field::template
      inject_constant_t<0>>;
867         using one = val<typename Field::template inject_constant_t<1>, typename Field::template
      inject_constant_t<0>>;
868 
869         template<typename v>
870         static constexpr bool is_in_field_v = (v::y == 0);
871 
872     private:
873         template<typename v1, typename v2>
```

```
874         struct add {
875             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>>;
876         };
877 
878         template<typename v1, typename v2>
879         struct sub {
880             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>>;
881         };
882 
883         template<typename v1, typename v2>
884         struct mul {
885             using type = val<
886                 typename Field::template add_t<
887                 typename Field::template mul_t<typename v1::x, typename v2::x>,
888                 typename Field::template mul_t<
889                 typename Field::template inject_constant_t<d>,
890                 typename Field::template mul_t<typename v1::y, typename v2::y>
891                 >
892                 >,
893                 typename Field::template add_t<
894                 typename Field::template mul_t<typename v1::x, typename v2::y>,
895                 typename Field::template mul_t<typename v1::y, typename v2::x>
896                 >
897             >;
898         };
899 
900         template<typename v1, typename v2>
901         struct div {
902         private:
903             using inner = typename Field::template div_t<
904                 typename Field::one, typename Field::template sub_t<
905                 typename Field::template mul_t<typename v2::x, typename v2::x>,
906                 typename Field::template mul_t<
907                 typename Field::template inject_constant_t<d>,
908                 typename Field::template mul_t<typename v2::y, typename v2::y>
909                 >
910                 >
911             >;
912             using inv_v2 = val<
913                 typename Field::template mul_t<typename v2::x, inner>,
914                 typename Field::template mul_t<typename Field::template minus_t<typename v2::y>, inner>
915             >;
916         public:
917             using type = typename mul<v1, inv_v2>::type;
918         };
919 
920       public:
921         static constexpr bool is_field = true;
922         static constexpr bool is_euclidean_domain = true;
923 
924         template<auto x>
925         using inject_constant_t = val<typename Field::template inject_constant_t<x>, typename
    Field::zero>;
926 
927         template<auto x, auto y>
928         using inject_values_t = val<typename Field::template inject_constant_t<x>, typename
    Field::template inject_constant_t<y>>;
929 
930         template<typename v1, typename v2>
931         using add_t = typename add<v1, v2>::type;
932 
933         template<typename v1, typename v2>
934         using sub_t = typename sub<v1, v2>::type;
935 
936         template<typename v1, typename v2>
937         using mul_t = typename mul<v1, v2>::type;
938 
939         template<typename v1, typename v2>
940         using div_t = typename div<v1, v2>::type;
941 
942         template<typename v>
943         using minus_t = sub_t<zero, v>;
944 
945         template<typename v1, typename v2>
946         static constexpr bool eq_v =
947             (Field::template eq_v<typename v1::x, typename v2::x>) &&
948             (Field::template eq_v<typename v1::y, typename v2::y>);
949 
950         template<typename v1, typename v2>
951         static constexpr bool gt_v =
952             (Field::template gt_v<v1::x, v2::x>) ||
953             ((Field::template eq_v<v1::x, v2::x>) && (Field::template gt_v<v1::y, v2::y>));
954 
955         template<typename v>
956         static constexpr bool pos_v = gt_v<v, zero>;
```

```
957
958         template<typename v1, typename v2>
959         using mod_t = zero;
960
961         template<typename v1, typename v2>
962         using gcd_t = v1;
963     };
964 }
965
966 // polynomial
967 namespace aerobus {
968     // coeffN x^N + ...
973     template<typename Ring>
974         requires IsEuclideanDomain<Ring>
975     struct polynomial {
976         static constexpr bool is_field = false;
977         static constexpr bool is_euclidean_domain = Ring::is_euclidean_domain;
978
979         template<typename coeffN, typename...  coeffs>
980         struct val {
982             static constexpr size_t degree = sizeof...(coeffs);
984             using aN = coeffN;
986             using strip = val<coeffs...>;
988             static constexpr bool is_zero_v = degree == 0 && aN::is_zero_v;
989
990         private:
991             template<size_t index, typename E = void>
992             struct coeff_at {};
993
994             template<size_t index>
995             struct coeff_at<index, std::enable_if_t<(index >= 0 && index <= sizeof...(coeffs))>> {
996                 using type = internal::type_at_t<sizeof...(coeffs) - index, coeffN, coeffs...>;
997             };
998
999             template<size_t index>
1000             struct coeff_at<index, std::enable_if_t<(index < 0 || index > sizeof...(coeffs))>> {
1001                 using type = typename Ring::zero;
1002             };
1003
1004         public:
1007             template<size_t index>
1008             using coeff_at_t = typename coeff_at<index>::type;
1009
1012             static std::string to_string() {
1013                 return string_helper<coeffN, coeffs...>::func();
1014             }
1015
1020             template<typename valueRing>
1021             DEVICE INLINED static constexpr valueRing eval(const valueRing& x) {
1022                 return eval_helper<valueRing, val>::template inner<0, degree +
      1>::func(static_cast<valueRing>(0), x);
1023             }
1024         };
1025
1026         // specialization for constants
1027         template<typename coeffN>
1028         struct val<coeffN> {
1029             static constexpr size_t degree = 0;
1030             using aN = coeffN;
1031             using strip = val<coeffN>;
1032             static constexpr bool is_zero_v = coeffN::is_zero_v;
1033
1034             template<size_t index, typename E = void>
1035             struct coeff_at {};
1036
1037             template<size_t index>
1038             struct coeff_at<index, std::enable_if_t<(index == 0)>> {
1039                 using type = aN;
1040             };
1041
1042             template<size_t index>
1043             struct coeff_at<index, std::enable_if_t<(index < 0 || index > 0)>> {
1044                 using type = typename Ring::zero;
1045             };
1046
1047             template<size_t index>
1048             using coeff_at_t = typename coeff_at<index>::type;
1049
1050             static std::string to_string() {
1051                 return string_helper<coeffN>::func();
1052             }
1053
1054             template<typename valueRing>
1055             DEVICE INLINED static constexpr valueRing eval(const valueRing& x) {
1056                 return static_cast<valueRing>(aN::template get<valueRing>());
1057             }
1058         };
```

```
1059
1061        using zero = val<typename Ring::zero>;
1063        using one = val<typename Ring::one>;
1065        using X = val<typename Ring::one, typename Ring::zero>;
1066
1067    private:
1068        template<typename P, typename E = void>
1069        struct simplify;
1070
1071        template <typename P1, typename P2, typename I>
1072        struct add_low;
1073
1074        template<typename P1, typename P2>
1075        struct add {
1076            using type = typename simplify<typename add_low<
1077                P1,
1078                P2,
1079                internal::make_index_sequence_reverse<
1080                std::max(P1::degree, P2::degree) + 1
1081                »::type>::type;
1082        };
1083
1084        template <typename P1, typename P2, typename I>
1085        struct sub_low;
1086
1087        template <typename P1, typename P2, typename I>
1088        struct mul_low;
1089
1090        template<typename v1, typename v2>
1091        struct mul {
1092            using type = typename mul_low<
1093                v1,
1094                v2,
1095                internal::make_index_sequence_reverse<
1096                v1::degree + v2::degree + 1
1097                »::type;
1098        };
1099
1100        template<typename coeff, size_t deg>
1101        struct monomial;
1102
1103        template<typename v, typename E = void>
1104        struct derive_helper {};
1105
1106        template<typename v>
1107        struct derive_helper<v, std::enable_if_t<v::degree == 0» {
1108            using type = zero;
1109        };
1110
1111        template<typename v>
1112        struct derive_helper<v, std::enable_if_t<v::degree != 0» {
1113            using type = typename add<
1114                typename derive_helper<typename simplify<typename v::strip>::type>::type,
1115                typename monomial<
1116                typename Ring::template mul_t<
1117                typename v::aN,
1118                typename Ring::template inject_constant_t<(v::degree)>
1119                >,
1120                v::degree - 1
1121                >::type
1122            >::type;
1123        };
1124
1125        template<typename v1, typename v2, typename E = void>
1126        struct eq_helper {};
1127
1128        template<typename v1, typename v2>
1129        struct eq_helper<v1, v2, std::enable_if_t<v1::degree != v2::degree» {
1130            static constexpr bool value = false;
1131        };
1132
1133
1134        template<typename v1, typename v2>
1135        struct eq_helper<v1, v2, std::enable_if_t<
1136            v1::degree == v2::degree &&
1137            (v1::degree != 0 || v2::degree != 0) &&
1138            (!Ring::template eq_v<typename v1::aN, typename v2::aN>)
1139            » {
1140            static constexpr bool value = false;
1141        };
1142
1143        template<typename v1, typename v2>
1144        struct eq_helper<v1, v2, std::enable_if_t<
1145            v1::degree == v2::degree &&
1146            (v1::degree != 0 || v2::degree != 0) &&
1147            (Ring::template eq_v<typename v1::aN, typename v2::aN>)
1148            » {
```

```
1149            static constexpr bool value = eq_helper<typename v1::strip, typename v2::strip>::value;
1150        };
1151
1152        template<typename v1, typename v2>
1153        struct eq_helper<v1, v2, std::enable_if_t<
1154            v1::degree == v2::degree &&
1155            (v1::degree == 0)
1156            » {
1157            static constexpr bool value = Ring::template eq_v<typename v1::aN, typename v2::aN>;
1158        };
1159
1160        template<typename v1, typename v2, typename E = void>
1161        struct lt_helper {};
1162
1163        template<typename v1, typename v2>
1164        struct lt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1165            static constexpr bool value = true;
1166        };
1167
1168        template<typename v1, typename v2>
1169        struct lt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1170            static constexpr bool value = Ring::template lt_v<typename v1::aN, typename v2::aN>;
1171        };
1172
1173        template<typename v1, typename v2>
1174        struct lt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1175            static constexpr bool value = false;
1176        };
1177
1178        template<typename v1, typename v2, typename E = void>
1179        struct gt_helper {};
1180
1181        template<typename v1, typename v2>
1182        struct gt_helper<v1, v2, std::enable_if_t<(v1::degree > v2::degree)» {
1183            static constexpr bool value = true;
1184        };
1185
1186        template<typename v1, typename v2>
1187        struct gt_helper<v1, v2, std::enable_if_t<(v1::degree == v2::degree)» {
1188            static constexpr bool value = Ring::template gt_v<typename v1::aN, typename v2::aN>;
1189        };
1190
1191        template<typename v1, typename v2>
1192        struct gt_helper<v1, v2, std::enable_if_t<(v1::degree < v2::degree)» {
1193            static constexpr bool value = false;
1194        };
1195
1196        // when high power is zero :  strip
1197        template<typename P>
1198        struct simplify<P, std::enable_if_t<
1199            std::is_same<
1200            typename Ring::zero,
1201            typename P::aN
1202            >::value && (P::degree > 0)
1203            »
1204        {
1205            using type = typename simplify<typename P::strip>::type;
1206        };
1207
1208        // otherwise :  do nothing
1209        template<typename P>
1210        struct simplify<P, std::enable_if_t<
1211            !std::is_same<
1212            typename Ring::zero,
1213            typename P::aN
1214            >::value && (P::degree > 0)
1215            »
1216        {
1217            using type = P;
1218        };
1219
1220        // do not simplify constants
1221        template<typename P>
1222        struct simplify<P, std::enable_if_t<P::degree == 0» {
1223            using type = P;
1224        };
1225
1226        // addition at
1227        template<typename P1, typename P2, size_t index>
1228        struct add_at {
1229            using type =
1230                typename Ring::template add_t<typename P1::template coeff_at_t<index>, typename
     P2::template coeff_at_t<index»;
1231        };
1232
1233        template<typename P1, typename P2, size_t index>
1234        using add_at_t = typename add_at<P1, P2, index>::type;
```

```
1235
1236        template<typename P1, typename P2, std::size_t...  I>
1237        struct add_low<P1, P2, std::index_sequence<I...» {
1238            using type = val<add_at_t<P1, P2, I>...>;
1239        };
1240
1241        // substraction at
1242        template<typename P1, typename P2, size_t index>
1243        struct sub_at {
1244            using type =
1245                typename Ring::template sub_t<typename P1::template coeff_at_t<index>, typename
      P2::template coeff_at_t<index»;
1246        };
1247
1248        template<typename P1, typename P2, size_t index>
1249        using sub_at_t = typename sub_at<P1, P2, index>::type;
1250
1251        template<typename P1, typename P2, std::size_t...  I>
1252        struct sub_low<P1, P2, std::index_sequence<I...» {
1253            using type = val<sub_at_t<P1, P2, I>...>;
1254        };
1255
1256        template<typename P1, typename P2>
1257        struct sub {
1258            using type = typename simplify<typename sub_low<
1259                P1,
1260                P2,
1261                internal::make_index_sequence_reverse<
1262                std::max(P1::degree, P2::degree) + 1
1263                »::type>::type;
1264        };
1265
1266        // multiplication at
1267        template<typename v1, typename v2, size_t k, size_t index, size_t stop>
1268        struct mul_at_loop_helper {
1269            using type = typename Ring::template add_t<
1270                typename Ring::template mul_t<
1271                typename v1::template coeff_at_t<index>,
1272                typename v2::template coeff_at_t<k - index>
1273                >,
1274                typename mul_at_loop_helper<v1, v2, k, index + 1, stop>::type
1275            >;
1276        };
1277
1278        template<typename v1, typename v2, size_t k, size_t stop>
1279        struct mul_at_loop_helper<v1, v2, k, stop, stop> {
1280            using type = typename Ring::template mul_t<typename v1::template coeff_at_t<stop>, typename
      v2::template coeff_at_t<0»;
1281        };
1282
1283        template <typename v1, typename v2, size_t k, typename E = void>
1284        struct mul_at {};
1285
1286        template<typename v1, typename v2, size_t k>
1287        struct mul_at<v1, v2, k, std::enable_if_t<(k < 0) || (k > v1::degree + v2::degree)» {
1288            using type = typename Ring::zero;
1289        };
1290
1291        template<typename v1, typename v2, size_t k>
1292        struct mul_at<v1, v2, k, std::enable_if_t<(k >= 0) && (k <= v1::degree + v2::degree)» {
1293            using type = typename mul_at_loop_helper<v1, v2, k, 0, k>::type;
1294        };
1295
1296        template<typename P1, typename P2, size_t index>
1297        using mul_at_t = typename mul_at<P1, P2, index>::type;
1298
1299        template<typename P1, typename P2, std::size_t...  I>
1300        struct mul_low<P1, P2, std::index_sequence<I...» {
1301            using type = val<mul_at_t<P1, P2, I>...>;
1302        };
1303
1304        // division helper
1305        template< typename A, typename B, typename Q, typename R, typename E = void>
1306        struct div_helper {};
1307
1308        template<typename A, typename B, typename Q, typename R>
1309        struct div_helper<A, B, Q, R, std::enable_if_t<
1310            (R::degree < B::degree) ||
1311            (R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
1312            using q_type = Q;
1313            using mod_type = R;
1314            using gcd_type = B;
1315        };
1316
1317        template<typename A, typename B, typename Q, typename R>
1318        struct div_helper<A, B, Q, R, std::enable_if_t<
1319            (R::degree >= B::degree) &&
```

```
1320            !(R::degree == 0 && std::is_same<typename R::aN, typename Ring::zero>::value)» {
1321        private:
1322            using rN = typename R::aN;
1323            using bN = typename B::aN;
1324            using pT = typename monomial<typename Ring::template div_t<rN, bN>, R::degree -
    B::degree>::type;
1325            using rr = typename sub<R, typename mul<pT, B>::type>::type;
1326            using qq = typename add<Q, pT>::type;
1327
1328        public:
1329            using q_type = typename simplify<typename div_helper<A, B, qq, rr>::q_type>::type;
1330            using mod_type = typename simplify<typename div_helper<A, B, qq, rr>::mod_type>::type;
1331            using gcd_type = typename simplify<rr>::type;
1332        };
1333
1334        template<typename A, typename B>
1335        struct div {
1336            static_assert(Ring::is_euclidean_domain, "cannot divide in that type of Ring");
1337            using q_type = typename div_helper<A, B, zero, A>::q_type;
1338            using m_type = typename div_helper<A, B, zero, A>::mod_type;
1339        };
1340
1341
1342        template<typename P>
1343        struct make_unit {
1344            using type = typename div<P, val<typename P::aN»::q_type;
1345        };
1346
1347        template<typename coeff, size_t deg>
1348        struct monomial {
1349            using type = typename mul<X, typename monomial<coeff, deg - 1>::type>::type;
1350        };
1351
1352        template<typename coeff>
1353        struct monomial<coeff, 0> {
1354            using type = val<coeff>;
1355        };
1356
1357        template<typename valueRing, typename P>
1358        struct eval_helper
1359        {
1360            template<size_t index, size_t stop>
1361            struct inner {
1362                DEVICE INLINED static constexpr valueRing func(const valueRing& accum, const valueRing&
    x) {
1363                    constexpr valueRing coeff = static_cast<valueRing>(P::template coeff_at_t<P::degree
    - index>::template get<valueRing>());
1364                    return eval_helper<valueRing, P>::template inner<index + 1, stop>::func(x * accum +
    coeff, x);
1365                }
1366            };
1367
1368            template<size_t stop>
1369            struct inner<stop, stop> {
1370                DEVICE INLINED static constexpr valueRing func(const valueRing& accum, const valueRing&
    x) {
1371                    return accum;
1372                }
1373            };
1374        };
1375
1376        template<typename coeff, typename...  coeffs>
1377        struct string_helper {
1378            static std::string func() {
1379                std::string tail = string_helper<coeffs...>::func();
1380                std::string result = "";
1381                if (Ring::template eq_v<coeff, typename Ring::zero>) {
1382                    return tail;
1383                }
1384                else if (Ring::template eq_v<coeff, typename Ring::one>) {
1385                    if (sizeof...(coeffs) == 1) {
1386                        result += 'X';
1387                    }
1388                    else {
1389                        result += "X^" + std::to_string(sizeof...(coeffs));
1390                    }
1391                }
1392                else {
1393                    if (sizeof...(coeffs) == 1) {
1394                        result += coeff::to_string() + " X";
1395                    }
1396                    else {
1397                        result += coeff::to_string() + " X^" + std::to_string(sizeof...(coeffs));
1398                    }
1399                }
1400
1401                if (!tail.empty()) {
```

```
1402                     result += " + " + tail;
1403                 }
1404 
1405                 return result;
1406             }
1407         };
1408 
1409         template<typename coeff>
1410         struct string_helper<coeff> {
1411             static std::string func() {
1412                 if (!std::is_same<coeff, typename Ring::zero>::value) {
1413                     return coeff::to_string();
1414                 }
1415                 else {
1416                     return "";
1417                 }
1418             }
1419         };
1420 
1421     public:
1424         template<typename P>
1425         using simplify_t = typename simplify<P>::type;
1426 
1430         template<typename v1, typename v2>
1431         using add_t = typename add<v1, v2>::type;
1432 
1436         template<typename v1, typename v2>
1437         using sub_t = typename sub<v1, v2>::type;
1438 
1439         template<typename v1>
1440         using minus_t = sub_t<zero, v1>;
1441 
1445         template<typename v1, typename v2>
1446         using mul_t = typename mul<v1, v2>::type;
1447 
1451         template<typename v1, typename v2>
1452         static constexpr bool eq_v = eq_helper<v1, v2>::value;
1453 
1457         template<typename v1, typename v2>
1458         static constexpr bool lt_v = lt_helper<v1, v2>::value;
1459 
1463         template<typename v1, typename v2>
1464         static constexpr bool gt_v = gt_helper<v1, v2>::value;
1465 
1469         template<typename v1, typename v2>
1470         using div_t = typename div<v1, v2>::q_type;
1471 
1475         template<typename v1, typename v2>
1476         using mod_t = typename div_helper<v1, v2, zero, v1>::mod_type;
1477 
1481         template<typename coeff, size_t deg>
1482         using monomial_t = typename monomial<coeff, deg>::type;
1483 
1486         template<typename v>
1487         using derive_t = typename derive_helper<v>::type;
1488 
1491         template<typename v>
1492         static constexpr bool pos_v = Ring::template pos_v<typename v::aN>;
1493 
1497         template<typename v1, typename v2>
1498         using gcd_t = std::conditional_t<
1499             Ring::is_euclidean_domain,
1500             typename make_unit<gcd_t<polynomial<Ring>, v1, v2»::type,
1501             void>;
1502 
1506         template<auto x>
1507         using inject_constant_t = val<typename Ring::template inject_constant_t<x>>;
1508 
1512         template<typename v>
1513         using inject_ring_t = val<v>;
1514     };
1515 }
1516 
1517 // big integers
1518 namespace aerobus {
1519     struct bigint {
1520         enum signs {
1521             positive,
1522             negative
1523         };
1524 
1525         template<signs s, uint32_t an, uint32_t...  as>
1526         struct val;
1527 
1528         template<uint32_t an, uint32_t...  as>
1529         struct to_hex_helper {
1530             static std::string func(const bool prefix = false) {
```

```
1531                 std::string head = prefix ?  std::format("{:08X}", an) :  std::format("{:X}", an);
1532                 return head + to_hex_helper<as...>::func(true);
1533             }
1534         };
1535 
1536         template<uint32_t x>
1537         struct to_hex_helper<x> {
1538             static std::string func(const bool prefix = false) {
1539                 return prefix ?  std::format("{:08X}", x) :  std::format("{:X}", x);
1540             }
1541         };
1542 
1543     private:
1544 
1545         template<signs s>
1546         struct opposite {
1547             static constexpr signs value = s == signs::positive ?  signs::negative :  signs::positive;
1548         };
1549 
1550         template<signs s>
1551         static constexpr signs opposite_v = opposite<s>::value;
1552 
1553         static std::string to_string(const signs& s) {
1554             switch (s) {
1555             case signs::negative:
1556                 return "-";
1557             case signs::positive:
1558             default:
1559                 return "+";
1560             }
1561         }
1562 
1563         template<signs s1, signs s2>
1564         static constexpr signs mul_sign() {
1565             if constexpr (s1 == signs::positive) {
1566                 return s2;
1567             }
1568 
1569             return opposite_v<s2>;
1570         }
1571 
1572         template<size_t ss, signs s, uint32_t aN, uint32_t...  as>
1573         struct shift_left_helper {
1574             using type = typename shift_left_helper<ss - 1, s, aN, as..., 0>::type;
1575         };
1576 
1577         template<signs s, uint32_t aN, uint32_t...  as>
1578         struct shift_left_helper<0, s, aN, as...>
1579         {
1580             using type = val<s, aN, as...>;
1581         };
1582 
1583     public:
1584 
1585         template<signs s, uint32_t an, uint32_t...  as>
1586         struct val {
1587             template<size_t ss>
1588             using shift_left = typename shift_left_helper<ss, s, an, as...>::type;
1589             static constexpr signs sign = s;
1590 
1591             template<size_t index, typename E = void>
1592             struct digit_at {};
1593 
1594             template<size_t index>
1595             struct digit_at<index, std::enable_if_t<(index <= sizeof...(as))>> {
1596                 static constexpr uint32_t value = internal::value_at<(sizeof...(as) - index), an,
     as...>::value;
1597             };
1598 
1599             template<size_t index>
1600             struct digit_at<index, std::enable_if_t<(index > sizeof...(as))>> {
1601                 static constexpr uint32_t value = 0;
1602             };
1603 
1604             using strip = val<s, as...>;
1605             static constexpr uint32_t aN = an;
1606             static constexpr size_t digits = sizeof...(as) + 1;
1607 
1608             static std::string to_string() {
1609                 return bigint::to_string(s) + std::to_string(aN) + "B^" + std::to_string(digits - 1) +
     " + " + strip::to_string();
1610             }
1611 
1612             static std::string to_hex() {
1613                 return bigint::to_string(s) + "0X" + to_hex_helper<an, as...>::func(false);
1614             }
1615 
```

```
1616            static constexpr bool is_zero_v = sizeof...(as) == 0 && an == 0;
1617
1618            using minus_t = val<opposite_v<s>, an, as...>;
1619        };
1620
1621        template<signs s, uint32_t a0>
1622        struct val<s, a0> {
1623            template<size_t ss>
1624            using shift_left = typename shift_left_helper<ss, s, a0>::type;
1625            static constexpr signs sign = s;
1626            static constexpr uint32_t aN = a0;
1627            static constexpr size_t digits = 1;
1628            template<size_t index, typename E = void>
1629            struct digit_at {};
1630            template<size_t index>
1631            struct digit_at<index, std::enable_if_t<index == 0>> {
1632                static constexpr uint32_t value = a0;
1633            };
1634
1635            template<size_t index>
1636            struct digit_at<index, std::enable_if_t<index != 0>> {
1637                static constexpr uint32_t value = 0;
1638            };
1639
1640            static std::string to_string() {
1641                return bigint::to_string(s) + std::to_string(a0);
1642            }
1643
1644            static std::string to_hex() {
1645                return bigint::to_string(s) + std::format("0X{:X}", a0);
1646            }
1647
1648            static constexpr bool is_zero_v = a0 == 0;
1649
1650            using minus_t = val<opposite_v<s>, a0>;
1651
1652        };
1653
1654        using zero = val<signs::positive, 0>;
1655        using one = val<signs::positive, 1>;
1656
1657    private:
1658
1659        template<typename I, typename E = void>
1660        struct simplify {};
1661
1662        template<typename I>
1663        struct simplify<I, std::enable_if_t<I::digits == 1 && I::aN != 0>> {
1664            using type = I;
1665        };
1666
1667        template<typename I>
1668        struct simplify<I, std::enable_if_t<I::digits == 1 && I::aN == 0>> {
1669            using type = zero;
1670        };
1671
1672        template<typename I>
1673        struct simplify<I, std::enable_if_t<I::digits != 1 && I::aN == 0>> {
1674            using type = typename simplify<typename I::strip>::type;
1675        };
1676
1677        template<typename I>
1678        struct simplify<I, std::enable_if_t<I::digits != 1 && I::aN != 0>> {
1679            using type = I;
1680        };
1681
1682        template<uint32_t x, uint32_t y, uint8_t carry_in = 0>
1683        struct add_digit_helper {
1684        private:
1685            static constexpr uint64_t raw = ((uint64_t)x + (uint64_t)y + (uint64_t)carry_in);
1686        public:
1687            static constexpr uint32_t value = (uint32_t)(raw & 0xFFFF'FFFF);
1688            static constexpr uint8_t carry_out = (uint32_t)(raw >> 32);
1689        };
1690
1691        template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1692        struct add_at_helper {
1693        private:
1694            static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1695            static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1696        public:
1697            static constexpr uint32_t value = add_digit_helper<d1, d2, carry_in>::value;
1698            static constexpr uint8_t carry_out = add_digit_helper<d1, d2, carry_in>::carry_out;
1699        };
1700
1701        template<uint32_t x, uint32_t y, uint8_t carry_in, typename E = void>
1702        struct sub_digit_helper {};
```

```
1703
1704        // x - y
1705        template<uint32_t x, uint32_t y, uint8_t carry_in>
1706        struct sub_digit_helper<x, y, carry_in, std::enable_if_t<
1707            (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in) > x)
1708            » {
1709
1710            static constexpr uint32_t value = static_cast<uint32_t>(
1711                static_cast<uint32_t>(x) + 0x1'0000'0000UL - (static_cast<uint64_t>(y) +
    static_cast<uint64_t>(carry_in))
1712                );
1713            static constexpr uint8_t carry_out = 1;
1714        };
1715
1716        template<uint32_t x, uint32_t y, uint8_t carry_in>
1717        struct sub_digit_helper<x, y, carry_in, std::enable_if_t<
1718            (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in) <= x)
1719            » {
1720
1721            static constexpr uint32_t value = static_cast<uint32_t>(
1722                static_cast<uint64_t>(x) - (static_cast<uint64_t>(y) + static_cast<uint64_t>(carry_in))
1723                );
1724            static constexpr uint8_t carry_out = 0;
1725        };
1726
1727        template<typename I1, typename I2, size_t index, uint8_t carry_in = 0>
1728        struct sub_at_helper {
1729        private:
1730            static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1731            static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1732            using tmp = sub_digit_helper<d1, d2, carry_in>;
1733        public:
1734            static constexpr uint32_t value = tmp::value;
1735            static constexpr uint8_t carry_out = tmp::carry_out;
1736        };
1737
1738        template<uint32_t x, uint32_t y, uint32_t carry_in>
1739        struct mul_digit_helper {
1740        private:
1741            static constexpr uint64_t tmp = static_cast<uint64_t>(x) * static_cast<uint64_t>(y) +
    static_cast<uint64_t>(carry_in);
1742        public:
1743            static constexpr uint32_t value = static_cast<uint32_t>(tmp & 0xFFFF'FFFFU);
1744            static constexpr uint32_t carry_out = static_cast<uint32_t>(tmp » 32);
1745        };
1746
1747        template<typename I1, uint32_t d2, size_t index, uint32_t carry_in = 0>
1748        struct mul_at_helper {
1749        private:
1750            static constexpr uint32_t d1 = I1::template digit_at<index>::value;
1751            using tmp = mul_digit_helper<d1, d2, carry_in>;
1752        public:
1753            static constexpr uint32_t value = tmp::value;
1754            static constexpr uint32_t carry_out = tmp::carry_out;
1755        };
1756
1757        template<typename I1, typename I2, size_t index>
1758        struct add_low_helper {
1759        private:
1760            using helper = add_at_helper<I1, I2, index, add_low_helper<I1, I2, index - 1>::carry_out>;
1761        public:
1762            static constexpr uint32_t digit = helper::value;
1763            static constexpr uint8_t carry_out = helper::carry_out;
1764        };
1765
1766        template<typename I1, typename I2>
1767        struct add_low_helper<I1, I2, 0> {
1768            static constexpr uint32_t digit = add_at_helper<I1, I2, 0, 0>::value;
1769            static constexpr uint32_t carry_out = add_at_helper<I1, I2, 0, 0>::carry_out;
1770        };
1771
1772        template<typename I1, typename I2, size_t index>
1773        struct sub_low_helper {
1774        private:
1775            using helper = sub_at_helper<I1, I2, index, sub_low_helper<I1, I2, index - 1>::carry_out>;
1776        public:
1777            static constexpr uint32_t digit = helper::value;
1778            static constexpr uint8_t carry_out = helper::carry_out;
1779        };
1780
1781        template<typename I1, typename I2>
1782        struct sub_low_helper<I1, I2, 0> {
1783            static constexpr uint32_t digit = sub_at_helper<I1, I2, 0, 0>::value;
1784            static constexpr uint32_t carry_out = sub_at_helper<I1, I2, 0, 0>::carry_out;
1785        };
1786
1787        template<typename I1, uint32_t d2, size_t index>
```

```
1788         struct mul_low_helper {
1789         private:
1790             using helper = mul_at_helper<I1, d2, index, mul_low_helper<I1, d2, index - 1>::carry_out>;
1791         public:
1792             static constexpr uint32_t digit = helper::value;
1793             static constexpr uint32_t carry_out = helper::carry_out;
1794         };
1795 
1796         template<typename I1, uint32_t d2>
1797         struct mul_low_helper<I1, d2, 0> {
1798             static constexpr uint32_t digit = mul_at_helper<I1, d2, 0, 0>::value;
1799             static constexpr uint32_t carry_out = mul_at_helper<I1, d2, 0, 0>::carry_out;
1800         };
1801 
1802         template<typename I1, uint32_t d2, typename I>
1803         struct mul_low {};
1804 
1805         template<typename I1, uint32_t d2, std::size_t...  I>
1806         struct mul_low<I1, d2, std::index_sequence<I...>» {
1807             using type = val<signs::positive, mul_low_helper<I1, d2, I>::digit...>;
1808         };
1809 
1810         template<typename I1, uint32_t d2, size_t shift>
1811         struct mul_row_helper {
1812             using type = typename simplify<
1813                 typename mul_low<
1814                 I1,
1815                 d2,
1816                 typename internal::make_index_sequence_reverse<I1::digits + 1>
1817                 >::type>::type::template shift_left<shift>;
1818         };
1819 
1820         template<typename I1, typename I2, size_t index>
1821         struct mul_row {
1822         private:
1823             static constexpr uint32_t d2 = I2::template digit_at<index>::value;
1824         public:
1825             using type = typename mul_row_helper<I1, d2, index>::type;
1826         };
1827 
1828         template<typename I1, typename...  Is>
1829         struct vadd;
1830 
1831         template<typename I1, typename I2, typename E = void>
1832         struct eq;
1833 
1834         template<typename I1, typename I2, typename I>
1835         struct mul_helper {};
1836 
1837         template<typename I1, typename I2, std::size_t...  I>
1838         struct mul_helper<I1, I2, std::index_sequence<I...>» {
1839             using type = typename vadd<typename mul_row<I1, I2, I>::type...>::type;
1840         };
1841 
1842         template<typename I, size_t index>
1843         struct div_2_digit {
1844             static constexpr uint32_t value = ((I::template digit_at<index + 1>::value & 1) « 31) |
     (I::template digit_at<index>::value » 1);
1845         };
1846 
1847         template<typename X, typename I>
1848         struct div_2_helper {};
1849 
1850         template<typename X, std::size_t...  I>
1851         struct div_2_helper<X, std::index_sequence<I...>» {
1852             using type = val<signs::positive, div_2_digit<X, I>::value...>;
1853         };
1854 
1855         template<typename X, typename E = void>
1856         struct div_2 {};
1857 
1858         template<typename X>
1859         struct div_2<X, std::enable_if_t<X::digits == 1>» {
1860             using type = val<X::sign, (X::aN » 1)>;
1861         };
1862 
1863         template<typename X>
1864         struct div_2<X, std::enable_if_t<X::digits != 1>» {
1865             using type = typename simplify<typename div_2_helper<X,
     internal::make_index_sequence_reverse<X::digits»::type>::type;
1866         };
1867 
1868         template<typename I1, typename I2, typename E = void>
1869         struct mul {};
1870 
1871         template<typename I1, typename I2>
1872         struct mul<I1, I2, std::enable_if_t<
```

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

```
1958            std::enable_if_t<
1959            !eq<I1, I2>::value&&
1960            I1::sign == I2::sign &&
1961            I1::sign == signs::negative
1962            » {
1963            static constexpr bool value = gt_helper<typename I2::minus_t, typename I1::minus_t>::value;
1964        };
1965
1966        template<typename I1, typename I2>
1967        struct gt_helper<I1, I2,
1968            std::enable_if_t<
1969            !eq<I1, I2>::value&&
1970            I1::sign == I2::sign &&
1971            I1::sign == signs::positive &&
1972            (I1::digits > I2::digits)
1973            » {
1974            static constexpr bool value = true;
1975        };
1976
1977        template<typename I1, typename I2>
1978        struct gt_helper<I1, I2,
1979            std::enable_if_t<
1980            !eq<I1, I2>::value&&
1981            I1::sign == I2::sign &&
1982            I1::sign == signs::positive &&
1983            (I1::digits < I2::digits)
1984            » {
1985            static constexpr bool value = false;
1986        };
1987
1988        template<typename I1, typename I2>
1989        struct gt_helper<I1, I2,
1990            std::enable_if_t<
1991            !eq<I1, I2>::value&&
1992            I1::sign == I2::sign &&
1993            I1::sign == signs::positive &&
1994            (I1::digits == I2::digits) && I1::digits == 1
1995            » {
1996            static constexpr bool value = I1::aN > I2::aN;
1997        };
1998
1999        template<typename I1, typename I2>
2000        struct gt_helper<I1, I2,
2001            std::enable_if_t<
2002            !eq<I1, I2>::value&&
2003            I1::sign == I2::sign &&
2004            I1::sign == signs::positive &&
2005            (I1::digits == I2::digits) && I1::digits != 1 && (I1::aN > I2::aN)
2006            » {
2007            static constexpr bool value = true;
2008        };
2009
2010        template<typename I1, typename I2>
2011        struct gt_helper<I1, I2,
2012            std::enable_if_t<
2013            !eq<I1, I2>::value&&
2014            I1::sign == I2::sign &&
2015            I1::sign == signs::positive &&
2016            (I1::digits == I2::digits) && I1::digits != 1 && (I1::aN < I2::aN)
2017            » {
2018            static constexpr bool value = false;
2019        };
2020
2021        template<typename I1, typename I2>
2022        struct gt_helper<I1, I2,
2023            std::enable_if_t<
2024            !eq<I1, I2>::value&&
2025            I1::sign == I2::sign &&
2026            I1::sign == signs::positive &&
2027            (I1::digits == I2::digits) && I1::digits != 1 && I1::aN == I2::aN
2028            » {
2029            static constexpr bool value = gt_helper<typename I1::strip, typename I2::strip>::value;
2030        };
2031
2032
2033
2034        template<typename I1, typename I2, typename E = void>
2035        struct add {};
2036
2037        template<typename I1, typename I2, typename E = void>
2038        struct sub {};
2039
2040
2041
2042        // 0 + x -> x
2043        template<typename I1, typename I2>
2044        struct add<I1, I2, std::enable_if_t<
```

```
2045            I1::is_zero_v && !I2::is_zero_v
2046            » {
2047            using type = I2;
2048        };
2049
2050        // x + 0 -> x
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            » {
2055            using type = I1;
2056        };
2057
2058        // 0 + 0 -> x
2059        template<typename I1, typename I2>
2060        struct add<I1, I2, std::enable_if_t<
2061            I2::is_zero_v && I1::is_zero_v
2062            » {
2063            using type = zero;
2064        };
2065
2066        // +x + +y -> x + y
2067        template<typename I1, typename I2>
2068        struct add<I1, I2, std::enable_if_t<
2069            !I2::is_zero_v && !I1::is_zero_v &&
2070            gt_helper<I1, zero>::value &&
2071            gt_helper<I2, zero>::value
2072            » {
2073            using type = typename simplify<
2074                typename add_low<
2075                I1,
2076                I2,
2077                typename internal::make_index_sequence_reverse<std::max(I1::digits, I2::digits) + 1>
2078                >::type>::type;
2079        };
2080
2081        // -x + -y -> -(x+y)
2082        template<typename I1, typename I2>
2083        struct add<I1, I2, std::enable_if_t<
2084            !I2::is_zero_v && !I1::is_zero_v &&
2085            gt_helper<zero, I1>::value &&
2086            gt_helper<zero, I2>::value
2087            » {
2088            using type = typename add<typename I1::minus_t, typename I2::minus_t>::type::minus_t;
2089        };
2090
2091        // x + (-y) -> x - y
2092        template<typename I1, typename I2>
2093        struct add<I1, I2, std::enable_if_t<
2094            !I1::is_zero_v && !I2::is_zero_v &&
2095            gt_helper<I1, zero>::value&&
2096            gt_helper<zero, I2>::value
2097            » {
2098            using type = typename sub<I1, typename I2::minus_t>::type;
2099        };
2100
2101        // -x + y -> y - x
2102        template<typename I1, typename I2>
2103        struct add<I1, I2, std::enable_if_t<
2104            !I1::is_zero_v && !I2::is_zero_v &&
2105            gt_helper<zero, I1>::value&&
2106            gt_helper<I2, zero>::value
2107            » {
2108            using type = typename sub<I2, typename I1::minus_t>::type;
2109        };
2110
2111        // I1 == I2
2112        template<typename I1, typename I2>
2113        struct sub<I1, I2, std::enable_if_t<
2114            eq<I1, I2>::value
2115            » {
2116            using type = zero;
2117        };
2118
2119        // I1 != I2, I2 == 0
2120        template<typename I1, typename I2>
2121        struct sub<I1, I2, std::enable_if_t<
2122            !eq<I1, I2>::value&&
2123            eq<I2, zero>::value
2124            » {
2125            using type = I1;
2126        };
2127
2128        // I1 != I2, I1 == 0
2129        template<typename I1, typename I2>
2130        struct sub<I1, I2, std::enable_if_t<
2131            !eq<I1, I2>::value&&
```

```
2132            eq<I1, zero>::value
2133            » {
2134            using type = typename I2::minus_t;
2135        };
2136
2137        // 0 < I2 < I1
2138        template<typename I1, typename I2>
2139        struct sub<I1, I2, std::enable_if_t<
2140            gt_helper<I2, zero>::value&&
2141            gt_helper<I1, I2>::value
2142            » {
2143            using type = typename simplify<
2144                typename sub_low<
2145                I1,
2146                I2,
2147                typename internal::make_index_sequence_reverse<std::max(I1::digits, I2::digits) + 1>
2148                >::type>::type;
2149        };
2150
2151        // 0 < I1 < I2
2152        template<typename I1, typename I2>
2153        struct sub<I1, I2, std::enable_if_t<
2154            gt_helper<I1, zero>::value&&
2155            gt_helper<I2, I1>::value
2156            » {
2157            using type = typename sub<I2, I1>::type::minus_t;
2158        };
2159
2160        // I2 < I1 < 0
2161        template<typename I1, typename I2>
2162        struct sub<I1, I2, std::enable_if_t<
2163            gt_helper<zero, I1>::value&&
2164            gt_helper<I1, I2>::value
2165            » {
2166            using type = typename sub<typename I2::minus_t, typename I1::minus_t>::type;
2167        };
2168
2169        // I1 < I2 < 0
2170        template<typename I1, typename I2>
2171        struct sub<I1, I2, std::enable_if_t<
2172            gt_helper<zero, I2>::value&&
2173            gt_helper<I2, I1>::value
2174            » {
2175            using type = typename sub<typename I1::minus_t, typename I2::minus_t>::type::minus_t;
2176        };
2177
2178        // I2 < 0 < I1
2179        template<typename I1, typename I2>
2180        struct sub<I1, I2, std::enable_if_t<
2181            gt_helper<zero, I2>::value&&
2182            gt_helper<I1, zero>::value
2183            » {
2184            using type = typename add<I1, typename I2::minus_t>::type;
2185        };
2186
2187        // I1 < 0 < I2
2188        template<typename I1, typename I2>
2189        struct sub<I1, I2, std::enable_if_t<
2190            gt_helper<zero, I1>::value&&
2191            gt_helper<I2, zero>::value
2192            » {
2193            using type = typename add<I2, typename I1::minus_t>::type::minus_t;
2194        };
2195
2196        // useful for multiplication
2197        template<typename I1, typename...  Is>
2198        struct vadd {
2199            using type = typename add<I1, typename vadd<Is...>::type>::type;
2200        };
2201
2202        template<typename I1, typename I2>
2203        struct vadd<I1, I2> {
2204            using type = typename add<I1, I2>::type;
2205        };
2206
2207        template<typename I, size_t s, typename E = void>
2208        struct shift_right_helper { };
2209
2210        template<typename I, size_t s>
2211        struct shift_right_helper<I, s, std::enable_if_t<(s >= I::digits)» {
2212            using type = zero;
2213        };
2214
2215        template<typename I, size_t s>
2216        struct shift_right_helper<I, s, std::enable_if_t<(s == 0)» {
2217            using type = I;
2218        };
```

```
2219
2220        template<typename I, size_t s>
2221        struct shift_right_helper<I, s, std::enable_if_t<(s != 0) && (s < I::digits)» {
2222        private:
2223            using digit = val<I::sign, I::template digit_at<s>::value>;
2224            using tmp = typename shift_right_helper<I, s + 1>::type;
2225        public:
2226            using type = typename add<
2227                digit,
2228                typename tmp::template shift_left<1>
2229            >::type;
2230        };
2231
2232        template<typename A, typename B, typename E = void>
2233        struct floor_helper {};
2234
2235        template<typename A, typename B>
2236        struct floor_helper<A, B, std::enable_if_t<gt_helper<B, A>::value» {
2237            using type = zero;
2238        };
2239
2240        template<typename A, typename B>
2241        struct floor_helper<A, B, std::enable_if_t<eq<A, B>::value» {
2242            using type = one;
2243        };
2244
2245        template<typename A, typename B>
2246        struct floor_helper<A, B, std::enable_if_t<gt_helper<A, B>::value && (A::digits == 1 &&
    B::digits == 1)» {
2247            using type = val<signs::positive, A::aN / B::aN>;
2248        };
2249
2250        template<typename A, typename B>
2251        struct floor_helper<A, B, std::enable_if_t<gt_helper<A, B>::value && (A::digits != 1 ||
    B::digits != 1)» {
2252            template<typename X, typename Y>
2253            using average_t = typename div_2<typename add<X, Y>::type>::type;
2254
2255            template<typename lowerbound, typename upperbound, typename E = void>
2256            struct inner {};
2257
2258            template<typename lowerbound, typename upperbound>
2259            struct inner<lowerbound, upperbound, std::enable_if_t<eq<
2260                typename add<lowerbound, one>::type, upperbound>::value
2261                » {
2262                using type = lowerbound;
2263            };
2264
2265            template<typename lowerbound, typename upperbound>
2266            struct inner<lowerbound, upperbound, std::enable_if_t<
2267                gt_helper<upperbound, typename add<lowerbound, one>::type>::value&&
2268                gt_helper<typename mul<average_t<upperbound, lowerbound>, B>::type, A>::value
2269                » {
2270                using type = typename simplify<typename inner<lowerbound, average_t<upperbound,
    lowerbound»::type>::type;
2271            };
2272
2273            template<typename lowerbound, typename upperbound>
2274            struct inner<lowerbound, upperbound, std::enable_if_t<
2275                gt_helper<upperbound, typename add<lowerbound, one>::type>::value &&
2276                !gt_helper<typename mul<average_t<upperbound, lowerbound>, B>::type, A>::value
2277                » {
2278                using type = typename simplify<typename inner<average_t<upperbound, lowerbound>,
    upperbound>::type>::type;
2279            };
2280
2281            // this type is ONLY used for division where we know this bound
2282            using type = typename inner<zero, val<signs::positive, 1, 1»::type;
2283        };
2284
2285        template<typename N, typename M, int64_t i>
2286        struct div_helper_inner {
2287            static_assert(N::sign == signs::positive);
2288            static_assert(M::sign == signs::positive);
2289            static constexpr size_t l = M::digits;
2290            static constexpr size_t k = N::digits;
2291            using Qm1 = typename simplify<typename div_helper_inner<N, M, i - 1>::Q>::type;
2292            using Rm1 = typename simplify<typename div_helper_inner<N, M, i - 1>::R>::type;
2293            using D = typename simplify<typename add<
2294                typename Rm1::template shift_left<1>,
2295                val<signs::positive, N::template digit_at<k - (i + l)>::value>
2296            >::type>::type;
2297            using Beta = typename simplify<typename floor_helper<D, M>::type>::type;
2298            using Q = typename simplify<typename add<typename Qm1::template shift_left<1>,
    Beta>::type>::type;
2299
2300            using R = typename simplify<typename sub<D, typename mul<M, Beta>::type>::type>::type;
```

```
2301         };
2302
2303         template<typename N, typename M>
2304         struct div_helper_inner<N, M, -1> {
2305             static_assert(N::sign == signs::positive);
2306             static_assert(M::sign == signs::positive);
2307             static constexpr size_t l = M::digits;
2308             static constexpr size_t k = N::digits;
2309             using Q = zero;
2310             using R = typename shift_right_helper<N, k - l + 1>::type; // first l-1 digits of N
2311         };
2312
2313         template<typename N, typename M, typename E = void>
2314         struct div_helper {};
2315
2316         template<typename N, typename M>
2317         struct div_helper<N, M, std::enable_if_t<
2318             M::sign == signs::positive &&
2319             N::sign == signs::positive &&
2320             !M::is_zero_v
2321             » {
2322             static constexpr size_t l = M::digits;
2323             static constexpr size_t k = N::digits;
2324             using Q = typename simplify<typename div_helper_inner<N, M, k - l>::Q>::type;
2325             using R = typename simplify<typename div_helper_inner<N, M, k - l>::R>::type;
2326         };
2327
2328         template<typename N, typename M>
2329         struct div_helper<N, M, std::enable_if_t<
2330             M::sign == signs::negative &&
2331             !M::is_zero_v
2332             » {
2333             using tmp = div_helper<N, typename M::minus_t>;
2334             using Q = typename tmp::Q::minus_t;
2335             using R = typename tmp::R;
2336         };
2337
2338         template<typename N, typename M>
2339         struct div_helper<N, M, std::enable_if_t<
2340             N::sign == signs::negative &&
2341             !M::is_zero_v
2342             » {
2343             using tmp = div_helper<typename N::minus_t, M>;
2344             using R_i = typename simplify<typename tmp::R>::type;
2345             using Q_i = typename simplify<typename tmp::Q>::type;
2346             using Q = std::conditional_t<R_i::is_zero_v, typename Q_i::minus_t, typename sub<typename
     Q_i::minus_t, one>::type>;
2347             using R = std::conditional_t<R_i::is_zero_v, zero, typename sub<M, R_i>::type>;
2348         };
2349
2350         template<string_literal S>
2351         struct digit_from_string {
2352             static constexpr size_t N = S.len();
2353
2354             template<size_t i>
2355             static constexpr char char_at = (i + 1 < N) ? S.template char_at<i>() :  '0';
2356
2357             template <char c>
2358             static constexpr uint32_t from_hex = (c >= '0' && c <= '9') ?  c - '0' :  10 + c - 'A';
2359
2360             template<size_t index>
2361             static constexpr uint32_t value() {
2362                 constexpr uint32_t d1 = from_hex<char_at<8 * index»;
2363                 constexpr uint32_t d2 = from_hex<char_at<8 * index + 1» « 4;
2364                 constexpr uint32_t d3 = from_hex<char_at<8 * index + 2» « 8;
2365                 constexpr uint32_t d4 = from_hex<char_at<8 * index + 3» « 12;
2366                 constexpr uint32_t d5 = from_hex<char_at<8 * index + 4» « 16;
2367                 constexpr uint32_t d6 = from_hex<char_at<8 * index + 5» « 20;
2368                 constexpr uint32_t d7 = from_hex<char_at<8 * index + 6» « 24;
2369                 constexpr uint32_t d8 = from_hex<char_at<8 * index + 7» « 28;
2370                 return d1 | d2 | d3 | d4 | d5 | d6 | d7 | d8;
2371             }
2372         };
2373
2374         template<string_literal S, typename I>
2375         struct from_hex_helper {};
2376
2377         template<string_literal S, std::size_t...  I>
2378         struct from_hex_helper<S, std::index_sequence<I...» {
2379             using type = typename simplify<val<signs::positive, digit_from_string<S>::template
     value<I>()...»::type;
2380         };
2381
2382     public:
2383         static constexpr bool is_euclidean_domain = true;
2384         static constexpr bool is_field = false;
2385
```

```
2386         template<typename v>
2387         using inject_ring_t = v;
2388 
2389         template<auto v>
2390         using inject_constant_t = val<(v < 0) ?  bigint::signs::negative :  bigint::signs::positive, (v
     >= 0 ?  v :  -v)>;
2391 
2394         template<string_literal S>
2395         using from_hex_t = typename from_hex_helper<S, internal::make_index_sequence_reverse<(S.len() -
     1) / 8 + 1>>::type;
2396 
2398         template<typename I>
2399         using minus_t = typename I::minus_t;
2400 
2402         template<typename I1, typename I2>
2403         static constexpr bool eq_v = eq<I1, I2>::value;
2404 
2406         template<typename I>
2407         static constexpr bool pos_v = I::sign == signs::positive && !I::is_zero_v;
2408 
2410         template<typename I1, typename I2>
2411         static constexpr bool gt_v = gt_helper<I1, I2>::value;
2412 
2414         template<typename I1, typename I2>
2415         static constexpr bool ge_v = eq_v<I1, I2> || gt_v<I1, I2>;
2416 
2418         template<typename I>
2419         using simplify_t = typename simplify<I>::type;
2420 
2422         template<typename I1, typename I2>
2423         using add_t = typename add<I1, I2>::type;
2424 
2426         template<typename I1, typename I2>
2427         using sub_t = typename sub<I1, I2>::type;
2428 
2430         template<typename I, size_t s>
2431         using shift_left_t = typename I::template shift_left<s>;
2432 
2434         template<typename I, size_t s>
2435         using shift_right_t = typename shift_right_helper<I, s>::type;
2436 
2438         template<typename I1, typename I2>
2439         using mul_t = typename mul<I1, I2>::type;
2440 
2442         template<typename...  Is>
2443         using vadd_t = typename vadd<Is...>::type;
2444 
2446         template<typename I>
2447         using div_2_t = typename div_2<I>::type;
2448 
2450         template<typename I1, typename I2>
2451         using div_t = typename div_helper<I1, I2>::Q;
2452 
2454         template<typename I1, typename I2>
2455         using mod_t = typename div_helper<I1, I2>::R;
2456 
2458         template<typename I1, typename I2>
2459         using gcd_t = gcd_t<bigint, I1, I2>;
2460 
2462         template<typename I1, typename I2, typename I3>
2463         using fma_t = add_t<mul_t<I1, I2>, I3>;
2464 
2465 
2466     };
2467 }
2468 
2469 // fraction field
2470 namespace aerobus {
2471     namespace internal {
2472         template<typename Ring, typename E = void>
2473             requires IsEuclideanDomain<Ring>
2474         struct _FractionField {};
2475 
2476         template<typename Ring>
2477             requires IsEuclideanDomain<Ring>
2478         struct _FractionField<Ring, std::enable_if_t<Ring::is_euclidean_domain>>
2479         {
2481             static constexpr bool is_field = true;
2482             static constexpr bool is_euclidean_domain = true;
2483 
2484         private:
2485             template<typename val1, typename val2, typename E = void>
2486             struct to_string_helper {};
2487 
2488             template<typename val1, typename val2>
2489             struct to_string_helper <val1, val2,
2490                 std::enable_if_t<
```

```
2491                Ring::template eq_v<val2, typename Ring::one>
2492                » {
2493                static std::string func() {
2494                    return val1::to_string();
2495                }
2496            };
2497
2498            template<typename val1, typename val2>
2499            struct to_string_helper<val1, val2,
2500                std::enable_if_t<
2501                !Ring::template eq_v<val2, typename Ring::one>
2502                » {
2503                static std::string func() {
2504                    return "(" + val1::to_string() + ") / (" + val2::to_string() + ")";
2505                }
2506            };
2507
2508        public:
2512            template<typename val1, typename val2>
2513            struct val {
2514                using x = val1;
2515                using y = val2;
2516
2518                static constexpr bool is_zero_v = val1::is_zero_v;
2519                using ring_type = Ring;
2520                using field_type = _FractionField<Ring>;
2521
2523                static constexpr bool is_integer = std::is_same<val2, typename Ring::one>::value;
2524
2528                template<typename valueType>
2529                DEVICE INLINED static constexpr valueType get() { return static_cast<valueType>(x::v) /
    static_cast<valueType>(y::v); }
2530
2533                static std::string to_string() {
2534                    return to_string_helper<val1, val2>::func();
2535                }
2536
2541                template<typename valueRing>
2542                static constexpr valueRing eval(const valueRing& v) {
2543                    return x::eval(v) / y::eval(v);
2544                }
2545            };
2546
2548            using zero = val<typename Ring::zero, typename Ring::one>;
2550            using one = val<typename Ring::one, typename Ring::one>;
2551
2554            template<typename v>
2555            using inject_t = val<v, typename Ring::one>;
2556
2559            template<auto x>
2560            using inject_constant_t = val<typename Ring::template inject_constant_t<x>, typename
    Ring::one>;
2561
2564            template<typename v>
2565            using inject_ring_t = val<typename Ring::template inject_ring_t<v>, typename Ring::one>;
2566
2567            using ring_type = Ring;
2568
2569        private:
2570            template<typename v, typename E = void>
2571            struct simplify {};
2572
2573            // x = 0
2574            template<typename v>
2575            struct simplify<v, std::enable_if_t<v::x::is_zero_v» {
2576                using type = typename _FractionField<Ring>::zero;
2577            };
2578
2579            // x != 0
2580            template<typename v>
2581            struct simplify<v, std::enable_if_t<!v::x::is_zero_v» {
2582
2583            private:
2584                using _gcd = typename Ring::template gcd_t<typename v::x, typename v::y>;
2585                using newx = typename Ring::template div_t<typename v::x, _gcd>;
2586                using newy = typename Ring::template div_t<typename v::y, _gcd>;
2587
2588                using posx = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
    minus_t<newx>, newx>;
2589                using posy = std::conditional_t<!Ring::template pos_v<newy>, typename Ring::template
    minus_t<newy>, newy>;
2590            public:
2591                using type = typename _FractionField<Ring>::template val<posx, posy>;
2592            };
2593
2594        public:
2597            template<typename v>
```

```
2598            using simplify_t = typename simplify<v>::type;
2599
2600        private:
2601
2602            template<typename v1, typename v2>
2603            struct add {
2604            private:
2605                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2606                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2607                using dividend = typename Ring::template add_t<a, b>;
2608                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2609                using g = typename Ring::template gcd_t<dividend, diviser>;
2610
2611            public:
2612                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
2613            };
2614
2615            template<typename v>
2616            struct pos {
2617                static constexpr bool value =
2618                    (Ring::template pos_v<typename v::x> && Ring::template pos_v<typename v::y>) ||
2619                    (!Ring::template pos_v<typename v::x> && !Ring::template pos_v<typename v::y>);
2620
2621            };
2622
2623            template<typename v1, typename v2>
2624            struct sub {
2625            private:
2626                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2627                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2628                using dividend = typename Ring::template sub_t<a, b>;
2629                using diviser = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2630                using g = typename Ring::template gcd_t<dividend, diviser>;
2631
2632            public:
2633                using type = typename _FractionField<Ring>::template simplify_t<val<dividend, diviser»;
2634            };
2635
2636            template<typename v1, typename v2>
2637            struct mul {
2638            private:
2639                using a = typename Ring::template mul_t<typename v1::x, typename v2::x>;
2640                using b = typename Ring::template mul_t<typename v1::y, typename v2::y>;
2641
2642            public:
2643                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
2644            };
2645
2646            template<typename v1, typename v2, typename E = void>
2647            struct div {};
2648
2649            template<typename v1, typename v2>
2650            struct div<v1, v2, std::enable_if_t<!std::is_same<v2, typename
    _FractionField<Ring>::zero>::value» {
2651            private:
2652                using a = typename Ring::template mul_t<typename v1::x, typename v2::y>;
2653                using b = typename Ring::template mul_t<typename v1::y, typename v2::x>;
2654
2655            public:
2656                using type = typename _FractionField<Ring>::template simplify_t<val<a, b»;
2657            };
2658
2659            template<typename v1, typename v2>
2660            struct div<v1, v2, std::enable_if_t<
2661                std::is_same<zero, v1>::value&& std::is_same<v2, zero>::value»  {
2662                using type = one;
2663            };
2664
2665            template<typename v1, typename v2>
2666            struct eq {
2667                static constexpr bool value =
2668                    std::is_same<typename simplify_t<v1>::x, typename simplify_t<v2>::x>::value&&
2669                    std::is_same<typename simplify_t<v1>::y, typename simplify_t<v2>::y>::value;
2670            };
2671
2672            template<typename TL, typename E = void>
2673            struct vadd {};
2674
2675            template<typename TL>
2676            struct vadd<TL, std::enable_if_t<(TL::length > 1)» {
2677                using head = typename TL::pop_front::type;
2678                using tail = typename TL::pop_front::tail;
2679                using type = typename add<head, typename vadd<tail>::type>::type;
2680            };
2681
2682            template<typename TL>
2683            struct vadd<TL, std::enable_if_t<(TL::length == 1)» {
```

```
2684                using type = typename TL::template at<0>;
2685            };
2686
2687            template<typename...  vals>
2688            struct vmul {};
2689
2690            template<typename v1, typename...  vals>
2691            struct vmul<v1, vals...> {
2692                using type = typename mul<v1, typename vmul<vals...>::type>::type;
2693            };
2694
2695            template<typename v1>
2696            struct vmul<v1> {
2697                using type = v1;
2698            };
2699
2700
2701            template<typename v1, typename v2, typename E = void>
2702            struct gt;
2703
2704            template<typename v1, typename v2>
2705            struct gt<v1, v2, std::enable_if_t<
2706                (eq<v1, v2>::value)
2707                » {
2708                static constexpr bool value = false;
2709            };
2710
2711            template<typename v1, typename v2>
2712            struct gt<v1, v2, std::enable_if_t<
2713                (!eq<v1, v2>::value) &&
2714                (!pos<v1>::value) && (!pos<v2>::value)
2715                » {
2716                static constexpr bool value = gt<
2717                    typename sub<zero, v1>::type, typename sub<zero, v2>::type
2718                >::value;
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 = true;
2727            };
2728
2729            template<typename v1, typename v2>
2730            struct gt<v1, v2, std::enable_if_t<
2731                (!eq<v1, v2>::value) &&
2732                (!pos<v1>::value) && (pos<v2>::value)
2733                » {
2734                static constexpr bool value = false;
2735            };
2736
2737            template<typename v1, typename v2>
2738            struct gt<v1, v2, std::enable_if_t<
2739                (!eq<v1, v2>::value) &&
2740                (pos<v1>::value) && (pos<v2>::value)
2741                » {
2742                static constexpr bool value = Ring::template gt_v<
2743                    typename Ring::template mul_t<v1::x, v2::y>,
2744                    typename Ring::template mul_t<v2::y, v2::x>
2745                >;
2746            };
2747
2748        public:
2749
2751            template<typename v1, typename v2>
2752            using add_t = typename add<v1, v2>::type;
2753
2755            template<typename v1, typename v2>
2756            using mod_t = zero;
2757
2761            template<typename v1, typename v2>
2762            using gcd_t = v1;
2763
2766            template<typename...  vs>
2767            using vadd_t = typename vadd<vs...>::type;
2768
2771            template<typename...  vs>
2772            using vmul_t = typename vmul<vs...>::type;
2773
2775            template<typename v1, typename v2>
2776            using sub_t = typename sub<v1, v2>::type;
2777
2778            template<typename v>
2779            using minus_t = sub_t<zero, v>;
2780
```

```
2782             template<typename v1, typename v2>
2783             using mul_t = typename mul<v1, v2>::type;
2784 
2786             template<typename v1, typename v2>
2787             using div_t = typename div<v1, v2>::type;
2788 
2790             template<typename v1, typename v2>
2791             static constexpr bool eq_v = eq<v1, v2>::value;
2792 
2794             template<typename v1, typename v2>
2795             static constexpr bool gt_v = gt<v1, v2>::value;
2796 
2798             template<typename v>
2799             static constexpr bool pos_v = pos<v>::value;
2800         };
2801 
2802         template<typename Ring, typename E = void>
2803             requires IsEuclideanDomain<Ring>
2804         struct FractionFieldImpl {};
2805 
2806         // fraction field of a field is the field itself
2807         template<typename Field>
2808             requires IsEuclideanDomain<Field>
2809         struct FractionFieldImpl<Field, std::enable_if_t<Field::is_field» {
2810             using type = Field;
2811             template<typename v>
2812             using inject_t = v;
2813         };
2814 
2815         // fraction field of a ring is the actual fraction field
2816         template<typename Ring>
2817             requires IsEuclideanDomain<Ring>
2818         struct FractionFieldImpl<Ring, std::enable_if_t<!Ring::is_field» {
2819             using type = _FractionField<Ring>;
2820         };
2821     }
2822 
2823     template<typename Ring>
2824         requires IsEuclideanDomain<Ring>
2825     using FractionField = typename internal::FractionFieldImpl<Ring>::type;
2826 }
2827 
2828 // short names for common types
2829 namespace aerobus {
2831     using q32 = FractionField<i32>;
2833     using fpq32 = FractionField<polynomial<q32»;
2835     using q64 = FractionField<i64>;
2837     using pi64 = polynomial<i64>;
2839     using pq64 = polynomial<q64>;
2841     using fpq64 = FractionField<polynomial<q64»;
2842 
2845     template<uint32_t...  digits>
2846     using bigint_pos = bigint::template val<bigint::signs::positive, digits...>;
2849     template<uint32_t...  digits>
2850     using bigint_neg = bigint::template val<bigint::signs::negative, digits...>;
2851 
2856     template<typename Ring, typename v1, typename v2>
2857     using makefraction_t = typename FractionField<Ring>::template val<v1, v2>;
2858 
2859     template<typename Ring, typename v1, typename v2>
2860     using addfractions_t = typename FractionField<Ring>::template add_t<v1, v2>;
2861     template<typename Ring, typename v1, typename v2>
2862     using mulfractions_t = typename FractionField<Ring>::template mul_t<v1, v2>;
2863 }
2864 
2865 // taylor series and common integers (factorial, bernouilli...)  appearing in taylor coefficients
2866 namespace aerobus {
2867     namespace internal {
2868         template<typename T, size_t x, typename E = void>
2869         struct factorial {};
2870 
2871         template<typename T, size_t x>
2872         struct factorial<T, x, std::enable_if_t<(x > 0)» {
2873         private:
2874             template<typename, size_t, typename>
2875             friend struct factorial;
2876         public:
2877             using type = typename T::template mul_t<typename T::template val<x>, typename factorial<T,
      x - 1>::type>;
2878             static constexpr typename T::inner_type value = type::template get<typename
     T::inner_type>();
2879         };
2880 
2881         template<typename T>
2882         struct factorial<T, 0> {
2883         public:
2884             using type = typename T::one;
```

```
2885            static constexpr typename T::inner_type value = type::template get<typename
    T::inner_type>();
2886        };
2887
2888        template<typename T, size_t k, size_t n, typename E = void>
2889        struct combination_helper {};
2890
2891        template<typename T, size_t k, size_t n>
2892        struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k <= (n / 2) && k > 0)» {
2893            using type = typename FractionField<T>::template mul_t<
2894                typename combination_helper<T, k - 1, n - 1>::type,
2895                makefraction_t<T, typename T::template val<n>, typename T::template val<k»>;
2896        };
2897
2898        template<typename T, size_t k, size_t n>
2899        struct combination_helper<T, k, n, std::enable_if_t<(n >= 0 && k > (n / 2) && k > 0)» {
2900            using type = typename combination_helper<T, n - k, n>::type;
2901        };
2902
2903        template<typename T, size_t n>
2904        struct combination_helper<T, 0, n> {
2905            using type = typename FractionField<T>::one;
2906        };
2907
2908        template<typename T, size_t k, size_t n>
2909        struct combination {
2910            using type = typename internal::combination_helper<T, k, n>::type::x;
2911            static constexpr typename T::inner_type value = internal::combination_helper<T, k,
    n>::type::template get<typename T::inner_type>();
2912        };
2913
2914        template<typename T, size_t m>
2915        struct bernouilli;
2916
2917        template<typename T, typename accum, size_t k, size_t m>
2918        struct bernouilli_helper {
2919            using type = typename bernouilli_helper<
2920                T,
2921                addfractions_t<T,
2922                    accum,
2923                    mulfractions_t<T,
2924                        makefraction_t<T,
2925                            typename combination<T, k, m + 1>::type,
2926                            typename T::one>,
2927                        typename bernouilli<T, k>::type
2928                    >
2929                >,
2930                k + 1,
2931                m>::type;
2932        };
2933
2934        template<typename T, typename accum, size_t m>
2935        struct bernouilli_helper<T, accum, m, m>
2936        {
2937            using type = accum;
2938        };
2939
2940        template<typename T, size_t m>
2941        struct bernouilli {
2942            using type = typename FractionField<T>::template mul_t<
2943                typename internal::bernouilli_helper<T, typename FractionField<T>::zero, 0, m>::type,
2944                makefraction_t<T,
2945                typename T::template val<static_cast<typename T::inner_type>(-1)>,
2946                typename T::template val<static_cast<typename T::inner_type>(m + 1)>
2947                >
2948            >;
2949
2950            template<typename floatType>
2951            static constexpr floatType value = type::template get<floatType>();
2952        };
2953
2954        template<typename T>
2955        struct bernouilli<T, 0> {
2956            using type = typename FractionField<T>::one;
2957
2958            template<typename floatType>
2959            static constexpr floatType value = type::template get<floatType>();
2960        };
2961
2962        // -1^k
2963        template<typename T, int k, typename E = void>
2964        struct alternate {};
2965
2966        template<typename T, int k>
2967        struct alternate<T, k, std::enable_if_t<k % 2 == 0» {
2968            using type = typename T::one;
2969            static constexpr typename T::inner_type value = type::template get<typename
```

```
    T::inner_type>();
2970        };
2971
2972        template<typename T, int k>
2973        struct alternate<T, k, std::enable_if_t<k % 2 != 0» {
2974            using type = typename T::template minus_t<typename T::one>;
2975            static constexpr typename T::inner_type value = type::template get<typename
    T::inner_type>();
2976        };
2977
2978        // pow
2979        template<typename T, auto p, auto n>
2980        struct pow {
2981            using type = typename T::template mul_t<typename T::template val<p>, typename pow<T, p, n -
    1>::type>;
2982        };
2983
2984        template<typename T, auto p>
2985        struct pow<T, p, 0> { using type = typename T::one; };
2986    }
2987
2990    template<typename T, size_t k, size_t n>
2991    using combination_t = typename internal::combination<T, k, n>::type;
2992
2995    template<typename T, size_t k, size_t n>
2996    constexpr typename T::inner_type combination_v = internal::combination<T, k, n>::value;
2997
3001    template<typename T, size_t n>
3002    using bernouilli_t = typename internal::bernouilli<T, n>::type;
3003
3008    template<typename FloatType, typename T, size_t n >
3009    constexpr FloatType bernouilli_v = internal::bernouilli<T, n>::template value<FloatType>;
3010
3013    template<typename T, int k>
3014    using alternate_t = typename internal::alternate<T, k>::type;
3015
3018    template<typename T, size_t k>
3019    constexpr typename T::inner_type alternate_v = internal::alternate<T, k>::value;
3020
3024    template<typename T, size_t i>
3025    using factorial_t = typename internal::factorial<T, i>::type;
3026
3030    template<typename T, size_t i>
3031    constexpr typename T::inner_type factorial_v = internal::factorial<T, i>::value;
3032
3033    template<typename T, auto p, auto n>
3034    using pow_t = typename internal::pow<T, p, n>::type;
3035
3036    namespace internal {
3037        template<typename, template<typename, size_t> typename, class>
3038        struct make_taylor_impl;
3039
3040        template<typename T, template<typename, size_t> typename coeff_at, size_t...  Is>
3041        struct make_taylor_impl<T, coeff_at, std::integer_sequence<size_t, Is...» {
3042            using type = typename polynomial<FractionField<T»::template val<typename coeff_at<T,
    Is>::type...>;
3043        };
3044    }
3045
3046    // generic taylor serie, depending on coefficients
3047    template<typename T, template<typename, size_t index> typename coeff_at, size_t deg>
3048    using taylor = typename internal::make_taylor_impl<T, coeff_at,
    internal::make_index_sequence_reverse<deg + 1»::type;
3049
3050    namespace internal {
3051        template<typename T, size_t i>
3052        struct exp_coeff {
3053            using type = makefraction_t<T, typename T::one, factorial_t<T, i»;
3054        };
3055
3056        template<typename T, size_t i, typename E = void>
3057        struct sin_coeff_helper {};
3058
3059        template<typename T, size_t i>
3060        struct sin_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 sin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
3066            using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i»;
3067        };
3068
3069        template<typename T, size_t i>
3070        struct sin_coeff {
3071            using type = typename sin_coeff_helper<T, i>::type;
3072        };
```

```
3073
3074        template<typename T, size_t i, typename E = void>
3075        struct sh_coeff_helper {};
3076
3077        template<typename T, size_t i>
3078        struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
3079            using type = typename FractionField<T>::zero;
3080        };
3081
3082        template<typename T, size_t i>
3083        struct sh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
3084            using type = makefraction_t<T, typename T::one, factorial_t<T, i»;
3085        };
3086
3087        template<typename T, size_t i>
3088        struct sh_coeff {
3089            using type = typename sh_coeff_helper<T, i>::type;
3090        };
3091
3092        template<typename T, size_t i, typename E = void>
3093        struct cos_coeff_helper {};
3094
3095        template<typename T, size_t i>
3096        struct cos_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 cos_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
3102            using type = makefraction_t<T, alternate_t<T, i / 2>, factorial_t<T, i»;
3103        };
3104
3105        template<typename T, size_t i>
3106        struct cos_coeff {
3107            using type = typename cos_coeff_helper<T, i>::type;
3108        };
3109
3110        template<typename T, size_t i, typename E = void>
3111        struct cosh_coeff_helper {};
3112
3113        template<typename T, size_t i>
3114        struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
3115            using type = typename FractionField<T>::zero;
3116        };
3117
3118        template<typename T, size_t i>
3119        struct cosh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
3120            using type = makefraction_t<T, typename T::one, factorial_t<T, i»;
3121        };
3122
3123        template<typename T, size_t i>
3124        struct cosh_coeff {
3125            using type = typename cosh_coeff_helper<T, i>::type;
3126        };
3127
3128        template<typename T, size_t i>
3129        struct geom_coeff { using type = typename FractionField<T>::one; };
3130
3131
3132        template<typename T, size_t i, typename E = void>
3133        struct atan_coeff_helper;
3134
3135        template<typename T, size_t i>
3136        struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1» {
3137            using type = makefraction_t<T, alternate_t<T, i / 2>, typename T::template val<i»;
3138        };
3139
3140        template<typename T, size_t i>
3141        struct atan_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0» {
3142            using type = typename FractionField<T>::zero;
3143        };
3144
3145        template<typename T, size_t i>
3146        struct atan_coeff { using type = typename atan_coeff_helper<T, i>::type; };
3147
3148        template<typename T, size_t i, typename E = void>
3149        struct asin_coeff_helper;
3150
3151        template<typename T, size_t i>
3152        struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
3153        {
3154            using type = makefraction_t<T,
3155                factorial_t<T, i - 1>,
3156                typename T::template mul_t<
3157                typename T::template val<i>,
3158                T::template mul_t<
3159                pow_t<T, 4, i / 2>,
```

```
3160                pow<T, factorial<T, i / 2>::value, 2
3161                >
3162                >
3163                »;
3164        };
3165
3166        template<typename T, size_t i>
3167        struct asin_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
3168        {
3169            using type = typename FractionField<T>::zero;
3170        };
3171
3172        template<typename T, size_t i>
3173        struct asin_coeff {
3174            using type = typename asin_coeff_helper<T, i>::type;
3175        };
3176
3177        template<typename T, size_t i>
3178        struct lnp1_coeff {
3179            using type = makefraction_t<T,
3180                alternate_t<T, i + 1>,
3181                typename T::template val<i»;
3182        };
3183
3184        template<typename T>
3185        struct lnp1_coeff<T, 0> { using type = typename FractionField<T>::zero; };
3186
3187        template<typename T, size_t i, typename E = void>
3188        struct asinh_coeff_helper;
3189
3190        template<typename T, size_t i>
3191        struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
3192        {
3193            using type = makefraction_t<T,
3194                typename T::template mul_t<
3195                alternate_t<T, i / 2>,
3196                factorial_t<T, i - 1>
3197                >,
3198                typename T::template mul_t<
3199                T::template mul_t<
3200                typename T::template val<i>,
3201                pow_t<T, (factorial<T, i / 2>::value), 2>
3202                >,
3203                pow_t<T, 4, i / 2>
3204                >
3205            >;
3206        };
3207
3208        template<typename T, size_t i>
3209        struct asinh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
3210        {
3211            using type = typename FractionField<T>::zero;
3212        };
3213
3214        template<typename T, size_t i>
3215        struct asinh_coeff {
3216            using type = typename asinh_coeff_helper<T, i>::type;
3217        };
3218
3219        template<typename T, size_t i, typename E = void>
3220        struct atanh_coeff_helper;
3221
3222        template<typename T, size_t i>
3223        struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 1»
3224        {
3225            // 1/i
3226            using type = typename FractionField<T>::  template val<
3227                typename T::one,
3228                typename T::template val<static_cast<typename T::inner_type>(i)»;
3229        };
3230
3231        template<typename T, size_t i>
3232        struct atanh_coeff_helper<T, i, std::enable_if_t<(i & 1) == 0»
3233        {
3234            using type = typename FractionField<T>::zero;
3235        };
3236
3237        template<typename T, size_t i>
3238        struct atanh_coeff {
3239            using type = typename asinh_coeff_helper<T, i>::type;
3240        };
3241
3242        template<typename T, size_t i, typename E = void>
3243        struct tan_coeff_helper;
3244
3245        template<typename T, size_t i>
3246        struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
```

```
3247            using type = typename FractionField<T>::zero;
3248        };
3249
3250        template<typename T, size_t i>
3251        struct tan_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
3252        private:
3253            // 4^((i+1)/2)
3254            using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
3255            // 4^((i+1)/2) - 1
3256            using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
    FractionField<T>::one>;
3257            // (-1)^((i-1)/2)
3258            using altp = typename FractionField<T>::template inject_t<alternate_t<T, (i - 1) / 2»;
3259            using dividend = typename FractionField<T>::template mul_t<
3260                altp,
3261                FractionField<T>::template mul_t<
3262                _4p,
3263                FractionField<T>::template mul_t<
3264                _4pm1,
3265                bernouilli_t<T, (i + 1)>
3266                >
3267                >
3268            >;
3269        public:
3270            using type = typename FractionField<T>::template div_t<dividend,
3271                typename FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
3272        };
3273
3274        template<typename T, size_t i>
3275        struct tan_coeff {
3276            using type = typename tan_coeff_helper<T, i>::type;
3277        };
3278
3279        template<typename T, size_t i, typename E = void>
3280        struct tanh_coeff_helper;
3281
3282        template<typename T, size_t i>
3283        struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) == 0» {
3284            using type = typename FractionField<T>::zero;
3285        };
3286
3287        template<typename T, size_t i>
3288        struct tanh_coeff_helper<T, i, std::enable_if_t<(i % 2) != 0» {
3289        private:
3290            using _4p = typename FractionField<T>::template inject_t<pow_t<T, 4, (i + 1) / 2»;
3291            using _4pm1 = typename FractionField<T>::template sub_t<_4p, typename
    FractionField<T>::one>;
3292            using dividend =
3293                typename FractionField<T>::template mul_t<
3294                _4p,
3295                typename FractionField<T>::template mul_t<
3296                _4pm1,
3297                bernouilli_t<T, (i + 1)>
3298                >
3299                >::type;
3300        public:
3301            using type = typename FractionField<T>::template div_t<dividend,
3302                FractionField<T>::template inject_t<factorial_t<T, i + 1»>;
3303        };
3304
3305        template<typename T, size_t i>
3306        struct tanh_coeff {
3307            using type = typename tanh_coeff_helper<T, i>::type;
3308        };
3309    }
3310
3311    namespace functions {
3315        template<typename T, size_t deg>
3316        using exp = taylor<T, internal::exp_coeff, deg>;
3317
3321        template<typename T, size_t deg>
3322        using expm1 = typename polynomial<FractionField<T»::template sub_t<
3323            exp<T, deg>,
3324            typename polynomial<FractionField<T»::one>;
3325
3329        template<typename T, size_t deg>
3330        using lnp1 = taylor<T, internal::lnp1_coeff, deg>;
3331
3335        template<typename T, size_t deg>
3336        using atan = taylor<T, internal::atan_coeff, deg>;
3337
3341        template<typename T, size_t deg>
3342        using sin = taylor<T, internal::sin_coeff, deg>;
3343
3347        template<typename T, size_t deg>
3348        using sinh = taylor<T, internal::sh_coeff, deg>;
3349
```

```
3353         template<typename T, size_t deg>
3354         using cosh = taylor<T, internal::cosh_coeff, deg>;
3355 
3359         template<typename T, size_t deg>
3360         using cos = taylor<T, internal::cos_coeff, deg>;
3361 
3365         template<typename T, size_t deg>
3366         using geometric_sum = taylor<T, internal::geom_coeff, deg>;
3367 
3371         template<typename T, size_t deg>
3372         using asin = taylor<T, internal::asin_coeff, deg>;
3373 
3377         template<typename T, size_t deg>
3378         using asinh = taylor<T, internal::asinh_coeff, deg>;
3379 
3383         template<typename T, size_t deg>
3384         using atanh = taylor<T, internal::atanh_coeff, deg>;
3385 
3389         template<typename T, size_t deg>
3390         using tan = taylor<T, internal::tan_coeff, deg>;
3391 
3395         template<typename T, size_t deg>
3396         using tanh = taylor<T, internal::tanh_coeff, deg>;
3397     }
3398 }
3399 
3400 // continued fractions
3401 namespace aerobus {
3404     template<int64_t...  values>
3405     struct ContinuedFraction {};
3406 
3407     template<int64_t a0>
3408     struct ContinuedFraction<a0> {
3409         using type = typename q64::template inject_constant_t<a0>;
3410         static constexpr double val = type::template get<double>();
3411     };
3412 
3413     template<int64_t a0, int64_t...  rest>
3414     struct ContinuedFraction<a0, rest...> {
3415         using type = q64::template add_t<
3416             typename q64::template inject_constant_t<a0>,
3417             typename q64::template div_t<
3418                 typename q64::one,
3419                 typename ContinuedFraction<rest...>::type
3420             »;
3421         static constexpr double val = type::template get<double>();
3422     };
3423 
3428     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>;
3431     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>;
3433     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>;
3435     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>;
3436 }
3437 
3438 // known polynomials
3439 namespace aerobus {
3440     // CChebyshev
3441     namespace internal {
3442         template<int kind, int deg>
3443         struct chebyshev_helper {
3444             using type = typename pi64::template sub_t<
3445                 typename pi64::template mul_t<
3446                 typename pi64::template mul_t<
3447                 pi64::inject_constant_t<2>,
3448                 typename pi64::X
3449                 >,
3450                 typename chebyshev_helper<kind, deg - 1>::type
3451                 >,
3452                 typename chebyshev_helper<kind, deg - 2>::type
3453             >;
3454         };
3455 
3456         template<>
3457         struct chebyshev_helper<1, 0> {
3458             using type = typename pi64::one;
3459         };
3460 
3461         template<>
3462         struct chebyshev_helper<1, 1> {
3463             using type = typename pi64::X;
3464         };
3465 
3466         template<>
```

```
3467         struct chebyshev_helper<2, 0> {
3468             using type = typename pi64::one;
3469         };
3470 
3471         template<>
3472         struct chebyshev_helper<2, 1> {
3473             using type = typename pi64::template mul_t<
3474                 typename pi64::inject_constant_t<2>,
3475                 typename pi64::X>;
3476         };
3477     }
3478 
3479     // Laguerre
3480     namespace internal {
3481         template<size_t deg>
3482         struct laguerre_helper {
3483         private:
3484             // Lk = (1 / k) * ((2 * k - 1 - x) * lkm1 - (k - 2)Lkm2)
3485             using lnm2 = typename laguerre_helper<deg - 2>::type;
3486             using lnm1 = typename laguerre_helper<deg - 1>::type;
3487             // -x + 2k-1
3488             using p = typename pq64::template val<
3489                 typename q64::template inject_constant_t<-1>,
3490                 typename q64::template inject_constant_t<2 * deg - 1»;
3491             // 1/n
3492             using factor = typename pq64::template inject_ring_t<
3493                 q64::val<typename i64::one, typename i64::template inject_constant_t<deg»>;
3494 
3495         public:
3496             using type = typename pq64::template mul_t <
3497                 factor,
3498                 typename pq64::template sub_t<
3499                     typename pq64::template mul_t<
3500                         p,
3501                         lnm1
3502                     >,
3503                     typename pq64::template mul_t<
3504                         typename pq64::template inject_constant_t<deg-1>,
3505                         lnm2
3506                     >
3507                 >
3508             >;
3509 
3510         };
3511 
3512         template<>
3513         struct laguerre_helper<0> {
3514             using type = typename pq64::one;
3515         };
3516 
3517         template<>
3518         struct laguerre_helper<1> {
3519             using type = typename pq64::template sub_t<typename pq64::one, typename pq64::X>;
3520         };
3521     };
3522 
3523     namespace known_polynomials {
3525         enum hermite_kind {
3526             probabilist,
3527             physicist
3528         };
3529     }
3530 
3531     namespace internal {
3532         template<size_t deg, known_polynomials::hermite_kind kind>
3533         struct hermite_helper {};
3534 
3535         template<size_t deg>
3536         struct hermite_helper<deg, known_polynomials::hermite_kind::probabilist> {
3537         private:
3538             using hnm1 = typename hermite_helper<deg - 1,
    known_polynomials::hermite_kind::probabilist>::type;
3539             using hnm2 = typename hermite_helper<deg - 2,
    known_polynomials::hermite_kind::probabilist>::type;
3540 
3541         public:
3542             using type = typename pi64::template sub_t<
3543                 typename pi64::template mul_t<typename pi64::X, hnm1>,
3544                 typename pi64::template mul_t<
3545                     typename pi64::template inject_constant_t<deg - 1>,
3546                     hnm2
3547                 >
3548             >;
3549         };
3550 
3551         template<size_t deg>
3552         struct hermite_helper<deg, known_polynomials::hermite_kind::physicist> {
```

```
3553         private:
3554             using hnm1 = typename hermite_helper<deg - 1,
     known_polynomials::hermite_kind::physicist>::type;
3555             using hnm2 = typename hermite_helper<deg - 2,
     known_polynomials::hermite_kind::physicist>::type;
3556
3557         public:
3558             using type = typename pi64::template sub_t<
3559                 // 2X Hn-1
3560                 typename pi64::template mul_t<typename pi64::val<typename i64::template
     inject_constant_t<2>, typename i64::zero>, hnm1>,
3561
3562                 typename pi64::template mul_t<
3563                     typename pi64::template inject_constant_t<2*(deg - 1)>,
3564                     hnm2
3565                 >
3566             >;
3567         };
3568
3569         template<>
3570         struct hermite_helper<0, known_polynomials::hermite_kind::probabilist> {
3571             using type = typename pi64::one;
3572         };
3573
3574         template<>
3575         struct hermite_helper<1, known_polynomials::hermite_kind::probabilist> {
3576             using type = typename pi64::X;
3577         };
3578
3579         template<>
3580         struct hermite_helper<0, known_polynomials::hermite_kind::physicist> {
3581             using type = typename pi64::one;
3582         };
3583
3584         template<>
3585         struct hermite_helper<1, known_polynomials::hermite_kind::physicist> {
3586             // 2X
3587             using type = typename pi64::template val<typename i64::template inject_constant_t<2>,
     typename i64::zero>;
3588         };
3589     }
3590
3591     namespace known_polynomials {
3594         template <size_t deg>
3595         using chebyshev_T = typename internal::chebyshev_helper<1, deg>::type;
3596
3599         template <size_t deg>
3600         using chebyshev_U = typename internal::chebyshev_helper<2, deg>::type;
3601
3604         template <size_t deg>
3605         using laguerre = typename internal::laguerre_helper<deg>::type;
3606
3609         template <size_t deg>
3610         using hermite_prob = typename internal::hermite_helper<deg, hermite_kind::probabilist>::type;
3611
3614         template <size_t deg>
3615         using hermite_phys = typename internal::hermite_helper<deg, hermite_kind::physicist>::type;
3616     }
3617 }
```

# 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